Solving the Equality Generalized Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm

Keld Helsgaun

MAY 2014 ROSKILDE UNIVERSITY COMPUTER SCIENCE RESEARCH REPORT #141

Keld Helsgaun

Computer Science
Roskilde University
P. O. Box 260
DK–4000 Roskilde
Denmark

Telephone: +45 4674 3839
Telefax: +45 4674 3072
Internet: http://www.ruc.dk/dat_en/
E-mail: datalogi@ruc.dk

ISSN 0109–9779

Research reports are available electronically from:

http://www.ruc.dk/dat_en/research/reports/

Solving the Equality Generalized Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm

Keld Helsgaun
E-mail: keld@ruc.dk

Computer Science
Roskilde University
DK-4000 Roskilde, Denmark

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 Traveling Salesman Problem (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 as a black box, 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 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

Mathematics Subject Classification: 90C27, 90C35, 90C59

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 $G = (V, E)$, where $V=\{v_1....v_n\}$ is the vertex set and $E=\{(v_i,v_j) : v_i, v_j \in V, i < j\}$ is the edge set. A non-negative cost c_{ij} is associated with each edge (v_i, v_j) and the vertex set V is partitioned into m mutual exclusive and exhaustive clusters $V_1....V_m$, 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 node 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*.

December 2013. Updated March 19, 2014

Figure 1 is an illustration of the problem. The lines depict a feasible cycle, called a *g-tour*.

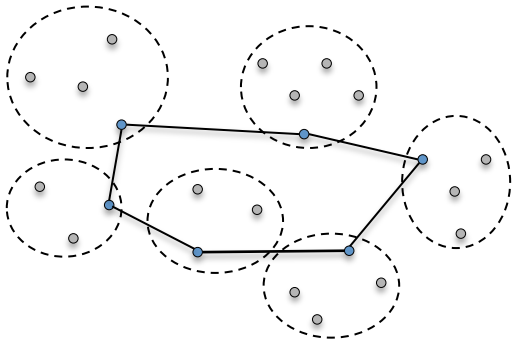

Figure 1 *Illustration of the E-GTSP for an instance with 6 clusters (n = 23, m =6).*

It is well known that any E-GTSP instance can be transformed into an asymmetric TSP instance containing the same number of vertices [2, 3, 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 M is 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. 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 as a black box 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 x 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 (using the Standard C Library function `popen()`).

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 Core i7 with 32 GB RAM. Version 2.0.7 of LKH was used.

The program was tested using E-GTSP instances generated from instances in TSPLIB [10] by applying the clustering method of Fischetti, Salazar, and Toth [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.

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

PROBLEM_FILE = GTSPLIB/<*instance name*>.gtsp
ASCENT_CANDIDATES = 500
MAX_CANDIDATES = 30
OPTIMUM = <*best known cost*>
PI_FILE = <π *file name*>
POPULATION_SIZE = 5

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

PROBLEM_FILE: The test instances have been placed in the directory GTSPLIB and have filename extension ".gtsp".

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.

PI_FILE: The penalties (π values) generated by the Held-Karp ascent are saved in a file such that subsequent test runs can reuse the values and skip the ascent.

POPULATION_SIZE: A genetic algorithm is used, in which 10 runs are performed (RUNS = 10 is default in LKH) with a population size of 5 individuals (TSP tours). That is, when 5 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.

Table 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. 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.

Value: the average cost value returned in the ten tests.

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 the 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 40 for the largest instances). However, the running times for GLKH are satisfactory and reasonable for practical purposes. Considering that GLKH uses LKH as a black box, without any modifications, its performance is surprisingly impressive.

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	15332	15332.0	0.00	100	0.0
14st70	316	316.0	0.00	100	0.0
16eil76	209	209.0	0.00	100	0.0
16pr76	64925	64925.0	0.00	100	0.0
20gr96	29440	29440.0	0.00	100	0.0
20rat99	497	497.0	0.00	100	0.0
20kroA100	9711	9711.0	0.00	100	0.0
20kroB100	10328	10328.0	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	27898	27898.0	0.00	100	0.0
24gr120	2769	2769.0	0.00	100	0.1
25pr124	36605	36605.0	0.00	100	0.1
26bier127	72418	72418.0	0.00	100	0.1
26ch130	2828	2828.0	0.00	100	0.1
28gr137	36417	36417.0	0.00	100	0.1
28pr136	42570	42570.0	0.00	100	0.2
29pr144	45886	45886.0	0.00	100	0.1
30ch150	2750	2750.0	0.00	100	0.5
30kroA150	11018	11018.0	0.00	100	0.1
30kroB150	12196	12196.0	0.00	100	0.1
31pr152	51576	51576.0	0.00	100	0.2
32u159	22664	22664.0	0.00	100	0.1
35si175	5564	5564.0	0.00	100	0.8
36brg180	4420	4420.0	0.00	100	0.2
39rat195	854	854.0	0.00	100	0.3
Average			0.00	100	

Table 1 *Results for small benchmark instances.*

Name	Opt.	Value	Error (%)	Opt. (%)	Time (s)
40d198	10557	10557.0	0.00	100	1.9
40kroa200	13406	13406.0	0.00	100	0.4
40krob200	13111	13111.0	0.00	100	0.6
41gr202	23301	23301.0	0.00	100	0.4
45ts225	68340	68340.0	0.00	100	1.9
45tsp225	1612	1612.0	0.00	100	2.3
46pr226	64007	64007.0	0.00	100	0.1
46gr229	71972	71972.0	0.00	100	0.3
53gil262	1013	1013.0	0.00	100	1.4
53pr264	29549	29549.0	0.00	100	0.6
56a280	1079	1079.0	0.00	100	0.9
60pr299	22615	22615.0	0.00	100	1.5
64lin318	20765	20765.0	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	101946	101946.0	0.00	100	7.6
88pr439	60099	60099.0	0.00	100	2.6
89pcb442	21657	21657.0	0.00	100	8.1
89rbg443 (asym.)	632	632.0	0.00	100	25.6
99d493	20023	20023.4	0.00	100	170.5
107ali535	128639	128639.0	0.00	100	18.4
107att532	13464	13464.0	0.00	100	12.0
107si535	13502	13502.0	0.00	100	34.5
113pa561	1038	1038.0	0.00	100	7.7
115u574	2388	2388.0	0.00	100	45.5
115rat575	16689	16689.0	0.00	100	26.5
131p654	27428	27428.0	0.00	100	14.0
132d657	22498	22498.0	0.00	100	490.8
134gr666	163028	163028.0	0.00	100	162.3
145u724	17272	17272.0	0.00	100	145.4
157rat783	3262	3262.9	0.03	70	764.4
200dsj1000	9187884	9187884.0	0.00	100	794.4
201pr1002	114311	114311.0	0.00	100	164.8
207si1032	22306	22306.0	0.00	100	1202.5
212u1060	106007	106029.5	0.02	50	2054.9
217vm1084	130704	130704.0	0.00	100	209.0
Average			0.00	98	

Table 2 *Results for large benchmark 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 3). 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, 10000, and 31623 uniformly distributed points in a square. The C-instances consist of 1000, 3162, 10000, and 31623 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 24978 locations in Sweden and 71009 locations in China.

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

The following 4 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 U.S. states, plus the District of Columbia. Figure 2 shows the current best g-tour for this instance. Figure 3 and 4 show the current best g-tour for 10C1k.0 and 200C1k.0, respectively.

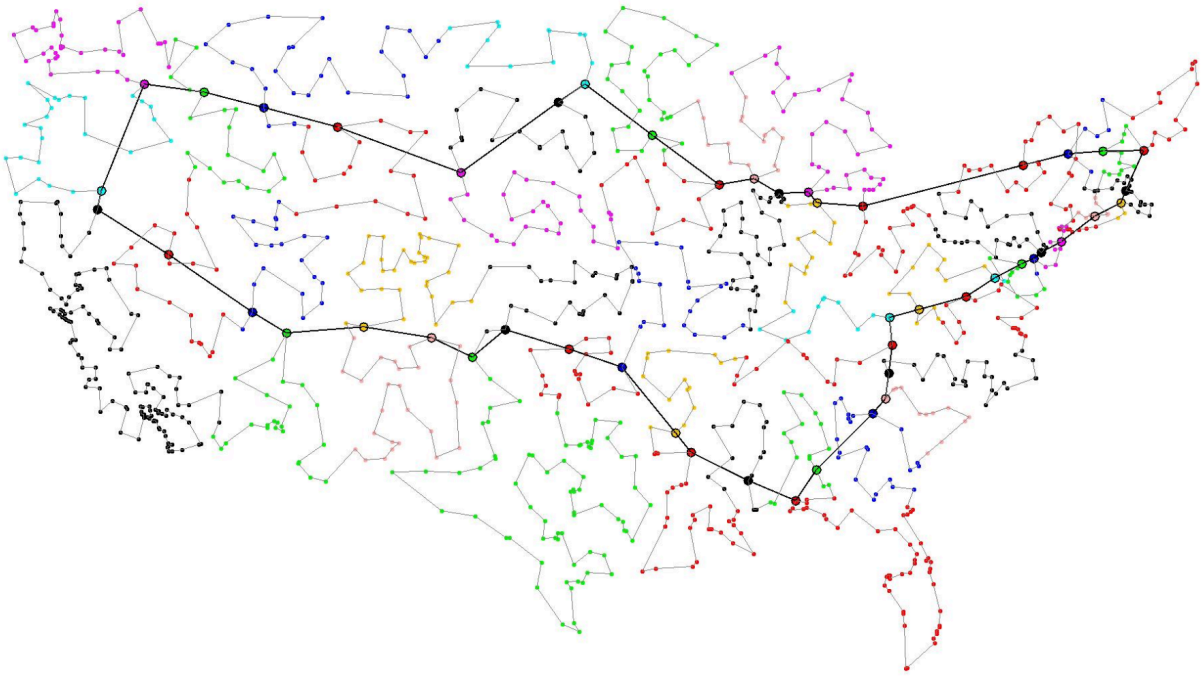

Figure 2 *Current best g-tour for 49usa1097 (length: 10,465,466 meters ≈ 6,503 miles).*

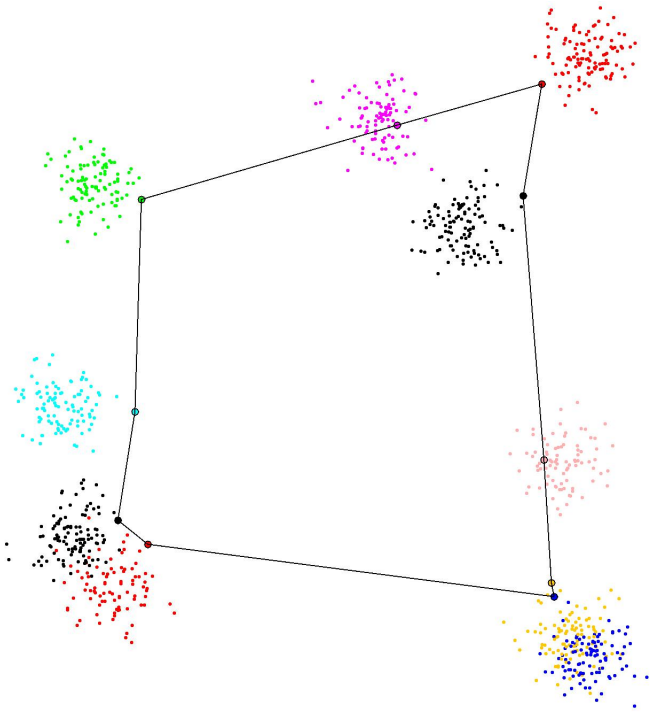

Figure 3 *Current best g-tour for 10C1k.0 (10 natural clusters).*

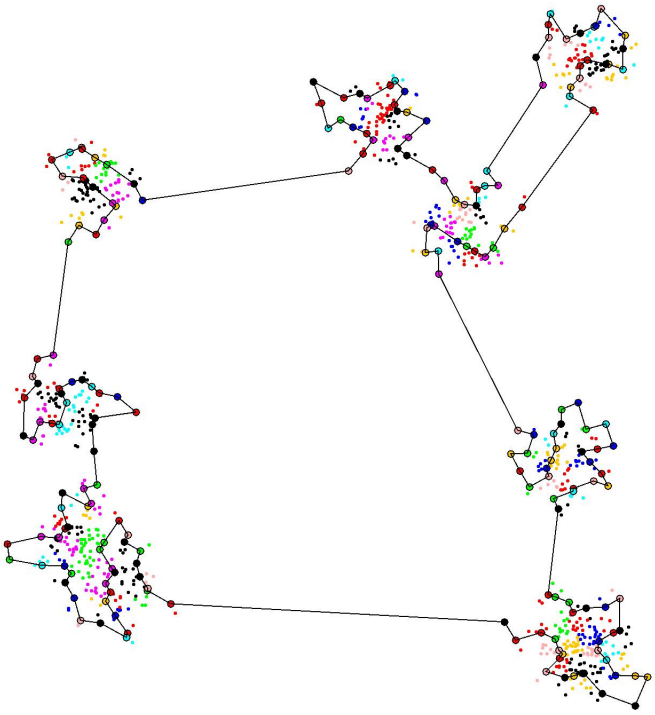

Figure 4 *Current best g-tour for 200C1k.0 (200 K-center clusters).*

The column *Best* of Table 3 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:

PROBLEM_FILE = GTSPLIB/<*instance name*>.gtsp
ASCENT_CANDIDATES = 500
INITIAL_PERIOD = 1000
INPUT_TOUR_FILE = <*input g-tour file name*>
MAX_CANDIDATES = 30
MAX_TRIALS = 1000
OPTIMUM = <*current best cost*>
OUTPUT_TOUR_FILE = <*output g-tour file name*>
PI_FILE = <π-*file name*>
POPULATION_SIZE = 1
PRECISION = 10
RUNS = 1

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 other columns of the table give the results when the parameter INPUT_ TOUR_FILE is omitted.

Name	Best	Value	Error (%)	Time (s)
10C1k.0	2522585	2522605	0.00	4.9
200C1k.0	6375154	6375154	0.00	133.7
200E1k.0	9662857	9670122	0.08	241.8
49usa1097	10465466	10465466	0.00	50.6
235pcb1173	23399	23669	1.15	367.4
259d1291	28400	28400	0.00	284.1
261rl1304	150468	150860	0.26	415.2
265rl1323	154023	154134	0.07	418.4
276nrw1379	20050	20194	0.72	398.3
280fl1400	15316	15316	0.00	119.6
287u1432	54482	54632	0.28	345.2
316fl1577	14182	14183	0.01	1294.2
331d1655	29443	29620	0.60	706.8
350vm1748	185459	185588	0.07	563.6
364u1817	25530	25667	0.54	724.0
378rl1889	184034	185246	0.66	694.0
421d2103	40049	40270	0.55	806.5
431u2152	27614	27719	0.38	815.7
464u2319	65758	66589	1.26	748.7
479pr2392	169874	171361	0.88	938.9
608pcb3038	52416	53565	2.19	1082.9
31C3k.0	3553142	3553142	0.00	482.3
633C3k.0	10255031	10255031	0.00	2833.9
633E3k.0	16197552	16484977	1.77	1218.0
759fl3795	18662	18691	0.16	3802.4
893fnl4461	63163	65060	3.00	1825.1
1183rl5915	309243	314927	1.84	3000.1
1187rl5934	295767	300618	1.64	3505.5
1480pla7397	12732870	12793563	0.48	5466.2
100C10k.0	6158999	6240251	1.32	3268.7
2000C10k.0	18044846	18284681	1.33	14027.2
2000E10k.0	28769011	29644352	3.04	5138.5
2370rl11849	427996	440652	2.96	7640.7
2702usa13509	10080705	10274251	1.92	8356.8
2811brd14051	176639	181912	2.99	8472.6
3023d15112	628259	649649	3.40	9787.7
3703d18512	234921	244558	4.10	11665.5
4996sw24978	417631	428690	2.65	20395.1
316C31k.0	10098861	10554017	4.51	13748.0
6325C31k.0	31834048	32105148	0.85	54866.3
6325E31k.0	50503475	52533090	4.02	27188.6
6762pla33810	28222961	29062848	2.97	38265.4
14202ch71009	2322839	2378232	2.38	90408.7
17180pla85900	54792193	56758297	3.59	121253.3
Average			1.38	

Table 3 *Results for the new very large benchmark instances.*

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.

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 [17] 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., Salazar González, 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. Dordrecht: Kluwer, 602–662 (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/

14. National traveling salesman problems. http://www.math.uwaterloo.ca/tsp/world/countries.html

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.: Generalized Traveling Salesman Problem Reduction Algorithms. Alg. Oper. Res., 4:144-154 (2009)

RECENT RESEARCH REPORTS

#143 Keld Helsgaun. Solving the Bottleneck Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm. 42 pp. May 2014, Roskilde University, Roskilde, Denmark.

#142 Keld Helsgaun. Solving the Clustered Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm. 13 pp. May 2014, Roskilde University, Roskilde, Denmark.

#141 Keld Helsgaun. Solving the Equality Generalized Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm. 15 pp. May 2014, Roskilde University, Roskilde, Denmark.

#140 Anders Barlach. *Effekt-drevet IT udvikling Eksperimenter med effekt-drevne systemudviklingsprojekter, der involverer CSC Scandihealth og kunder fra det danske sundhedsvæsen*. PhD thesis, Roskilde, Denmark, November 2013.

#139 Mai Lise Ajspur. *Tableau-based Decision Procedures for Epistemic and Temporal Epistemic Logics*. PhD thesis, Roskilde, Denmark, October 2013.

#138 Rasmus Rasmussen. *Electronic Whiteboards in Emergency Medicine Studies of Implementation Processes and User Interface Design Evaluations*. PhD thesis, Roskilde, Denmark, April 2013.

#137 Christian Theil Have. *Efficient Probabilistic Logic Programming for Biological Sequence Analysis*. PhD thesis, Roskilde, Denmark, January 2013.

#136 Sine Zambach. *Regulatory Relations Represented in Logics and Biomedical Texts*. PhD thesis, Roskilde, Denmark, February 2012.

#135 Ole Torp Lassen. *Compositionality in probabilistic logic modelling for biological sequence analysis*. PhD thesis, Roskilde, Denmark, November 2011.

#134 Philippe Blache, Henning Christiansen, Verónica Dahl, and Jørgen Villadsen, editors. *Proceedings of the 6th International Workshop on Constraints and Language Processing*, Roskilde, Denmark, October 2011.

#133 Jens Ulrik Hansen. *A logic toolbox for modeling knowledge and information in multi-agent systems and social epistemology*. PhD thesis, Roskilde, Denmark, September 2011.

#132 Morten Hertzum and Magnus Hansen, editors. *Proceedings of the Tenth Danish Human-Computer Interaction Research Symposium (DHRS2010)*, Roskilde, Denmark, November 2010.

#131 Tine Lassen. *Uncovering Prepositional Senses*. PhD thesis, Roskilde, Denmark, September 2010.

#130 Gourinath Banda. *Modelling and Analysis of Real Time Systems with Logic Programming and Constraints*. PhD thesis, Roskilde, Denmark, August 2010.