LKH
User Guide

Version 1.3 (July 2002)

by
Keld Helsgaun
E-mail: keld@ruc.dk

1. Introduction

The Lin-Kernighan heuristic [1] is generally considered to be one of the most successful methods for generating optimal or near-optimal solutions for the symmetric traveling salesman problem.

LKH is an implementation of a modified Lin-Kernighan heuristic. The new heuristic differs in many details from the original one. The most notable difference is found in the search strategy. The new heuristic uses larger (and more complex) search steps than the original one. Also new is the use of sensitivity analysis to direct and restrict the search. For a detailed description of the modified heuristic, see [2][3].

Computational experiments have shown that LKH is highly effective. Even though the algorithm is approximate, optimal solutions are produced with an impressively high frequency. LKH has produced optimal solutions for all solved problems we have been able to obtain, including a 15112-city problem (at the time of writing, the largest nontrivial problem solved to optimality). Furthermore, the algorithm has improved the best known solutions for a series of large-scale problems with unknown optima, among these an 85900-city problem.

2. Installation

LKH has been implemented in the programming language C. The software, approximately 4000 lines of code, is entirely written in ANSI C and portable across a number of computer platforms and C compilers.

The code is distributed for research use. The author reserves all rights to the code.

The code is available at the internet at:

http://homer.ruc.dk/~keld/public_html/research/LKH

The software is available in two formats:

LKH-1.3.tgz	(gzipped tar file, 5.9 MB)
LKH-1.3.sit	(Stuffit archive, 4.6 MB)

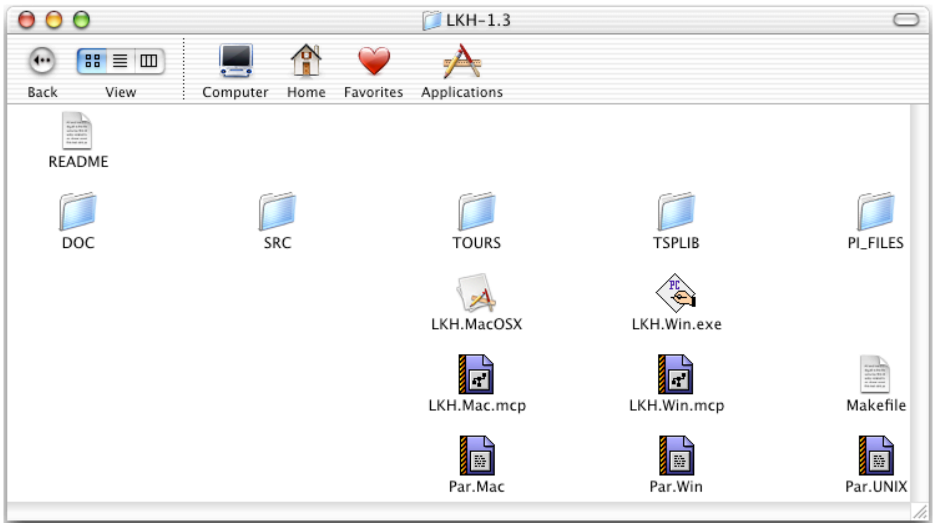
If a UNIX machine is used, download the software in the first format. Next execute the following UNIX commands:

```
gzip -d LKH-1.3.tgz
tar xvf LKH-1.3.tar
cd LKH-1.3
make
```

An executable file called LKH.UNIX is now available in the directory LKH-1.3.

If a MacOSX or a Windows machine is used, download the software in the second format. Next unstuff it with StuffIt Expander™ (freeware available at http://www.aladdinsys.com).

On a MacOSX machine the following files and folders can be found in the folder LKH-1.3. Similar files and directories can be found on a Windows machine and on a UNIX machine.



The README file contains instructions for installing the software.

The DOC folder contains the following documentation:
(1) LKH_GUIDE.pdf, this user guide, (2) LKH_REPORT.pdf, a report that describes the implementation and performance of the sofware, (3) ADDENDUM.pdf, an addendum to the report, and (4) TSPLIB_DOC.PS, a description of the publicly available library of problem instances TSPLIB.

The SRC folder contains source code.

The TSPLIB folder contains the symmetric, asymmetric and Hamiltonian tour problems of TSPLIB.

The TOURS folder contains optimum and best known tours for these problems.

The PI_FILES folder contains penalties produced by the ascent of the algorithm (see the report in the DOC folder).

The files `LKH.Mac.mcp` and `LKH.Win.mpc` are Metrowerks projects. They can be used to recompile the source files on a MacOSX machine and a Windows machine, respectively.

The file `Makefile` is used by the command `make` on a UNIX machine.

The files `LKH.MacOSX` and `LKH.Win.exe` are used to run the LKH program on a MacOSX machine and a Windows machine, respectively.

The files `ParMac`, `ParWin` and `ParUNIX` are files that can be used to test the installation of the software.

To verify that the software has been installed correctly, run the LKH program (`LKH.UNIX` on a UNIX machine, `LKH.MacOSX` on a MacOSX machine, and `LKH.Win` on a Windows machine).

The program asks for a filename

```
PARAMETER_FILE =
```

On a MacOSX machine type `Par.Mac`. On a Windows machine type `Par.Win`. On a UNIX machine type `Par.UNIX`. Then press return.

The program should now solve a problem with 532 cities (the problem `att532.tsp` in TSPLIB). The output should be similar to the following (produced by a 550 MHz PowerBook G4):

```
PARAMETER_FILE = Par.Mac

PARAMETER_FILE = Par.Mac
ASCENT_CANDIDATES = 50
BACKTRACK_MOVE_TYPE = 0
CANDIDATE_FILE =
EXCESS = 0.001880
INITIAL_PERIOD = 266
INITIAL_STEP_SIZE = 1
INITIAL_TOUR_FILE =
INPUT_TOUR_FILE =
MAX_CANDIDATES = 5
MAX_SWAPS = 532
MAX_TRIALS = 532
MERGE_TOUR_FILE_1 =
MERGE_TOUR_FILE_2 =
MOVE_TYPE = 5
OPTIMUM = 27686
PI_FILE = :PI_FILES:att532.pi
PRECISION = 100
PROBLEM_FILE = :TSPLIB:att532.tsp
RESTRICTED_SEARCH = YES
RUNS = 10
SEED = 1
SUBGRADIENT = YES
TOUR_FILE =
TRACE_LEVEL = 1

Lower bound = 27415.7, Gap = 1.0%, Ascent time = 0 sec.
Preprocessing time = 5 sec.
* 1: Cost = 27706, Time = 0 sec.
* 11: Cost = 27693, Time = 1 sec.
* 35: Cost = 27686, Time = 2 sec.
Cost = 27686, Seed = 1, Time = 2 sec.
* 1: Cost = 27857, Time = 1 sec.
* 2: Cost = 27705, Time = 1 sec.
* 7: Cost = 27693, Time = 2 sec.
* 18: Cost = 27686, Time = 3 sec.
Cost = 27686, Seed = 2, Time = 3 sec.
* 1: Cost = 27706, Time = 0 sec.
* 22: Cost = 27693, Time = 2 sec.
* 25: Cost = 27686, Time = 2 sec.
Cost = 27686, Seed = 3, Time = 2 sec.
* 1: Cost = 27686, Time = 0 sec.
Cost = 27686, Seed = 4, Time = 0 sec.
* 1: Cost = 27705, Time = 1 sec.
* 10: Cost = 27693, Time = 1 sec.
* 19: Cost = 27686, Time = 2 sec.
Cost = 27686, Seed = 5, Time = 2 sec.
* 1: Cost = 27712, Time = 0 sec.
* 9: Cost = 27705, Time = 1 sec.
* 95: Cost = 27703, Time = 4 sec.
* 109: Cost = 27693, Time = 4 sec.
* 149: Cost = 27686, Time = 7 sec.
Cost = 27686, Seed = 6, Time = 7 sec.
* 1: Cost = 27693, Time = 0 sec.
* 21: Cost = 27686, Time = 2 sec.
Cost = 27686, Seed = 7, Time = 2 sec.
* 1: Cost = 27870, Time = 1 sec.
* 6: Cost = 27837, Time = 1 sec.
* 35: Cost = 27725, Time = 2 sec.
* 38: Cost = 27712, Time = 2 sec.
* 44: Cost = 27707, Time = 3 sec.
* 48: Cost = 27693, Time = 3 sec.
* 62: Cost = 27686, Time = 4 sec.
Cost = 27686, Seed = 8, Time = 4 sec.
* 1: Cost = 27718, Time = 1 sec.
* 2: Cost = 27714, Time = 1 sec.
* 6: Cost = 27704, Time = 1 sec.
* 42: Cost = 27703, Time = 3 sec.
* 91: Cost = 27686, Time = 4 sec.
Cost = 27686, Seed = 9, Time = 5 sec.
* 1: Cost = 27693, Time = 0 sec.
* 27: Cost = 27686, Time = 3 sec.
Cost = 27686, Seed = 10, Time = 3 sec.

Successes/Runs = 10/10
Cost.min = 27686, Cost.avg = 27686.0, Cost.max = 27686
Gap.min = 0.000%, Gap.avg = 0.000%, Gap.max = 0.000%
MinTrials = 1, Trials.avg. = 44.8
Time.min = 0 sec., Time.avg. = 3.0 sec.
```

3. User interface

The software includes code both for reading problem instances and for printing solutions.

Input is given in two separate files:

(1) the *problem file* and
(2) the *parameter file*.

The *problem file* contains a specification of the problem instance to be solved. The file format is the same as used in TSPLIB [4], a publicly available library of sample instances for the TSP The library may be downloaded from

http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB

The current version of the software allows specification of symmetric, asymmetric, as well as Hamiltonian tour problems.

Distances (costs, weights) may be given either explicitly in matrix form (in a full or triangular matrix), or implicitly by associating a 2- or 3-dimensional coordinate with each node. In the latter case distances may be computed by either a Euclidean, Manhattan, maximum, geographical or pseudo-Euclidean distance function. See the `TSPLIB_DOC.PS` file for details. At present, all distances must be integral.

Problems may be specified on a complete or sparse graph, and there is an option to require that certain edges must appear in the solution of the problem.

The *parameter file* contains control parameters for the solution process. The solution process is typically carried out using default values for the parameters. The default values have proven to be adequate in many applications. Actually, almost all computational tests reported in this paper have been made using these default settings. The only information that cannot be left out is the name of the problem file.

The format is as follows:

PROBLEM_FILE = <*string*>
Specifies the name of the problem file.

Additional control information may be supplied in the following format:

ASCENT_CANDIDATES = <*integer*>
The number of candidate edges to be associated with each node during the ascent.
The candidate set is complemented such that every candidate edge is associated
with both its two end nodes.
Default: 50.

BACKTRACK_MOVE_TYPE = <*integer*>
Specifies the backtrack move type to be used in local search.
A backtrack move allows for backtracking up to a certain level of the local search.
A value of 2, 3, 4 or 5 signifies that a backtrack 2-opt, 3-opt, 4-opt or 5-opt move is to be used as the first move in the search. The value 0 signifies that no backtracking is to be used.
Default: 0.

CANDIDATE_FILE = <*string*>
Specifies the name of a file to which the candidate sets are to be written.
If the file already exists, and the PI_FILE exists, the candidate edges are read from the file.
Each line contains a node number, the number of the dad of the node in the minimum spanning tree (0, if the node has no dad), the number of candidate edges emanating from the node, followed by these candidate edges. For each candidate edge its end node number and □-value are given.

COMMENT : <*string*>
A comment.

EOF
Terminates the input data. The entry is optional.

EXCESS = <*real*>
The maximum □-value allowed for any candidate edge is set to EXCESS times the absolute value of the lower bound of a solution tour (determined by the ascent).
Default: 1.0/DIMENSION.

INITIAL_PERIOD = <*integer*>
The length of the first period in the ascent.
Default: DIMENSION/2 (but at least 100).

INITIAL_STEP_SIZE = <*integer*>
The initial step size used in the ascent.
Default: 1.

INITIAL_TOUR_FILE = <string>
Specifies the name of a file containing a tour to be used as the initial tour in the search.
The tour is given by a list of integers giving the sequence in which the nodes are visited in the tour. The tour is terminated by a -1.

INPUT_TOUR_FILE = <*string*>
Specifies the name of a file containing a tour. The tour is given by a list of integers giving the sequence in which the nodes are visited in the tour. The tour is terminated by a -1. The tour is used to limit the search (the last edge to be removed in a non-gainful move must not belong to the tour). In addition, the Alpha field of its edges is set to zero.

MAX_CANDIDATES = <*integer*> { SYMMETRIC }
The maximum number of candidate edges to be associated with each node.
The integer may be followed by the keyword SYMMETRIC, signifying that the candidate set is to be complemented such that every candidate edge is associated with both its two end nodes.
Default: 5.

MAX_SWAPS = <*integer*>
Specifies the maximum number of swaps (flips) in any search for a tour improvement.
Default: DIMENSION.

MAX_TRIALS = <*integer*>
The maximum number of trials in each run.
Default: number of nodes (DIMENSION, given in the problem file).

MERGE_TOUR_FILE_1 = <*string*>
Specifies the name of a tour to be merged. The edges of the tour are added to the candidate sets with □-values equal to 0.

MERGE_TOUR_FILE_2 = <*string*>
Specifies the name of a tour to be merged. The edges of the tour are added to the candidate sets with □--values equal to 0.

MOVE_TYPE = <*integer*>
Specifies the move type to be used in local search. The value can be 2, 3, 4 or 5 and signifies whether 2-opt, 3-opt, 4-opt or 5-opt move is to be used.
Default: 5.

OPTIMUM = <*real*>
Known optimal tour length. A run will be terminated as soon as a tour length less than or equal to optimum is achieved.
Default: -DBL_MAX.

PI_FILE = <*string*>
Specifies the name of a file to which penalties (π-values determined by the ascent) are to be written. If the file already exits, the penalties are read from the file, and the ascent is skipped. Each line of the file is of the form <integer> <integer>, where the first integer is a node number, and the second integer is the π-value associated with the node.

PRECISION = <*integer*>
The internal precision in the representation of transformed distances:
d_{ij} = PRECISION$*c_{ij} + \pi_i + \pi_j$, where d_{ij}, c_{ij}, π_i and π_j are all integral.
Default: 100 (which corresponds to 2 decimal places).

RESTRICTED_SEARCH: [YES | NO]
Specifies whether the following search pruning technique is to be used:
The first edge to be broken in a move must not belong to the currently best solution tour.
When no solution tour is known, then it must not belong to the minimum spanning 1-tree.
Default: YES.

RUNS = <*integer*>
The total number of runs.
Default: 10.

SEED = <*integer*>
Specifies the initial seed for random number generation.
Default: 1.

SUBGRADIENT: [YES | NO]
Specifies whether the π-values should be determined by subgradient optimization.
Default: YES.

TOUR_FILE = <*string*>
Specifies the name of a file to which the best tour is to be written.

TRACE_LEVEL = <*integer*>
Specifies the level of detail of the output given during the solution process.
The value 0 signifies a minimum amount of output. The higher the value is the more information is given.
Default: 1.

During the solution process information about the progress being made is written to standard output. The user may control the level of detail of this information (by the value of the TRACE_LEVEL parameter).

Before the program terminates, a summary of key statistics is written to standard output, and, if specified by the TOUR_FILE parameter, the best tour found is written to a file (in TSPLIB format).

The user interface is somewhat primitive, but it is convenient for many applications. It is simple and requires no programming in C by the user. However, the current implementation is modular, and an alternative user interface may be implemented by rewriting a few modules. A new user interface might, for example, enable graphical animation of the solution process.

References

[1] S. Lin & B. W. Kernighan,
"An Effective Heuristic Algorithm for the Traveling-Salesman Problem",
Oper. Res. **21**, 498-516 (1973).

[2] K. Helsgaun,
"An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic",
Datalogiske Skrifter **81**, Roskilde University (1998).

[3] K. Helsgaun,
"An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic",
European Journal of Operational Research **126** (1), 106-130 (2000).

[4] G. Reinelt,
"TSPLIB - A Traveling Salesman Problem Library",
ORSA J. Comput. **3-4**, 376-385 (1991).