Computers & Operations Research 37 (2010) 981–990

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

Heuristics for the rural postman problem

Kaj Holmberg*

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

ARTICLE INFO

Available online 25 August 2009

Keywords:
Snow removal
Rural postman problem
Heuristics

ABSTRACT

When addressing the problem of snow removal for secondary roads, a tool for solving the rural postman problem can be very useful. We present some ideas for heuristics for this problem. The heuristics are of the same type as the classical Frederickson heuristic. The ideas concern the order of the main steps in such a method, namely constructing a connected graph with all vertices having even degree, containing all the required edges. We also propose two postprocessing heuristics for improving the tours and removing unnecessary detours. The computational tests show that the ideas are interesting alternatives to the classical approach, and that running times are acceptable. We study problem characteristics that may indicate which method to choose.

1. Introduction

Snow removal is an interesting problem in many countries. The planning typically includes questions about when to treat specific links and how the snow removal equipment should travel. Snow removal for primary roads is well treated in the literature, see for example [15], and is often subject to significant regulations, such as that the roads of a certain type must be cleared before the thickness of the snow reaches a certain limit. Furthermore, it is usually uninteresting to consider moving the equipment without removing snow.

In this paper we deal with snow removal for secondary roads, which is a fairly different problem. By "secondary roads" we here mean narrow roads such as bicycle paths, sidewalks and paths in pedestrian areas, but also other roads that are treated differently from primary roads. Often such roads are handled by other equipment than primary roads, and usually such roads can be considered as undirected for the snow removing equipment.

Another important characteristic of this problem is that all links are not intended to be cleared. Usually there are (primary) roads that are already cleared of snow and can be used for transportation of the equipment. A machine made for a special type of roads, such as bicycle paths, often cannot be used for clearing other roads. Such a machine can often drive much faster while not removing snow, and some links may be used only for transportation of the machine.

One might possibly formulate an optimization model for deciding which links to clear of snow, but we believe that this decision might be better to leave to the planners, due to the amount of "non-mathematical" considerations that must be taken into account. However, it could be very useful with a tool that helps the planner to evaluate different possibilities.

The tool we have in mind is simply to find the best tour of one such machine, given which links to clear. With such a tool, the planner may try a number of different plans. If this tool is going to be used operationally, it must not take too long to get a solution. Preferably it should be a matter of seconds, not minutes. If this tool is used by an operator which has a contract only for a part of a city, the graphs will not be extremely large.

The problem we will solve will be the *rural postman problem*, which is specified as follows. Let $G=(V,E)$ be an undirected graph with vertex set V and edge set E. We assume that G is connected. Also let c_{ij} be the cost of edge (i,j), and assume that $c \geq 0$. The undirected Chinese postman problem (CPP) is to find the minimum cost tour in G that covers each edge in E at least once.

An Euler tour is a tour that uses each edge exactly once. A famous result by Euler is that a connected graph contains an Euler tour if and only if each vertex has even degree. Therefore, a graph where all vertices have even degree is called Eulerian. Furthermore, it is easy to find an Euler tour in an Eulerian graph.

Clearly, an Euler tour is, if it exists, an optimal solution to the CPP. If G is not Eulerian, some edges have to be used more than once, the choice of which is described in the next section.

Now let $R \subseteq E$ be the *required* set of edges. The undirected rural postman problem (RPP) is to find the minimum cost tour in G that covers each edge in R at least once. Unless R forms an Eulerian connected subgraph, solving the RPP amounts to choosing which

* Fax: +46 13 136053.
E-mail address: kahol@mai.liu.se.

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.08.004

edges to add to R in order to form an Eulerian connected graph. (Since parallel edges may be used, we will in this paper by "graph" really mean "multigraph", i.e. allow parallel edges.)

The RPP is *NP*-hard, which is easily seen in an instance where the required edges form a number of separate "villages", as parallels to the traveling salesman problem are obvious. For this reason, we focus our interest on heuristic solution methods, i.e. methods that hopefully give good solutions, but are not guaranteed to give an optimal solution. Furthermore, if such a method happens to find an optimal solution, it will not verify that the solution is optimal, so we will not know that it is optimal.

The perhaps most well-known heuristic for the RPP is that of Frederickson [7]. This method is actually quite efficient, i.e. it finds good solutions fast in general. We will describe the method in a following section. We will study variations of this basic heuristic, and do computational tests in order to see which variant is the best.

Much work has been done on the RPP and similar problems, see for example [6,3,8,9].

The RPP application in our interest is snow removal of secondary roads, so R is the set of links that need to be cleared of snow by the machine under consideration, while E also contains links that can be used for transportation of the machine. The costs c will be the cost for clearing each link, and may for example be the time required plus other factors. We can assume that all costs are strictly positive. It is not unlikely that the costs will satisfy the triangle inequality, but we cannot be sure of this.

Since the speed of the machine is higher during transportation than when clearing the road from snow, the costs will be different for transportation and for covering required edges. However, since the set of required edges is given, the cost for removing snow is fixed and the optimization only concerns transportation. Especially, using a required edge a second time is regarded as transportation, not snow removal. Therefore this fact does not affect the optimization.

In the calculations of the objective function value, we will use the same edge costs for transportation and covering required edges. (Using two different values would only increase the relative size of the constant part of the objective function value, and thus decrease the difference between different solutions.)

2. The Chinese postman method

As mentioned above, the CPP is very easy to solve if G is Eulerian, i.e. all vertices have even degree. If G is not Eulerian, some edges need to be used more than once. Using an edge twice can be seen as duplicating the edge and using each copy once. Therefore, in order to solve the CPP, we need to decide which edges to duplicate. One can show that it is not necessary to use an edge more than twice in an optimal CPP tour.

The optimal objective function value consists of two parts, one which is the same for all feasible solutions, namely $\sum_{(i,j)\in E} c_{ij}$, i.e. the cost for using each edge once, and one part which may differ between different feasible solutions, namely the cost for using some edges a second time. This second part can be seen as the cost for duplicating some edges, i.e. the additional cost needed to make the graph Eulerian. The optimization only concerns this second part, so the optimization is simply to minimize the additional costs for making the graph Eulerian.

This can be done polynomially and exactly as follows. First we find the shortest paths between all pairs of vertices with odd degree, for example by using Floyd–Warshall's method. Then, treating each shortest path as one edge, we find a minimum cost perfect matching containing all the vertices with odd degree. Each edge in the matching is then replaced by the shortest path it represents, and these edges are added to the graph.

This way, one is added to the degree of all vertices that had odd degree, so the graph becomes Eulerian. Furthermore, since the paths and the matching are of minimum cost, we have found the cheapest way of making the graph Eulerian. Now any Euler tour in the augmented graph is an optimal solution to the CPP. This way an optimal CPP tour can be found in polynomial time. Let us denote this method by CPM.

Actually, if the required set of edges, R, in the RPP forms a connected set, an optimal solution to the RPP can be found in exactly the same way. Let V^R be the set of "required vertices", i.e. vertices adjacent to a required edge, i.e. $V^R = \{i : (i,j) \in R \text{ or } (j,i) \in R\}$. Furthermore, let d_i^R be the degree of required edges for vertex i, and let the set of "odd" vertices $V^O \subseteq V^R$ be the vertices with odd d_i^R.

To solve the special case of the RPP when R forms a connected set, we find a minimum cost perfect matching for all odd vertices, V^O, using edges representing the shortest paths between the odd vertices. Adding the edges in the shortest paths used in the matching to the required graph, produces a connected Eulerian subgraph containing all the required edges. As all this is done at minimal cost, any Euler tour in the augmented required graph is an optimal solution to the RPP. Thus, if the required subgraph is connected, the RPP is polynomially solvable. Let us denote this method by CRPM.

However, if the required subgraph is not connected, this procedure is not guaranteed to form a connected subgraph. It can be used to form a number of disconnected Euler tours, but as a single connected Euler tour is required, this has to be enforced as discussed in the next section.

3. Heuristics

The Frederickson heuristic, [7], is a natural extension of the previously discussed method. One starts by making the required set connected, and then applies the CRPM method.

The question is how to connect the disconnected required parts at minimal cost. Here guaranteed optimality is lost. There is no known polynomial way of doing this so that the final solution is guaranteed to be of minimal cost.

In Frederickson's method, one simply finds a minimal spanning tree (MST) that connects the disconnected parts. More precisely, each connected set of required edges, C_i, is contracted into a vertex, and for each set of thereby created parallel edges, only the one with minimal cost is kept. In this graph, an MST is found, and the edges in the MST, E^{MST}, are added to the required set, R. Then the CRPM method is applied.

Obviously this method is polynomial. If the costs satisfy the triangle inequality, the cost of the obtained solution is no more than 1.5 times the optimal value. Let *method CE*1 denote this method.

Let us now discuss some variations of this approach. The two basic steps of the method are the following.

- Make the graph connected.
- Make the degrees even.

One may consider two different types of instances. The first type is when the required set forms a number of clearly separated "villages", often such that the cost of a path between two villages is larger than the cost of a link within a village. This might correspond to the situation of a real life rural postman.

However, other applications, such as for example snow removal, may create the other type of instances, namely where the sets of required edges are much more intermingled. Here the cost between two separate parts may be less than the cost on a link within a connected part.

In this second type of instance, separate sets C_i might become connected automatically if we make the degrees even, before making the graph connected. After this we still need to make sure that the graph is connected, but hopefully at a lower cost. Doing this, as above, with the help of an MST, might produce vertices with odd degree, so we need to finish up by making all degrees even (again).

The second method, denoted *method ECE*, is basically as follows.

1. Make the degrees even.
2. Make the graph connected.
3. Make the degrees even.

In some more detail, first apply the CRPM method, i.e. find a minimum cost perfect matching for all odd vertices, V^O, using edges representing the shortest paths between the odd vertices, and add the edges in the shortest paths used in the matching. *If* the resulting graph is connected, proceed to find an Euler tour.

If the resulting graph is not connected, find an MST connecting the disconnected parts, and add the edges in the MST, E^{MST}. Then apply the CRPM procedure for making the degrees even a second time.

In [14], a similar method is proposed. There one first applies the CRPM method, contracts each connected component to a single point and finds an MST. After removing duplicated edges from the MST-solution, the edges in the solution are added *twice* in order to produce a feasible tour. In the ECE method, the CRPM procedure is used instead of adding the edges twice, and this might take somewhat more time, but will probably give better solutions.

Another variation of the basic idea is to try to avoid creating odd degrees when connecting the different sets. Suppose that a certain connected set of required edges contains some vertices with even degree and some with odd. Then, if the MST is only allowed to connect to vertices with odd degree, no new vertices with odd degree will be created, and hopefully some vertices with odd degree will get even degree. That way, the subsequent matching may get a lower cost.

Unfortunately, some connected parts have no odd vertices, i.e. all vertices have even degree. (In this case, this separate part is Eulerian in itself.) Then we allow any node to be used. Let *method CE*2 denote this method.

*Method CE*0 uses the same basic steps as method CE1, but the connected sets of required edges are not contracted into single vertices. The MST is found in the original graph, where all the required edges are temporarily given cost equal to zero. These edges will in principle be the first ones to be chosen by the MST algorithm, so the MST will contain a tree in each set of required edges, C_i, plus non-required edges that are needed to make the solution connected. Since this solution may contain non-required edges that do not help connecting required parts, non-required edges are removed from the solution as long as the solution remains connected. The part of R that is not in E^{MST} is added afterwards, and then the degrees are made even as usual.

The main motivation for this method is to save time. Some preprocessing is not done, but on the other hand, a larger MST has to be solved. In general, however, one should not expect the solution to be better than for CE1.

In preliminary tests, using the visual package Vineopt (www.vineopt.com), it was observed that sometimes these methods result in apparently non-optimal parts of the tour. Therefore, we developed some heuristic improvements of the tours, as discussed in the next section.

4. Postprocessing

In some cases, the methods discussed above result in unwanted repetition of edges. Therefore we have constructed some postprocessing procedures that straightens out and improves the tours. One could obviously use simple 2- or 3-swap methods, as in [8], or other metaheuristics, such as tabu or simulated annealing. However, here we only consider procedures that are directly aimed at removing certain imperfections. (In other words, we are not aiming at escaping local optima.)

The first type of heuristic is based on the idea that if an edge is used more than once, improvements might be possible. We call it *heuristic H*1. Let us go through one case at a time.

Case 1: An edge is used *three times in the same direction*. Assume that this happens between vertex i and vertex j. Then the tour consists of (i,j), followed by a path, called A, from j to i, then (i,j) again, then a path, called B, from j to i, then (i,j) again, and finally a path, C, from j to i.

Here it is enough to use (i,j) once, regardless whether it is required or not. This can be achieved by removing two of the three passes through (i,j), and reversing one of A, B and C. Let rev(B) denote reversing the direction of path B, so that rev(B) leads from i to j. We get the following change:

Initial tour: i–j–A–i–j–B–i–j–C. → New tour: i–j–A–rev(B)–C.

Case 2: An edge is used *two times in one direction and once in the other*. Then the tour consists of (i,j), followed by a path, called A, from j to i, then a path, called B, from i to j, then (j,i), possibly followed by a cycle, C, starting and ending at i, then (i,j) again, and finally a path, D, from j to i.

Then it is enough to use (i,j) once, regardless whether it is required or not. Here we do not need to reverse any path.

A simple case is if the cycle C is non-existing. Then the tour contains (i,j) and (j,i) in direct succession, which we call an "appendix", which can be removed.

Initial tour: i–j–A–B–j–i–i–j–D. → New tour: i–j–A–B–D.

If there is a cycle C, we first move this part of the tour to after D, in order to obtain the simple case above.

Initial tour: i–j–A-B–j–i–C–i–j–D. → Modified tour: i–j–A–B–j–i–i–j–D–C. → New tour: i–j–A–B–D–C.

Case 3: An edge is used *two times in the same direction and is not required*. Then the tour consists of (i,j), then a path, A, from j to i, then (i,j) again, followed by a path, B, from j to i.

Here we do not need to use (i,j) at all. We remove both usages of (i,j) and reverse B.

Initial tour: i–j–A–i–j–B. → New tour: A–rev(B).

Case 4: An edge is used *once in each direction and is not required*. This case is a bit more complicated.

A simple special case is the following. The tour consists of (i,j), then (j,i), then a path, A, from j to i, and finally a path, B, from j to i. Then we simply remove the appendix, i.e. both usages of (i,j).

Initial tour: i–j–j–i–A–B. → New tour: A–B.

However, there are other cases that might occur. One possibility is that the tour separates into two if we remove both (i,j) and (j,i). If this happens, we cannot remove the usage of (i,j).

Another possibility is that the tour consists of three or more parts except (i,j) and (j,i). Then there are several possibilities, and one or more of the parts have to be reversed. In order to handle this case, we move parts of the tour, in order to try to create an appendix, as above. However, here there are cases which we consider too complicated to handle at present.

Another postprocessing heuristic, denoted by *heuristic H*2, is as follows. Following the tour around, we note which edges are required. Each required edge is only noted as required once (the first time it is encountered), so if the same edge is encountered a second time on the tour, it is not noted as required. After this, the whole tour is partitioned into a number of required and not required paths, i.e. the edges in a path are either all required or all not required.

We do not change the required paths, but each non-required path, say between vertices i and j, can be replaced by any shorter path between i and j. For this, the result of an all-to-all shortest path method, such as Floyd–Warshall's method, can be used. Thus, each connected non-required path in the tour is replaced by the shortest path between the two vertices. Especially we note that any cycle with only non-required edges can be directly removed from the tour.

Clearly the result depends on where on the tour the labeling of required edges starts, and one might try different starting points, in order to find the best choice. We have not tried this in our tests.

In this context, we might mention that a solution is often composed of several cycles, which may be placed in any order. While the order does not change the cost of the solution, it might affect the results of the subsequent improvement heuristics. We have not investigated the possibility of changing this order explicitly (except for what is done in some cases in H1).

Obviously the idea of using shortcuts is not new. It is an essential part of Christofides heuristic for the traveling salesman problem, and the idea is taken further in [11], where an improvement of roughly 5% is mentioned. In [13], the idea is used on the general routing problem, and is found to improve the results of Frederickson's heuristic by 3–5%.

5. Computational tests

5.1. General test instances

The methods have been tested on several groups of instances, of different types. Unfortunately it is, for various reasons, not possible to report results for real life data from the snow removal application under consideration. Instead we have made an effort to test the methods on many different instances of various kinds. We have actually solved more instances than in any other paper we have found, so any conclusions that we can draw from these tests will be fairly well founded. On the other hand, there is unfortunately a risk that many different structures within the same group make conclusions hard to draw, especially for the total set of test problems. Therefore we will study specific groups more, and try to draw conclusions based on the characteristics of the instances. In Section 5.3 we will study some instances that have a structure that resembles more what the snow removal application gives.

Initially we made some small instances, U, for illustration and debugging purposes. (If coordinates are available for the vertices, the graphs can be displayed using Vineopt, which helped much in the development of the heuristics.)

Then we started out with 58 different networks, used in [10]. These networks are of several different types and structures, and are created randomly, but in fairly advanced ways. Some are based on the well known Sioux Falls network, used in traffic planning research, some are complete graphs and some are sparse graphs structured in levels. See [10] for more information. For these networks, we randomly generated which edges should be required, in four different ways (with increasing numbers of required edges). The methods we wish to compare are identical (and optimal) for the case when the required edges form a connected set, so all such instances were removed. This left a set of only 32 instances, denoted by E.

We also found several sets on instances used by other authors. The instances, A and C, used in [8], were supplied by the authors. The graphs have vertices randomly generated in the plane and in group C all vertices have degree equal to four.

At http://www.iwr.uni-heidelberg.de/groups/comopt/people/theis/GRPLIB, [17], several sets of test instances are kept. Some instances are from problems that are slightly different than the rural postman problem, and some problems have unclear origin. However, our main aim is to have as many different test instances as possible, regardless of origin. We only used the edge lists, edge costs and required edges from these data. The problems were grouped as follows.

Group G1 contains 335 fairly easy instances (under the name "angel-wpp"). Group G2 contains 68 instances (under the name "angelwpp-A"), which are somewhat more difficult to solve. Group G3 contains 27 instances (under the name "angel-wpp-big"), which seems to be slightly larger versions of those in G2. We do not know much about the structure of the graphs in these groups, so we classify them as unstructured.

Group G4 contains 64 instances, used in [4] (divided into groups G41, with instances named I, G42 named ALBA, G43 named GRP and G44 named MADR), which are quite easy to solve. The graphs are based on the cities Albaida (G41, G42 and G43) and Madrigueras (G44), with required edges selected in various random ways.

Group G5 contains 80 instances used in [5], generated in various ways. This group contains both random graphs and grid graphs. Group G6 contains 24 instances used in [12], based on roads in Lancashire. The numbers of connected components are low and the instances are very easy to solve.

Group G7 contains 18 quite large instances (under the name "hcp"), with up to 5000 vertices. Of these, we only use group G7A, which consists of the 14 instances with at most 3000 vertices. Group G8A contains 40 instances (under the name "hertz") with up to 2000 vertices, while group G8B contains 20 larger instances of the same type, with up to 3000 vertices. Group G9 contains instances used in [1], and we divided them into group G9A with 64 instances with up to 1000 vertices, and group G9B with 20 instances with 2500 vertices.

Group G10 contains a number of instances used in [16]. Sorting out the uninteresting cases (with no required edges) left 24 instances. Group G11, finally, contains eight instances used in [2]. These instances come from vehicle routing with sparse feasibility graphs, which means that each node only has a few other nodes to which it can be adjacent in a route. Together this yields a total of 939 instances.

In Table 1, we give the maximal and average number of vertices, $|V|$, edges, $|E|$, and required edges, $|R|$, as well as connected components of required edges, $|C|$. The total number of problems in each group is denoted by #. Obviously large numbers of vertices and edges make the instances harder to solve. Also a large number of connected components will in a way make the problem harder. We note that groups G6, U and G41 have small numbers of connected components. Since relative numbers may be more interesting, we also give various such characteristics below.

In Table 2, we give the following statistics of the test instances. We give the averages of the density of the graph, $n_E = |E|/|V|$, the required density, $n_R = |R|/|V|$, the required proportion of edges, $n_P = |R|/|E|$, the relation between the number of connected components and the number of nodes, $n_C = |C|/|V|$, the number of required edges per connected component, $n_Q = |R|/|C|$, the minimal, $\underline{c}$, and maximal, $\bar{c}$, edge cost, the minimal, $\underline{d}$, and maximal, $\bar{d}$, distance (cost) between two connected components, and finally the relations between the distances between connected components and edge costs, $n_1 = \underline{d}/\underline{c}$ and $n_2 = \bar{d}/\bar{c}$. (If there is any instance in a group with $c_{ij} = 0$, n_1 cannot be calculated.)

The parameter n_1 is a very important parameter, since it in a way indicates the probability for the method ECE of "accidentally" connecting components when making the degree even. For very large values of n_1, this is unlikely to happen. If n_1 cannot be calculated, we may get some similar information from n_2, although not as trustworthy, since the most expensive edges are probably not used.

Looking for significant values, we find that groups G8B, G41, G8A and G6 have the largest values of n_R, followed by C, E, G3 and G43, while G9B, G9A, G5 and G7A have small values of n_R. Considering n_P, G6 has the largest value, followed by G43, G44, G41, G42 and C, while G9B, G9A, A, G11, G5 and G7A have small values. For n_C, we find that G6, G8B and G8A have small values. The number of

Table 1
Characteristics of the test instances.

Group	#	$\lvert V\rvert_{max}$	$\lvert V\rvert_{ave}$	$\lvert E\rvert_{max}$	$\lvert E\rvert_{ave}$	$\lvert R\rvert_{max}$	$\lvert R\rvert_{ave}$	$\lvert C\rvert_{max}$	$\lvert C\rvert_{ave}$
U	9	23	9.9	47	17.9	15	6.8	4	1.9
E	32	150	75.5	431	193.7	146	59.0	32	8.4
A	38	350	184.2	1717	805.7	187	77.8	61	26.3
C	72	250	149.3	500	298.4	330	129.8	56	19.0
G1	335	196	80.5	316	146.3	141	50.6	42	13.1
G2	68	749	377.4	2288	900.5	613	295.9	114	49.3
G3	27	999	878.6	3073	2285.4	854	728.3	146	104.6
G41	24	50	25.1	184	58.9	78	28.1	8	5.0
G42	10	116	116.0	174	174.0	126	75.1	34	21.1
G43	15	196	196.0	316	316.0	238	158.3	42	21.8
G44	15	116	116.0	174	174.0	122	85.7	23	13.0
G5	80	320	240.0	640	420.0	68	48.2	51	32.8
G6	24	140	108.5	190	144.0	190	109.9	6	3.0
G7A	14	3000	2571.4	5999	5139.3	906	630.9	528	402.2
G8A	40	2000	1050.0	13 312	5967.7	2944	1112.0	222	66.6
G8B	20	3000	2550.0	20 992	15 383.2	4266	2804.8	324	138.0
G9A	64	1000	501.6	2000	815.1	193	69.4	135	50.7
G9B	20	2500	2500.0	5000	4062.5	470	327.8	315	242.5
G10	24	1416	391.0	13 024	2487.0	723	193.8	691	188.8
G11	8	857	205.9	3403	804.2	318	83.2	112	29.1
Total	939								

Table 2
Further characteristics of the test instances.

Group	n_E	n_R	n_P	n_C	n_Q	$\underline{c}$	$\bar{c}$	$\underline{d}$	$\bar{d}$	n_1	n_2
U	1.64	0.55	0.33	0.25	2.42	7.40	13.60	12.40	37.00	1.75	2.69
E	3.18	0.83	0.30	0.11	14.84	42.75	139.78	48.31	885.25	–	8.78
A	4.38	0.42	0.10	0.14	3.86	6.34	151.87	16.58	1362.55	–	10.44
C	2.00	0.86	0.43	0.13	10.63	6.15	962.43	24.29	1533.31	–	1.65
G1	1.96	0.55	0.29	0.17	3.43	4.76	193.81	19.82	796.79	5.54	3.41
G2	2.17	0.74	0.37	0.14	5.66	2.17	297.63	9.59	1810.96	5.73	9.20
G3	2.55	0.83	0.34	0.12	7.08	1.00	164.04	2.25	1662.29	2.25	12.48
G41	2.15	1.00	0.48	0.24	5.52	1.08	19.88	3.58	34.83	–	1.79
G42	1.50	0.65	0.43	0.18	6.44	12.00	388.00	22.80	1627.60	1.90	4.19
G43	1.61	0.81	0.50	0.11	25.41	20.00	290.00	33.00	1562.33	1.65	5.39
G44	1.50	0.74	0.49	0.11	12.58	12.00	388.00	21.87	1606.00	1.82	4.14
G5	1.75	0.20	0.12	0.14	1.48	84.73	284.48	89.32	1044.74	1.05	3.72
G6	1.31	0.91	0.69	0.03	29.69	2.50	97.50	23.33	456.83	9.56	4.65
G7A	2.00	0.25	0.12	0.16	1.55	0.00	8.00	0.00	31.93	–	3.99
G8A	5.40	1.00	0.19	0.07	65.77	11.75	701.95	46.15	14 175.80	12.19	23.00
G8B	6.01	1.10	0.19	0.06	63.56	1.80	392.40	10.75	14 377.35	–	37.23
G9A	1.63	0.14	0.09	0.10	1.37	1.00	10.00	1.30	49.94	1.30	4.99
G9B	1.62	0.13	0.08	0.10	1.33	1.00	10.00	1.00	70.15	1.00	7.02
G10	2.74	0.43	0.18	0.41	1.05	4195	264 412	53 964	368 859	5.58	1.54
G11	3.83	0.44	0.11	0.15	2.88	1.25	172.62	1.75	252.25	1.44	1.45

required edges per connected component, n_Q, is high for G8A, G8B, G6 and G43, and low for G10, G9B, G9A, G5 and G7A. Finally we note that G9B, G5, G9A and G11 have low values of n_1, while G8A has the largest, followed by G6, G2, G10 and G1.

5.2. Objective function values

We have four methods for constructing solutions, CE0, CE1, CE2, ECE, and two postprocessing heuristics, H1, H2. In our computational tests, we have either used both the two postprocessing heuristics after each of the main methods, or none of them.

In order to sum up the results in a readable way, we have for each instance found the best objective function value, and then listed the methods that gave this objective function value. (Often some of the methods yield the same objective function value.) We have then summed up to see for how many instances a certain method yields the best objective function value. The results are given in Tables 3 and 4.

We have also calculated normalized objective function values, i.e. divided by the best objective function value found. (The objective function values include the costs for the required edges.) These values are given in Tables 5 and 6. (In Tables 4, 5 and 6, the groups G41, G42, G43 and G44 are treated as one group, G4.)

The first conclusion from these tables is that Frederickson's method (CE1) is still competitive. Taking the average over all the test problems, it would be the winner. However, all the methods seems to give fairly good solutions in general. Taking a closer look, one finds that CE1 performs well on many easy problems, especially in groups A, G1, G2 and G4. Obviously the totals in Tables 3 and 4 are much affected by this fact. Therefore we study the different groups more.

Stepping through the groups, Table 3 shows the following. The instances in group U are too small to have any significance. For group E, CE2 is the best method, while for groups A and C, method CE1 is best. CE1 is also the best for groups G1, G2 and G3 (which probably have similar structure). A difference is that CE0 is fairly good for G1

Table 3
Number of best objective function values, without H1 and H2.

Group	Total	CE0	CE1	CE2	ECE
U	9	7	8	8	7
E	32	13	16	22	14
A	38	0	23	8	11
C	72	15	48	31	8
G1	335	146	202	123	126
G2	68	8	55	30	3
G3	27	0	24	22	0
G41	24	20	17	10	5
G42	10	2	7	4	3
G43	15	5	10	6	3
G44	15	5	9	7	2
G5	80	0	6	0	74
G6	24	21	18	15	6
G7A	14	0	7	7	7
G8A	40	3	21	19	9
G8B	20	2	12	13	4
G9A	64	2	17	3	46
G9B	20	0	2	0	18
G10	24	7	10	12	10
G11	8	0	1	0	7
Total	939	256	513	340	363

Table 4
Number of best objective function values, with H1 and H2.

Group	Total	CE0	CE1	CE2	ECE
U	9	8	8	8	7
E	32	17	19	21	12
A	38	6	18	13	7
C	72	27	41	35	6
G1	335	189	203	154	134
G2	68	26	37	22	1
G3	27	3	22	19	0
G4	64	35	40	32	12
G5	80	0	11	7	63
G6	24	21	18	24	9
G7A	14	0	6	7	7
G8A	40	8	24	17	4
G8B	20	5	12	13	1
G9A	64	7	21	13	34
G9B	20	0	8	2	11
G10	24	10	9	10	12
G11	8	1	1	0	7
Total	939	363	498	397	327

Table 5
Relative objective function values, without H1 and H2.

Group	CE0	CE1	CE2	ECE
U	1.019	1.004	1.004	1.024
E	1.042	1.010	1.009	1.015
A	1.049	1.006	1.019	1.039
C	1.024	1.004	1.012	1.063
G1	1.015	1.007	1.020	1.026
G2	1.009	1.001	1.010	1.032
G3	1.008	1.000	1.001	1.029
G4	1.013	1.007	1.021	1.051
G5	1.203	1.064	1.080	1.002
G6	1.000	1.003	1.004	1.063
G7A	1.121	1.024	1.032	1.098
G8A	1.010	1.002	1.004	1.010
G8B	1.006	1.001	1.002	1.011
G9A	1.093	1.043	1.056	1.009
G9B	1.104	1.043	1.058	1.001
G10	1.045	1.015	1.015	1.052
G11	1.040	1.024	1.022	1.001

Table 6
Relative objective function values, with H1 and H2.

Group	CE0	CE1	CE2	ECE
U	1.006	1.006	1.006	1.014
E	1.019	1.008	1.007	1.016
A	1.019	1.007	1.014	1.038
C	1.015	1.004	1.009	1.056
G1	1.008	1.006	1.012	1.020
G2	1.005	1.002	1.008	1.031
G3	1.005	1.000	1.001	1.027
G4	1.010	1.007	1.013	1.037
G5	1.124	1.039	1.041	1.004
G6	1.001	1.003	1.000	1.024
G7A	1.095	1.013	1.018	1.064
G8A	1.005	1.002	1.004	1.011
G8B	1.002	1.000	1.001	1.012
G9A	1.051	1.021	1.024	1.016
G9B	1.072	1.019	1.025	1.009
G10	1.022	1.006	1.006	1.018
G11	1.020	1.021	1.020	1.001

but much worse for G2 and G3. On the other hand, CE2 is almost as good as CE1 on G3. Method ECE is not good for G2 and G3.

For group G41, CE0 is the best, while for G42, G43 and G44, CE1 is best. For group G5, however, method ECE is very much better than any other method. For group G6, CE0 is best, followed by CE1. For group G7A all methods except CE0 are equally good.

For group G8A, CE1 is best, closely followed by CE2, while for group G8B, CE2 is slightly better than CE1. For groups G9A and G9B, ECE is clearly the best method, while for group G10, CE2 is best, followed by ECE and CE1. Finally, for group G11, ECE is a clear winner.

Let us now look at the results from the point of view of the methods. The goal is to identify classes of instances for which a method performs better than the others.

Method ECE is best for groups G5, G9A, G9B and G11. These groups have the *smallest* values of n_1. We also find significantly low values of n_R, n_P and n_Q in these groups.

Method CE2 is best for groups E, G8B and G10, and rather good for C, G3, G7A and G8A. Here the pattern is not so clear. One can find large values of n_1, n_R and n_Q, but also small values. An important factor for method CE2 is the number of connected components with even degree of all nodes after the first stage of the method, and this is not captured by the parameters we have recorded.

Method CE0 is best for groups G6 and G41. Here we find rather large values of n_1, n_Q, n_P and n_R, and the smallest value of n_C.

Method CE1 is best for groups A, C, G1–3, G42–44 and G8A. These groups do not have very extreme values of the parameters, but we note that the values of n_1, n_R, n_P and n_Q are *not* very small.

A clear conclusion is that ECE seems to be best for instances with low values of n_1, n_P, n_Q and possibly n_R. It is harder to predict the difference between CE0, CE1 and CE2. One might say that ECE and CE1 are complementary, i.e. CE1 is good when ECE is bad and vice versa, and that CE2 captures a little of both.

The results are similar with the postprocessing heuristics, but somewhat less pronounced. If a method fails to find a good solution, there is probably a higher potential for improvement for the postprocessing heuristics.

There are some minor changes after the postprocessing, namely that CE2 surpasses CE0 for G6 and that ECE surpasses CE2 for G10 when it comes to the number of best objective function values.

We conclude that ECE seems to be preferable if the number of required edges is low compared to the number of vertices and edges, if the number of required edges in average in each connected component is small and if the minimal distances between connected components is not much larger the minimal edge costs.

Many instances come from slightly different problems, such as the general routing problem, where also vertices can be required,

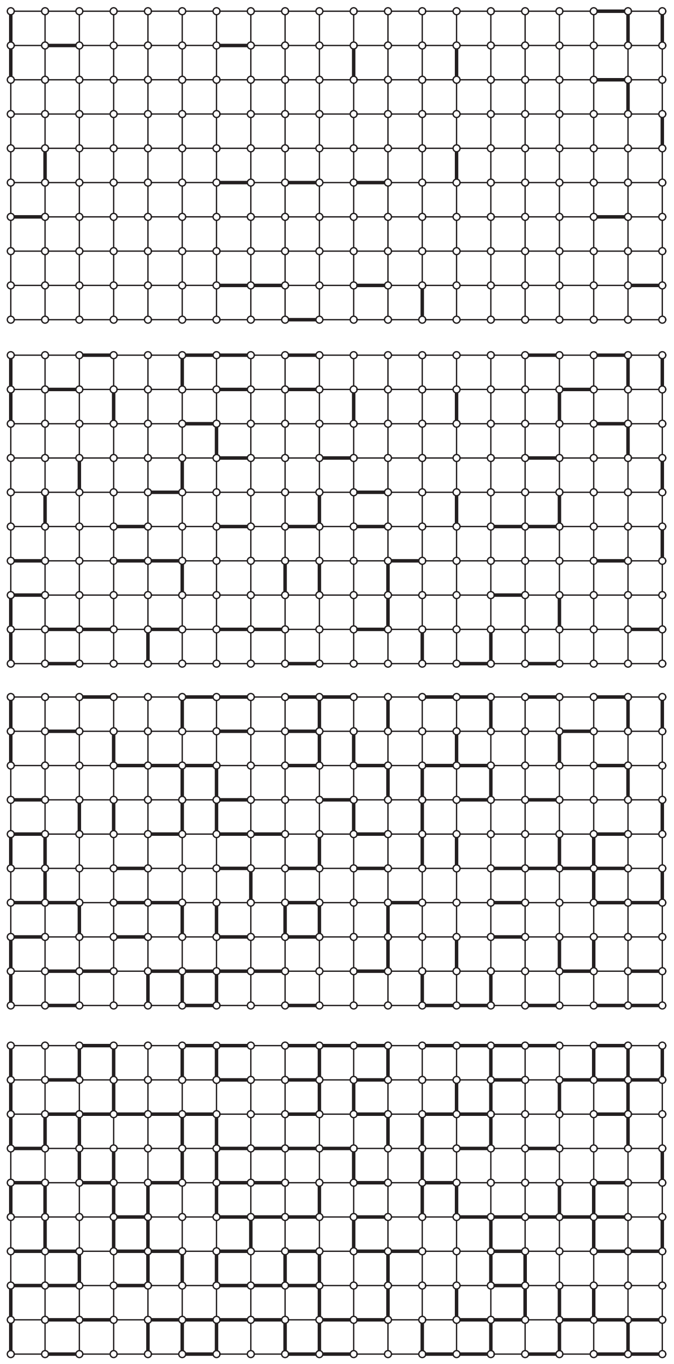

Fig. 1. Grid instances Gr1-1, Gr1-3, Gr1-5 and Gr1-7.

or the Steiner tree problem, where only vertices can be required. In such cases, the objective function values we find cannot be compared to the ones published. Comparing to the few relevant values we found, an optimal solution seems to be found frequently for the easier problems.

5.3. Grid instances

We also made some more specific tests, based on the application of snow removal in an urban network. In such a case, the underlying graph is quite similar to a grid. A vertex is a street crossing, so the degrees of the nodes are usually four, with some exceptions, especially at the borders of the area.

Table 7
Characteristics of the grid instances Gr1.

Inst.	$\lvert R\rvert$	$\lvert C\rvert$	n_R	n_P	n_C	n_Q	n_1	n_2
Gr1-1	25	21	0.12	0.07	0.10	1.19	2.36	9.02
Gr1-2	45	33	0.23	0.12	0.17	1.36	2.18	9.33
Gr1-3	71	48	0.35	0.19	0.24	1.48	1.45	9.85
Gr1-4	99	67	0.49	0.27	0.34	1.48	1.00	9.85
Gr1-5	125	47	0.62	0.34	0.23	2.66	1.00	9.85
Gr1-6	152	25	0.76	0.41	0.12	6.08	1.09	9.85
Gr1-7	173	11	0.86	0.47	0.06	15.73	1.45	9.85

Table 8
Results for the grid instances Gr1.

Inst.	CE0	CE1	CE2	ECE	Best
Gr1-1	1.138	1.000	1.005	1.018	CE1
Gr1-2	1.194	1.054	1.086	1.000	ECE
Gr1-3	1.051	1.029	1.074	1.000	ECE
Gr1-4	1.047	1.041	1.057	1.000	ECE
Gr1-5	1.020	1.027	1.027	1.000	ECE
Gr1-6	1.003	1.000	1.010	1.001	CE1
Gr1-7	1.002	1.003	1.000	1.038	CE2

We first constructed a grid network of size 10×20, which gave 200 vertices and 370 edges. The edge costs were randomly generated between 10 and 200. This structure corresponds fairly well to a city network for a relevant area. Note that if the edge costs correspond to distances, a regular grid is not a correct map of the area. Instead the network should be more skewed in various ways. Some edges might get a higher cost than if the triangle inequality would hold, and non-required such edges will not be used. In effect we then get certain crossings with less than four streets, which is realistic.

We then graphically made seven sets of required edges of increasing size, each one containing the previous one, with the help of Vineopt. Some instances are shown in Fig. 1 and the characteristics are given in Table 7. For all instances we have $|V| = 200$, $|E| = 370$ and $n_E = 1.85$.

We see that the number of connected components first increase as the set of required edges increase, but then starts to decrease. In Table 8 we give the relative objective function values (without postprocessing) for these instances. We find that ECE is well suited for this structure, except when there are very few required edges, or when the number of required edges is so large that the number of connected components is small. Again we find that small values of n_1, n_R and n_P favors ECE.

We then generated two more groups of random instances of grid networks, Gr2 with 10 instances of size 10×20, which gives 200 nodes and 370 edges, and Gr3 with 10 instances of size 20×50, which gives 1000 nodes and 1930 edges. Required edges were chosen randomly, according to an increasing probability. The edge costs were randomly generated between 50 and 150.

Some characteristics of the instances are given in Table 9. For Gr2 we have $|V| = 200$, $|E| = 370$ and $n_E = 1.85$, and for Gr3 we have $|V| = 1000$, $|E| = 1930$ and $n_E = 1.93$.

In Table 10 we give the results of the computational tests, without using postprocessing heuristics, as relative objective function values.

Summing up we find that for group Gr2, method CE2 gives the best objective function value for 8 of the 10 instances. For group Gr3, method CE2 gives the best objective function value for 7 of the 10 instances, while method ECE gives best objective function for three instances. Thus we find that for these problems CE2 is clearly the best method.

For these two groups, we note that n_1 is very small for all instances. We find that ECE is less competitive as n_R, n_P and n_Q grows.

Table 9
Characteristics of the grid instances Gr2 and Gr3.

Inst.	$\|R\|$	$\|C\|$	n_R	n_P	n_C	n_Q	n_1	n_2
Gr2-0	88	40	0.44	0.24	0.20	2.20	1.00	17.68
Gr2-1	97	44	0.48	0.26	0.22	2.20	1.00	18.06
Gr2-2	125	31	0.62	0.34	0.15	4.03	1.00	17.27
Gr2-3	136	38	0.68	0.37	0.19	3.58	1.00	17.91
Gr2-4	172	18	0.86	0.46	0.09	9.56	1.02	17.64
Gr2-5	187	13	0.94	0.51	0.07	14.38	1.00	16.74
Gr2-6	208	7	1.04	0.56	0.04	29.71	1.06	17.83
Gr2-7	215	4	1.07	0.58	0.02	53.75	1.02	17.76
Gr2-8	232	4	1.16	0.63	0.02	58.00	1.00	17.85
Gr2-9	256	3	1.28	0.69	0.01	85.33	1.08	16.88
Gr3-0	483	199	0.48	0.25	0.20	2.43	1.00	42.49
Gr3-1	562	175	0.56	0.29	0.17	3.21	1.00	44.03
Gr3-2	633	150	0.63	0.33	0.15	4.22	1.00	43.85
Gr3-3	768	121	0.77	0.40	0.12	6.35	1.00	44.42
Gr3-4	842	81	0.84	0.44	0.08	10.40	1.00	42.72
Gr3-5	962	51	0.96	0.50	0.05	18.86	1.00	44.19
Gr3-6	1023	37	1.02	0.53	0.04	27.65	1.00	44.55
Gr3-7	1168	17	1.17	0.61	0.02	68.71	1.00	44.80
Gr3-8	1231	11	1.23	0.64	0.01	111.91	1.02	42.97
Gr3-9	1331	5	1.33	0.69	0.01	266.20	1.00	42.03

Table 10
Results for the grid instances Gr2 and Gr3.

Inst.	CE0	CE1	CE2	ECE	Best
Gr2-0	1.060	1.031	1.000	1.002	CE2
Gr2-1	1.093	1.032	1.022	1.000	ECE
Gr2-2	1.028	1.019	1.000	1.009	CE2
Gr2-3	1.030	1.003	1.000	1.020	CE2
Gr2-4	1.029	1.024	1.000	1.031	CE2
Gr2-5	1.009	1.002	1.000	1.051	CE2
Gr2-6	1.005	1.001	1.000	1.043	CE2
Gr2-7	1.001	1.001	1.000	1.065	CE2
Gr2-8	1.004	1.004	1.000	1.066	CE2
Gr2-9	1.000	1.000	1.001	1.051	CE0/1
Gr2 all	1.026	1.012	1.002	1.034	CE2
Gr3-0	1.125	1.048	1.046	1.000	ECE
Gr3-1	1.107	1.062	1.050	1.000	ECE
Gr3-2	1.055	1.024	1.016	1.000	ECE
Gr3-3	1.023	1.011	1.000	1.009	CE2
Gr3-4	1.013	1.009	1.000	1.025	CE2
Gr3-5	1.016	1.012	1.000	1.052	CE2
Gr3-6	1.003	1.000	1.000	1.053	CE1/2
Gr3-7	1.005	1.005	1.000	1.056	CE2
Gr3-8	1.002	1.001	1.000	1.053	CE2
Gr3-9	1.002	1.001	1.000	1.058	CE2
Gr3 all	1.035	1.017	1.011	1.031	CE2

The good performance of CE2 probably means that there are very few connected components where all nodes have even degree, since the difference between CE2 and the other methods mainly lies in the treatment of connected components containing nodes with odd degree.

5.4. Solution times

We also wish to take the solution times into account. The CPU-times on a 2.9 GHz PC running Linux are presented in Tables 11 and 12 (in the same way as the objective functions). For the easier problems, the best solution time was often zero (i.e. less than 0.01 s) for some instance in the group, and then the normalized time cannot be computed. These groups are omitted from Table 12.

In order to further compare the solution times, we have summed up the total time for solving all problems in a group, see Table 13. For each method, we give the time needed by the method, M, by heuristic H1, by heuristic H2 and the total time, *T*.

Our first conclusion here is that many of the test instances are a bit too easy for these methods. For example, for group G1, some of the methods found fairly good solutions for 335 problems in less than 10 s. A practical conclusion is that many of the easier test instances are possible to solve in real-time with heuristics.

Table 11
Number of best times, with H1 and H2.

Group	Total	CE0	CE1	CE2	ECE
U	9	9	9	8	8
E	32	25	20	21	25
A	38	31	9	12	19
C	72	55	20	21	44
G1	335	291	198	190	245
G2	68	41	9	4	42
G3	27	27	0	0	0
G4	64	53	34	32	48
G5	80	57	0	1	46
G6	24	15	12	15	12
G7A	14	12	0	0	2
G8A	40	29	5	2	6
G8B	20	15	0	0	5
G9A	64	39	17	19	41
G9B	20	18	0	0	2
G10	24	24	8	9	10
G11	8	8	3	3	4
Total	939	749	344	337	559

Table 12
Relative times, with H1 and H2.

Group	CE0	CE1	CE2	ECE
G2	1.104	2.001	2.219	1.148
G3	1.000	2.602	3.245	1.027
G5	1.053	1.739	1.648	1.126
G7A	1.002	10.561	11.947	2.282
G8B	1.004	2.820	3.584	1.058
G9B	1.000	4.639	4.380	1.056
Total	1.027	4.060	4.504	1.283

Table 13
Total solution time (seconds) for each group.

Group	CE0				CE1				CE2				ECE			
	M	H1	H2	*T*	M	H1	H2	*T*	M	H1	H2	*T*	M	H1	H2	*T*
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0
A	2	0	2	4	6	0	1	7	8	0	0	8	4	0	1	5
C	3	0	2	5	3	0	4	7	5	0	2	7	6	0	1	7
G1	6	0	2	8	9	0	1	10	11	0	2	13	9	0	3	12
G2	44	1	26	71	142	0	30	172	174	0	29	203	67	0	35	102
G3	115	1	74	190	408	1	88	497	743	1	106	850	222	2	110	334
G4	3	0	0	3	3	0	1	4	3	0	1	4	2	0	1	3
G5	10	0	5	15	24	0	4	28	16	0	8	24	12	0	5	17
G6	1	0	0	1	1	0	0	1	0	0	1	1	1	0	0	1
G7A	1231	60	1095	2386	22 209	160	931	23 300	24 503	158	914	25 575	3617	112	916	4645
G8A	561	2	309	872	1352	5	305	1662	1725	0	317	2042	669	0	314	983
G8B	2274	7	1131	3412	5933	8	1125	7066	7386	7	1126	8519	2744	6	1155	3905
G9A	95	1	90	186	321	5	96	422	358	6	117	481	97	0	90	187
G9B	1037	33	1003	2073	8431	31	1001	9463	7511	34	1007	8552	1225	4	1006	2235
G10	81	2	43	126	3682	3	40	3725	5245	1	43	5289	1250	4	41	1295
G11	4	0	2	6	13	0	3	16	14	1	2	17	5	0	2	7
Total	5468	107	3784	9359	42 538	213	3630	46 381	47 703	208	3675	51 586	9930	128	3680	13 738

The main conclusion, however, is the perhaps surprising difference in solution time between the methods. The pattern here is more consistent, i.e. holds for most of the problem groups.

The general result is that methods CE1 and CE2 take much longer time than CE0 and ECE. We find that method CE0 is the fastest, and method ECE is only slightly slower while methods CE1 and CE2 are definitely slower, method CE2 being the slowest.

The total time for solving all the instances is for CE1 and CE2 around 4–5 times the time needed by CE0 and ECE. While summing the number of "bests" takes the easier instances more into account, summing up the time takes the harder instances more into account. If we had to choose between these two criteria, we would consider the latter as more important.

We find that for most instances we can run *both* CE0 and ECE without exceeding the time needed by CE1. Actually it may be a very good practical idea to run both CE0 and ECE, and simply choose the best of the two solutions.

Considering the additional grid instances, the solution times for groups Gr1 and Gr2 are very small. For group Gr3, we get the following summed relative solution times: method CE0: 1.008, method CE1: 2.209, method CE2: 2.984 and method ECE: 1.127.

Finally we should perhaps mention that the codes are not optimized. It is probably possible to speed them up, but we do not expect that this would change our general conclusions.

5.5. Postprocessing

Now we consider the improvement obtained by the two heuristics H1 and H2. In Table 14, we give the average improvement obtained by the two postprocessing heuristics for the different methods. We find that heuristic H1 is very quick, but gives improvement which often is larger than that of H2. (However, here we must remember that H1 is used before H2.) The time needed by H2 is significantly larger than H1.

Maybe the most important effect of the postprocessing heuristics is to remove obvious flaws in the tours. Without the post-heuristics, the tours contained unmotivated repetitions, which would be very apparent at a visual investigation. If the tours are to be used in real life, and postprocessing heuristics are not used, the clients/users might not trust the optimization approach very much. One might even say that the primary objective of the postprocessing heuristics is to make the tours "presentable".

Table 14
Average improvement of the postprocessing heuristics, in %.

Group	CE0		CE1		CE2		ECE	
	H1	H2	H1	H2	H1	H2	H1	H2
U	0.0	1.3	0.0	0.0	0.0	0.0	0.0	1.2
E	1.4	1.3	0.9	0.2	0.8	0.1	0.5	0.3
A	0.4	4.1	0.5	1.1	0.7	1.4	0.1	1.6
C	1.0	1.0	0.7	0.4	0.8	0.6	0.7	1.0
G1	0.9	0.6	0.8	0.3	1.1	0.5	0.5	0.9
G2	0.8	0.7	0.6	0.4	0.9	0.4	0.4	0.8
G3	0.6	0.8	0.6	0.4	0.6	0.4	0.4	0.7
G4	0.8	0.4	0.6	0.3	0.9	0.8	0.6	1.5
G5	3.6	3.7	2.7	0.3	3.6	0.6	0.1	0.4
G6	0.0	0.1	0.1	0.0	0.5	0.1	2.0	1.8
G7A	3.4	1.0	3.0	0.0	3.3	0.1	4.5	0.4
G8A	0.1	0.8	0.1	0.4	0.1	0.4	0.0	0.4
G8B	0.1	0.5	0.1	0.3	0.2	0.3	0.0	0.3
G9A	3.3	1.7	2.8	0.5	3.6	0.7	0.1	0.5
G9B	3.4	0.6	3.2	0.1	4.0	0.1	0.1	0.2
G10	0.8	2.9	0.6	1.9	0.7	1.9	2.0	2.8
G11	1.2	1.3	0.5	0.4	0.1	0.7	0.0	0.6
Total	1.3	1.3	1.0	0.4	1.3	0.5	0.7	0.9

6. Conclusions

In the process of planning snow removal for secondary roads, the rural postman problem appears as an interesting problem. We propose some heuristics for the rural postman problem of the same type as the classical Frederickson method. The ideas are to make the degree even before making the graph connected, and to favor odd nodes when making the graph connected. We also present two heuristics for improving the obtained tours.

We report extensive computational tests on several different groups of test instances, and find that, while the original method still performs well, the alternative ideas performs better for certain types of test instances, and are therefore worth considering. This is especially pronounced for instances resembling instances for the snow removal application. Furthermore, the solution times are quite different, and some of the new methods are quicker.

Finally we find that it is practically possible to solve instances of relevant sizes in real time.

Acknowledgments

Preliminary tests of some of the methods were made in the Master Thesis by Roeya Elyasi Pour and Hoosang Farhadi, 2005, supervised by Kaj Holmberg.

References

[1] Beasley JE. An SST-based algorithm for the Steiner problem in graphs. Networks 1989;19:1–16.
[2] Beasley JE, Christofides N. Vehicle routing with a sparse feasibility graph. European Journal of Operational Research 1997;98:499–511.
[3] Cook C, Schoenfeld DA, Wainwright RL. Finding rural postman tours. In: Proceedings of the 1998 ACM symposium on applied computing. New York: ACM Press; 1998. p. 318–26.
[4] Corberán A, Letchford AN, Sanchis J. A cutting plane algorithm for the general routing problem. Mathematical Programming 2001;90:291–316.
[5] Duin C, Voß S. Efficient path and vertex exchange in Steiner tree algorithms. Networks 1997;29:89–105.
[6] Eiselt HA, Gendreau M, Laporte G. Arc routing problems, part II: the rural postman problem. Operations Research 1995;43:399–414.
[7] Frederickson GN. Approximation algorithms for some postman problems. Journal of the Association for Computing Machinery 1979;26:538–54.
[8] Ghiani G, Laganá D, Musmanno R. A constructive heuristic for the undirected rural postman problem. Computers & Operations Research 2006;33:3450–7.
[9] Groves G, Vuuren J. Efficient heuristics for the rural postman problem. ORiON 2005;21:33–51.
[10] Holmberg K, Hellstrand J. Solving the uncapacitated network design problem by a Lagrangean heuristic and branch-and-bound. Operations Research 1998; 46:247–59.
[11] Johnson DS, McGeoch LA. The traveling salesman problem: a case study in local optimization. In: Aarts EHL, Lenstra JK, editors. Local search in combinatorial optimization. New York: Wiley; 1997. p. 215–310.
[12] LYO Li, Eglese RW. An interactive algorithm for vehicle routing for winter-gritting. The Journal of the Operational Research Society 1996;47:217–28.
[13] Muyldermans L, Beullens P, Cattrysse D, Oudheusden DV. Exploring variants of 2-opt and 3-opt for the general routing problem. Operations Research 2005;53:982–95.
[14] Pearn W, Wu T. Algorithms for the rural postman problem. Computers & Operations Research 1995;22:819–28.
[15] Razmara G. Snow removal routing problems—theory and applications. PhD dissertation, Linköping University, Sweden, 2004; Linköping Studies in Science and Technology. Dissertation no. 888.
[16] Reinelt G. The traveling salesman problem: computational solutions for TSP applications. in: Lecture Notes in Computer Science Berlin: Springer; vol. 840, 1994.
[17] Theis DO. Polyhedra and algorithms for the general routing problem. PhD thesis, University of Heidelberg, 2005.