

# A branch-and-dive heuristic for single vehicle snow removal

Roghayeh Hajizadeh<sup>1</sup> | Kaj Holmberg<sup>1</sup>

Department of Mathematics, Linköping Institute of Technology, Linköping, Sweden

**Correspondence**

Kaj Holmberg, Department of Mathematics, Linköping Institute of Technology, SE-581 83 Linköping, Sweden.  
Email: kaj.holmberg@liu.se

**Funding information**

This research was supported by the Swedish Research Council, Grant/Award Number: 2015-04313.

**Abstract**

This paper deals with planning of a tour for a vehicle to clear a certain set of streets in a city of snow. Our previous results on the problem contain a heuristic based on reformulation to an asymmetric traveling salesman problem (ATSP) which yields feasible solutions and upper bounds, and a relaxation of a MIP model for obtaining lower bounds. The goal now is to try to improve the solutions and bounds. In this paper we describe a branch-and-dive heuristic which is based on branch-and-bound principles. We discuss how branching can be done so that the fixations can be utilized in both the relaxation and the ATSP model, and how the search for better solutions can be done. The heuristic has been implemented and applied to real life city networks. The method is shown to outperform two other heuristics for the ATSP with precedence constraints.

**KEYWORDS**

arc routing, asymmetric traveling salesman, branch-and-bound, heuristic, precedences, turning penalties

## 1 | INTRODUCTION

Snow removal is an important problem in Nordic countries. In Figure 1 the maximal snow depth (December to April) in Stockholm (Observatorielunden) is given for each year from 1904 to 2018, and the large variation is evident. (The data are taken from Stockholm Stad [26]. The horizontal line is the average, and the slightly sloping horizontal line is the linear trend.) The total costs for snow removal in the city of Stockholm for each year from 2008 to 2013 (as reported by Stockholm Stad) were 130, 146, 265, 238, 224, and 192 million SEK (10 SEK  $\approx$  1 Euro). Note that this is only for one city, not the whole country, and that Stockholm is not situated in the northern parts of Sweden, where there is much more snow. The costs of course depend on the maximal snow depth, but also on how and when the snow comes. Large amounts of snow in a short time give high costs. It seems hard to predict the amount of snow next year, as a year with much snow is often followed by a year with much less, and vice versa. Using last year's plan is often not possible, so one must be able to quickly plan new tours.

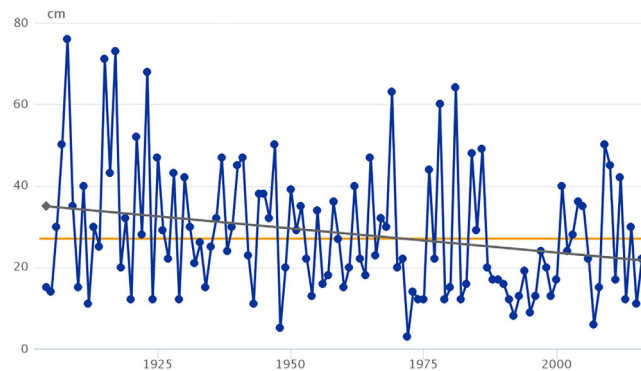
Our work is aimed at Swedish circumstances, as specified in communication with the contractors that do the work, and with the municipalities that pay for it. In this paper we deal with a few aspects (described later) that are different from previous work from other countries.

We consider removing snow after a snowfall, which means that each street only needs to be cleared once. Furthermore, we consider snow removal in urban areas, not in rural areas. In general, the streets in a city need to be cleared of snow by a limited number of vehicles within a certain time. Tours for the vehicles should be planned in order to minimize the time or/and cost. Snow should be cleared from all the streets including bicycle paths, pedestrian paths, bus stops, intersections, and so forth.

Roghayeh Hajizadeh and Kaj Holmberg contributed equally to this study.

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2020 The Authors. *Networks* published by Wiley Periodicals LLC.



**FIGURE 1** Snow depths in Stockholm over the years [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]

In practice (in Sweden), a city is divided into different areas and for each area a contractor takes care of the snow removal. It is sufficient to consider the snow removal problem on one area at a time, since the contractors do not cooperate. In [13] the allocation of streets to identical vehicles is studied. Now we consider the optimization of the snow removal tour facing a single vehicle (a single snow remover). This means that the relevant size of an area to be covered is decreased even more, since the total area of a contractor is usually divided up between the vehicles. We conclude that the networks we need to deal with are not very large. However, there are various details in the problem which make it quite complex.

Turning a vehicle around takes more time than driving straight, so there is a kind of turning penalty. The number of sweeps required to clear a street of snow is different, depending on the type of street. Also crossings and turning spaces need to be cleared. The snow removal problem thus contains both multiple arc routing and node routing. Finally, a vehicle can use streets for transporting itself without removing snow.

The structure of the paper is as follows. In Section 2, details of the snow removal problem are given and we describe how to pass from a real problem to a mathematical model. Related work is discussed in Section 3. In Section 4, a method to find a feasible solution and a relaxation of the original problem are given in order to find upper and lower bounds. In Section 5, we describe the branching procedure as well as the branch-and-bound based heuristic, which we call “branch-and-dive,” since it is more a question of sequential fixations, where optimality cannot be guaranteed. Computational results are discussed in Section 6, and some conclusions are presented in Section 7.

## 2 | SPECIFYING THE OPTIMIZATION MODEL

In order to solve the problem, a mathematical model is needed. First the total work is divided into small unit tasks, where clearing a street usually requires several tasks. Examples of unit tasks are: clearing the right side of the street, clearing the left side of the street, clearing the middle of the street, clearing a turning space, clearing a crossing, and so on. Some unit tasks, like clearing a crossing, are done on nodes and some, like clearing the right side of the street, are done on links in the street network. (We use “link” for a street, not to confuse it with the arcs in the graph in which we will solve the problem.)

The set of tasks on a link or a node depends on the kind of street or crossing. For a node it depends on the number of adjacent streets. For links, the following classification of streets, determined by the number of the sweeps needed, is used.

- 1 Narrow streets which need one sweep which can be done in any direction. There is only one undirected task on this kind of streets.
- 2 Small ordinary streets which need two sweeps, one on the right side and one on the left side. Two directed tasks are defined on these streets.
- 3 Normal ordinary streets which need three sweeps, one in the middle, one on the right side and one on the left side.
- 4 Larger ordinary streets which need four sweeps, two in each direction.

In the streets with three sweeps, the middle sweep is done to enable traffic as soon as possible and must be done before the side sweeps. In addition, a middle sweep can be done in any direction. Hence, one undirected task (the middle sweep) and two directed tasks (the side sweeps) are defined on the streets with three sweeps. Having a higher number of sweeps and a mix of undirected and directed tasks makes the problem complicated.

The undirected tasks handled the following way. For any undirected task, we define two different directed tasks, one in each direction and let them form a *group*. In addition, each directed task is considered as a group with only one task. Then the requirement is that exactly one task from each group should be done.

A street must be cleared before the crossing and turning areas connected to that street. Specifically, all the tasks in-going to a node must be done before the node task is done, but outgoing tasks can be done after the node task. Also, the middle sweep must be done before the side sweeps. Therefore, precedence constraints are needed.

Various indata are required. The time (cost) for each task, the time (cost) for transportation on a street (without removing snow), as well as start and end node of each task are needed. Turning penalties can now be seen as times needed for switching between two tasks, and these times are needed. These data can be found in different databases. In [11,12] it is described how to get street networks and corresponding task characteristics from OpenStreetMap.

In [10] a detailed mixed integer programming model (P1) for the complete snow removal problem with several vehicles is given. Computational results are reported and show that the model is not solvable for realistic city networks. In order to find lower bounds, different relaxations are considered by removing constraint sets and/or using aggregate variables. Among the relaxations, one model, P2, was found to give good lower bounds in rather short time. The model uses aggregate variables and omits switching and precedences constraints.

In [15] the problem is restricted to one vehicle, and the problem, including all constraints, is formulated as an asymmetric traveling salesman problem (ATSP) in an extended graph, with additional constraints. In addition, a heuristic for finding a feasible solution is given. Model P2S, which is P2 restricted to one vehicle, is given in Section 4.2 and we will briefly explain the reformulation to ATSP in Section 4.1.

Thus we have a method which gives (rather good) feasible solutions, and upper bounds on the optimal objective function value, as well as a relaxation which gives a lower bound. The question now is how to design a procedure, using these tools, for improving the solution and bounds.

### 3 | RELATED WORK

Snow removal in Canada and North America is described in [19,23]. Many aspects of snow removal are covered by the suite [20-23]. The aspect of the turning penalties has been studied in [3,5,6,25].

In [2], a branch-and-cut method for ATSP with precedence constraints is presented, using many classes of valid inequalities. Problems with up to 380 nodes (or 444 without precedences) are solved. Moon et al. [17] present a genetic algorithm for the same problem, using topological sort and a new crossover procedure. Sung and Jeong [27] present an adaptive evolutionary algorithm using topological sorting based on precedences, with the expressed purpose of only working with feasible solutions. Problems with up to 100 nodes are treated.

The sequential ordering problem (SOP) is a closely related problem. Gambardella and Dorigo [8] describe a hybrid ant colony system for the SOP, using a new local search method, SOP3. The method improved many best-known solutions for TSPLIB, with up to 380 nodes. In the master thesis [1], two-tier methods for the SOP are tried, combining population based methods (particle swarm optimization, genetic algorithms) with local search techniques. Problems with up to 100 nodes are treated.

In [7], an improvement heuristic framework for the generalized traveling salesman problem (GTSP) is presented in the context of laser cutting. The GTSP with precedences is treated in [4] in the context of toolpath optimization for minimizing airtime during machining, using a SOP-heuristic. In the master thesis [24], some methods for the GTSP with precedences are tested, including local search, path preserving 3-opt, and ant colony optimization. Problems with up to 216 nodes are solved.

### 4 | OBTAINING BOUNDS

The goal is now to design a procedure, using our available tools for finding a feasible solution and an upper bound, and a relaxation, giving a lower bound, for improving the solutions and bounds.

We consider the well-known branch-and-bound principles, which however cannot be directly applied to our problem. The main reason is a belief that the upper bounds are rather good, while the lower bounds are weaker. Therefore, it would probably not be fruitful to base the branch-and-bound only on the relaxation. The primal ATSP heuristic should be guided by the branchings, so that it provides even better solutions and better upper bounds (in addition to better lower bounds from the relaxation).

One may consider different possibilities of how to branch, but the crucial aspect is that the branching can be introduced in both models, P2S and the ATSP, which are described briefly in the following sections.



FIGURE 2 Task arc

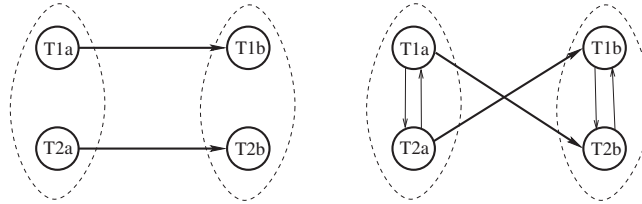


FIGURE 3 Graph structure for a group for an undirected task

#### 4.1 | Finding a feasible solution

The solution method for finding a feasible solution based on transforming the problem to an ATSP in [15] is briefly explained below. The graph  $G = (N, L)$  corresponding to the city network has node set  $N$  (which we call “real nodes,” not to be confused with the nodes in the directed graph we will construct) and link set  $L$  (which contain both edges  $E$  and arcs  $A$ ).

A directed task graph,  $G^T = (N^T, A^T)$ , is created as follows. Two nodes for each task and an arc from the starting node to the ending node, see Figure 2, with arc cost equal to the task cost, are introduced. Then arcs are added from the end node of each task arc to the start node of all tasks that may succeed it. The costs of these arcs are set to the costs of the shortest paths between the corresponding real nodes. In other words, when a task is done and another task is chosen to be done next, the vehicle has to be moved between the two corresponding real nodes along a shortest path. If those two nodes correspond to the same real node, the cost is the switching time between the two tasks. To handle precedence constraints, the arcs which should not be used because of precedences are not added to the extended graph. This increases the probability for the precedences constraints to be satisfied. For example, if task 1 should precede task 2, arc  $(T2b, T1a)$  is not added.

Undirected tasks which are formulated by groups lead to the asymmetric generalized traveling salesman problem, AGTSP (where exactly one node from each set must be visited) which can be transformed to a normal ATSP [18]. In order to do this, arcs with cost zero are added in a cycle between the nodes in a set. Then, the outgoing arcs are switched one step along the cycle. For instance, consider an undirected task which yields to two directed tasks 1 and 2 and only one of them must be done. Figure 3 shows the reformulation of this group to the normal ATSP.

Note that the diagonal arcs in Figure 3 are the only arcs leading out from the first set, and the only arcs leading into the second set. Therefore, at least one of them must be used. Furthermore, at most one of the zero cost arcs in a group can be used (otherwise a node would be visited twice). The path  $T1a-T2a-T1b-T2b$  means doing task T1. In order to discourage the cycle from using both diagonal arcs, a fixed cost can be added to both of them. (In [18] a fixed cost larger than the sum of all arc costs is used. In our experiments a much smaller costs was sufficient.)

The vehicle should start from the depot and go back to the depot at the end of the tour. To handle this, an extra node is added which represents the depot. Then, arcs are added from this node to the starting node of the tasks which do not have any predecessors as well as from the ending node of the tasks which do not have any successors. The costs of these arcs are the length of the shortest path between corresponding nodes. (By splitting the depot node into two, one where the tour starts, and one where it ends, the problem can be equivalently stated as a SOP.)

The resulting task graph  $G^T$ , in which the problem will be solved, has  $|N^T| = 2p + 1$ , where  $p$  is the number of tasks, which is between  $4|L| + |N|$  and  $|L| + |N|$ , depending of the types of the streets. The size of the constructed task graph is thus much larger than the city networks, but the number of nodes is  $O(n^3)$ , where  $n$  is the number of real nodes, so the increase is polynomial. The increase in size is the price we have to pay to get the problem into a solvable form. Comparing to other AGTSPs, our graph has a very specific structure. The groups are very small, and many nodes do not belong to a group (with more than one node). We believe that it is necessary to utilize this special structure to be able to solve the problem.

The resulting ATSP is solved by using the LKH heuristic [9], which contains more heuristic improvements than we could realistically implement ourselves. The solution to ATSP gives an order for doing the tasks. If the order of the tasks is given, one can find a feasible solution by increasing the time step-by-step and build up the tour for the vehicle.

The order could however include failed precedences, either for node tasks or from incorrect choice of tasks from groups. In [15] a procedure for reordering the problematic tasks, called a “retasking” procedure is described. Failed precedences for node tasks are easy to correct, by simply moving the node tasks to the last time the node is visited in the sequence. Failed precedences in groups are harder to correct, and sometimes the vehicle needs to switch direction of a sweep, which means that it has to take a small tour in order to approach the link from the other side. If it was known in advance which of the two tasks in a group

that shall be done, this problem would not appear. Therefore, tasks in groups with failed precedences are good candidates to branch on.

## 4.2 | Mathematical model for a relaxation

Model P2, first developed in [10], is a relaxation of our total model with aggregate variables and without switching and precedences constraints. It was found to give relatively good lower bounds compared to other relaxations rather quickly. It is adapted to one vehicle in [15], and the mathematical model, here called P2S, is given below. It is based on the city network,  $G = (N, L = E \cup A)$ , where  $T$  is the set of tasks,  $\Gamma$  is the set of task groups, and  $T_g$  is the set of tasks that belong to group  $g$ . The number of nodes is denoted by  $n$ , the number of links by  $m$ , the number of tasks by  $p$ , and the number of groups by  $pg$ .

The starting node and the ending node of a link  $l$  are denoted by  $s_l^L$  and  $e_l^L$ , and the starting node and ending node of task  $j$  are denoted by  $s_j^T$  and  $e_j^T$  respectively. In addition, the set of tasks starting at node  $i$  is  $S_i^T = \{j \in T : i = s_j^T\}$ , the set of tasks ending at node  $i$  is  $E_i^T = \{j \in T : i = e_j^T\}$ , the set of links starting at node  $i$  is  $S_i^L = \{l \in L : i = s_l^L\}$ , and the set of links ending at node  $i$  is  $E_i^L = \{l \in L : i = e_l^L\}$ . Furthermore, the time needed for the vehicle to do task  $j$  is  $d_j^T$ , and the time needed for transporting the vehicle along link  $l$  is  $d_l^L$ . Finally  $D$  is the depot where the vehicle should start and end.

The variables are the following.  $z_j^T = 1$  if the vehicle does task  $j$ .  $z_l^{FL}$  is the number of times the vehicle is transported forwards on link  $l$  (from  $s_l^L$  to  $e_l^L$ ) and  $z_l^{BL}$  is the number of times the vehicle is transported backwards on link  $l$  (from  $e_l^L$  to  $s_l^L$ ). Model P2S is given below.

$$\begin{aligned} \min \quad & \sum_{l \in L} d_l^L z_l^{FL} + \sum_{l \in E} d_l^L z_l^{BL} + \sum_{j \in T} d_j^T z_j^T \\ \text{s.t.} \quad & \sum_{j \in T_g} z_j^T = 1 \quad \forall g \in \Gamma \end{aligned} \quad (1)$$

$$\sum_{j \in E_i^T} z_j^T + \sum_{l \in E_i^L} z_l^{FL} + \sum_{l \in S_i^L \cap E} z_l^{BL} = \sum_{j \in S_i^T} z_j^T + \sum_{l \in S_i^L} z_l^{FL} + \sum_{l \in E_i^L \cap E} z_l^{BL} \quad \forall i \in N, \quad (2)$$

$$\sum_{j \in S_D^T} z_j^T + \sum_{l \in S_D^L} z_l^{FL} + \sum_{l \in E_D^L \cap E} z_l^{BL} \geq 1, \quad (3)$$

$$\sum_{j \in E_D^T} z_j^T + \sum_{l \in E_D^L} z_l^{FL} + \sum_{l \in S_D^L \cap E} z_l^{BL} \geq 1, \quad (4)$$

$$z_j^T \in \{0, 1\} \quad \forall j \in T, \quad (5)$$

$$z_l^{FL}, z_l^{BL} \geq 0, \text{ integer} \quad \forall l \in L. \quad (6)$$

The objective function contains the cost for transportation (forward and backward) and for doing the tasks. Constraint (1) states that one task in each group is done, constraint (2) is node equilibrium constraints, and constraints (3) and (4) guarantee that the vehicle passes the depot. Constraint (1) actually fixes  $z_j^T = 1$  for the groups that contain only one task.

## 5 | THE BRANCH-AND-DIVE APPROACH

The goal is to use both the model P2S which gives a lower bound and the ATSP based heuristic which gives a feasible solution and an upper bound. Although P2 was found to give better lower bounds in reasonable time than the other relaxations tried in [10], the bounds still seem to be rather weak. The ATSP heuristic is designed to always try to produce a feasible solution, so if there are failed precedences, it tries to correct them. This in general leads to a solution without failed precedences, but worse than the initial one. The question now is how these tools can be used to improve the solution and bounds, and one possible way is to construct a branching procedure around the models.

The aim of a standard branch-and-bound method for MIP problems is to improve bounds and find an optimal solution. In addition, branching is based on only one problem, the LP-relaxation. We have two models, P2S and ATSP, and branching in P2S improves the lower bound and branching in ATSP improves the solution and the upper bound. Hence, it is necessary that fixations induced by the branching can be introduced in both problems.

## 5.1 | How to branch?

The only aspect that stops our ATSP heuristic from giving the optimal solution is the failed precedences for groups, so this seems like a perfect candidate for branching. In the ATSP based heuristic, first the ATSP is solved, often yielding a solution with failed precedences. Then the retasking yields a feasible solution. Before the retasking there are problematic variables associated with the tasks in a group with failed precedences. One can branch on these variables, that is, fix which task of a group to do ( $z_j^T = 1$ ) in one branch and fix not to do it ( $z_j^T = 0$ ) in the other branch. Since our groups only contain two variables, the other variable also will be fixed. This branching can improve the solution of ATSP. It can also be used in P2S, since it fixes  $z^T$ -variables, and may therefore increase the lower bound.

Since P2S may become large for real life problems, an option is to solve the LP-relaxation of P2S to get a lower bound. That might yield noninteger values for some variables, and then another possibility could be to branch on a noninteger  $z_j^T$ . Technically such a branching could be imposed on the ATSP heuristic too, but it is not certain that the group is problematic when it comes to precedences, so the branching may have no effect on the ATSP solution. At present, proving optimality within reasonable time seems to be out of reach, so our main goal is to find better solutions (i.e., upper bounds). Since this way of branching might not help in this respect, we will not use it.

There are other possibilities of branching if we only consider one of the problems, such as over the sequence variables in the ATSP, which however are not present in P2S, or to branch over transportation. One could set  $z_i^{FL} \geq 1$  (or  $z_i^{BL} \geq 1$ ) in one branch and fix  $z_i^{FL} = 0$  (or  $z_i^{BL} = 0$ ) in the other branch. However, the question of which transportation variable to select to branch on does not have a good answer.

Summing up, it seems clear that the best way of branching is over the groups, as follows. Choose a task  $j$  with failed precedences before retasking in ATSP, and create one branch with  $z_j^T = 1$  and one with  $z_j^T = 0$ . Since  $z_j^T$  had one of these values before the branching, one of the new problems has already been solved. Fixing the variable will not change the solution, but it might change the lower bound. Solving P2S with that fixation would give a lower bound only valid in that node of the tree. There is nothing more that could be done with that branch, so it will simply be cut off.

This leaves the other branch, where  $z_j^T$  is fixed to the other value, and both ATSP and P2S are solved. Hence, there will always be only one branch. This is why we call this method branch-and-dive.

## 5.2 | Cutting branches

In a standard branch-and-bound method a branch is cut if a feasible solution for the original problem is found, if there is no feasible solution for the current problem, or if it is certain that the solution will be worse. In our method, we already have a feasible solution in one branch and the goal is to improve the feasible solution and bounds on the objective value. Hence, when the lower bound is close enough to the upper bound, the method is terminated. Since there is no guarantee that the gap can be made small enough, there must be other stopping criteria as well.

The fixations are implemented by changing the ATSP graph, which can be seen as a suggestion to go in another direction. Sometimes it is not possible to go in the other direction, and as the ATSP heuristic always tries to find a solution, an invalid solution may be obtained, using artificial arcs, with a very high cost. Then the question is if we should cut the branch, or keep on trying to increase the lower bound.

In addition, if a feasible solution is not found for the LP-relaxation of model P2S, the method must come to a halt, because the relaxation has no feasible solution. For practical reasons there is also an upper bound on the number of branchings.

Since there is just one branch, cutting the branch is equivalent to stopping the algorithm. Therefore, if at least one of the following criteria is satisfied, the algorithm is terminated. Here Lowb is the lower bound and Upb the upper bound. In the computational tests, we used  $\eta = 100$  and  $\epsilon = 0.01$ .

- Error =  $(\text{Upb} - \text{Lowb})/\text{Upb} \leq \epsilon$ , for some small positive  $\epsilon$ .
- The number of branchings is more than  $\eta$ .
- The LP has no feasible solution.
- No solution is found in the ATSP.
- There are no failed precedences.

## 5.3 | The branch-and-dive method

In each iteration, the following is done. First the ATSP is solved to get an upper bound. Both the tour before the retasking, which usually is infeasible, due to violated precedences, and the feasible tour after the retasking are saved. The upper bound is obviously the cost of the feasible solution, while the violated precedences are used for branching. Since branching on failed

precedences in the tour before retasking was chosen, the groups with failed precedences are checked and the first task  $j$  with failed precedences selected to branch on. Let  $\bar{z}_j^T$  be the value obtained in the solution.

In one branch, the task to do in the group is the same as was obtained in the solution, that is, fix  $z_j^T = \bar{z}_j^T$ , and in the other, the opposite task is done, that is, fix  $z_j^T = 1 - \bar{z}_j^T$ . Usually we already have a feasible solution in the first branch, obtained from the ATSP solution.

Then we fix the other task in that group, dive down in the second branch, that is, the opposite of what the ATSP solution gave, and solve both the ATSP and (the LP-relaxation of) P2S, and repeat the above. There is always one branch in the tree, and at each stage we need to save the branchings and the upper and lower bounds. The detailed algorithm is seen in Algorithm 1.

---

**Algorithm 1.** Branch-and-dive
 

---

```

1: Upb = +∞, Lowb = -∞
2: Failed_preced = ∅
3: Fixed = ∅
4: Branch = 0
5: Optional: Solve (the LP-relaxation of) P2S, yields objective value  $v^{P2S}$ . Set Lowb =  $v^{P2S}$ .
6: while Stopping criteria not satisfied do
7:   Solve ATSP, yields tour1: tour before retasking, tour2: tour after retasking,  $v^A$ : cost of tour2.
8:   update Failed_preced
9:   if  $v^A < \text{Upb}$  then
10:    Upb =  $v^A$ 
11:    Set Best tour to Tour2.
12:   if Failed_preced  $\neq \emptyset$  then
13:    Branch = Branch + 1
14:    Pick first task  $j$  from Failed_preced
15:    Fixed = Fixed  $\cup \{z_j^T = 1 - \bar{z}_j^T\}$ 
16:    Solve (the LP-relaxation of) P2S, yields objective value  $v^{P2S}$ .
17:    if  $v^{P2S} > \text{Lowb}$  then
18:      Lowb =  $v^{P2S}$ 
19:   Error = (Upb - Lowb)/Upb

```

---

The stopping criteria are those mentioned in Section 5.2. The retasking is described in Section 4.1. *Failed\_preced* is simply a list of groups with failed precedences, which is recalculated in each iteration. The optional first lower bound obtained before the branching is a global lower bound, while the lower bound obtained after branchings is only valid for that node in the branch-and-bound tree. It can be used to cut that branch, but not to estimate the quality of a solution found in another branch.

Variations of the method which we have tried are:

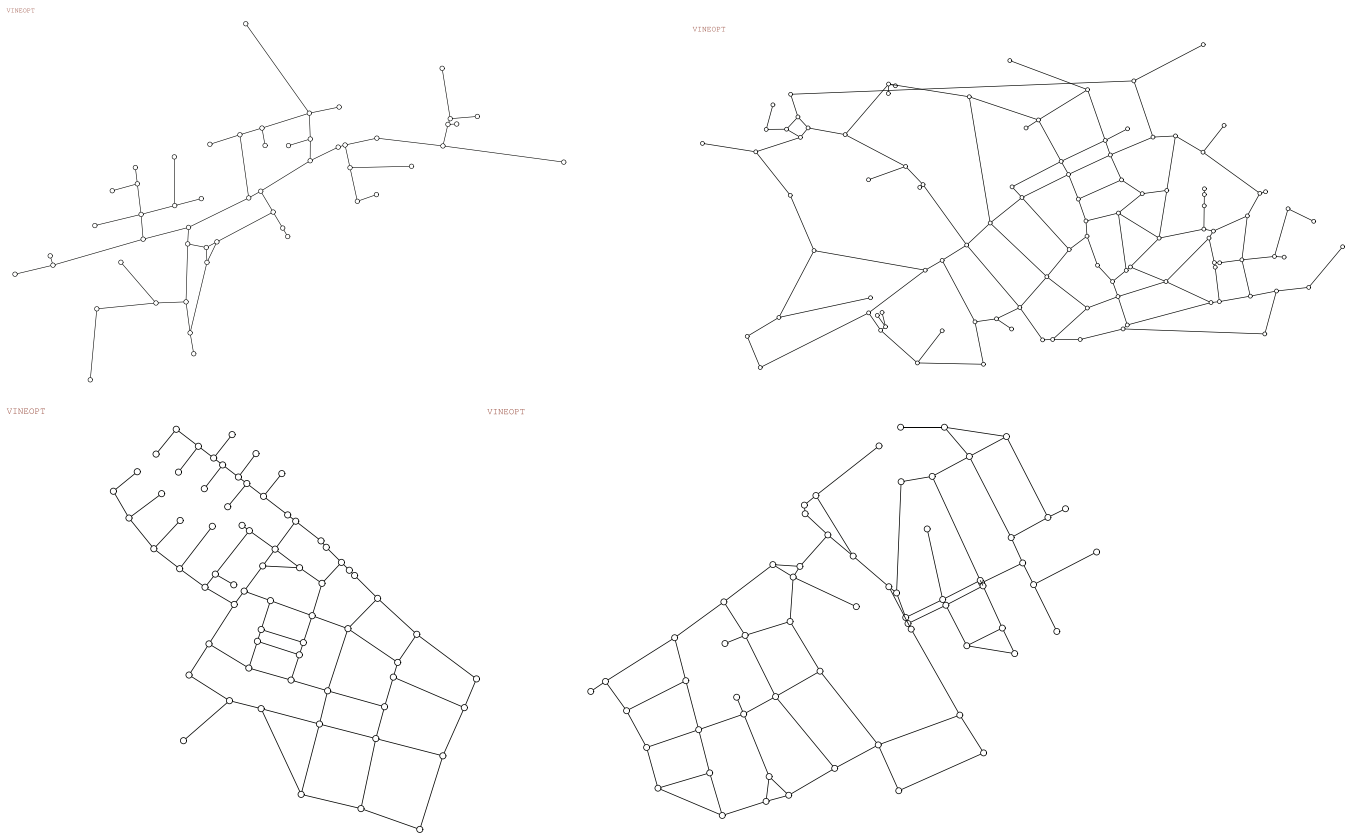
- 1 Standard (as specified in the algorithm, without the optional step).
- 2 Var 1: Solve the integer problem P2S in the first optional step (slower).
- 3 Var 2: Cut off the branch/stop the method when Upb increases (faster).

Variation 1 is simply an attempt to get a better lower bound, although it takes more time. Variation 2, on the other hand, is an attempt to speed the method up. One might think that when the upper bound starts to increase, there will be no more improvement. However, our tests show that sometimes a later improvement may occur.

## 6 | COMPUTATIONAL TESTS

The branch-and-dive method was implemented in Python, and was run on a Lenovo Legion Y520T-25IKL Desktop, Intel Core I7-7700, 3.6 GHz, 8 GB DDR4, SSD 256 GB, running Fedora Core 30. The asymmetric traveling salesman networks were constructed with an implementation in C, and solved with LKH [9]. Model P2S was solved with GLPK 4.64. Data communication was made via files.

A number of small cities in Sweden were obtained from OpenStreetMap, see [12,14] for details about the instances and the extraction procedure. Data for the cities are available in [16]. The city networks were then divided into parts suitable for one vehicle. Some of the networks are pictured in Figure 4 and the sizes of the instances are given in Table 1, where also  $n^T$ , the number of nodes in the ATSP, and  $m^T$ , the number of arcs in the ATSP, are given. We note that the number of nodes in the ATSP is up to 1000 and the number of arcs is up to 250 000.



**FIGURE 4** Askeby: 53 nodes, 56 links; Borensberg-b: 103 nodes, 131 links; Skanninge-a-1: 68 nodes, 86 links; Vikingstad-a-2: 63 nodes, 86 links [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]

For a street network, tasks and groups are created in the code for the ATSP subproblem. All switching and precedences are considered. In [15] two cases are discussed, one with the middle sweep and one without for certain streets. Here the middle sweep is included.

To initialize the method, the ATSP is solved. Both tours before retasking and after that are saved. The first one is to find the failed precedences and selecting branching item and the second one is to find a feasible solution and an upper bound. Likewise, the ATSP with consideration to fixation is solved in each iteration in order to update feasible solution and upper bound. The ATSP heuristic must follow the fixation in each iteration.

In order to get lower bounds, model P2S is used. Solving this model to optimality in a short time is often not possible, the LP-relaxation of P2S is used. We also consider solving the integer problem P2S, but with time limited to 60 seconds. If a MIP-solver is interrupted before it is finished, it usually returns the objective function of the best integer solution. However, that is not the lower bound we are looking for, so care has to be taken to correctly extract the best lower bound obtained.

## 6.1 | Initial tests

We first did some computational experiments with different versions of the method for the small city of Askeby, 518 inhabitants, 53 nodes, 56 links, which gives 275 tasks, 445 nodes, and 49 398 arcs in the ATSP.

The standard setting gave the following result for Askeby. The first upper bound was 400, and the first lower bound was 318. After the first branching, the upper bound decreased to 398, and the lower bound increased to 319. Further branchings did not improve the upper bound, but the lower bound was increased to 320, 321, and finally 322. After that, GLPK failed to find a feasible solution, and the method was terminated. Overall six ATSPs and six LPs were solved, and the total time was 19 seconds.

Then a version where the calculations of lower bounds were omitted was tried, which is possible since the branching procedure is based on the ATSP-solution. If the lower bounds do not really help, it might be better to focus on the upper bound, and not spend time on calculating lower bounds. However, that did not improve the upper bound, and we got no lower bound. On the other hand, the method continued to branch until the ATSP became infeasible, so this time we solved 10 ATSPs (and no LPs), and the time increased to 29 seconds. One conclusion of this is that the time for solving ATSPs dominates, while the time for solving LPs is much less.

Another variation tried was to cut the branch when the upper bound started to increase. This means only doing the first few iterations compared to the standard setting, so the final result was an upper bound of 398 and a lower bound of 320, and a solution time of 9 seconds.

If the lower bound is weak, it might be better to solve the integer problem P2S, instead of the LP-relaxation, but with a time limit of 30 seconds. This gave a final lower bound 332, which confirms that 320 is a weak lower bound. The total time increased to 168 seconds.

We also started a run with the integer problem P2S, without time limits. It was interrupted after a couple of hours, when the first MIP had not been solved. At that time the upper bound was 360 and the lower bound 335 for that MIP, and there were thousands of remaining branches, so we conclude that it is not possible to solve this MIP exactly.

A final test was to solve the integer version of P2S initially, with a time limit of 60 seconds, but use the LP-relaxation after the branching. This gave the global lower bound 330, and took 79 seconds (out of which 60 seconds were spent on the integer problem).

We draw the following conclusions of these tests. The lower bound can be improved by using an integer problem at least once, for a limited time, but it is rather time consuming. The lower bounds from only LP-problems increase gradually, but are quite weak. However, an infeasible LP-relaxation is a good signal to terminate the algorithm.

We decided to use the standard setting for the main test, but also use two of the variations above, namely to solve one IP-problem in the beginning to get a better lower bound (Var 1), and to cut off when the upper bound starts to increase (Var 2).

## 6.2 | Comparison with other methods

There are other methods that might be used within our solution approach, see Section 3. However, our graph has a very specific structure. We are not looking for (or proposing) a general method for GTSP with precedences. Instead we are trying to use the structure of our problem as efficiently as possible.

There is also an issue about size. Consider as an example the small city of Askeby, with 53 nodes and 56 links. It gives 275 tasks, and the task graph has 445 nodes and 49 398 arcs. It should be noted that the construction for handling the groups adds no nodes, only a few arcs. This is actually larger than the maximal sizes mentioned in Section 3. Our largest problem has 1000 nodes, and it seems unlikely that the published approaches will be able to handle that in a reasonable time.

After the construction of  $G^T$  the problem is an ATSP with precedences, and we have compared our method to two methods for that problem, namely an adaptive evolutionary algorithm described in [27], and an ant colony hybrid method including a new local search method, SOP3, described in [8], both claimed to be better than other methods at the time of publication.

These methods are probably designed for problems in general networks (and will probably work best on complete graphs), while our networks have a quite specific structure. It turned out that these methods were not capable of solving our city instances, so we used a set of smaller instances, mini, for comparison. Details about these instances can be found in [10] (there called “ex”). They are based on smaller artificial maps, but the networks are constructed as described in this paper, so the graph structure is still there.

The methods were implemented in Python, using numpy for matrix calculations and random selection and networkx for graph creation and topological sort, and the results are presented in Table 1. In the implementation of the SOP3 local search heuristic, we had help from the description in [24]. In the table *Upb* denotes the upper bound, *Lowb* the lower bound, and *Time* the solution time in seconds.

The smaller problems were solved accurately by all methods, but as soon as the size starts increasing, the solutions of the two heuristics start to deteriorate, and the solution times increase. In Figure 5, a green-yellow dot denotes the branch-and-dive method, a blue plus denotes the ant colony method, and a red cross denotes the evolutionary method. To the left in the figure, the objective function values are given, and to the right the solution times in seconds. At the top of the figure, all three methods are given, while below the worst of them in each graph is removed, to enable better comparison of the other two.

It seems that the ant colony method gives better solutions than the evolutionary algorithm, but needs significantly more time. Since the largest of the mini-instances is about the same size as the smallest of our city-instances, we deduce that these methods cannot be used to solve real life instances. (Obviously this conclusion only holds for the specific graph structure we have.)

## 6.3 | Main tests

The main computational tests were made on the set of small cities. In Table 2 the bounds obtained without branching are given, for comparison, to see what improvements are obtained. Here *Upb* denotes the upper bound, *Lowb* the lower bound and *Err* the relative gap  $(Upb - Lowb)/Upb$ . The results of the three variations of the branch-and-dive method can be found in Table 3. Please note that the lower bounds are the final lower bounds, after branching, which are not global lower bounds, so the errors reported refer to the last node in the tree, except for Var 1, where the lower bounds are found before branching, and thus are

TABLE 1 Comparisons with other codes on smaller problems

Problem			Branch-and-dive			Adapt Evo Alg		Ant colony hybrid	
	$n^T$	$m^T$	Upb	Lowb	Time	Upb	Time	Upb	Time
mini-01a	10	21	13.0	13.0	0.363	13.0	0.196	13.0	0.574
mini-01b	10	19	13.0	13.0	0.027	13.0	0.212	13.0	0.559
mini-01c	14	37	13.0	13.0	0.028	17.0	0.249	17.0	0.736
mini-02a	16	57	19.0	17.0	0.044	22.4	0.297	19.0	1.282
mini-02b	16	53	19.0	19.0	0.027	22.8	0.317	19.0	1.144
mini-02c	20	85	19.0	19.0	0.040	23.8	0.368	23.0	1.879
mini-02d	20	83	19.0	19.0	0.026	26.2	0.401	22.6	1.613
mini-03a	32	241	26.0	25.0	0.072	43.0	0.858	30.0	7.249
mini-03b	32	237	32.0	28.0	0.075	45.8	0.794	33.6	7.045
mini-03c	32	235	31.0	30.0	0.082	48.8	0.842	34.6	6.874
mini-03d	32	235	31.0	30.0	0.053	49.6	0.874	35.6	6.841
mini-03e	44	443	32.0	31.0	0.050	63.2	1.426	40.2	11.688
mini-04a	54	703	40.0	37.0	0.076	97.8	1.914	53.2	35.447
mini-04b	54	693	49.0	48.0	0.072	121.8	1.725	65.4	34.398
mini-04c	54	685	51.0	51.0	0.048	103.6	2.280	72.2	34.255
mini-05a	32	241	23.0	21.0	0.053	40.0	0.850	27.4	8.028
mini-05b	32	231	29.0	29.0	0.031	46.6	1.168	31.8	5.956
mini-05c	52	621	29.0	29.0	0.030	70.4	2.117	42.6	24.804
mini-06a	64	993	48.0	42.0	0.113	99.4	3.115	59.0	57.408
mini-06b	64	983	58.0	51.0	0.118	116.6	2.472	65.4	46.551
mini-06c	100	2393	59.0	59.0	0.053	176.6	7.256	89.4	100.292
mini-08a	80	1541	57.0	54.0	0.220	125.2	4.264	67.8	78.663
mini-08b	80	1541	57.0	54.0	0.222	122.0	4.530	68.2	91.564
mini-08c	80	1541	57.0	54.0	0.325	126.0	4.233	70.2	68.774
mini-09a	142	4971	78.0	71.0	0.444	215.2	14.576	109.6	470.344
mini-09b	142	4919	110.0	103.0	0.149	269.6	14.963	138.6	360.207
mini-09c	142	4919	110.0	103.0	0.147	257.0	15.530	139.0	356.324

TABLE 2 Size of the cities and ATSPs, and results without branching

Name	$n$	$m$	$p$	$pg$	$n^T$	$m^T$	Upb	Lowb	Err	Time
Askeby	53	56	275	220	445	49 398	400.0	318.0	0.205	3.101
Atvid-a-1	91	113	593	455	1005	252 232	668.0	579.0	0.133	22.034
Atvid-a-2	71	96	455	359	769	147 648	547.0	451.0	0.176	9.271
Atvid-a-3	34	49	230	181	393	38 514	256.0	218.0	0.148	2.008
Bankekind	30	35	168	134	277	19 116	213.0	179.0	0.160	1.020
Borensberg-b	103	131	605	485	1005	252 288	760.0	637.0	0.162	19.219
Boxholm-1	90	119	570	449	961	230 642	684.0	574.0	0.161	15.887
Brokind-1	89	101	479	385	781	152 316	649.0	541.0	0.166	10.889
Ekangen-1	74	82	380	309	613	93 822	528.0	434.0	0.178	5.824
Mantorp-b	68	83	400	317	665	110 390	514.0	420.0	0.183	7.683
Norsholm	64	77	368	293	609	92 574	451.0	378.0	0.162	6.254
Rimforsa-a-1	64	74	354	283	581	84 254	482.0	392.0	0.187	4.743
Rimforsa-a-2	30	34	158	128	257	16 460	219.0	181.0	0.174	0.900
Skanninge-a-1	68	86	410	325	685	117 138	526.0	422.0	0.198	7.891
Skanninge-a-2	45	55	263	209	437	47 636	329.0	265.0	0.195	2.623
Ull1	41	43	199	163	317	25 064	298.0	243.0	0.185	1.375
Ull2	44	47	226	182	365	33 224	308.0	257.0	0.166	1.855
Valla-a	40	55	250	200	421	44 220	298.0	251.0	0.158	2.414
Valla	69	85	393	316	649	105 162	504.0	418.0	0.171	6.483
Vikingstad-a-1	32	35	168	135	273	18 570	239.0	192.0	0.197	0.880
Vikingstad-a-2	63	86	407	321	689	118 508	489.0	405.0	0.172	7.327
Vikingstad-a-3	15	17	83	66	137	4658	116.0	92.0	0.207	0.212

TABLE 3 Results for the variations of the branch-and-dive method

Name	Standard				Var 1				Var 2			
	Upb	Lowb	Err	Time	Upb	Lowb	Err	Time	Upb	Lowb	Err	Time
Askeby	398	322	0.19	18.9	398	330	0.17	78.8	398	320	0.20	9.1
Atvid-a-1	668	586	0.12	259.4	668	588	0.12	319.9	668	579	0.13	45.1
Atvid-a-2	533	454	0.15	90.8	533	461	0.14	150.3	533	452	0.15	29.3
Atvid-a-3	248	219	0.12	13.2	248	225	0.09	73.0	256	218	0.15	4.5
Bankekind	207	185	0.11	10.5	207	192	0.07	70.4	207	181	0.13	3.1
Borensberg-b	749	646	0.14	371.3	749	646	0.14	432.0	760	637	0.16	39.0
Boxholm-1	684	576	0.16	105.2	684	584	0.15	173.8	684	574	0.16	33.7
Brokind-1	649	544	0.16	54.6	649	551	0.15	116.6	649	542	0.16	21.9
Ekangen-1	528	440	0.17	49.0	528	445	0.16	110.6	528	435	0.18	12.2
Mantorp-b	510	426	0.16	69.3	510	431	0.16	133.6	510	422	0.17	23.0
Norsholm	451	382	0.15	42.4	451	389	0.14	104.7	451	378	0.16	11.6
Rimforsa-a-1	482	398	0.17	67.0	482	403	0.16	129.7	482	394	0.18	10.3
Rimforsa-a-2	205	187	0.09	5.2	205	193	0.06	65.6	205	183	0.11	2.5
Skanninge-a-1	510	425	0.17	91.1	510	433	0.15	157.2	510	422	0.17	23.4
Skanninge-a-2	303	271	0.11	25.6	303	274	0.10	87.6	303	266	0.12	8.0
Ull1	285	254	0.11	18.2	285	256	0.10	79.3	296	245	0.17	4.5
Ull2	300	271	0.10	26.9	300	271	0.10	88.3	300	259	0.14	5.4
Valla-a	284	260	0.08	43.6	284	261	0.08	105.5	284	253	0.11	10.7
Valla	504	425	0.16	47.1	504	429	0.15	111.1	504	419	0.17	13.2
Vikingstad-a-1	227	200	0.12	10.1	227	204	0.10	70.8	227	193	0.15	2.9
Vikingstad-a-2	485	407	0.16	69.9	485	416	0.14	134.0	485	405	0.16	23.1
Vikingstad-a-3	112	96	0.14	1.2	112	96	0.14	1.6	116	92	0.21	0.5

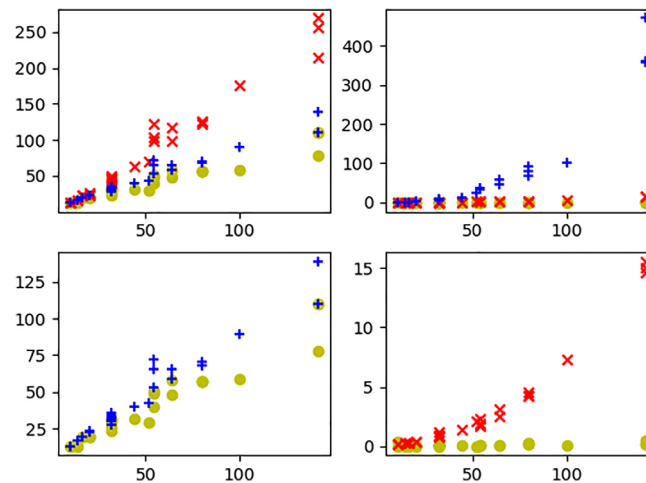


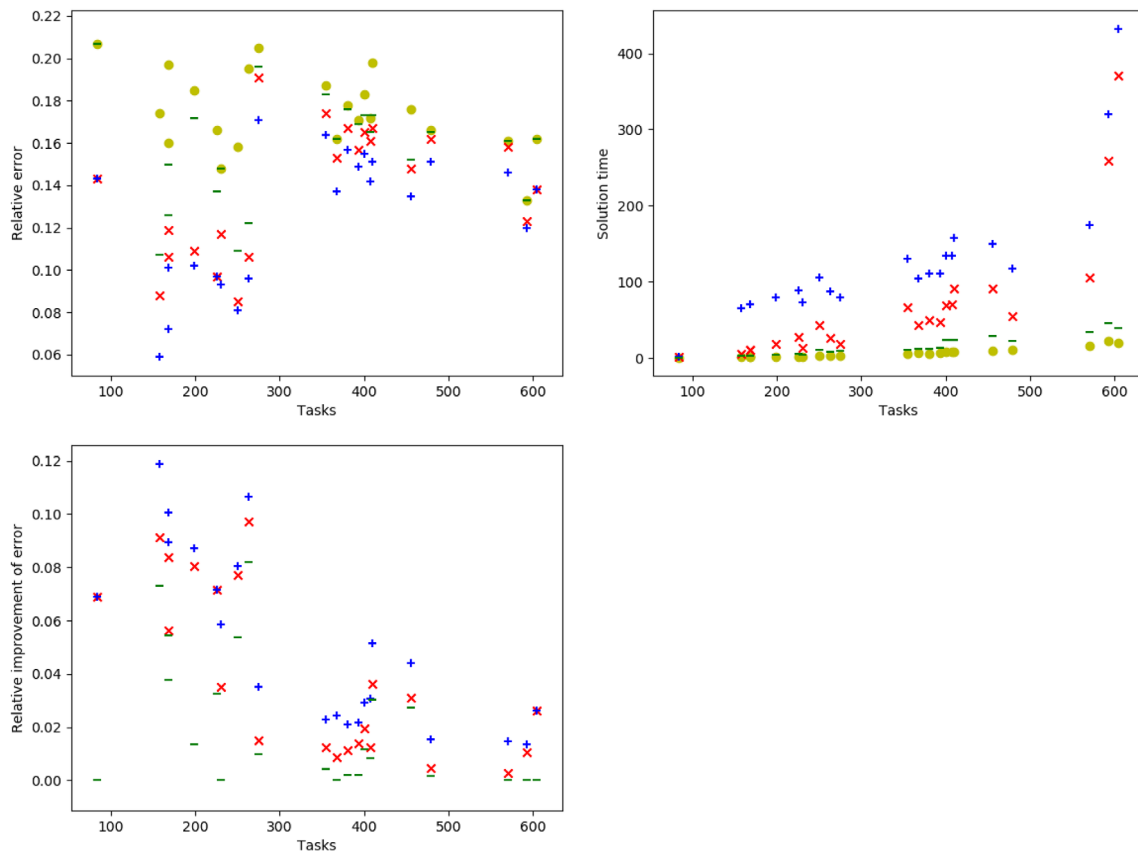
FIGURE 5 Performance, that is, objective function value and solution time vs number of nodes [Color figure can be viewed at wileyonlinelibrary.com]

global lower bounds, and the relative errors correctly reflect the quality of the solutions. (On the other hand, we firmly believe that the upper bounds lie much closer to the optimal values than the lower bounds.)

The results could probably be improved much by using a faster solver than GLPK, like Gurobi or Cplex, for P2S, especially if it enabled solving the integer problem.

Comparing results without and with branching, Tables 2 and 3, using the standard setting, we find that the upper bound is improved in 15 out of 22 cases. The lower bound is improved by the branching in all cases, so the relative error (in the final node) is decreased in all cases. Solving the integer P2S improves most lower bounds more, but the solution time increases significantly. Stopping as soon as the upper bound starts to increase yields four worse upper bounds, while the solution time decreases significantly.

The results are illustrated in Figure 6. Here a green-yellow dot denotes the method without branching, a red cross denotes the standard method with branching, a blue plus denotes the method with spending 60 seconds on trying to solve the initial MIP,



**FIGURE 6** Relative error, solution time and relative improvement of the relative error vs number of tasks [Color figure can be viewed at [wileyonlinelibrary.com](http://wileyonlinelibrary.com)]

and a green dash denotes the method that stopped as soon as no improvement was made. The horizontal axis gives the number of tasks. The vertical axes give relative error, solution time, and the relative improvement of the relative error.

We are still a bit unsure about the quality of the upper bounds. Since P2S is a relaxation of P1, there might still be a gap even if P2S is solved optimally in each iteration. It is possible that the upper bounds that we fail to improve actually are optimal, in which case they obviously cannot be improved by any method. If we were to aim at proving optimality, stronger ways of increasing the lower bound would be required.

## 7 | CONCLUSIONS

We have constructed a branch-and-dive heuristic based on branch-and-bound principles, for the problem of finding the best tour for one snow removal vehicle. The problem includes arc and node routing, multiple link tasks, node tasks, turning penalties, and precedence requirements. Branching is done based on the solutions from an ATSP heuristic and enforced in the ATSP heuristic and in a MIP model used for finding lower bounds. Tests show that the method outperforms two other successful heuristics for the ATSP with precedences. We have run the method on small real life city networks in areas of a size suitable for single vehicles, and find that the results are improved by the branching. Even though the solution times increase, it is still feasible in practice to solve these problems daily. Utilizing the special structure of the graph enables us to find solutions for rather large problems.

### ORCID

Roghayeh Hajizadeh  <https://orcid.org/0000-0002-7402-6292>

Kaj Holmberg  <https://orcid.org/0000-0001-5907-0087>

### REFERENCES

- [1] C.A. Anthonissen, A population-based approach to sequential ordering problems, Master thesis, Stellenbosch University, 2007.

- [2] N. Ascheuer, M. Jünger, and G. Reinelt, *A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints*, *Comput. Optim. Appl.* **17** (2000), 61–84.
- [3] E. Benavent and D. Soler, *The directed rural postman problem with turn penalties*, *Transp. Sci.* **33** (1999), 408–418.
- [4] K. Castelino, R. D'Souza, and P.K. Wright, *Toolpath optimization for minimizing airtime during machining*, *J. Manuf. Syst.* **22** (2002), 173–180.
- [5] J. Clossey, G. Laporte, and P. Soriano, *Solving arc routing problems with turn penalties*, *J. Oper. Res. Soc.* **52** (2001), 433–439.
- [6] A. Corberan, R. Martí, E. Martinez, and D. Soler, *The rural postman problem on mixed graphs with turn penalties*, *Comput. Oper. Res.* **29** (2002), 887–903.
- [7] R. Dewil, P. Vansteenwegen, D. Cattrysse, M. Laguna, and T. Vossen, *An improvement heuristic framework for the laser cutting tool path problem*, *Int. J. Prod. Res.* **53** (2015), 1761–1776.
- [8] L.M. Gambardella and M. Dorigo, *An ant colony system hybridized with a new local search for the sequential ordering problem*, *INFORMS J. Comput.* **12** (2000), 237–255.
- [9] K. Helsgaun, *An effective implementation of the Lin-Kernighan traveling salesman heuristic*, *Eur. J. Oper. Res.* **126** (2000), 106–130.
- [10] K. Holmberg, *Urban snow removal: Modeling and relaxations*, Research report LiTH-MAT-R-2014/08-SE, Department of Mathematics, Linköping University, Sweden, 2014.
- [11] K. Holmberg, *Map matching by optimization*, Research report LiTH-MAT-R-2015/01-SE, Department of Mathematics, Linköping University, Sweden, 2015.
- [12] K. Holmberg, *On using OpenStreetMap and GPS for optimization*, Research report LiTH-MAT-R-2015/15-SE, Department of Mathematics, Linköping University, Sweden, 2015.
- [13] K. Holmberg, *Heuristics for the weighted k-rural postman problem with applications to urban snow removal*, *J. Vehicle Routing Algorithms* **1** (2018), 105–119.
- [14] K. Holmberg, *Prepared test instances extracted from openstreetmap data using different network reductions*, Research report LiTH-MAT-R-2018/04-SE, Department of Mathematics, Linköping University, Sweden, 2018.
- [15] K. Holmberg, *The (over) zealous snow remover problem*, *Transp. Sci.* **53** (2019), 867–881.
- [16] K. Holmberg, 2020. Networks extracted from OpenStreetMap. <http://www.kajh.se/vineopt/problemdata/osm/>.
- [17] C. Moon, J. Kim, G. Choi, and Y. Seo, *An efficient genetic algorithm for the traveling salesman problem with precedence constraints*, *Eur. J. Oper. Res.* **140** (2002), 606–617.
- [18] C.E. Noon and J.C. Bean, *An efficient transformation of the generalized traveling salesman problem*, *INFOR* **31** (1993), 39–44.
- [19] N. Perrier, A. Langevin, and C.A. Amaya, *Vehicle routing for urban snow plowing operations*, *Transp. Sci.* **42** (2008), 44–56.
- [20] N. Perrier, A. Langevin, and J.F. Campbell, *A survey of models and algorithms for winter road maintenance: Part I: System design for spreading and plowing*, *Comput. Oper. Res.* **33** (2006), 209–238.
- [21] N. Perrier, A. Langevin, and J.F. Campbell, *A survey of models and algorithms for winter road maintenance: Part II: System design for snow disposal*, *Comput. Oper. Res.* **33** (2006), 239–262.
- [22] N. Perrier, A. Langevin, and J.F. Campbell, *A survey of models and algorithms for winter road maintenance: Part III: Vehicle routing and depot location for spreading*, *Comput. Oper. Res.* **34** (2007), 211–257.
- [23] N. Perrier, A. Langevin, and J.F. Campbell, *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** (2007), 258–294.
- [24] R. Salman, Algorithms for the precedence constrained generalized travelling salesperson problem, Master thesis, Chalmers University of Technology, 2015.
- [25] D. Soler, E. Martinez, and J. Mico, *A transformation for the mixed general routing problem with turn penalties*, *J. Oper. Res. Soc.* **59** (2008), 540–547.
- [26] Stockholm Stad: Miljöbarometern., Snödjup. <http://miljobarometern.stockholm.se/klimat/klimat-och-vaderstatistik/snodjup/> (Accessed January 10, 2020).
- [27] J. Sung and B. Jeong, *An adaptive evolutionary algorithm for traveling salesman problem with precedence constraints*, *Sci. World J.* **2014** (2014), 1–11.

**How to cite this article:** Hajizadeh R, Holmberg K. A branch-and-dive heuristic for single vehicle snow removal. *Networks*. 2020;1–13. <https://doi.org/10.1002/net.21989>