# Mathematical Programming
# Programming Exercise Report 2

Mathias Hofer (01226806)

May 2018

## 1 Model

Objective function

$$\min \Sigma_{e \in E} w_e x_e$$

Make sure that we have a connected tree, thus we only select arcs from nodes which have some selected incoming arc.

$$\Sigma_{(i,j) \in \delta^-(j)} a_{ij} \geq a_{jk} \qquad \forall j \in V, \forall (j,k) \in \delta^+(j)$$

Linking constraints for arc and edge variables, which also ensure that we at most select one arc between all adjacent nodes

$$a_{ij} + a_{ji} \leq x_e \qquad \forall e = (i,j) \in E$$

Make sure that we select $k$ arcs

$$\Sigma_{(i,j) \in A} a_{ij} = k$$

Make sure that we do not select any arc back to the artificial root node

$$a_{i0} = 0 \qquad \forall i \in V$$

Make sure that we only select one of the artificial zero-weight arcs from the root node

$$\Sigma_{(0,i) \in \delta^+(0)} a_{0i} \leq 1$$

Cycle elimination constraints

$$\Sigma_{e \in C} x_e \leq |C| - 1 \qquad \forall C \subseteq E, |C| \geq 2, C \text{ forms a cycle}$$

Edge selection variables

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

Arc selection variables

$$a_{ij} \in \{0,1\} \qquad (i,j) \in A$$

# 2 Separation Method

## 2.1 Non-Fractional Case

The procedure for the non-fractional case is basically the one from the slides. Hence, we are looking for a cycle C s.t. $\Sigma_{(i,j)\in C}(1-\overline{a}_{ij}) < 1$ where $\overline{a}$ denotes a solution. This is done by building a directed graph with weight $\overline{w}_{ij} = 1 - \overline{a}_{ij}$ for each arc $(i,j) \in A$. Then we are looking for the cheapest path $P$ from $v$ to $u$ for each arc $(u,v) \in A$. If we have found some path s.t. $\Sigma_{(i,j)\in P}\overline{w}_{ij} + \overline{w}_{uv} < 1$ we have identified some a violated cycle and thus add the inequality to our formulation.

## 2.2 Fractional Case

For the fractional case we use the same procedure as above, but with some modifications to the directed graph. Since in the fractional case we have $a_{ij} \in [0,1]$ and thus for example have $a_{ij} = 1/3$ and $a_{ji} = 2/3$ for some $i,j \in V$. What we do to apply the above procedure is we set $\overline{w}_{ij} = 1 - 1 = 0$ if $a_{ij} < a_{ji}$ and $\overline{w}_{ij} = 1 - 0 = 1$ otherwise. Intuitively, we just select one of the arcs, i.e. the one with the lower value, if we have fractional values $> 0$ for both directions. So basically we just "guess" an integer solution, based on the fractional one and then preemptively add cycle elimination inequalities.

# 3 Results

| | **\|V\|** | **K** | **Objective** | **Runtime in Seconds** | **B&B Nodes** | **User Cuts** |
|---|---|---|---|---|---|---|
| g01 | 10 | 2 | 46 | 0.09 | 0 | 0 |
| | | 5 | 447 | 0.02 | 0 | 0 |
| g02 | 20 | 4 | 373 | 0.05 | 0 | 0 |
| | | 10 | 1390 | 0.08 | 0 | 2 |
| g03 | 50 | 10 | 725 | 0.03 | 0 | 1 |
| | | 25 | 3074 | 0.39 | 32 | 17 |
| g04 | 70 | 14 | 909 | 0.20 | 12 | 6 |
| | | 35 | 3292 | 0.28 | 4 | 11 |
| g05 | 100 | 20 | 1235 | 0.25 | 2 | 7 |
| | | 50 | 4898 | 1.16 | 341 | 50 |
| g06 | 200 | 40 | 2068 | 6.58 | 1081 | 45 |
| | | 100 | 6705 | 11.22 | 1,348 | 138 |
| g07 | 300 | 60 | 1335 | 5.08 | 24 | 8 |
| | | 150 | 4534 | 19.25 | 64 | 35 |
| g08 | 400 | 80 | 1620 | 11.56 | 8 | 13 |
| | | 200 | 5787 | 182.39 | 5,505 | 252 |

Table 1: Results on a Intel Core i7-5500U 2.4GHz

# 4 Interpretation of Results

In comparison to the compact models, the CEC approach outperforms them quite significantly. The most extreme gap occured on instance g08, for which the SCF formulation "exploded" for some reason, s.t. it took over 1 hour to find a solution whereas the CEC approach just takes about 3 minutes. Thus, in percent, the CEC approach takes only 3.93% percent of the time and only 0.15% of the B&B nodes of the SCF formulation. But also for not so extreme instances like g06 it only takes about 1/7th of the time and significantly less B&B nodes.

The amount of added inequalities during the cycle elimination procedure also has an impact on the runtime. The first implementation just added all found valid inequalities. However, after some experiments we stop the search for further cycles after 5-6 added inequalities. If we add less then the whole overhead of the graph generation and path search is too high and if we add more then we spend to much time with the current solution. Thus, performance was best with at most 5-6 added valid inequalities per call.

As for the fractional separation method, it turned out that given two arcs $a_{ij}$ and $a_{ji}$ it works best when we select, i.e. assign the value 1 to it whereas the inverse becomes 0, the one with the lower fractional value. This may seem a bit surprising, since this arc may have a lower value because it has less impact on the objective function. However, from my perspective, we try by the selection of the not so promising arc that we preemptively cut off not so promising parts from the search space. For most instances the runtime improvement was not so significant, but for instance g08 it took about a minute less to find a solution with this separation method. As for the amount of added valid inequalities, the same arguments as above apply, thus best results were achieved with a value of at most 5-6 added valid inequalities per call.