Solving the Clustered Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm

Keld Helsgaun

MAY 2014 ROSKILDE UNIVERSITY COMPUTER SCIENCE RESEARCH REPORT #142

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 Clustered 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 Clustered Traveling Salesman Problem (CTSP) is an extension of the Traveling Salesman Problem (TSP) where the set of cities is partitioned into clusters, and the salesman has to visit the cities of each cluster consecutively. It is well known that any instance of CTSP can be transformed into a standard 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 CTSP 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: Clustered traveling salesman problem, CTSP, Traveling salesman problem, TSP, Lin-Kernighan

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

1. Introduction

The Clustered Traveling Salesman Problem (CTSP) is an extension of the Traveling Salesman Problem (TSP) where the set of cities is partitioned into clusters, and the salesman has to visit the cities of each cluster consecutively. The CTSP coincides with the TSP whenever all clusters are singletons. The problem has numerous applications, including vehicle routing, disk defragmentation, and timetabling [1].

The CTSP 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 CTSP can be stated as the problem of finding a minimum cost cycle, where all vertices of each cluster must be visited consecutively. Furthermore, the clusters can be visited in any order.

December 2013. Updated March 19, 2014

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 *c-tour*.

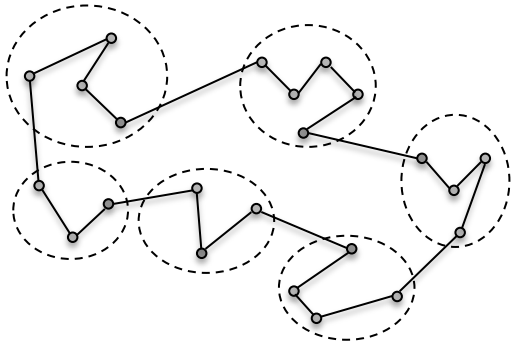

Figure 1 *Illustration of the CTSP for an instance with 6 clusters (n = 23, m =6).*

It is well known that any CTSP instance can be transformed into a TSP instance containing the same number of vertices [2]. 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) Define $c'_{ij} = c_{ij}$, when v_i and v_j belong to the same cluster.

c) When v_i and v_j belong to different clusters, define $c'_{ij} = c_{ij}+M$, where M is a sufficiently large constant. It suffices that M is larger than the largest cost.

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 exactly m inter-cluster edges. Thus, the cost of the c-tour for the CTSP is the cost of the TSP tour minus mM.

The transformation allows one to solve CTSP instances using a TSP solver. However, in the past this approach has had 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 CTSP solution, heuristic TSP solvers may have difficulties in solving this type of instances. 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 [3, 4] is a powerful local search heuristic for the TSP based on the variable depth local search of Lin and Kernighan [5]. 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/CLKH.

The following section describes how LKH can be used as a black box to solve the CTSP.

2. Implementing a CTSP Solver Based on LKH

The input to LKH is given in two files:

(1) A *problem file* in TSPLIB format [6], 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.

A CTSP solver based on LKH should therefore be able to read a CTSP instance, transform it into a TSP instance, produce the two input files required by LKH, and let LKH solve the TSP instance. The c-tour is the obtained TSP tour. A more precise algorithmic description is given below:

1. Read the CTSP instance.
2. Transform it into a 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.

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`/2, where `INT_MAX` is the maximal value that can be stored in an `int` variable. The transformation results in an 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 some 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 CTSP 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()`).

3. Computational Evaluation

The program, which is named CLKH, 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 instances generated from instances in TSPLIB [6] by applying the clustering method of Fischetti, Salazar, and Toth [7]. 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 [8] and from the national instances on the TSP web page of William Cook et al. [9].

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 = 7
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 [10]. 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 7 was chosen.

OPTIMUM: This parameter may be used to supply a best known solution cost. The algorithm will use this value as the value for *M* in the CTSP-to-TSP transformation and stop if this value is reached during the search process. If this parameter is not specified, *M* will be chosen as `INT_MAX`/*m*, where `INT_MAX` is the maximal value that can be stored in an `int` variable.

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 [7]. 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 [11]. 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.

Best: the best known solution cost. The exact solution cost (optimum) is not known for any of the instances.

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.

Best (%): 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.

Table 2 shows that all large benchmark instances are solved quite quickly too. Considering that CLKH uses LKH as a black box, without any modifications, its performance is surprisingly impressive.

Name	Best	Value	Error (%)	Best (%)	Time (s)
3burma14	3819	3819.0	0.00	100	0.0
4br17 (asym.)	39	39.0	0.00	100	0.0
4gr17	2178	2178.0	0.00	100	0.0
5gr21	2933	2933.0	0.00	100	0.0
5gr24	1289	1289.0	0.00	100	0.0
5ulysses22	7367	7367.0	0.00	100	0.0
6bayg29	1671	1671.0	0.00	100	0.0
6bays29	2056	2056.0	0.00	100	0.0
6fri26	937	937.0	0.00	100	0.0
7ftv33 (asym.)	1406	1406.0	0.00	100	0.0
8ftv35 (asym.)	1631	1631.0	0.00	100	0.0
8ftv38 (asym.)	1697	1697.0	0.00	100	0.0
9dantzig42	759	759.0	0.00	100	0.0
10att48	11516	11516.0	0.00	100	0.0
10gr48	5205	5205.0	0.00	100	0.0
10hk48	12130	12130.0	0.00	100	0.0
11berlin52	8122	8122.0	0.00	100	0.0
11eil51	446	446.0	0.00	100	0.0
12brazil58	26581	26581.0	0.00	100	0.0
14st70	733	733.0	0.00	100	0.0
16eil76	579	579.0	0.00	100	0.0
16pr76	113014	113014.0	0.00	100	0.0
20gr96	55606	55606.0	0.00	100	0.0
20rat99	1301	1301.0	0.00	100	0.0
20kroA100	21536	21536.0	0.00	100	0.0
20kroB100	22869	22869.0	0.00	100	0.0
20kroC100	21343	21343.0	0.00	100	0.0
20kroD100	22677	22677.0	0.00	100	0.0
20kroE100	23541	23541.0	0.00	100	0.0
20rd100	8418	8418.0	0.00	100	0.0
21eil101	670	670.0	0.00	100	0.0
21lin105	14545	14545.0	0.00	100	0.0
22pr107	44326	44326.0	0.00	100	0.0
24gr120	7396	7396.0	0.00	100	0.1
25pr124	60535	60535.0	0.00	100	0.1
26bier127	121798	121798.0	0.00	100	0.1
26ch130	6317	6317.0	0.00	100	0.1
28gr137	74442	74442.0	0.00	100	0.1
28pr136	104405	104405.0	0.00	100	0.1
29pr144	58813	58813.0	0.00	100	1.8
30ch150	6764	6764.0	0.00	100	0.1
30kroA150	27577	27577.0	0.00	100	0.1
30kroB150	27392	27392.0	0.00	100	0.1
31pr152	5430	5430.0	0.00	100	0.3
32u159	2557	2557.0	0.00	100	0.1
35si175	3819	3819.0	0.00	100	0.1
36brg180	39	39.0	0.00	100	0.5
39rat195	2178	2178.0	0.00	100	0.0
Average			0.00	100	

Table 1 *Results for small GTSPLIB instances.*

Name	Best	Value	Error (%)	Best (%)	Time (s)
40d198	16185	16185.0	0.00	100	0.1
40kroa200	30905	30905.0	0.00	100	0.1
40krob200	31191	31191.0	0.00	100	0.1
41gr202	41997	41997.0	0.00	100	0.1
45ts225	142704	142704.0	0.00	100	0.4
45tsp225	4189	4189.0	0.00	100	0.1
46pr226	81227	81227.0	0.00	100	0.0
46gr229	140584	140743.0	0.00	100	2.4
53gil262	2528	2528.0	0.00	100	0.1
53pr264	54537	54537.0	0.00	100	1.3
56a280	2739	2739.0	0.00	100	0.6
60pr299	50910	50910.0	0.00	100	0.7
64lin318	43490	43490.0	0.00	100	0.0
65rbg323 (asym.)	4602	4602.0	0.00	100	0.2
72rbg358 (asym.)	5249	5249.0	0.00	100	0.3
80rd400	15899	15899.0	0.00	100	0.3
81rbg403 (asym.)	6572	6572.0	0.00	100	9.3
84fl417	12242	12242.0	0.00	100	36.6
87gr431	178174	178174.0	0.00	100	0.4
88pr439	111891	111891.0	0.00	100	0.4
89pcb442	52856	52856.0	0.00	100	0.3
89rbg443 (asym.)	6252	6252.0	0.00	100	14.1
99d493	35998	35998.0	0.00	100	0.8
107ali535	208419	208419.0	0.00	100	0.8
107att532	29072	29072.0	0.00	100	0.6
107si535	48592	48592.0	0.00	100	5.7
113pa561	2988	2988.0	0.00	100	0.2
115u574	38749	38749.0	0.00	100	0.5
115rat575	7257	7257.0	0.00	100	5.7
131p654	35424	35424.0	0.00	100	14.6
132d657	50829	50828.0	0.00	100	0.6
134gr666	303984	303984.0	0.00	100	7.2
145u724	43911	43911.0	0.00	100	0.7
157rat783	9324	9324.0	0.00	100	1.7
200dsj1000	19593614	19593614.0	0.00	100	24.4
201pr1002	270613	270613.0	0.00	100	3.6
207si1032	95354	95354.0	0.00	100	70.6
212u1060	231746	231746.0	0.00	100	63.0
217vm1084	248578	248578.0	0.00	100	7.7
Average			0.00	100	

Table 2 *Results for 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 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 [8]. 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 [9]. 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 c-tour for this instance. Figure 3 and 4 show the current best c-tour for 10C1k.0 and 200C1k.0, respectively.

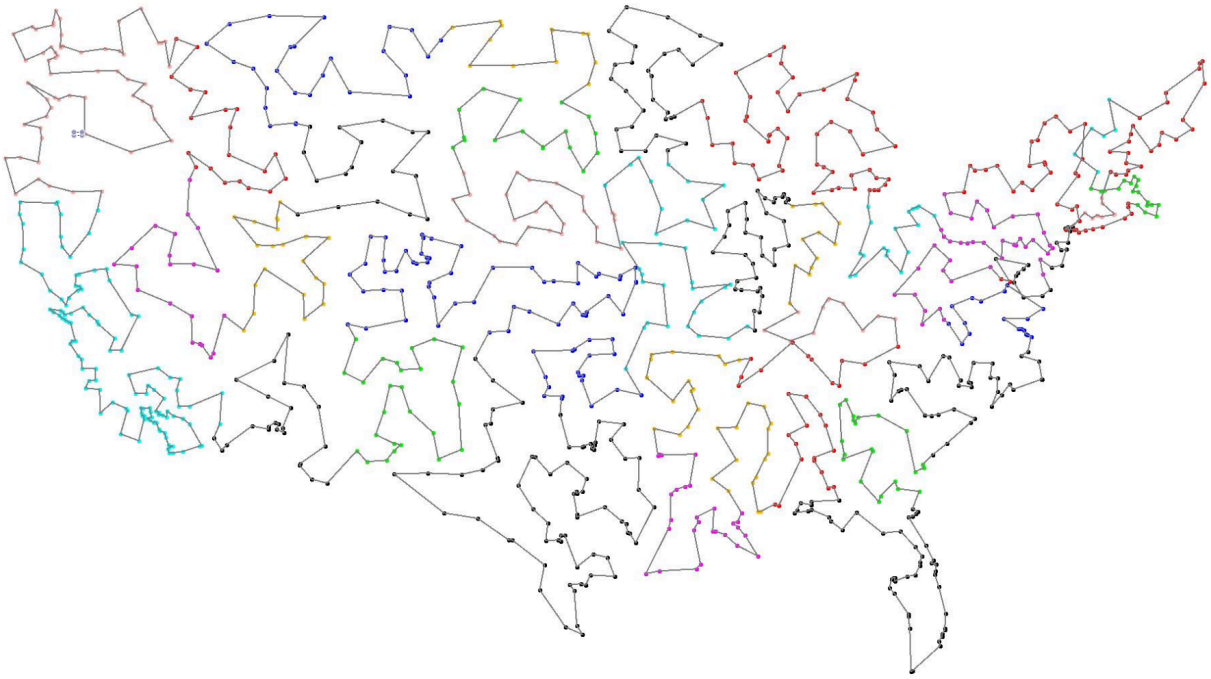

Figure 2 *Current best c-tour for 49usa1097 (length:* 77,563,429 *meters ≈ 48,196 miles).*

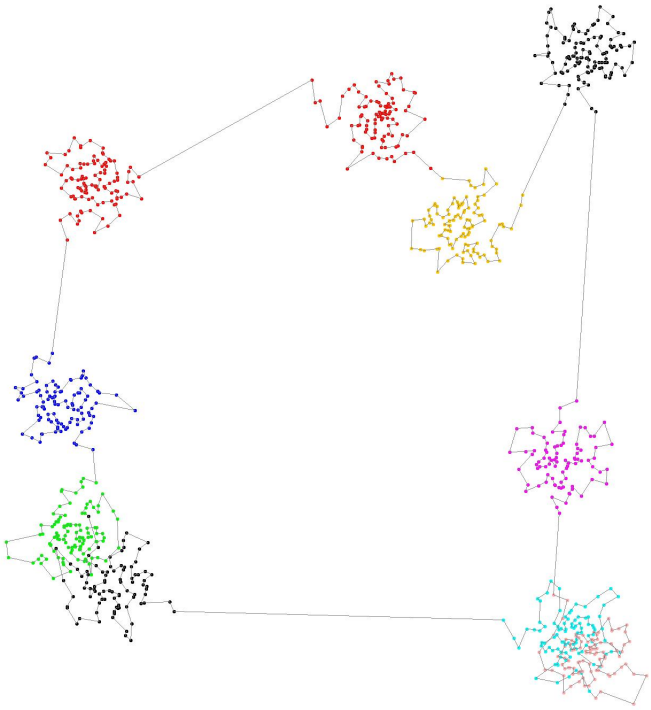

Figure 3 *Current best c-tour for 10C1k.0 (10 natural clusters).*

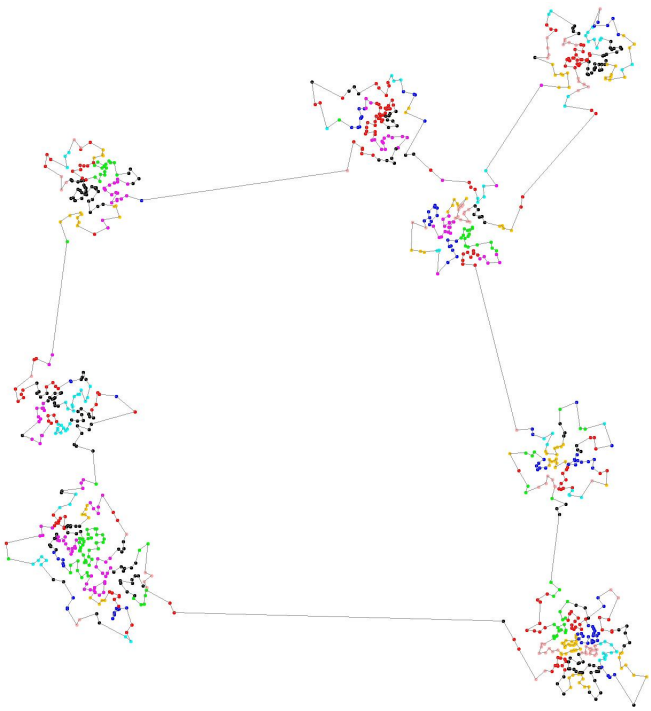

Figure 4 *Current best c-tour for 200C1k.0 (200 K-center clusters).*

The column *Best* of Table 3 shows the current best solution costs found by CLKH. These costs were found using several runs of CLKH where in each run the current best c-tour was used as input tour to CLKH and using the following non-default parameter settings:

PROBLEM_FILE = GTSPLIB/<*instance name*>.gtsp
ASCENT_CANDIDATES = 500
INITIAL_PERIOD = 1000
INPUT_TOUR_FILE = <*input c-tour file name*>
MAX_CANDIDATES = 7
MAX_TRIALS = 1000
OPTIMUM = <*current best cost*>
OUTPUT_TOUR_FILE = <*output c-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 c-tour from two or more given c-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	12139627	12139627	0.00	22.5
200C1k.0	11929315	11929315	0.00	9.6
200E1k.0	24468822	24468822	0.00	12.2
49usa1097	77583052	77583052	0.00	33.5
235pcb1173	59796	59796	0.00	13.5
259d1291	55978	55978	0.00	27.0
261rl1304	261132	261132	0.00	16.4
265rl1323	280004	280188	0.07	30.7
276nrw1379	60473	60485	0.02	37.1
280fl1400	20229	20255	0.13	292.4
287u1432	162151	162151	0.00	24.5
316fl1577	23023	23023	0.00	54.9
331d1655	65871	65946	0.11	64.4
350vm1748	348244	348244	0.00	29.1
364u1817	61879	61916	0.06	60.4
378rl1889	323040	323663	0.19	41.5
421d2103	91637	91748	0.12	104.5
431u2152	69876	69876	0.00	42.3
464u2319	246707	246707	0.00	28.3
479pr2392	397707	397867	0.04	90.9
608pcb3038	146351	146362	0.01	168.9
31C3k.0	20058457	20160054	0.49	413.7
633C3k.0	20160074	20162594	0.01	169.7
633E3k.0	42697510	42699039	0.00	109.5
759fl3795	29582	29582	0.00	129.4
893fnl4461	193834	193834	0.00	80.4
1183rl5915	599142	599223	0.00	233.0
1187rl5934	588104	589496	0.24	229.3
1480pla7397	23926551	23957171	0.13	1008.5
100C10k.0	36350972	37014518	1.83	1200.0
2000C10k.0	34571660	34685387	0.33	776.8
2000E10k.0	75508805	75515101	0.01	629.5
2370rl11849	977547	977628	0.01	694.9
2702usa13509	20836277	20842290	0.03	886.7
2811brd14051	496893	496984	0.02	963.6
3023d15112	1658277	1658278	0.02	1180.1
3703d18512	683914	684099	0.03	1465.0
4996sw24978	893094	893456	0.04	1840.7
316C31k.0	63148541	63236932	0.14	4008.6
6325C31k.0	62618284	62831680	0.34	2475.2
6325E31k.0	133745969	133786294	0.03	2956.9
6762pla33810	69318480	69337616	0.03	3784.1
14202ch71009	4779103	4780524	0.03	2350.1
17180pla85900	149096064	149176474	0.05	12532.9
Average			0.10	

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

4. Conclusion

This paper has evaluated the performance of LKH on CTSP instances that are transformed into standard TSP instances. 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 GTSP benchmark instances, GTSPLIB, could be solved quickly, and it was possible to find high-quality solutions for a series of new large-scale CTSP instances with up to 17,180 clusters and 85,900 vertices.

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 c-tours for these instances via http://www.ruc.dk/~keld/research/CLKH.

References

1. Laporte G., Palekar U.: Some applications of the clustered travelling salesman problem. J. Oper. Res. Soc., 53(9):972-976 (2002)

2. Chrisman, J. A.: The clustered traveling salesman problem. Comput. Oper. Res., 2(2):115–119 (1975)

3. Helsgaun, K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. Eur. J. Oper. Res., 126(1):106-130 (2000)

4. Helsgaun, K.: General k-opt submoves for the Lin-Kernighan TSP heuristic. Math. Prog. Comput., 1(2-3):119-163 (2009)

5. Lin, S, Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Oper. Res., 21(2):498-516 (1973)

6. Reinelt, G.: TSPLIB - a traveling salesman problem library. ORSA J. Comput., 3(4):376-384 (1991)

7. 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)

8. 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/

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

10. Held, M, Karp, R.M.: The traveling salesman problem and minimum spanning trees. Oper. Res., 18(6):1138-1162 (1970)

11. Gutin, G., Karapetyan, D.: A memetic algorithm for the generalized traveling salesman problem. Nat. Comput., 9(1):47-60 (2010)

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.