Math. Prog. Comp. (2015) 7:269–287
DOI 10.1007/s12532-015-0080-8

FULL LENGTH PAPER

Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun Algorithm

Keld Helsgaun[1]

Received: 23 August 2013 / Accepted: 6 March 2015 / Published online: 29 April 2015

Abstract The equality generalized traveling salesman problem (E-GTSP) is an extension of the traveling salesman problem (TSP) where the set of cities is partitioned into clusters, and the salesman has to visit every cluster exactly once. It is well known that any instance of E-GTSP can be transformed into a standard asymmetric instance of the TSP, and therefore solved with a TSP solver. This paper evaluates the performance of the state-of-the art TSP solver Lin–Kernighan–Helsgaun (LKH) on transformed E-GTSP instances. Although LKH is used without any modifications, the computational evaluation shows that all instances in a well-known library of benchmark instances, GTSPLIB, could be solved to optimality in a reasonable time. In addition, it was possible to solve a series of new very-large-scale instances with up to 17,180 clusters and 85,900 vertices. Optima for these instances are not known but it is conjectured that LKH has been able to find solutions of a very high quality. The program's performance has also been evaluated on a large number of instances generated by transforming arc routing problem instances into E-GTSP instances. The program is free of charge for academic and non-commercial use and can be downloaded in source code.

Keywords Equality generalized traveling salesman problem · E-GTSP · Traveling salesman problem · TSP · Lin–Kernighan · Heuristics · Arc routing problems

Mathematics Subject Classification 90C27 · 90C35 · 90C59

✉ Keld Helsgaun
keld@ruc.dk

[1] Computer Science, Roskilde University, 4000 Roskilde, Denmark

1 Introduction

The equality generalized traveling salesman problem (E-GTSP) is an extension of the traveling salesman problem (TSP) where the set of cities is partitioned into clusters, and the salesman has to visit every cluster exactly once. The E-GTSP coincides with the TSP whenever all clusters are singletons. The problem has numerous applications, including airplane routing, computer file sequencing, and postal delivery [1].

The E-GTSP is defined on a complete graph $G = (V, A)$, where $V = \{v_1 \ldots v_n\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V\}$ is the set of directed arcs ($i \neq j$) or undirected edges ($i < j$). A non-negative cost c_{ij} is associated with each arc or edge (v_i, v_j) and the vertex set V is partitioned into m mutual exclusive and exhaustive clusters $V_1 \ldots V_m$, i.e., $V = V_1 \cup V_2 \cup V_m$ with $V_i \cap V_j = \varnothing$, for all $i, j, i \neq j$. The E-GTSP can be stated as the problem of finding a minimum cost cycle that includes exactly one vertex from each cluster.

If the cost matrix $C = (c_{ij})$ is symmetric, i.e., $c_{ij} = c_{ji}$ for all $i, j, i \neq j$, the problem is called *symmetric*. Otherwise it is called *asymmetric*.

Figure 1 is an illustration of the problem. The lines depict a feasible cycle, called a *g-tour*, a closed path visiting exactly one vertex of each cluster.

It is well known that any E-GTSP instance can be transformed into an *asymmetric* TSP instance containing the same number of vertices [2–4]. The transformation can be described as follows, where V' and c' denote the vertex set and cost matrix of the transformed instance:

(a) V' is equal to V.
(b) Create an arbitrary directed cycle of the vertices within each cluster and define $c'_{ij} = 0$, when v_i and v_j belong to the same cluster and v_j succeeds v_i in the cycle.
(c) When v_i and v_j belong to different clusters, define $c'_{ij} = c_{kj} + M$, where v_k is the vertex that succeeds v_i in a cycle, and M is a sufficiently large constant. It suffices that the constant M be larger than the sum of the n largest costs.
(d) Otherwise, define $c'_{ij} = 2M$.

This transformation works since having entered a cluster at a vertex v_i, an optimal TSP tour always visits all other vertices of the cluster before it moves to the next cluster.

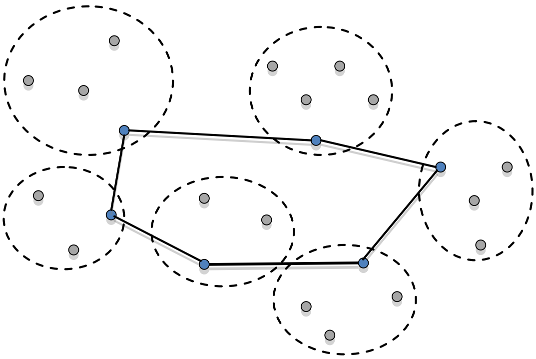

Fig. 1 Illustration of the E-GTSP for a symmetric instance with 6 clusters ($n = 23, m = 6$)

The optimal TSP tour must have zero cost inside the cluster and must have exactly m inter-cluster edges. Thus, the cost of the g-tour for the E-GTSP is the cost of the TSP tour minus mM. The g-tour can be extracted by picking the first vertex from each cluster in the TSP tour.

The transformation allows one to solve E-GTSP instances using an asymmetric TSP solver. However, in the past this approach has had very little application, because the produced TSP instances have an unusual structure, which is hard to handle for many existing TSP solvers. Since a near-optimal TSP solution may correspond to an infeasible E-GTSP solution, heuristic TSP solvers are often considered inappropriate [5,6]. In this paper, it is shown that this need not be the case if the state-of-the-art heuristic TSP solver LKH is used.

LKH [7,8] is a powerful local search heuristic for the TSP based on the variable depth local search of Lin and Kernighan [9]. Among its characteristics may be mentioned its use of 1-tree approximation for determining a candidate edge set, extension of the basic search step, and effective rules for directing and pruning the search. LKH is available free of charge for scientific and educational purposes from http://www.ruc.dk/~keld/research/LKH. The following section describes how LKH can be used to solve the E-GTSP.

2 Implementing an E-GTSP Solver Based on LKH

The input to LKH is given in two files:

1. A *problem file* in TSPLIB format [10], which contains a specification of the TSP instance to be solved. A problem may be symmetric or asymmetric. In the latter case, the problem is transformed by LKH into a symmetric one with $2n$ vertices.
2. A *parameter file*, which contains the name of the problem file, together with some parameter values that control the solution process. Parameters that are not specified in this file are given suitable default values.

An E-GTSP solver based on LKH should therefore be able to read an E-GTSP instance, transform it into an asymmetric TSP instance, produce the two input files required by LKH, let LKH solve the TSP instance, and extract the g-tour from the obtained TSP tour. A more precise algorithmic description is given below:

1. Read the E-GTSP instance.
2. Transform it into an asymmetric TSP instance.
3. Write the TSP instance to a problem file.
4. Write suitable parameter values to a parameter file.
5. Execute LKH given these two files.
6. Extract the g-tour from the TSP solution tour.
7. Perform post-optimization of the g-tour.

Comments:

1. The instance must be given in the GTSPLIB format, an extension of the TSPLIB format, which allows for specification of the clusters. A description of the GTSPLIB format can be found at http://www.cs.rhul.ac.uk/home/zvero/GTSPLIB/.

2. The constant M is chosen as `INT_MAX/4`, where `INT_MAX` is the maximal value that can be stored in an `int` variable. The transformation results in an asymmetric $n \times n$ cost matrix.
3. The problem file is in TSPLIB format with EDGE_WEIGHT_TYPE set to EXPLICIT, and EDGE_WEIGHT_FORMAT set to FULL_MATRIX.
4. The transformation induces a fair amount of degeneracy, which makes the default parameter settings of LKH inappropriate. For example, tests have shown that it is necessary to work with candidate edge set that is larger than by default. For more information, see the next section.
5. The E-GTSP solver has been implemented in C to run under Linux. This has made it possible to execute LKH as a child process.
6. The g-tour is easily found by picking the first vertex from each cluster during a sequential traversal of the TSP tour. The g-tour is checked for feasibility.
7. In this step, attempts are made to optimize the g-tour by two means: (1) using LKH for local optimization as described above but now on an instance with m vertices (the vertices of the g-tour). (2) Performing so-called *cluster optimization*, a well-known post-optimization heuristic for the E-GTSP [11]. This heuristic attempts to find a g-tour that visits the clusters in the same order as the current g-tour, but is cheaper than this. It is implemented as a shortest path algorithm and runs in $O(nm^2)$ time. If the smallest cluster has a size of $O(1)$, the algorithm may be implemented to run in O(nm) time. A detailed description of the heuristic and its implementation can be found in [6,12]. Local optimization and cluster optimization are performed as long as it is possible to improve the current best g-tour.

3 Computational evaluation

The program, which is named GLKH, was coded in C and run under Linux on an iMac 3.4 GHz Intel i7-3770 (Ivy Bridge) with 8 MB cache and 32 GB RAM. Version 2.0.7 of LKH was used. The program uses only one of the computer's four CPU cores.

The program was tested using E-GTSP instances generated from instances in TSPLIB [10] by applying the clustering method of Fischetti et al. [12]. This method, known as *K-center clustering*, clusters the vertices based on proximity to each other. For a given instance, the number of clusters is fixed to $m = \lceil n/5 \rceil$.

In addition, the program has been tested on a series of large-scale instances generated from clustered instances taken from the 8th DIMACS Implementation Challenge [13] and from the national instances on the TSP web page of William Cook et al. [14].

The number of clusters in the test instances varies between 4 and 17,180, and the number of vertices varies between 14 and 85,900.

Finally, the program has been tested on a large number of instances generated by transforming arc routing problem instances into E-GTSP instances.

3.1 Results for standard GTSPLIB instances

For instances with at most 1084 vertices, the following non-default parameter settings for LKH were chosen and written to a parameter file:

ASCENT_CANDIDATES = 500
MAX_CANDIDATES = 30
OPTIMUM = $\langle$*best known cost*$\rangle$
POPULATION_SIZE = 5

Below is given the rationale for the choice of the parameters:

ASCENT_CANDIDATES: The candidate sets that are used in the Lin–Kernighan search process are found using a Held–Karp subgradient ascent algorithm based on minimum 1-trees [15]. In order to speed up the ascent, the 1-trees are generated in a sparse graph. The value of the parameter ASCENT_CANDIDATES specifies the number of edges emanating from each vertex in this graph. The default value in LKH is 50. However, the unusual structure of the transformed problem made it necessary to use a larger value. After preliminary experiments, the value 500 was chosen.

MAX_CANDIDATES: This parameter is used to specify the size of the candidate sets used during the Lin–Kernighan search. Its value specifies the maximum number of candidate edges emanating from each vertex. The default value in LKH is 5. But also here it is necessary to use a larger value. After some preliminary experiments, the value 30 was chosen.

OPTIMUM: This parameter may be used to supply a best known solution cost. The algorithm will stop if this value is reached during the search process.

POPULATION_SIZE: A genetic algorithm is used, in which ten runs are performed (RUNS = 10 is default in LKH) with a population size of five individuals (TSP tours). That is, when five different tours have been obtained, the remaining runs will be given initial tours produced by combining individuals from the population.

LKH's default basic move type, MOVE_TYPE = 5, is used. LKH offers the possibility of using higher-order and/or non-sequential move types in order to improve the solution quality [8]. However, the relatively large size of the candidate set makes the local search too time-consuming for such move types.

Tables 1 and 2 show the test results for instances with at most 1084 vertices. This set of benchmark instances is commonly used in the literature. Each test was repeated ten times. To ease a comparison, the tables follow the format used in [16]. The column headers are as follows:

Name the instance name. The prefix number is the number of clusters of the instance; the suffix number is the number of vertices.

Opt. the best known solution cost. The exact solution cost (optimum) is known for all instances with at most 89 clusters and 443 vertices [11]. For the rest of the instances the best known solutions costs are taken from [16].

Value the average cost value returned in the ten tests.

Table 1 Results for small benchmark instances (STOP_AT_OPTIMUM = YES)

Name	Opt.	Value	Error (%)	Opt. (%)	Time (s)
3burma14	1805	1805.0	0.00	100	0.0
4br17 (asym.)	31	31.0	0.00	100	0.0
4gr17	1389	1389.0	0.00	100	0.0
5gr21	4539	4539.0	0.00	100	0.0
5gr24	334	334.0	0.00	100	0.0
5ulysses22	5307	5307.0	0.00	100	0.0
6bayg29	707	707.0	0.00	100	0.0
6bays29	822	822.0	0.00	100	0.0
6fri26	481	481.0	0.00	100	0.0
7ftv33 (asym.)	476	476.0	0.00	100	0.0
8ftv35 (asym.)	525	525.0	0.00	100	0.0
8ftv38 (asym.)	511	511.0	0.00	100	0.0
9dantzig42	417	417.0	0.00	100	0.0
10att48	5394	5394.0	0.00	100	0.0
10gr48	1834	1834.0	0.00	100	0.0
10hk48	6386	6386.0	0.00	100	0.0
11berlin52	4040	4040.0	0.00	100	0.0
11eil51	174	174.0	0.00	100	0.0
12brazil58	15,332	15,332.0	0.00	100	0.0
14st70	316	316.0	0.00	100	0.0
16eil76	209	209.0	0.00	100	0.0
16pr76	64,925	64,925.0	0.00	100	0.0
20gr96	29,440	29,440.0	0.00	100	0.0
20rat99	497	497.0	0.00	100	0.0
20kroA100	9711	9711.0	0.00	100	0.0
20kroB100	10,328	10,328.0	0.00	100	0.0
20kroC100	9554	9554.0	0.00	100	0.0
20kroD100	9450	9450.0	0.00	100	0.0
20kroE100	9523	9523.0	0.00	100	0.0
20rd100	3650	3650.0	0.00	100	0.0
21eil101	249	249.0	0.00	100	0.1
21lin105	8213	8213.0	0.00	100	0.0
22pr107	27,898	27,898.0	0.00	100	0.0
24gr120	2769	2769.0	0.00	100	0.1
25pr124	36,605	36,605.0	0.00	100	0.1
26bier127	72,418	72,418.0	0.00	100	0.1
26ch130	2828	2828.0	0.00	100	0.1
28gr137	36,417	36,417.0	0.00	100	0.1
28pr136	42,570	42,570.0	0.00	100	0.2
29pr144	45,886	45,886.0	0.00	100	0.1

Table 1 continued

Name	Opt.	Value	Error (%)	Opt. (%)	Time (s)
30ch150	2750	2750.0	0.00	100	0.5
30kroA150	11,018	11,018.0	0.00	100	0.1
30kroB150	12,196	12,196.0	0.00	100	0.1
31pr152	51,576	51,576.0	0.00	100	0.2
32u159	22,664	22,664.0	0.00	100	0.1
35si175	5564	5564.0	0.00	100	0.4
36brg180	4420	4420.0	0.00	100	0.2
39rat195	854	854.0	0.00	100	0.3
Average			0.00	100	

Table 2 Results for large benchmark instances (STOP_AT_OPTIMUM = YES)

Name	Opt.	Value	Error (%)	Opt. (%)	Time (s)
40d198	10,557	10,557.0	0.00	100	1.9
40kroa200	13,406	13,406.0	0.00	100	0.4
40krob200	13,111	13,111.0	0.00	100	0.6
41gr202	23,301	23,301.0	0.00	100	0.4
45ts225	68,340	68,340.0	0.00	100	1.9
45tsp225	1612	1612.0	0.00	100	2.3
46pr226	64,007	64,007.0	0.00	100	0.1
46gr229	71,972	71,972.0	0.00	100	0.3
53gil262	1013	1013.0	0.00	100	1.4
53pr264	29,549	29,549.0	0.00	100	0.6
56a280	1079	1079.0	0.00	100	0.9
60pr299	22,615	22,615.0	0.00	100	1.5
64lin318	20,765	20,765.0	0.00	100	1.7
65rbg323 (asym.)	471	471.0	0.00	100	0.3
72rbg358 (asym.)	693	693.0	0.00	100	0.8
80rd400	6361	6361.0	0.00	100	8.1
81rbg403 (asym.)	1170	1170.0	0.00	100	3.9
84fl417	9651	9651.0	0.00	100	2.0
87gr431	101,946	101,946.0	0.00	100	7.6
88pr439	60,099	60,099.0	0.00	100	2.6
89pcb442	21,657	21,657.0	0.00	100	8.1
89rbg443 (asym.)	632	632.0	0.00	100	25.6
99d493	20,023	20,023.4	0.00	100	170.5
107ali535	128,639	128,639.0	0.00	100	18.4
107att532	13,464	13,464.0	0.00	100	12.0
107si535	13,502	13,502.0	0.00	100	34.5

Table 2 continued

Name	Opt.	Value	Error (%)	Opt. (%)	Time (s)
113pa561	1038	1038.0	0.00	100	7.7
115u574	16,689	16,689.0	0.00	100	26.5
115rat575	2388	2388.0	0.00	100	45.5
131p654	27,428	27,428.0	0.00	100	14.0
132d657	22,498	22,498.0	0.00	100	490.8
134gr666	163,028	163,028.0	0.00	100	162.3
145u724	17,272	17,272.0	0.00	100	145.4
157rat783	3262	3262.9	0.03	70	764.4
200dsj1000	9,187,884	9,187,884.0	0.00	100	794.4
201pr1002	114,311	114,311.0	0.00	100	164.8
207si1032	22,306	22,306.0	0.00	100	1202.5
212u1060	106,007	106,029.5	0.02	50	2054.9
217vm1084	130,704	130,704.0	0.00	100	209.0
Average			0.00	98	

Error (%) the error, in percent, of the average cost above the best known solution cost.

Opt. (%) the number of tests, in per cent, in which the best known solution cost was reached.

Time (s) the average CPU time, in seconds, used for one test.

As can be seen in Table 1, the small benchmark instances are quickly solved to optimality.

Table 2 shows that all large benchmark instances are solved to optimality too. In comparison with the results obtained for the same instances by Gutin and Karapetyan's state-of-the-art solver GK [16, p. 58], the optimality percentage for GLKH is higher (98 versus 81 %). This higher success rate is obtained at the expense of worse running times (a factor of about 50 for the largest instances). However, the running times for GLKH are satisfactory and reasonable for practical purposes.

In the real world, the optimum cost is not known in advance. Table 3 shows the performance of GLKH for the set of small instances when the OPTIMUM parameter is left out and LKH is given a time limit of 1 s. Table 4 shows the performance for the set of large instances when the time limit is set to 1 min.

Considering that GLKH uses LKH without any modifications, its performance is surprisingly impressive. In this connection it may be mentioned that Gutin and Karapetyan have proposed several adaptions of the Lin–Kernighan heuristic for the GTSP that does not use problem transformation but exploits the special structure of this problem type [17]. However, none of these adaptions were as successful as their state-of-the-art solver, GK, when it came to tour quality.

Table 3 Results for small benchmark instances (STOP_AT_OPTIMUM = NO, TIME_LIMIT = 1, RUNS = 1)

Name	Opt.	Value	Error (%)	Opt. (%)	Time (s)
3burma14	1805	1805.0	0.00	100	0.0
4br17 (asym.)	31	31.0	0.00	100	0.1
4gr17	1389	1389.0	0.00	100	0.0
5gr21	4539	4539.0	0.00	100	0.0
5gr24	334	334.0	0.00	100	0.0
5ulysses22	5307	5307.0	0.00	100	0.0
6bayg29	707	707.0	0.00	100	0.0
6bays29	822	822.0	0.00	100	0.0
6fri26	481	481.0	0.00	100	0.0
7ftv33 (asym.)	476	476.0	0.00	100	0.3
8ftv35 (asym.)	525	525.0	0.00	100	0.3
8ftv38 (asym.)	511	511.0	0.00	100	0.6
9dantzig42	417	417.0	0.00	100	0.4
10att48	5394	5394.0	0.00	100	0.2
10gr48	1834	1834.0	0.00	100	0.2
10hk48	6386	6386.0	0.00	100	0.2
11berlin52	4040	4040.0	0.00	100	0.2
11eil51	174	174.0	0.00	100	0.5
12brazil58	15,332	15,332.0	0.00	100	0.4
14st70	316	316.0	0.00	100	0.5
16eil76	209	209.0	0.00	100	0.5
16pr76	64,925	64,925.0	0.00	100	0.5
20gr96	29,440	29,440.0	0.00	100	1.0
20rat99	497	497.0	0.00	100	0.9
20kroA100	9711	9711.0	0.00	100	1.0
20kroB100	10,328	10,328.0	0.00	100	1.0
20kroC100	9554	9554.0	0.00	100	1.0
20kroD100	9450	9450.0	0.00	100	1.0
20kroE100	9523	9523.0	0.00	100	1.0
20rd100	3650	3650.0	0.00	100	1.0
21eil101	249	249.0	0.00	100	1.0
21lin105	8213	8213.0	0.00	100	1.0
22pr107	27,898	27,898.0	0.00	100	1.0
24gr120	2769	2769.0	0.00	100	1.0
25pr124	36,605	36,605.0	0.00	100	1.0
26bier127	72,418	72,418.0	0.00	100	1.0
26ch130	2828	2828.0	0.00	100	1.0
28gr137	36,417	36,417.0	0.00	100	1.0
28pr136	42,570	42,570.0	0.00	100	1.0

Table 3 continued

Name	Opt.	Value	Error (%)	Opt. (%)	Time (s)
29pr144	45,886	45,886.0	0.00	100	1.0
30ch150	2750	2750.0	0.00	100	1.0
30kroA150	11,018	11,018.0	0.00	100	1.0
30kroB150	12,196	12,196.0	0.00	100	1.0
31pr152	51,576	51,576.0	0.00	100	1.0
32u159	22,664	22,664.0	0.00	100	1.0
35si175	5564	5564.0	0.00	100	1.0
36brg180	4420	4420.0	0.00	100	1.0
39rat195	854	854.0	0.00	100	1.0
Average			0.00	100	

Table 4 Results for large benchmark instances (STOP_AT_OPTIMUM = NO, TIME_LIMIT = 60, RUNS = 1)

Name	Opt.	Value	Error (%)	Opt. (%)	Time (s)
40d198	10,557	10,557.0	0.00	100	24.0
40kroa200	13,406	13,406.0	0.00	100	4.3
40krob200	13,111	13,111.0	0.00	100	5.3
41gr202	23,301	23,301.0	0.00	100	7.9
45ts225	68,340	68,340.0	0.00	100	9.5
45tsp225	1612	1612.5	0.03	90	9.7
46pr226	64,007	64,007.0	0.00	100	11.0
46gr229	71,972	71,972.0	0.00	100	9.6
53gil262	1013	1013.0	0.00	100	11.5
53pr264	29,549	29,549.0	0.00	100	28.2
56a280	1079	1079.0	0.00	100	17.3
60pr299	22,615	22,615.0	0.00	100	14.1
64lin318	20,765	20,765.0	0.00	100	17.6
65rbg323 (asym.)	471	471.0	0.00	100	60.2
72rbg358 (asym.)	693	693.0	0.00	100	60.4
80rd400	6361	6361.0	0.00	100	29.0
81rbg403 (asym.)	1170	1170.0	0.00	100	61.7
84fl417	9651	9651.0	0.00	100	60.3
87gr431	101,946	101,946.0	0.00	100	27.9
88pr439	60,099	60,099.0	0.00	100	56.0
89pcb442	21,657	21,657.0	0.00	100	33.7
89rbg443 (asym.)	632	632.2	0.03	80	60.7
99d493	20,023	20,033.8	0.05	30	57.6
107ali535	128,639	128,640.4	0.00	90	60.3

Table 4 continued

Name	Opt.	Value	Error (%)	Opt. (%)	Time (s)
107att532	13,464	13,464.0	0.00	100	60.2
107si535	13,502	13,505.2	0.02	80	59.5
113pa561	1038	1038.0	0.00	100	48.7
115u574	16,689	16,711.5	0.13	60	60.3
115rat575	2388	2388.9	0.04	90	59.2
131p654	27,428	27,428.0	0.00	100	60.6
132d657	22,498	22,498.0	0.12	20	60.4
134gr666	163,028	163,529.4	0.31	30	60.4
145u724	17,272	17,321.9	0.29	30	60.4
157rat783	3262	3272.4	0.32	0	60.5
200dsj1000	9,187,884	9,218,751.6	0.34	20	61.1
201pr1002	114,311	114,311.0	0.07	50	60.9
207si1032	22,306	22,352.0	0.21	0	64.1
212u1060	106,007	106,524.9	0.49	10	61.0
217vm1084	130,704	131,274.5	0.44	10	61.0
Average			0.07	77	

3.2 Results for new very large GTSPLIB instances

To provide some very-large-scale instances for research use, GTSPLIB has been extended with 44 instances ranging in size from 1000 to 85,900 vertices (see Table 5). The instances are generated from TSPLIB instances with the following exceptions:

- The instances 200E1k.0, 633E3k.0, 2000E10k.0, 6325E31k.0, 200C1k.0, 633C3k, and 6325C31k.0 are generated from instances used in the 8th DIMACS Implementation Challenge [13]. The E-instances consist of 1000, 3162, 10,000, and 31,623

Table 5 Results for the new very large benchmark instances

Name	Best	Value	Error (%)	Time (s)
10C1k.0	2,522,585	2,522,605	0.00	4.9
200C1k.0	6,375,154	6,375,154	0.00	133.7
200E1k.0	9,662,857	9,670,122	0.08	241.8
49usa1097	10,465,466	10,465,466	0.00	50.6
235pcb1173	23,399	23,669	1.15	367.4
259d1291	28,400	28,400	0.00	284.1
261rl1304	150,468	150,860	0.26	415.2
265rl1323	154,023	154,134	0.07	418.4

Table 5 continued

Name	Best	Value	Error (%)	Time (s)
276nrw1379	20,050	20,194	0.72	398.3
280fl1400	15,316	15,316	0.00	119.6
287u1432	54,482	54,632	0.28	345.2
316fl1577	14,182	14,183	0.01	1294.2
331d1655	29,443	29,620	0.60	706.8
350vm1748	185,459	185,588	0.07	563.6
364u1817	25,530	25,667	0.54	724.0
378rl1889	184,034	185,246	0.66	694.0
421d2103	40,049	40,270	0.55	806.5
431u2152	27,614	27,719	0.38	815.7
464u2319	65,758	66,589	1.26	748.7
479pr2392	169,874	171,361	0.88	938.9
608pcb3038	52,416	53,565	2.19	1082.9
31C3k.0	3,553,142	3,553,142	0.00	482.3
633C3k.0	10,255,031	10,255,031	0.00	2833.9
633E3k.0	16,197,552	16,484,977	1.77	1218.0
759fl3795	18,662	18,691	0.16	3802.4
893fnl4461	63,163	65,060	3.00	1825.1
1183rl5915	309,243	314,927	1.84	3000.1
1187rl5934	295,767	300,618	1.64	3505.5
1480pla7397	1,273,2870	12,793,563	0.48	5466.2
100C10k.0	6,158,999	6,240,251	1.32	3268.7
2000C10k.0	18,044,846	18,284,681	1.33	14,027.2
2000E10k.0	28,769,011	29,644,352	3.04	5138.5
2370rl11849	427,996	440,652	2.96	7640.7
2702usa13509	10,080,705	10,274,251	1.92	8356.8
2811brd14051	176,639	181,912	2.99	8472.6
3023d15112	628,259	649,649	3.40	9787.7
3703d18512	234,921	244,558	4.10	11,665.5
4996sw24978	417,631	428,690	2.65	20,395.1
316C31k.0	10,098,861	10,554,017	4.51	13,748.0
6325C31k.0	3,183,4048	32,105,148	0.85	54,866.3
6325E31k.0	50,503,475	52,533,090	4.02	27,188.6
6762pla33810	28,222,961	29,062,848	2.97	38,265.4
14202ch71009	2,322,839	2,378,232	2.38	90,408.7
17180pla85900	54,792,193	56,758,297	3.59	121,253.3
Average			1.38	

uniformly distributed points in a square. The C-instances consist of 1000, 3162, 10,000, and 31,623 clustered points. For a given size n of a C-instance, its points are clustered around $\lfloor n/10 \rfloor$ randomly chosen centers in a square.

- The instances 4996sw24978 and 14202ch71009 are generated from the National TSP benchmark library [14]. They consist, respectively, of 24,978 locations in Sweden and 71,009 locations in China.

All instances mentioned above were generated using Fischetti et al.'s clustering algorithm.

The following four instances in which clusters correspond to natural clusters have been added: 49usa1097, 10C1k.0, 31C3k.0, 100C10k.0, and 316C31k.0. The instance 49usa1097 consists of 1097 cities in the adjoining 48 US states, plus the District of Columbia. Figure 2 shows the current best g-tour for this instance. Figures 3 and 4 show the current best g-tour for 10C1k.0 and 200C1k.0, respectively.

The column *Best* of Table 5 shows the current best solution costs found by GLKH. These costs were found using several runs of GLKH where in each run the current best g-tour was used as input tour to GLKH and using the following non-default parameter settings:

ASCENT_CANDIDATES = 500
INITIAL_PERIOD = 1000
INPUT_TOUR_FILE = $\langle input\ g\text{-}tour\ file name\rangle$
MAX_CANDIDATES = 30
MAX_TRIALS = 1000
OPTIMUM = $\langle best\ known\ cost\rangle$
OUTPUT_TOUR_FILE = $\langle output\ g\text{-}tour\ file name\rangle$
PRECISION = 10
RUNS = 1

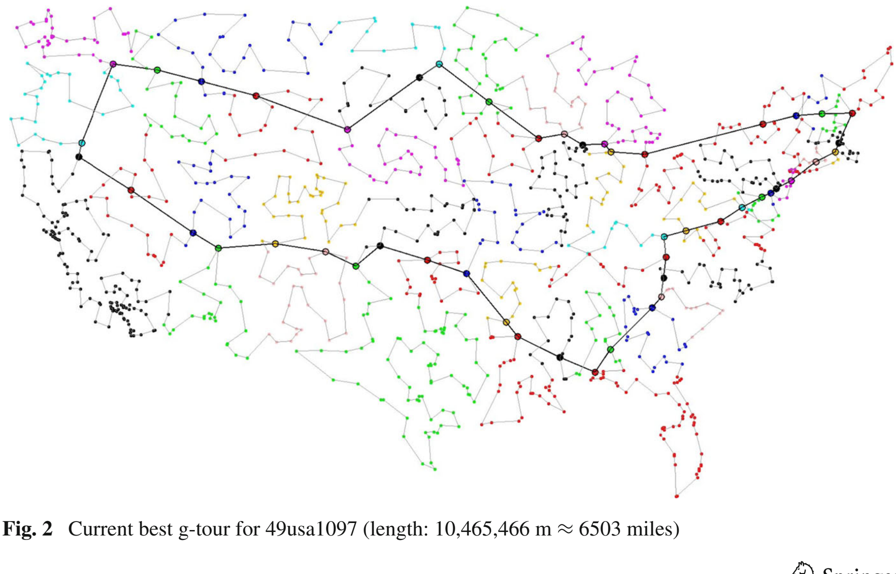

Fig. 2 Current best g-tour for 49usa1097 (length: 10,465,466 m ≈ 6503 miles)

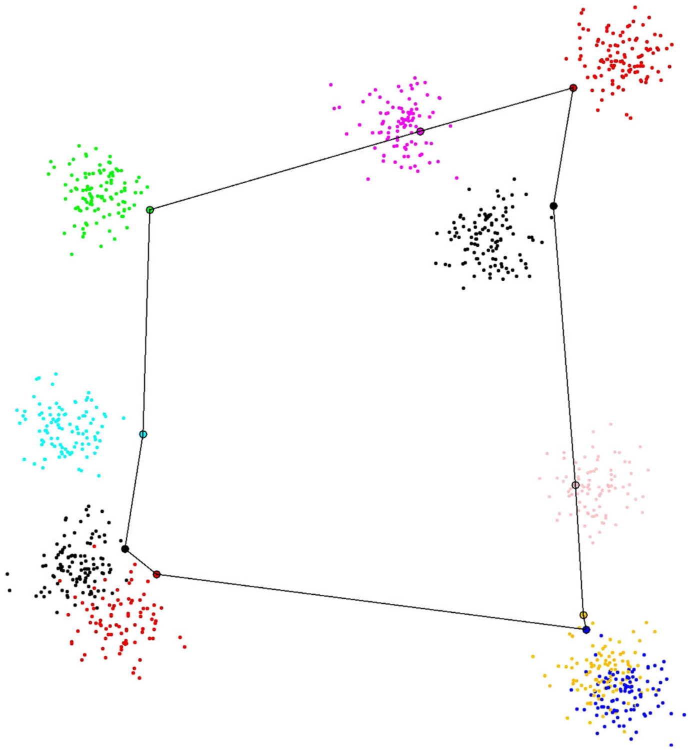

Fig. 3 Current best g-tour for 10C1k.0 (10 natural clusters)

The parameter INITIAL_PERIOD specifies the length of the first period in the Held–Karp ascent (default is $n/2$). MAX_TRIALS specifies the maximum number of trials (iterations) in the iterated Lin-Kernighan procedure (default is n). For some of the instances, the transformed costs are so large that the default precision in the π-transformed costs of LKH cannot be maintained but has to be reduced. The default precision of 100, which corresponds to two decimal places, is reduced to 10, which corresponds to one decimal place. The number of RUNS is set to 1 (default is 10).

It may also be mentioned that the parameter MERGE_TOUR_FILE can be used in attempts to produce a best possible g-tour from two or more given g-tours. Edges that are common to the corresponding TSP tours are fixed in the Lin-Kernighan search process.

The last three columns of the table give the results when the parameter INPUT_TOUR_FILE is omitted.

3.3 Results for arc routing problems

The goal of arc routing problems (ARPs) is to determine a minimum cost closed walk passing through some arcs and edges of a graph. Formally, ARPs are defined on a graph $G = (V, A, E)$ where $V = \{v_1, \ldots, v_n\}$ is a set of vertices, A is a set of (directed)

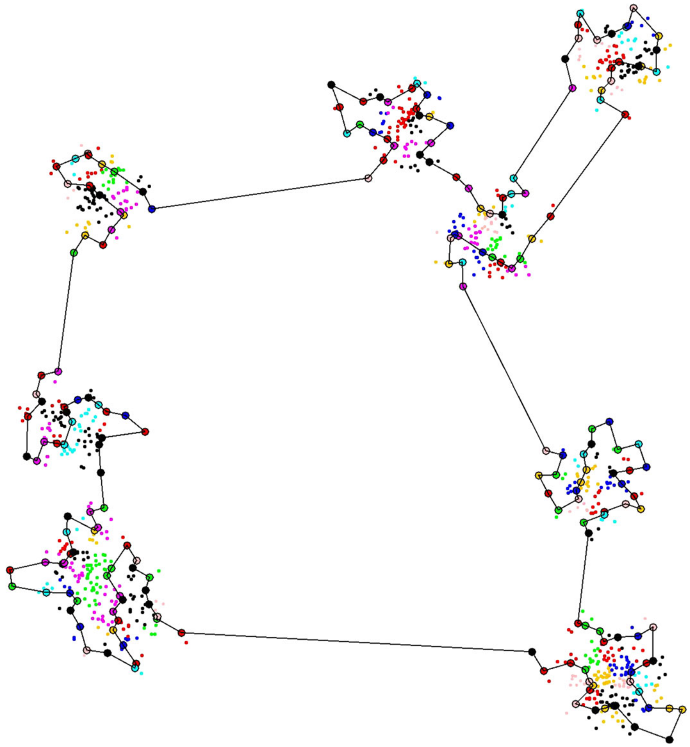

Fig. 4 Current best g-tour for 200C1k.0 (200 K-center clusters)

arcs $a_{ij}(i \neq j)$, and E is a set of (undirected) edges $e_{ij}(i < j)$. Non-negative costs c_{ij} and d_{ij} are associated with arcs a_{ij} and with edges e_{ij}, respectively. It is not necessary to traverse all arcs or edges. Denote by A_R and E_R the subsets of required arcs and edges, respectively. The aim is to determine a least cost closed walk on G including all required arcs and edges at least once.

If the walk also has to pass through a certain subset of required vertices, $V_R \subseteq V$, we have the general routing problem (GRP).

Depending on problem properties, some well-known classes of routing problems can be obtained from this definition. In this paper the following classes will be tackled:

- The Mixed Chinese Postman Problem (MCPP): $A_R = A \neq \varnothing, E_R = E \neq \varnothing, d_{ij} = d_{ji}$ for all $i, j, V_R = \varnothing$.
- The Windy Postman Problem (WPP): $A = \varnothing, E_R = E \neq \varnothing, d_{ij} \neq d_{ji}$ for at least one edge $e_{ij}, V_R = \varnothing$.
- The Undirected, Mixed and Windy Rural Postman Problems (URPP, MRPP, WRPP), which are defined similarly, except that now $\mathrm{A}_R \subset \mathrm{A}$ or $\mathrm{E}_R \subset \mathrm{E}$.
- The General Routing Problem (GRP): $A = \varnothing, E_R = E \neq \varnothing, d_{ij} = d_{ji}$ for all $i, j, V_R \neq \varnothing$.
- The Mixed General Routing Problem (MGRP): $A = \varnothing, E_R = E \neq \varnothing, d_{ij} = d_{ji}$ for all $i, j, V_R \neq \varnothing$.

- The Windy General Routing Problem (WGRP): $A = \varnothing$, $E_R = E \neq \varnothing$, $d_{ij} \neq d_{ji}$ for at least one edge e_{ij}, $V_R \neq \varnothing$.

All these problems can easily be transformed into E-GTSP [18]. The transformed problem is defined on a graph $H = (W, B)$. In this graph W consists of one vertex w_{ij} for each required arc v_{ij} of G, one vertex w_{ii} for each required vertex v_i, and two vertices w_{ij} and w_{ji} for each required edge e_{ij} (one for each of the corresponding opposite arcs, only one of which is required). B is the set of all arcs linking two vertices of W. Each vertex pair (w_{ki}, w_{lj}) in the transformed problem defines an arc of W with a cost equal to $s_{il} + c_{lj}$, where s_{il} denotes the cost of a shortest path from v_i to v_l on G.

We have thus transformed the original arc routing problem into E-GTSP, where each cluster consists of either one or two vertices. Clusters consisting of two vertices correspond to required edges in the original problem, whereas single vertex clusters correspond to required arcs and required vertices in the original problem.

Coberán et al. have provided a large library of test instances for arc routing problems [19]. The library includes 1042 instances of URPP, GRP, MCPP, MRPP, MGRP, WPP, WRPP and WGRP. All these instances have be transformed into E-GTSP and then solved by GLKH using the following non-default parameter settings:

ASCENT_CANDIDATES = 500
INITIAL_PERIOD = 1000
MAX_CANDIDATES = 12
MAX_TRIALS = 1000
OPTIMUM = ⟨*best known cost*⟩
RUNS = 1

Table 6 summarizes the computational results. The table contains average values over all considered instances of the respective type. The columns '*n*' and '*m*' report the average number of vertices and clusters in the E-GTSP instances.

The following observations can be made:

- The solution quality is good for all instances. Optima are found for about half of the instances and the average deviation from the optimal solution is less than 3 %.
- The instances in MCPP and WPP are the most difficult for GLKH. Other parameter settings might lead to a better solution quality. However, this will probably be at the expense of unacceptable running times. For these large instances, GLKH cannot compete with the highly sophisticated exact algorithm of Corberán et al. [20].
- Tests similar to those reported in Table 6 have been conducted by Drexl [21, p. 10]. Using Gutin and Karapetyan's heuristic E-GTSP solver GK, he found that GK performed acceptable for instances with up to about 200 clusters. However, for instances with more than 500 clusters, the gap to the optimal solutions usually exceeded 10 %. As seen, GLKH performs better than GK for instances with many clusters.

Currently, optima are known for 998 out of the 1042 instances. It may be mentioned, that until now GLKH has been able to find new best upper bounds for 22 of the remaining 44 instances:

Table 6 Results for arc routing problems

Instance classes	# of inst.	$\|V\|$	$\|A\|+\|E\|$	$\|A_R\|+\|E_R\|$	n	m	Error (%)	Opt. (%)	Time (s)
URPP: UR500	12	446	1129	616	1232	616	0.32	41.7	141.1
URPP: UR750	12	666	1698	907	1813	907	0.40	25.0	231.4
URPP: UR1000	12	886	2290	1215	2430	1215	0.50	25.0	358.4
GRP: Alba	15	116	174	86	196	110	0.00	100.0	0.2
GRP: Madr	15	196	316	158	347	189	0.00	100.0	1.5
GRP: GRP	10	116	174	75	178	102	0.00	100.0	0.2
MCPP: MA05	12	500	1158	1158	1773	1158	0.56	8.3	1028.9
MCPP: MB05	12	500	1210	1210	1836	1210	0.24	25.0	740.2
MCPP: MA10	12	1000	2319	2319	3555	2319	0.86	0.0	2958.7
MCPP: MB10	12	1000	2442	2442	3702	2442	0.61	16.7	2283.6
MCPP: MA15	12	1500	3479	3479	5330	3479	1.06	0.0	4686.8
MCPP: MB15	12	1500	3631	3631	5511	3631	0.57	8.3	3726.4
MCPP: MA20	12	2000	4645	4645	7108	4645	1.09	0.0	7031.0
MCPP: MB20	12	2000	4829	4829	7329	4829	0.58	0.0	5226.9
MCPP: MA30	12	3000	6959	6959	10,664	6959	1.20	0.0	12,627.1
MCPP: MB30	12	3000	7131	7131	10,877	7131	0.75	0.0	8511.3
MRPP: RB	18	449	1134	610	1376	610	0.02	72.2	141.0
MRPP: RD	18	900	2315	1230	2759	1230	0.08	33.3	530.9
MGRP: Alba	25	116	174	88	177	118	0.00	100.0	0.2
MGRP: Alda	31	214	351	168	324	217	0.00	93.5	7.1
MGRP: Madr	25	196	316	158	301	205	0.00	100.0	1.7
MGRP: GB	18	500	1218	610	980	661	0.03	83.3	110.5
MGRP: GD	18	1000	2450	1230	1958	1330	0.08	44.4	558.3
WPP: WA05	12	500	1160	1160	2321	1160	2.00	0.0	489.8
WPP: WB05	12	500	1213	1213	2426	1213	1.28	0.0	386.3
WPP: WA10	12	1000	2317	2317	4634	2317	2.44	0.0	1230.2
WPP: WB10	12	1000	2434	2434	4868	2434	2.15	0.0	899.4
WPP: WA15	12	1500	3493	3493	6986	3493	2.68	0.0	2229.5
WPP: WB15	12	1500	3655	3655	7309	3655	2.17	0.0	1678.3
WPP: WA20	12	2000	4645	4645	9289	4645	2.86	0.0	3412.7
WPP: WB20	12	2000	4826	4826	9652	4826	2.36	0.0	2561.3
WPP: WA30	12	3000	6961	6961	13,922	6961	2.97	0.0	7375.6
WPP: WB30	12	3000	7141	7141	14,282	7141	2.35	0.0	5060.1
WRPP: A100	72	116	174	102	204	102	0.01	94.4	1.1
WRPP: A500	27	401	1268	481	963	481	1.13	3.7	134.8
WRPP: A1000	27	848	2522	1149	2297	1149	1.71	0.0	505.4
WRPP: B	24	446	1132	610	1220	610	0.28	12.5	125.2
WRPP: C	24	673	1706	918	1837	918	0.40	8.3	170.8
WRPP: D	24	895	2287	1222	2443	1222	0.58	12.5	251.1

Table 6 continued

Instance classes	# of inst.	$\|V\|$	$\|A\|+\|E\|$	$\|A_R\|+\|E_R\|$	n	m	Error (%)	Opt. (%)	Time (s)
WRPP: M	72	196	316	187	374	187	0.04	68.1	10.8
WRPP: HD	54	86	173	85	170	85	0.01	96.3	0.9
WRPP: HG	54	83	149	77	154	77	0.00	100.0	0.6
WRPP: P	144	25	59	28	56	28	0.00	100.0	0.0
WGRP: A	27	500	1135	575	1223	648	1.18	0.0	191.1
WGRP: G	24	500	1210	599	1255	656	0.30	12.5	124.8

$$
\begin{array}{ll}
\text{MCPP MA3067}\ 6{,}529{,}588 & \text{WPP WB3061}: 178{,}684 \\
\text{MCCP MB2052}: 125{,}566 & \text{WPP WB3062}: 177{,}765 \\
\text{MCPP MB3052}: 151{,}284 & \text{WRPP C422}: 21{,}181 \\
\text{MCPP MB3065}: 201{,}187 & \text{WRPP D322}: 23{,}784 \\
\text{MGRP GD422}: 32{,}057 & \text{WRPP D421}: 24{,}539 \\
\text{MGRP GD425}: 37{,}581 & \text{WRPP D422}: 23{,}943 \\
\text{MGRP GD522}: 34{,}482 & \text{WGRP GB321}: 20{,}549 \\
\text{MGRP GD525}: 40{,}077 & \text{WGRP GB322}: 20{,}328 \\
\text{WPP WA3065}: 4{,}500{,}431 & \text{WGRP GB421}: 20{,}774 \\
\text{WPP WB3035}: 83{,}596 & \text{WGRP GB422}: 20{,}452 \\
\text{WPP WB3055}: 133{,}501 & \text{WGRP GB622}: 24{,}102
\end{array}
\qquad (1)
$$

4 Conclusion

This paper has evaluated the performance of LKH on E-GTSP instances that are transformed into standard asymmetric TSP instances using the Noon-Bean transformation [2,3]. Despite that LKH is not modified in order to cater for the unusual structure of the TSP instances, its performance is quite impressive. All instances in a well-known library of E-GTSP benchmark instances, GTSPLIB, could be solved to optimality in a reasonable time, and it was possible to find high-quality solutions for a series of new large-scale E-GTSP instances with up to 17,180 clusters and 85,900 vertices. Furthermore, it was possible to find solutions of good quality to large-scale undirected, mixed, and windy postman and general routing problem instances.

A possible future path for research would be to find a method for reducing the size of the candidate set. This would not only reduce running time but also allow LKH's high-order k-opt submoves to come into play and probably improve the solution quality. The algorithms for problem reduction presented in [22] might be useful here.

The developed software is free of charge for academic and non-commercial use and can be downloaded in source code together with an extended version of GTSPLIB and current best g-tours for these instances via http://www.ruc.dk/~keld/research/GLKH/.

References

1. Laporte, G., Asef-Vaziri, A., Sriskandarajah, C.: Some applications of the generalized travelling salesman problem. J. Oper. Res. Soc. **47**(12), 1461–1467 (1996)

2. Noon, C.E., Bean, J.C.: An efficient transformation of the generalized traveling salesman problem. INFOR **31**(1), 39–44 (1993)
3. Behzad. A., Modarres, M.: A New efficient transformation of generalized traveling salesman problem into traveling salesman problem. In: Proceedings of the 15th International Conference of Systems Engineering, ICSE (2002)
4. Ben-Arieh, D., Gutin, G., Penn, M., Yeo, A., Zverovitch, A.: Transformations of generalized ATSP into ATSP. Oper. Res. Lett. **31**(5), 357–365 (2003)
5. Laporte, G., Semet, F.: Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. INFOR **37**(2), 114–120 (1999)
6. Karapetyan, D., Gutin, G.: Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. Eur. J. Oper. Res. **219**(2), 234–251 (2012)
7. Helsgaun, K.: An effective implementation of the Lin–Kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**(1), 106–130 (2000)
8. Helsgaun, K.: General *k*-opt submoves for the Lin–Kernighan TSP heuristic. Math. Prog. Comput. **1**(2–3), 119–163 (2009)
9. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Oper. Res. **21**(2), 498–516 (1973)
10. Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
11. Fischetti, M., Salazar González, J.J., Toth, P.: A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. Oper. Res. **45**(3), 378–394 (1997)
12. Fischetti, M., SalazarGonzález, J.J., Toth, P.: The generalized traveling salesman and orientering problems. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and its Variations, pp. 602–662. Kluwer, Dordrecht (2002)
13. Johnson, D.S., McGeoch, L.A., Glover, F., Rego, C.: 8th DIMACS implementation challenge: the traveling salesman problem (2000). http://dimacs.rutgers.edu/Challenges/TSP/. Accessed 24 Apr 2015
14. National traveling salesman problems. http://www.math.uwaterloo.ca/tsp/world/countries.html. Accessed 24 Apr 2015
15. Held, M., Karp, R.M.: The traveling salesman problem and minimum spanning trees. Oper. Res. **18**(6), 1138–1162 (1970)
16. Gutin, G., Karapetyan, D.: A memetic algorithm for the generalized traveling salesman problem. Nat. Comput. **9**(1), 47–60 (2010)
17. Karapetyan, D., Gutin, G.: Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem. Eur. J. Oper. Res. **208**(3), 221–232 (2011)
18. Blais, M., Laporte, G.: Exact solution of the generalized routing problem through graph transformations. J. Oper. Res. Soc. **54**(8), 906–910 (2003)
19. Corberán, Á., Plana I., Sanchis, J.M.: Arc routing problems: data instances. http://www.uv.es/corberan/instancias.htm. Accessed 24 Apr 2015
20. Corberán, A., Oswald, M., Plana, I., Reinelt, G., Sanchis, J.M.: New results on the Windy Postman problem. Math. Program. Ser. A **132**, 309–332 (2012)
21. Drexl, M.: On the generalized directed rural postman problem. J. Oper. Res. Soc. (2013). doi:10.1057/jors.2013.60
22. Karapetyan, D., Gutin, G.: Generalized traveling salesman problem reduction algorithms. Algorithm Oper. Res. **4**, 144–154 (2009)