KoptLS User Manual

July 5, 2024

1 Installation

The suite is comprised of the following `.c` and `.h` files

- `proc4opt.c`, `proc4opt.h`
- `proc3opt.c`, `proc3opt.h`
- `proc2opt.c`
- `stopwatch.c`, `stopwatch.h`
- `io4opt.c`, `io4opt.h`
- `heap.c`, `heap.h`
- `woeginger.c`
- `cmd_convergenceKopt.c`
- `tspmain.h`

The command

```
make KoptLS
```

compiles the program and creates an exectuable with name `KoptLS` in the current directory.

By default, the program has a cost matrix `cost[i][j]` stored in memory. Optionally, for large instances (e.g., n way above 20,000), when there might be not enough memory to store an $n \times n$ matrix, one can edit the `makefile` and enable the compiler conditional compilation

```
-DCOSTS_ONFLY
```

When `COSTS_ONFLY` is defined, the cost of each edge ij is computed whenever it is needed (e.g., in Euclidean instances, as the distance between two points in the square).

In addition to the main suite, there is also a utility file

```
convertTSPLIB.c
```

that can be used to convert a file in the `TSP` format of TSPLIB into the format that our program requires. To compile, give the command

```
gcc convertTSPLIB.c -lm -o convert
```

To run, give the command

```
./convert inputf outputf
```

where `inputf` is the name of a TSPLIB file in format `TSP` (usually, with extension `.tsp`) and `outputf` is the name of the file on which the instance will be written, converted into our format.

2 Introduction

`KoptLS` is a local search algorithm for the TSP, based on the k-OPT neighborhood. It implements 2-OPT, 3-OPT and 4-OPT moves, either as Brute Force (complete enumeration) or some Smart Force variants ([1,2,3]). The local search can be run from a random tour or from an input tour on a file. One can specify how many steps of LS to perform, or if LS should be continued until a local optimum is found. Furthermore, one can specify how many (random) starts should be considered for the given instance. The intermediate tours generated during a search can be saved on separate files, or just the final locally optimal tour can be saved on file.

For each k-OPT neighborhood there are several versions of search algorithms implemented, from Brute Force (BF) to Smart Force (SF) declined in variants described in the papers where we introduced them.

3 Usage

`KoptLS` [instance] [search params] [other params] [stat/results]

The instance specification is mandatory, while most of other parameters are optional. At least one k-OPT strategy must be specified. The order of parameters is irrelevant.

Instance specification

The instance is specified through one of the following:

`-RU <n>` : creates a random uniform input on n nodes.

Default: `<n> = 100`

`-RE <n>` : creates a random euclidean input on n nodes.

Default: `<n> = 100`

`-F <filename>` : reads input graph from file.

Default: none

Search parameters

`-A2 <alg2>` : algorithm to use for 2-OPT (see [3]).

0: BF

1: basic SF (best pivot+ all completions)

2: same as 1: but w/o duplicates

3: SF ppairs lexi

4: SF ppairs heap

5: H(delta) heap ppairs

Default: `<alg2> = -1` (no 2-OPT)

`-A3 <alg3>` : algorithm to use for 3-OPT (see [1]).

0: BF

1: basic SF

Default: `<alg3> = -1` (no 3-OPT)

`-A4 <alg4>` : algorithm to use for 4-OPT (see [2]).

0: BF

1: SF w/o split

2: SF w/split

3: Woeginger D.P.

4: Glover D.P.

5: basic SF similar to 3OPT (w/4 heaps)

Default: `<alg4> = -1` (no 4-OPT)

`-O4 <orbits>` : determines the 4opt orbits in the form of a string over 1..7. E.g., "2" (only orbit 2) "135" (orbits 1,3 and 5) "1234567" (all orbits).

Default: `<orbits> = 1234567`

`-LOOP <k-ord>` : determines in which order to run the k opts. `<k-ord>` is a permutation of {2,3,4}.

Default: `<k-ord> = 234`

`-SWITCH2COEF <val>` : after `<val>`$\times n$ steps of LS, switches 2-OPT search (if any) to CE

Default: `<val> = INFINITE`

Other parameters

`-SEEDT <seed>` : fixes seed for random number generator for tours and permutations.

Default: `<seed> = 1234`

`-SEEDC <seed>` : fixes seed for random number generator for edge costs.

Default: `<seed> = 1234`

`-NINS <numinst>` : creates more instances of the same type (valid only with `-RU` and `-RE`).

Default: `<numinst> = 1`

`-NS <numsteps>` : how many steps of local search for each convergence.

Default: `<numsteps> = INFINITE` (run up to local opt).

`-NT <numtests>` : how many convergences of the same type on each instance.

Default: `<numsteps> = 1`

`-ST <start tour|NN>` : file name of tour to start from (if "NN" start from nearest neighbor tour).

Default: none (start from a random tour).

`-CMIN <lbval>` : sets lower bound to random edge costs in uniform instances, i.e., uar in `[CMIN,CMAX)`.

Default: `<lbval> = 0.0`

`-CMAX <ubval>` : sets upper bound to random edge costs in uniform instances, i.e., uar in `[CMIN,CMAX)`.

Default: `<ubval> = 1.0`

`-PREC <pval>` : sets the decimal precision for costs to pval digits.

Default: `<pval> = 6` digits.

`-TLIM <tsec>` : sets time limit for overall procedure, over multiple runs, in secs. (** NOTE: not very precise **)

Default: `<tsec> = INFINITE` (no time limit).

`-DOFINDBEST <impr>` : choose between best-improvement and first-improvement (1=yes, 0=no).

Default: `<impr> = 1`

`-DOSHOWMOVE <show>` : show on screen the moves found at each LS step (1=yes, 0=no).

Default: `<show> = 0`

Statistics and results

-WT <root_fname> : writes intermediate tours on file.
Default: <root_fname> = "" (do not write intermediate tours).

-WTL <root_fname> : writes last tour of each convergence on file.
Default: <root_fname> = "" (do not write last tours).

-WOPT <fname> : writes best tour found overall on file.
Default: <fname> = "" (do not write best tour).

-LTF <lapstep> : takes cumulative time/stats also for the steps lapstep, lapstep+1, ...
Default: <lapstep> = 0 (do not use).

-Wtime <fname> : writes log of time for each move on file.
Default: <fname> = "" (do not write log of moves time).

-Wmove <fname> : writes log of # evaluations for each move on file.
Default: <fname> = "" (do not write log of moves evaluations).

-Wval <fname> : writes log of tours value time on file.
Default: <fname> = "" (do not write log of tours value).

-TSPOUT <fname> : writes the read or generated graph in TSP format on file.
Default: <fname> = "" (do not write the instance on a file).

4 Examples

Full local search

- `KoptLS -RU 200 -A2 1 -A3 1 -LOOP 423`

  Runs a full-local search on a uniform random instance of 200 nodes, using only 2-OPT and 3-OPT, smart force, in this order

- `KoptLS -RE 300 -A4 2 -DOFINDBEST 0`

  Runs a full-local search on a random euclidean instance of 300 nodes, using only 4-OPT, smart force, with first-improvement strategy

Only one move

- `KoptLS -RU 200 -A2 0 -A3 1 -LOOP 423 -NS 1 -NT 10`

  Creates a uniform random instance of 200 nodes, and finds the best move, on 10 random starting tours using 2-OPT, complete enumeration and 3-OPT, smart force, in this order

- `KoptLS -F a.txt -A4 2 -NS 1 -ST NN`

  Given the input instance `a.txt`, computes a nearest neighbor starting tour and finds on it the best 4-OPT move, using smart force

5 Files format

As usual in `c`, all indices start at `0`.

Input instance. An instance is stored on a text file with the following format:

```
n
i_1 j_1 c(i_1 j_1)
...
i_m j_m c(i_m j_m)
```

The first value `n` is the number of nodes. Then, a list of $m \leq n(n-1)/2$ edges follows. The graph will be complete even if $m < n(n-1)/2$, and in this case the missing edges have value $+\infty$. For each edge, the file contains the endpoints i_k, j_k and the edge cost $c(i_k, j_k)$. For the indices, it must be

$$0 \leq i_k < j_k \leq n-1$$

Input/output tour. In output, a tour is written on a text file, with first the number of nodes, and then their permutation, of $n+1$ entries starting and ending with node 0. E.g. the tour on 5 nodes given by permutation $(3, 2, 4, 0, 1)$ will be stored as

```
5
0 1 3 2 4 0
```

When the tour is read (input) from a file, the format is the same but the last `0` can be missing with no harm (i.e., only n nodes are read, and they must start with `0`).

6 References

[1] G. Lancia and M. Dalpasso, Finding the Best 3-OPT Move in Subcubic Time, Algorithms 2020, 13(11)

[2] G. Lancia and M. Dalpasso, Algorithmic strategies for a fast exploration of the TSP 4-OPT neighborhood, Journal of Heuristics 2023, Open access

[3] G. Lancia and P. Vidoni, Average case sub-quadratic exact and heuristic procedures for the traveling salesman 2-OPT neighborhood, INFORMS J. On Computing 2024, (to appear)