Cost-Balanced Traveling Salesperson Problem

Heuristic Optimization Techniques, 2018WS

October 11, 2018

1 Problem description

We consider the *Cost-Balanced Traveling Salesperson Problem*[†] (CBTSP), an adaption of the classical TSP. Given an undirected, simple, connected graph $G = (V, E)$ with edge costs $d_{i,j} \in \mathbb{Z}$, $\forall (i, j) \in E$, find a Hamiltonian cycle $C \subset E$ with cost closest to 0, i.e., it minimizes

$$\left| \sum_{(i,j)\in C} d_{i,j} \right| .$$

The main difference to the original TSP is that edge costs can also be negative and therefore also the cost of a Hamiltonian cycle.

2 Handling of infeasible solutions

Since the graph is not necessarily complete, it will be difficult for some metaheuristics to always find a feasible solution. To handle those difficulties a well known method is penalization. Infeasible solutions are considered feasible during the execution of the algorithm but have an objective value greater than any feasible solution. To achieve this in our problem, we can make our graph complete by introducing edges with large constant costs. Such kind of constant is called big-M. In our case this means that non-existing edges can be used in the solutions, but have costs $M \in \mathbb{N}$.

3 Instances & Solution format

As usual, $n := |V|, m := |E|$. The instances are plain ASCII files of the following format:

```
<n> <m>
<vertex_1_of_edge_1> <vertex_2_of_edge_1> <cost_of_edge_1>
...
<vertex_1_of_edge_m> <vertex_2_of_edge_m> <cost_of_edge_m>
```

A simple example file looks like:

```
5  8
0  1  2
0  2  1
0  3  3
1  2 -2
1  3 -1
2  3  1
2  4  1
3  4 -2
```

[†]problem formulation and instances from a competition at the MESS2018

Vertices are identified by their indices starting with 0: $V = \{0, \ldots, n-1\}$. A solution is described by a permutation of V, a single line with whitespace between the vertex ids, for example:

```
0 3 4 2 1
```

Since the graph is not necessarily complete, only the subset of permutations corresponds to valid solutions where there is an edge between every subsequent vertex and from the last vertex to the first.

4 Reports

For each programming exercise we require you to hand in a short report (about 4-6 pages) via TUWEL containing (if not otherwise specified) at least:

- A description of the implemented algorithms (the adaptions, selected options, etc.—not the general procedure)
- Experimental setup (Machine, tested algorithm configurations)
- Best objective values and runtimes (+ Mean/Std. dev. for randomized algorithms over multiple runs) for all published instances and algorithms. Infeasible solutions must be excluded from this calculations. Furthermore list the number of runs which returned feasible/infeasible solutions for all published instances and algorithms.
- Do not use excessive runtimes for your algorithms, limit the maximum cpu time (not wall clock time!) to 15 minutes per instance.

What we don't want:

- Multithreading – use only single-threading
- UML diagrams (or any other form of source code description)
- Repetition of the problem description

5 Solution submission

We require you to hand in your best solutions for each instance and each algorithm in TUWEL before the deadline. Make sure that the reported best solution and the uploaded solution match. The upload can be repeated multiple times—only the best solution is shown. If one of your algorithms yields infeasible solutions only, still upload the one with the least missing edges.

The uploaded solutions are then checked for correctness and entered in the ranking table. The ranking tables shows information about group rankings (best three groups per instance & algorithm) and solution values to give you an estimate of your algorithms performance. Your ranking does not influence your grade!

6 AC cluster

To provide a standardized experiment environment it is possible to use the AC computing cluster. Login using ssh on `USERNAME@eowyn.ac.tuwien.ac.at` or `USERNAME@behemoth.ac.tuwien.ac.at`. Both machines run `Ubuntu 16.04 LTS` and provide you with a `gcc 5.4.1` toolchain and `Java 1.8`. External libraries required by your program should be installed in your home directory and the environment variables should be set accordingly.

Before submitting a command to the computing cluster create an executable e.g. a bash script setting up your environment and invoking your program. It is possible to supply additional command line arguments to your program. To submit a command to the cluster use:

```
 qsub -l h_rt=00:15:00 [QSUB_ARGS] COMMAND [CMD_ARGS]
```

The `qsub` commmand is a command for the Sun Grid Engine and the command above will submit your script with a maximum runtime of 15 minutes (hard) to the correct cluster nodes. Information about your running/pending jobs can be queried via `qstat`. Sometimes you might want to delete (possible) wrongly submitted jobs. This can be easily done by typing `qdel <job_id>`. You can find additional information under:
`https://www.ac.tuwien.ac.at/students/grid-engine/`