TU Wien
Institute for Logic and Computation
Algorithms and Complexity Group

Heuristic Optimization Techniques, WS 2024

Programming Exercise: General Information

v1, 2024-10-16

The programming exercise consists of two assignments. This document contains information valid for both assignments, please read it carefully. If you have questions, do not hesitate to contact us either after the lectures or via `heuopt@ac.tuwien.ac.at`.

1 The Minimum Weighted Crossings with Constraints Problem

We consider the *Minimum Weighted Crossings with Constraints Problem* (MWCCP), which is a generalization of the *Minimum Crossings Problem*. In the MWCCP we are given an undirected weighted bipartite graph $G = (U \cup V, E)$ with node sets $U = \{1, \ldots, m\}$ and $V = \{m+1, \ldots, n\}$ corresponding to the two partitions, edge set E, and a set of constraints C. The node sets U and V are disjoint. The edges $e = (u, v) \in E \subset U \times V$ have associated weights $w_e = w_{u,v}$. Precedence constraints C are given in the form of a set of ordered node pairs $C \subset V \times V$, where $(v, v') \in C$ means that node v must appear before node v' in a feasible solution.

The nodes of the graph G are to be arranged in two layers. The first layer contains all nodes of set U in fixed label order 1,...,m, while the second layer contains the nodes of set V in an order to be determined. The goal of the MWCCP is to find an ordering of the nodes V such that the weighted edge crossings are minimized while satisfying all constraints C.

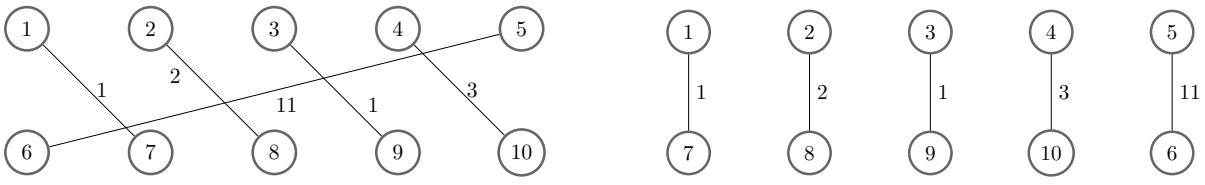
A candidate solution is thus represented by a permutation $\pi = (\pi_{m+1}, \ldots, \pi_n)$ of the nodes V. It is only feasible if all of the constraints C are fulfilled, i.e., $\text{pos}_\pi(v) < \text{pos}_\pi(v')$, $\forall (v, v') \in C$, where pos(v) refers to the position of a node $v \in V$ in the permutation π.

The objective function to be minimized is

$$f(\pi) = \sum_{(u,v) \in E} \sum_{(u',v') \in E | u < u'} (w_{u,v} + w_{u',v'}), \cdot \delta_\pi((u, v), (u', v')) \tag{1}$$

where $\delta_\pi(\cdot, \cdot)$ is an indicator function yielding one if the two considered edges cross each other in the solution given by the permutation π and zero otherwise, i.e.,

$$\delta_\pi((u, v), (u', v')) = \begin{cases} 1 & \text{if } \text{pos}_\pi(v) > \text{pos}_\pi(v') \\ 0 & \text{else.} \end{cases} \tag{2}$$



(a) a feasible solution (b) an optimal solution

Figure 1: Two exemplary solutions to an MWCCP instance; assume $C = \{(7, 10), (8, 9)\}$.

Figure 1 gives an example of two solutions for the same MWCCP instance. The left solution has objective value $f(\langle 6, 7, 8, 9, 10 \rangle) = 12 + 13 + 12 + 14 = 51$, while the right solution has objective value $f(\langle 7, 8, 9, 10, 6 \rangle) = 0$ and is an optimum.

2 Instances & Solution Format

A problem instance is given as plain ASCII file and contains in the first line, separated by a space, the number of nodes in set U, the number of nodes in set V, the number of constraints and the number of edges.

The second part of the file separated by the string `#constraints` contains the constraint information by node pairs in which the first node must appear before the second node.

The third part of the file separated by the string `#edges` contains for each edge in E the two connected nodes from U and V followed by the weight.

```
<|U|> <|V|> <|C|> <|E|>
#constraints
<node> <node>
...
<node> <node>
#edges
<node> <node> <weight>
...
<node> <node> <weight>
```

The MWCCP instance used in Figure 1 is specified by the following file `test`:

```
1:5 6:10 2 5
#constraints
7 10
8 9
#edges
1 7 1
2 8 2
3 9 1
4 10 3
5 6 11
```

In our programming assignment, the instance filename uniquely identifies a problem instance.

The format for the solution submission is as follows: The name of the problem instance in the first line followed by a single line containing the permutation of the nodes in V separated by a whitespace. The solution file for the example Figure 1b is thus:

```
test
7 8 9 10 6
```

3 Reports

For each programming exercise you are expected to hand in a concise report via TUWEL containing (if not otherwise specified) at least:

- A description of the implemented algorithms (the adaptions, problem-specific aspects, parameters etc., not the general procedure).
- Experimental setup (machine, tested algorithm configurations).
- Best objective values and runtimes plus (mean/std. deviation for randomized algorithms over multiple runs) for each published instance and algorithm. Infeasible solutions must be excluded from these calculations.
- Do not use excessive runtimes for your algorithms, limit the maximum runtime to, e.g., 15 minutes per instance on the machine you use.
- Use the instances of various sizes provided in TUWEL. Use the `tuning_instances` to tune the parameters of your different algorithms. Report the results on the instances in `test_instances` in your report.

What we do not want:

- Multithreading and multiprocessing, GPU usage – use only single CPU threads.
- UML diagrams (or any other form of source code description).
- Repetition of the problem description.

4 Solution & Source Code Submission

Hand in your best solutions for each instance and each algorithm **and** a zip-archive of your source code in TUWEL before the deadline. Make sure that the reported best solutions and the uploaded solutions match.

We will provide a competition server where you can upload solutions for certain instances. The uploaded solutions are then checked for correctness and, if okay, entered in a ranking table. The ranking table shows information about group rankings (best three groups per instance & algorithm) and solution values to give you an estimate of your algorithms performance in comparison to your colleagues' algorithms. Your ranking does not influence your grade. However, the finally best three groups will win small prizes!

5 Development Environment & AC Group's Cluster

You are free to use any programming language and development environment you like.

It is also possible to use the AC group's computing cluster:
Login using ssh on `USERNAME@eowyn.ac.tuwien.ac.at` or `USERNAME@behemoth.ac.tuwien.ac.at`. Both machines run `Ubuntu 18.04.6 LTS` and provide you with `Julia 1.11`, a `gcc 7.5.0` toolchain, `Java openJDK 11.`, and `Python 3.6`. You may install other programming languages or language versions or software packages on your own in your home directory but take care to stay in your 5GB disk quota limit.

Possible starting points may be the following open source frameworks maintained by our group, which provide generic implementations of VNS, GRASP, LNS etc. and examples for the TSP, MAXSAT, and graph coloring:

- `https://github.com/ac-tuwien/MHLib.jl` for Julia
- `https://github.com/ac-tuwien/pymhlib` for Python

The usage of any other suitable packages, e.g., for handling graphs or visualization purposes, also is allowed.

Do not run heavy compute jobs directly on behemoth or eowyn but instead submit jobs to the cluster in a batch fashion.

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` command 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 done by `qdel <job_id>`. You can find additional information under `https://www.a.c.tuwien.ac.at/students/compute-cluster/`.