Programming Exercise Part 2 for Mathematical Programming 186.835

Benjamin Schwendinger
e1225371

May 31, 2019

1 k-Node Minimum Spanning Tree (k-MST) Problem

Given:

- graph $G = (V, E, w)$
- edge weights $w(e) \in \mathbb{R}^+_\nu, \forall e \in E$
- integer $0 \leq k \leq |V|$

Goal: Find a minimum weight tree, spanning exactly k nodes.

I use a directed graph to model the problem. Hence for each edge in E, I use two directed arcs instead. Moreover, the weight of each arc is the same as the corresponding weight of the undirected graph. Furthermore a root node (I call it 0) and edges from 0 to each other node with weight 0 are added. I write the set of neighbors of a vertex v as $N(v)$.

2 Formulations

2.1 Variables:

2.1.1 Binary variables:

$$x_e = \begin{cases} 1 & \text{if edge } e \text{ is in the tree} \\ 0 & \text{otherwise} \end{cases}$$

$$y_a = \begin{cases} 1 & \text{if arc } a \text{ is in the tree} \\ 0 & \text{otherwise} \end{cases}$$

$$z_v = \begin{cases} 1 & \text{if vertex } v \text{ is in the tree} \\ 0 & \text{otherwise} \end{cases}$$

2.1.2 Integer variables:

2.2 CEC

$$\min_{e\in E} c_e x_e \tag{1a}$$

$$\text{subject to } \sum_{a\in A} y_a = k+1 \tag{1b}$$

$$\sum_{v\in V} z_v = k+1 \tag{1c}$$

$$\sum_{i\in N(j)} y_{ij} \le z_j \qquad \forall j \in V \tag{1d}$$

$$\sum_{i\in N(0)} y_{i0} = \sum_{i\in N(0)} y_{0i} = 1 \tag{1e}$$

$$y_{ij} \le z_i, z_j \qquad \forall i,j \in V \tag{1f}$$

$$y_{ij} + y_{ji} = x_e \qquad \forall e = \{i,j\}, e \in E \tag{1g}$$

$$\sum_{e\in C} x_e \ge |C| - 1 \qquad \forall C \subseteq E, |C| \ge 2, C \text{ forms a cycle} \tag{1h}$$

$$z_v \in \{0,1\} \qquad \forall v \in V \tag{1i}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{1j}$$

$$y_a \in \{0,1\} \qquad \forall a \in A \tag{1k}$$

2.3 DCC

$$\min_{e\in E} c_e x_e \tag{2a}$$

$$\text{subject to } \sum_{a\in A} y_a = k+1 \tag{2b}$$

$$\sum_{v\in V} z_v = k+1 \tag{2c}$$

$$\sum_{i\in N(j)} y_{ij} \le z_j \qquad \forall j \in V \tag{2d}$$

$$\sum_{i\in N(0)} y_{i0} = \sum_{i\in N(0)} y_{0i} = 1 \tag{2e}$$

$$y_{ij} \le z_i, z_j \qquad \forall i,j \in V \tag{2f}$$

$$y_{ij} + y_{ji} = x_e \qquad \forall e = \{i,j\}, e \in E \tag{2g}$$

$$\sum_{e\in\delta^-(S)} x_e \ge z_v \qquad \forall S \subseteq V, \forall v \in S \tag{2h}$$

$$z_v \in \{0,1\} \qquad \forall v \in V \tag{2i}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{2j}$$

$$y_a \in \{0,1\} \qquad \forall a \in A \tag{2k}$$

The objective is modelled by Equation (a) which states the weight of the found tree has to be minimal.
The constraints (b) to (g) are the same for both formulations.

(b) Our found directed graph has to have $k+1$ arcs. Hence I get a tree on by deleting the arcs from and to 0.

(c) Our found tree has to have $k+1$ nodes. This is true, since I am searching for a tree on k for graph without our added root node.

(d) The number of ingoing arcs to every node is less or equal then the binary variable represent the node. This enforces that the found graph is a forest.

(e) The number of ingoing and outgoing arcs from 0 is 1. This enforces that really $k-1$ edges of the input graph are used and not only our added zero weight edges from and to 0.

(f) An arc can only be in the tree if both ends of it are in the tree.

(g) An edge is in the graph if one and at most one of the corresponding arcs is in the directed graph.

The constraints $(1h)$ ensures that the solution does not contain a cycle.
The constraints $(2h)$ ensures that the solution has only 1 connected component.
The constraints i to k set the domains of my binary variables.

3 Seperation

Since both approaches got exponential many constraints due to the constraints $(1h)$ and $(2h)$, I use a variation of the Branch-and-Cut method. Here I also allow that the relaxed LP program does not contain all constraints of my MILP, but these constraints are still added if needed. This kind of constraints are called **lazy constraints**, since I add them in a lazy manner. Whenever a lazy constraint is violated in the seperation process, I add it to my set of active constraints. Hence, my lazy constraints can also cut off invalid integer solutions which were still valid in the relaxed program.

3.1 CEC

Here I check if the solution graph contains a cycle and if so, I remove the current solution of the set of valid solutions (enforce constraint 1h). This is done by the convenient function $cycle_basics(G)$ which returns all cycles in a given graph G.

3.2 DCC

Here I check if the solution graph consists of more than 1 connected component. If there are more connected components than I enforce that at least two of these components have to be connected by an edge.

Inputs		CEC				DCC			
Instance	k	Objective	Nodes explored	Time(s)	#v iol.Ineq	Objective	Nodes explored	Time(s)	#v iol.Ineq
1	2	46	0	0.018	0	46	0	0.038	0
1	5	477	0	0.020	0	477	0	0.052	0
2	4	373	0	0.047	0	373	0	0.066	2
2	10	1390	0	0.054	3	1390	0	0.079	12
3	10	725	0	0.079	1	725	0	0.038	2
3	25	3074	265	0.226	13	3074	0	0.108	6
4	14	909	0	0.216	1	909	24	0.204	4
4	35	3292	116	0.248	9	3292	23	0.218	25
5	20	1235	0	0.286	5	1235	0	0.333	8
5	50	4898	1625	0.843	36	4898	263	0.814	100
6	40	2068	10 510	4.632	38	2068	1532	5.901	265
6	100	6705	4369	3.591	111	6705	4949	10.073	357
7	60	1335	0	4.696	5	1335	555	19.555	193
7	150	4534	315	4.523	28	4534	10 490	268.619	1364
8	80	1619	6	3.908	6	1620	4016	114.485	946
8	200	5787	7362	18.430	234	5787	18 222	400.209	2491

Table 1: Result table for different formulations

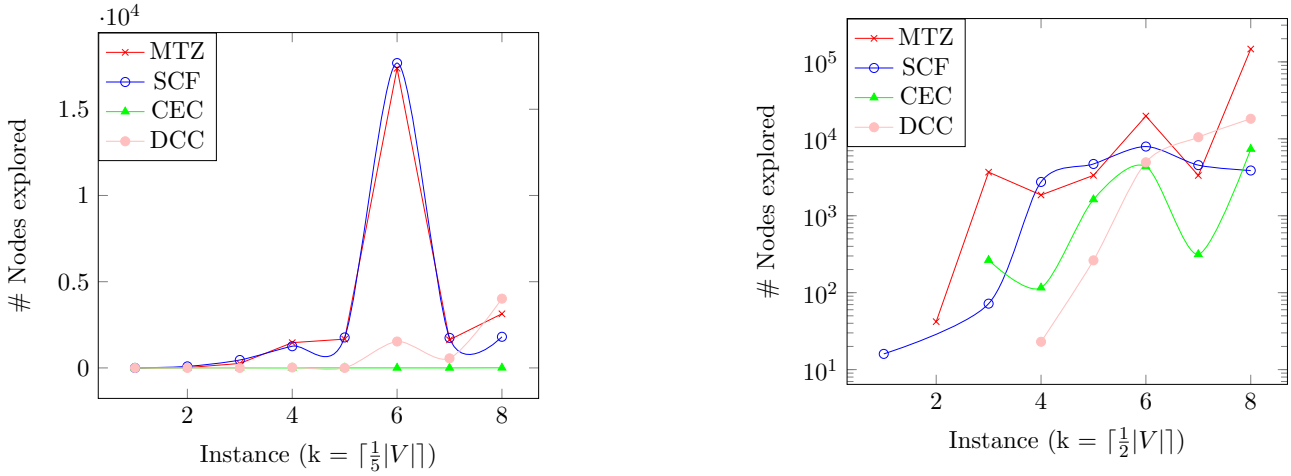

Figure 1: Number of explored nodes for different formulations

4 Computational Results

The implementations are done in Python and I use Gurobi 8 to solve our ILP approach. All tests were performed on a single core of an Intel Xeon E5540 processor with 2.53 GHz using at most 3GB RAM.

According to Table 1 and Figure 1 I see that the least nodes have to be explored for CEC. The running times according to Table 1 and Figure 2 show that CEC is the fastest approach. This is probably also due to use of the convenient function from networkx. Moreover, we see that all optimal values are reached with both developed formulations. In 1 and Figure 3 I see that the least number of violated inequalities are added for the CEC approach.

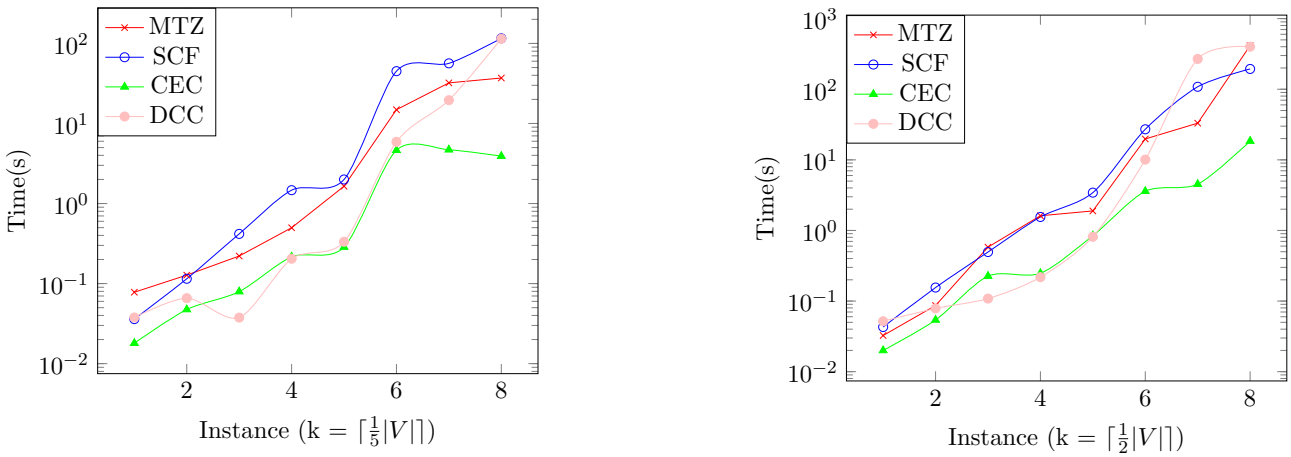

Figure 2: Running times for different formulations

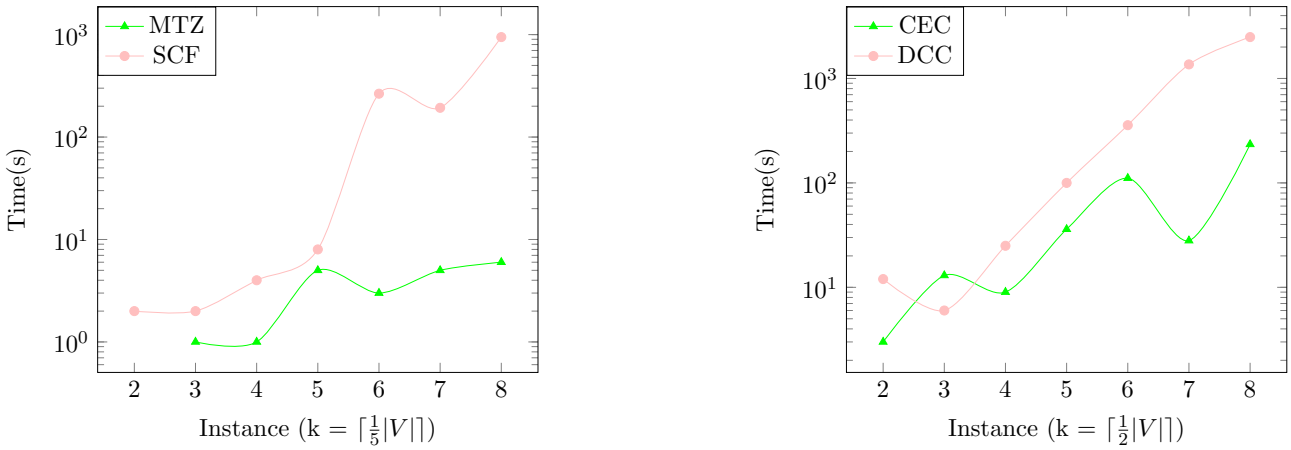

Figure 3: Number of violations added for different formulations