TU Wien
Institute for Logic and Computation
Algorithms and Complexity Group

186.112 Heuristic Optimization Techniques, WS 2021

Programming Exercise: General Information

v1, 2021-10-25

The programming exercise consists of two assignments. This document contains information valid for both assignments. Please read it carefully and contact us via `heuopt@ac.tuwien.ac.at` if you have any questions.

1 Problem

This section presents and discusses the problem you will have to solve in the programming assignments.

1.1 Motivation

A telecommunication company modernizes their cable network in a city to increase the bandwidth of their customers. To do so, they need to connect specific points in the city with each other using fiber optic cables. At the same time another company wants to establish a district heating system. They therefore need to connect the producer of the heat with all the consumers. In both cases it is necessary to dig up parts of the street. Since both, fiber optic cables and pipes, can be laid with only digging up once, the two companies decide to cooperate. Possible routes and their intersections have already been combined into a graph. It is your task now to design the two networks such that the total costs are minimized. The total costs consist of costs for laying pipes and costs for laying fiber optic cables subtracted by the savings due to shared passages of both networks.

1.2 Specification

We call the formalization of the problem of finding two such networks the *Double Steiner Tree Problem* (DSTP). It is an adaption of the well-known classical Steiner Tree Problem (STP). We are given

- an undirected simple connected graph $G = (V, E)$ with vertex set V, edge set $E \subseteq V \times V$, and edge weights $w_{i,j} \in \mathbb{N}$, $\forall (i, j) \in E$,
- two subsets of vertices called terminals $T_1, T_2 \subseteq V$ and
- factors $\alpha_1, \alpha_2 \in \mathbb{N}$ and $\gamma \in \mathbb{Z}, \gamma \geq \max(-\alpha_1, -\alpha_2)$.

Let $w(X) := \sum_{(i,j)\in X} w_{i,j}$ for $X \subseteq E$. The aim is to find two **connected, cycle-free** subgraphs $S_k = (V_k, E_k), k \in \{1, 2\}$ of G that span the terminal sets, i.e., $T_k \subseteq V_k, k \in \{1, 2\}$, and minimize

$$\sum_{k\in\{1,2\}} \alpha_k \cdot w(E_k) + \gamma \cdot w(E_1 \cap E_2). \tag{1}$$

The subgraphs S_1 and S_2 are called Steiner trees.

1.3 Example

A DSTP example instance with a corresponding solution is shown in Figure 1. Since $|E_1| = 5$, $|E_2| = 4$, $|E_1 \cap E_2| = 1$, and all edge weights are 1, the objective value results in $2 \cdot 5 + 3 \cdot 4 - 2 \cdot 1 = 20$.

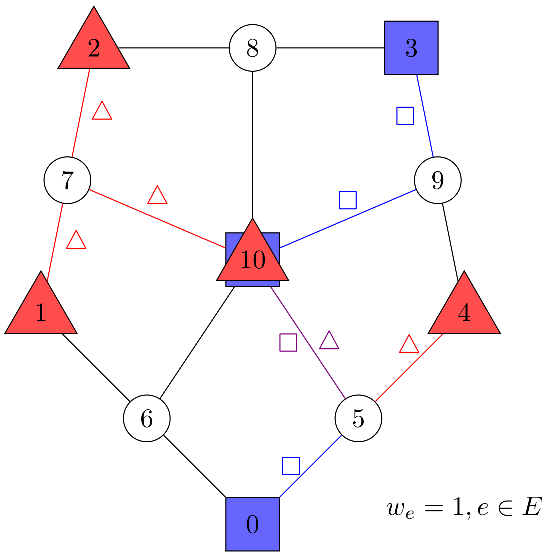


Figure 1: An example instance to the DSTP. All edge weights are 1, $\alpha_1 = 2$, $\alpha_2 = 3$ and $\gamma = -2$. Red triangles are terminals in T_1, blue squares are terminals in T_2. Vertex 10 is contained in both, T_1 and T_2. Edge labels and colors indicate the two Steiner trees of a solution to the instance.

1.4 Notes on the Problem Structure

Relation to the STP. The classical STP only gives, besides the weighted graph G, one set of terminals T and asks for a minimum weight connected subgraph S of G that spans at least those terminals. When restricting the search within the DSTP to one of the trees of a solution (i.e., fixing the other tree), one essentially solves the STP with adapted weights. Since $\gamma \geq \max(-\alpha_1, -\alpha_2)$, the adapted edge weights are not negative. Therefore the considerations for the STP are also useful for the DSTP. It furthermore allows you to consult the extensive literature on the STP for inspiration.

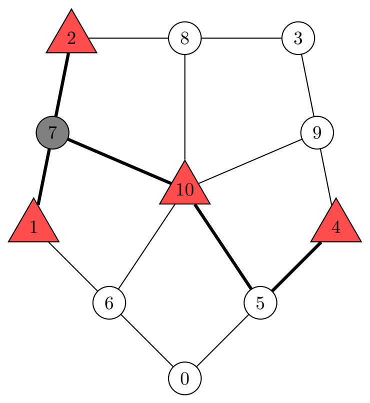
Key-nodes and key-paths. Consider a single Steiner tree S for the terminal set T. A *key-node* is a vertex of S that is not a terminal and has degree at least three within S. Denote the set of key-nodes with N. A *key-path* is a path between two vertices in $N \cup T$ that does not contain any vertices of $N \cup T$ in between. The key-node and key-paths of the first Steiner tree in the example solution are illustrated in Figure 2. Observe that in an optimal Steiner tree all key-paths are shortest paths.

Consider an optimal Steiner tree S of graph G for the terminal set T with key-nodes N. Construct the distance graph D of $N \cup T$ in G, i.e., the complete graph with nodes $N \cup T$ and distances as edge weights. Taking the minimum spanning tree of D and replacing its edges by the corresponding shortest paths in G results again in the Steiner tree S. If it would not, S was not optimal, since the construction would then give a Steiner tree of smaller weight. Therefore an optimal solution can be represented by its key-nodes only. Hence a search procedure can also try to find a good set of key-nodes and operate on this set.

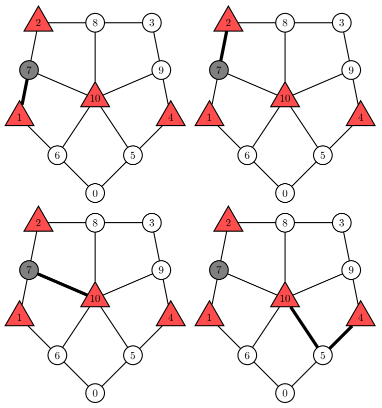
1.5 Instance and Solution Format

The instances are given as plain ASCII files of the following format:

S $\lvert V\rvert$ $\lvert E\rvert$ $\lvert T_1\rvert$ $\lvert T_2\rvert$	
F α_1 α_2 γ	
E u v $w_{u,v}$	$\forall(u,v) \in E$
T1 v	$\forall v \in T_1$
T2 v	$\forall v \in T_2$



(a) The Steiner tree with the single key-node 7 highlighted.



(b) The four key-paths of the Steiner tree.

Figure 2: Illustration of the terms *key-node* and *key-path* for the first Steiner tree of the example in Figure 1.

Vertices are identified by their indices starting with 0: $V = \{0, \dots, n-1\}$.

Solutions are also plain ASCII files and list the edges of the two Steiner trees $S_k, k \in \{1, 2\}$ in arbitrary order in the following format. $|V|$, $|E|$, $|T_1|$ and $|T_2|$ are used by the submission tool to determine the corresponding instance.

S $|V|$ $|E|$ $|T_1|$ $|T_2|$	
S1 u v	$\forall (u, v) \in E_1$
S2 u v	$\forall (u, v) \in E_2$

The instance in Figure 1 can be represented by a file with the following contents:

```
S 11 15 4 3
F 2 3 -2
E 0 5 1
E 0 6 1
E 1 6 1
E 1 7 1
E 2 7 1
E 2 8 1
E 3 8 1
E 3 9 1
E 4 9 1
E 4 5 1
E 5 10 1
E 6 10 1
E 7 10 1
E 8 10 1
E 9 10 1
T1 1
T1 2
T1 4
T1 10
T2 0
T2 3
T2 10
```

The highlighted solution in Figure 1 can be represented as follows:

```
S 11 15 4 3
S1 1 7
S1 2 7
S1 7 10
S1 10 5
S1 5 4
S2 0 5
S2 5 10
S2 10 9
S2 9 3
```

Note that the first vertex of an edge is not required to be lower than the second vertex.

2 Reports

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

- A description of the implemented algorithms (the adaptions, selected options, etc. – not the general procedure).
- The experimental setup (machine, tested algorithm configurations).
- Best objective values and runtimes 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 do not want:

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

3 Solution and Source Code Submission

We require you to hand in your best solutions for each instance and each algorithm **and** an 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 the best solutions are shown.

The uploaded solutions are then checked for correctness and 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. Your ranking does not influence your grade.

4 Development Environment and Cluster Usage

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

It is also possible to use the computer cluster of the Algorithms and Complexity Group. Log in using ssh via `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 1.6` (via `/home1/share/julia/bin/julia`), `gcc 7.5.0`, `Java 1.8` and `Python 3.7`. Other programming language versions or software packages may be installed in your home directory.

A good starting point may be one of the following open source frameworks maintained by our group:

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

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/`