TU Wien
Institute for Logic and Computation
Algorithms and Complexity Group

Heuristic Optimization Techniques, WS 2023

Programming Exercise: General Information

v3, 2023-11-09

The programming exercise consists of two assignments. This document contains information valid for both assignments, please read it carefully. If you have questions, please contact us either after the lecture units or by e-mail via `heuopt@ac.tuwien.ac.at`.

1 The Weighted s-Plex Editing Problem

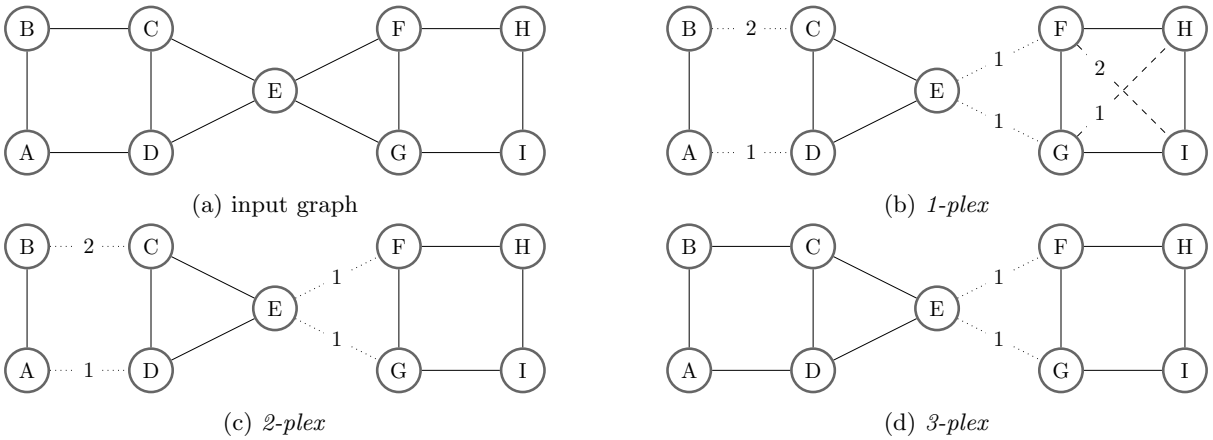
We consider the *Weighted s-Plex Editing Problem* (WsPEP), which is a generalization of the *s-Plex Editing Problem* which in itself is a generalization of the *Graph Editing Problem*. We are given an undirected graph $G = (V, E)$, a positive integer value s, and a symmetric weight matrix W which assigns every (unordered) vertex pair $(i, j) : i, j \in V, i \neq j$ a non-negative integer weight w_{ij}.

An *s-plex* of a graph is a subset of vertices $S \subseteq V$ such that each vertex has degree at least $|S| - s$ and there exist no edges to vertices outside of the *s-plex*, i.e., $i, j \in V,\ i \in S,\ j \notin S \implies (i, j) \notin E$. Note that a clique (complete graph on a vertex subset) is a *1-plex*.

The goal is to edit the edges of the graph by deleting existing edges and/or inserting new edges such that the edited graph consist only of non-overlapping *s-plexes* and such that the sum over all weights of the edited edges is minimal.

Let a candidate solution be represented by variables $x_{ij} \in \{0, 1\}$, $i, j \in V,\ i < j$, where a value 1 indicates that edge (i, j) is either inserted (if $(i, j) \notin E$) or deleted (if $(i, j) \in E$) and a value 0 indicates that the edge is not edited. The objective function is then given by

$$\min f(x) = \sum_{(i,j) \in E} w_{ij} x_{ij}$$



(a) input graph (b) *1-plex* (c) *2-plex* (d) *3-plex*

Figure 1: *s-plex* editing example.

Figure 1 gives three examples of *s-plex* editing. The input graph G is shown in Figure 1a. Figure 1b depicts a solution for the *1-plex* editing problem with value $f(x) = 8$, Figure 1c a solution for the *2-plex* editing problem with value $f(x) = 5$, and Figure 1d a solution for the *3-plex* editing problem with a value $f(x) = 2$. Dashed edges represent inserted edges while dotted edges represent deleted edges. The numbers on the edges indicate the corresponding constant weights (for simplicity only weights of edited edges are displayed).

2 Instances & solution format

A problem instance is given as plain ASCII file and contains in the first line, separated by a space, value s, the number of nodes $n = |V|$, the number $m = |E|$ of edges in G, and the number of the following lines ($l = n(n-1)/2$), from which graph G and weight matrix W can be compiled: For all pairs of nodes $i, j \in V = \{1, \ldots, n\}$, $i < j$, there is one line containing (separated by spaces) i, j, value 1 if $(i, j) \in E$ and 0 otherwise, and the weight w_{ij}. The example file `test` can be seen below:

```
<s> <n> <m> <l>
<start_node_1> <target_node_1> <edge_present_1> <weight_1>
<start_node_2> <target_node_2> <edge_present_2> <weight_2>
...
<start_node_l> <target_node_l> <edge_present_l> <weight_l>
```

The example file `test` represents the instance shown in Figure 1 and is shown bellow:

```
2 9 12 36
1 2 1 5
1 3 0 6
1 4 1 1
1 5 0 9
1 6 0 10
1 7 0 1
1 8 0 1
1 9 0 1
2 3 1 2
2 4 0 4
2 5 0 6
2 6 0 1
2 7 0 1
2 8 0 1
2 9 0 1
3 4 1 3
3 5 1 1
3 6 0 4
3 7 0 2
3 8 0 1
3 9 0 1
4 5 1 6
4 6 0 4
4 7 0 3
4 8 0 1
4 9 0 1
5 6 1 1
5 7 1 1
5 8 0 3
5 9 0 3
6 7 1 1
6 8 1 1
6 9 0 2
7 8 0 1
7 9 1 1
8 9 1 1
```

This represents the adjacency and weight matrices

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{1}$$

$$W = \begin{pmatrix} 0 & 5 & 6 & 1 & 9 & 10 & 1 & 1 & 1 \\ 5 & 0 & 2 & 4 & 6 & 1 & 1 & 1 & 1 \\ 6 & 2 & 0 & 3 & 1 & 4 & 2 & 1 & 1 \\ 1 & 4 & 3 & 0 & 6 & 4 & 3 & 1 & 1 \\ 9 & 6 & 1 & 6 & 0 & 1 & 1 & 3 & 3 \\ 10 & 1 & 4 & 4 & 1 & 0 & 1 & 1 & 2 \\ 1 & 1 & 2 & 3 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 3 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 3 & 2 & 1 & 1 & 0 \end{pmatrix} \tag{2}$$

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 lines containing pairs of nodes (i, j), with $i < j$, without parentheses and separated by a whitespace, for which the corresponding edges (i, j) are to be edited, i.e., either inserted if $(i, j) \notin E$ or deleted if $(i, j) \in E$. The solution for the example instance shown above in Figure 1c is thus:

```
test
1 4
2 3
5 6
5 7
```

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 (+ mean/std. deviation for randomized algorithms over multiple runs) for all published instances and algorithms. Infeasible solutions must be excluded from these calculations.
- Do not use excessive runtimes for your algorithms, limit the maximum runtime to 15 minutes per instance on the machine you use.
- Use the instances in `inst_tuning` to tune the parameters of your different algorithms. Report the results on the instances in `test_instances` and `inst_competition` in your report and upload the results on the instance in `inst_competition` to the respective section in TUWEL.

What we do not want:

- Multithreading and multiprocessing, GPU usage – use only single 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. The upload can be repeated multiple times, only best solutions are shown.

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 LTS` and provide you with `Julia 1.9` (via `/home1/share/julia/bin/julia`), a `gcc 7.5.0` toolchain, `Java openJDK 11.`, and `Python 3.6`. Other programming language versions or software packages may be installed in your home directory and the environment variables should be set accordingly.

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.

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