Mathematical Programming
Programming Exercise
Part 1: Compact Formulations

Helmuth Breitenfellner

Abstract

This report describes the formulations of the k-nodes Minimal Spanning Tree (k-MST). The formulations are based on *Single-Commodity Flows*, *Multi-Commodity Flows* and the *Sequential Formulation* based on Miller, Tucker and Zemlin [1]. These formulations have then been implemented using `CPLEX` in C++.

The implementation can be found on GitHub:
`https://github.com/helmuthb/mathprog/tree/master/ProgrammingExercise`

k-nodes Minimal Spanning Tree

The k-MST task is to find the *Minimal Spanning Tree* which contains exactly k nodes and minimizes the *weight sum* of the edges.
Existing formulations from the regular MST can be applied, but some adjustment is necessary.
For the regular MST one can choose any node as the root of the spanning tree, since each node will be contained. This is in the literature usually the node 1. However, for the *k-MST* this is not the case. Therefore one introduces an artificial *root node*, which has edges to all other nodes with weight 0.
This root node is then used in the formulations where normally the node 1 is used. Additional constraints are necessary to ensure that only *one* connection leaves the root node.
One has to allow for $k+1$ nodes in the resulting k-MST, as the artificial root node will be contained and is to be be removed from the results.
Also it requires additional variables to specify the nodes selected - since not all nodes will be in the k-MST..
Most formulations performed better when based on a *directed* formulation. Therefore, arcs where introduced, which exist for all edges in both directions - except for the root node, which only has arcs going out from it but not into it.
In all formulations the following values are predefined:

- n ... Number of nodes (*Vertices*)
- m ... Number of edges
- a ... Number of arcs

All formulations correspond 1-to-1 to the implemented program which was then executed. Constraints were introduced where try-and-error indicated better performance - even if they were redundant or stronger formulations seem possible.

Single-Commodity Flows Formulation (*SCF*)

The following variables are introduced:

$$
\begin{aligned}
z_i \in \{0,1\} &\ \text{...}\ \text{Node } i \text{ is selected} \\
x_{ij} \in \{0,1\} &\ \text{...}\ \text{Edge } i,j \text{ is selected} \\
y_{ij} \in \{0,1\} &\ \text{...}\ \text{Arc } i,j \text{ is taken in the tree, going from } i \text{ to } j \\
f_{ij} \in [0-k] &\ \text{...}\ \textit{Flow} \text{ from node } i \text{ to node } j
\end{aligned}
$$

With this variables one can define the *Single-Commodity Flows* formulation for the k-MST as follows:

$$f_{ij} \leq k \cdot y_{ij} \quad \forall i,j \neq 0 \tag{1}$$

$$y_{ij} + y_{ji} = x_{ij} \tag{2}$$

$$x_{ij} \leq z_i, \ \ x_{ij} \leq z_j \tag{3}$$

$$f_{0i} = k \cdot y_{0i} \tag{4}$$

$$\sum_j f_{ij} - \sum_j f_{ji} = -1 \cdot z_i \quad \forall i \neq 0 \tag{5}$$

$$\sum_{i,j} x_{ij} = k \tag{6}$$

$$\sum_j z_j = k+1 \tag{7}$$

$$\sum_i y_{0i} = 1 \tag{8}$$

Multi-Commodity Flows Formulation (*MCF*)

The following variables are introduced:

$$
\begin{aligned}
z_i \in \{0,1\} &\ \text{...}\ \text{Node } i \text{ is selected} \\
x_{ij} \in \{0,1\} &\ \text{...}\ \text{Edge } i,j \text{ is selected} \\
y_{ij} \in \{0,1\} &\ \text{...}\ \text{Arc } i,j \text{ is taken in the tree, going from } i \text{ to } j \\
f^l_{ij} \in [0-1] &\ \text{...}\ \textit{Flow} \text{ of type } l \text{ from node } i \text{ to node } j,\ l \in \{1,...,n\}
\end{aligned}
$$

With this variables one can define the *Multi-Commodity Flows* formulation for the k-MST as follows:

$$f^l_{ij} \leq y_{ij} \quad \forall l \neq 0 \tag{9}$$

$$y_{ij} + y_{ji} = x_{ij} \tag{10}$$

$$x_{ij} \leq z_i \tag{11}$$

$$x_{ij} \leq z_j \tag{12}$$

$$\sum_i y_{ij} = z_j \tag{13}$$

$$\sum_j f^l_{ij} - \sum_j f^l_{ji} = 0 \quad \forall i \neq l, i \neq 0, l \neq 0 \tag{14}$$

$$\sum_j f^i_{0j} = z_i \quad \forall i \neq 0 \tag{15}$$

$$\sum_j f^i_{ij} - \sum_j f^i_{ji} = -1 \cdot z_i \quad \forall i \neq 0 \tag{16}$$

$$\sum_{i,j} y_{ij} = k \tag{17}$$

$$\sum_i x_{0i} = 1 \tag{18}$$

Sequential Formulation (*MTZ*)

The Sequential Formulation was introduced by Miller, Tucker and Zemlin in [1].
It is an extremely compact formulation (having only a small number of conditions) but the LP-relaxation has a large Polyhedron (see e.g. [2]).
The basic idea is to give each node an *order* number, which denotes in which order it will appear in the final k-MST.
The following variables are introduced:

$z_i \in \{0,1\}$ ... Node i is selected
$x_{ij} \in \{0,1\}$ ... Edge i,j is selected
$y_{ij} \in \{0,1\}$ ... Arc i,j is taken in the tree, going from i to j
$u_i \in [0-k]$ ... *Order* of node i in the selected k-MST

With this variables one can define the *Sequence (MTZ)* formulation for the k-MST as follows:

$$y_{ij} + y_{ji} = x_{ij} \tag{19}$$

$$u_j \geq u_i + 1 - (1 - y_{ij}) \cdot k \tag{20}$$

$$\sum_j y_{ji} \leq z_i \tag{21}$$

$$x_{ij} \leq z_i \tag{22}$$

$$x_{ji} \leq z_i \tag{23}$$

$$\sum_{i,j} x_{ij} = k \tag{24}$$

$$\sum_j z_j = k + 1 \tag{25}$$

$$\sum_i x_{0i} = 1 \tag{26}$$

$$u_0 = 0 \tag{27}$$

Implementation Details

The implementation used the framework provided in the lecture.
Some adaptations include:

- Use subclasses of the `kMST_ILP` class for the algorithms

- Allow output of results with a `-v` option
- Don't add callbacks (they come lated for *CEC* and *DCC* model)

Other than that the framework was (hopefully) used as supposed.

Output Table

The optimizations have been executed multiple times - 100 times for the smaller graphs down to 5 times for the last graph - and the median of the CPU time has been taken. *The other results were consistent for each round.*
The runtimes refer to the output from the program and indicate CPU time for the solver - the actual time was of course larger.
The system used for the evaluation has an Intel® Xeon® CPU E31245 @ 3.30 GHz with 32 GB RAM.

			BnB-Nodes			CPU Time		
file	k	Objective	*SCF*	*MCF*	*MTZ*	*SCF*	*MCF*	*MTZ*
`g01.dat`	2	46	0	0	0	0.02	0.03	0.02
`g01.dat`	5	477	0	0	0	0.02	0.04	0.02
`g02.dat`	4	373	0	0	0	0.04	0.02	0.03
`g02.dat`	10	1390	0	0	0	0.10	0.10	0.05
`g03.dat`	10	725	0	0	0	0.07	0.42	0.10
`g03.dat`	25	3074	0	0	807	0.14	0.54	0.56
`g04.dat`	14	909	0	30	3	0.15	4.28	0.17
`g04.dat`	35	3292	0	0	355	0.22	1.62	0.43
`g05.dat`	20	1235	0	0	186	0.27	3.19	0.52
`g05.dat`	50	4898	0	0	2390	0.81	3.17	3.63
`g06.dat`	40	2068	251	27	4957	5.54	116.41	10.16
`g06.dat`	100	6705	537	33	5077	22.33	267.45	10.81
`g07.dat`	60	1335	487	0	911	20.37	165.32	11.16
`g07.dat`	150	4534	974	0	1791	404.69	500.42	18.04
`g08.dat`	80	1620	462	0	998	160.90	406.22	12.14
`g08.dat`	200	5787	988	0	11894	428.01	9808.82	33.49

Interpretation

Performance of SCF is quite good. It could be used for problems of smaller and medium sized problems scale within acceptable time.
Performance of MCF for smaller graphs is also good. However, once graphs are larger than 100 nodes its performance degrades very much. Running the evaluation for the last graph took hours to complete.
It's quite possible that some tweaking with the constraints could yield better results.
The overall "winner" so far (if this were a competition) is the MTZ formulation. Its compact size makes it a good choice for both small and large graphs.

Branch-and-Bound Nodes

The number of Branch-and-Bound Nodes can be seen as a measurement of the strength of the Polyhedron of the LP-relaxation. The smaller the Polyhedron is, the higher the

probability that no branch-and-bound is needed and that the best (LP) solution is already integer.
This interpretation is generally consistent with the results. MTZ is known to have a large Polyhedron for the LP-relaxation, and consistently the number of BnB-nodes is largest for it.
MCF is known to have a very tight Polyhedron for the LP-relaxation, and consistently the number of BnB-nodes is smallest for it.
However, due to the excess number of variables and conditions its performance is degraded.
One surprise is the high number of BnB-nodes for the medium sized graphs with the MTZ algorithm. I do not have a conclusive explanation for this, other than that CPLEX is a sophisticated piece of software which uses a combination of optimization algorithms combined with heuristics, and that these optimizations might influence the number of BnB-nodes beyond the characteristics of the MILP.

References

[1] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[2] Manfred Padberg and Ting-Yi Sung. Analytical comparison of different formulations of the travelling salesman problem. *Math. Program.*, 52:315–357, 05 1991.