Exercise Sheet Telecommunication

CO@Work

October 5, 2009

In this set of exercises you will formulate and solve various telecommunication network design problems using the modelling language ZIMPL and the integer linear programming solver SCIP.

The four exercises in this series build on each other. Starting with the very basic Steiner Tree Problem, the problems (and the resulting mixed integer programming models) become more complicated in each step.

Before starting with the implementation, please read the technical information at the end of this exercise worksheet.

Exercise 1: The Steiner Tree Problem

Formulate and solve the Steiner Tree Problem for all test instances.

All edge costs are strictly positive, so the Steiner Tree Problem can be described as follows:

Steiner Tree Problem	
Input	undirected graph (V, E) with edge costs $l(e)$, $e \in E$ terminal set $T \subseteq V$
Solution	edge set $E' \subseteq E$ such that (V, E') contains an (s, t)-path for each pair of distinct nodes $s, t \in T$.
Goal	$\min \sum_{e \in E'} l_e$

Use the sets and functions constructed in `SNDParser.zpl`. A detailed description can be found in the beginning of the file `SNDParser.zpl`. The cost l_{ij} for installing edge $ij \in E$ is given by the function $l(i, j)$.

Hint: Implement the directed multi-commodity flow formulation from the lecture. The parser `SNDParser.zpl` already constructs the directed arc set A, the terminal set T, and a single-element set $S1 = \{r\}$ containing a root node $r \in T$. You only need to:

1. Create a unit-demand commodity from the root to each other terminal node (in ZIMPL: from each node in $S1$ to each node in $T - S1$) and add continuous arc-flow variables and flow conservation constraints for each of these commodities.
2. Create binary variables for each edge and inequalities ensuring that an edge is installed whenever some commodity flows across the corresponding arcs. Recall that you get a stronger (but also larger) formulation by adding flows on antiparallel arcs.

Alternatively, you could also try the weaker but smaller model, where the $|T| - 1$ units of flow are sent from the root to the other terminals as a single commodity flow.

Solutions:

atlanta	4.5550000000e+06
cost266	5.6727000000e+05
di-yuan	1.8490000000e+05
france	1.0608000000e+04
nobel-eu	6.9620000000e+04
nobel-germany	2.2050000000e+04
nobel-us	5.1290000000e+04
pdh	4.8463500000e+05
polska	1.8420000000e+03

Exercise 2: The Survivable Network Design Problem

Formulate and solve the Survivable Network Design Problem with edge connectivity requirements 1 and 2 for all test instances.

Formally, this problem is given as follows:

$\{1,2\}$-Survivable Network Design Problem (Edge Connectivity)	
Input	undirected graph (V, E) with edge costs $l(e)$, $e \in E$ connectivity requests $\rho_v \in \{1, 2\}$
Solution	edge set $E' \subseteq E$ such that (V, E') contains $\min\{\rho_s, \rho_t\}$ edge disjoint (s, t)-paths for each pair of distinct nodes $s, t \in V$.
Goal	$\min \sum_{e \in E'} l_e$

All necessary sets and functions are constructed in `SNDParser.zpl`. We assume that

$$\rho_v := \begin{cases} 2 & v \in T \\ 1 & v \in V \setminus T \end{cases} \qquad \text{for all } v \in V.$$

The cost l_{ij} for installing edge ij is given by the function $l(i, j)$ again.

Hint: Transform the undirected problem into a directed one and use a directed multi-commodity flow formulation similar to that for the Steiner Tree Problem. A very good model is obtained as follows:

1. Pick a root node $r \in T = \{v \in V | \rho_v = 2\}$ (in ZIMPL, use the set $S1 = \{r\}$ created by `SNDParser.zpl`, again) and create a unit-demand commodity from r to each node in $V \setminus \{r\}$. These commodities form the first commodity set. Add arc-flow variables and flow conservation constraints for these commodities.

2. Create a second commodity from r to each node in $T \setminus \{r\}$. These commodities form the second commodity set. Add extra arc-flow variables and flow conservation constraints for these commodities, too.

3. Add inequalities to ensure that, for each node $t \in T \setminus \{r\}$, the two commodities from r to t are routed along edge disjoint paths.

4. Create binary variables for each edge and inequalities ensuring that an edge is installed whenever some commodity flows across the corresponding arcs.

   You get a stronger (but also larger) formulation by adding certain flows on antiparallel arcs.

Attention: Think carefully about the constraints linking the directed flow variables and the undirected edge variables. You can combine anti-parallel flows for commodities within the same commodity set but not for commodities from different commodity sets.

Solutions:

atlanta	1.6095000000e+07
cost266	1.7801100000e+06
di-yuan	3.5250000000e+05
france	3.8932000000e+04
nobel-eu	1.5270000000e+05
nobel-germany	4.7530000000e+04
nobel-us	1.0530000000e+05
polska	5.2120000000e+03

Exercise 3: The Capacitated Network Design Problem

Formulate and solve the Capacitated Network Design Problem. You should be able to solve the test instance `di-yuan` to optimality and the instances `atlanta`, `polska`, and `nobel*` to an optimality gap of at most 15%.

In this exercise, we consider the undirected problem version with modular edge capacities and edge setup costs:

Capacitated Network Design Problem	
Given	undirected graph $G = (V, E)$ with edge setup costs l_e capacity u_{ek} and cost c_{ek} for all edges $e \in E$ and all capacity modules $k \in K$ undirected commodities $C \subset V^2$ with demands d_{st} for each $(s, t) \in C$
Solution	integers y_{ek} for all $e \in E$ and $k \in K$ such that the edge capacities $x_e := \sum_{k \in K} u_{ek} y_{ek}$ permit a (fractional) multi-commodity flow routing all commodities
Goal	minimize the total edge setup cost plus the total capacity installation cost, i.e. $\min \sum_{e \in E:\, x_e > 0} l_e + \sum_{e \in E} \sum_{k \in K} c_{ek} y_{ek}$

All necessary sets and functions are constructed in `SNDParser.zpl`. The capacity and cost values u_{ek} and c_{ek} can be accessed via the functions $cost(e, k)$ and $cap(e, k)$, respectively.

Hints: Use the model presented in the lecture and add extra binary variables z_e, $e \in E$, expressing whether an edge is set up or not. Add constraints linking these variables either to the capacity installation variables y_{ek} or to the flow variables f. These inequalities must ensure that z_e is set to 1 if at least one capacity unit is installed on e or, alternatively, if some commodity is routed arcoss the corresponding arcs.

You may also aggregate multiple commodities that start from a common source node to a single commodity with one source and multiple targets. Using flow variables only for these aggregate commodities leads to much smaller (but slightly weaker) formulations.

Solutions:

atlanta	1.1717500000e+08
di-yuan	2.4304000000e+06
france	5.8694000000e+04
nobel-eu	1.1014100000e+06 (<3% gap)
nobel-germany	2.2942000000e+05 (<2% gap)
nobel-us	2.6283700000e+06
pdh	1.2435812000e+07
polska	3.1024000000e+04

Exercise 4: The Capacitated Network Design Problem with 1+1 Protection

Formulate and solve the Capacitated Network Design Problem with 1+1 Protection. You should be able to solve the test instance `di-yuan` to optimality and the instances `atlanta`, `polska`, and `nobel*` to an optimality gap of at most 5%.

We consider the undirected problem version with modular edge capacities and edge setup costs and the vertex disjoint variant of 1+1 protection. The problem input and the optimization goal is the same as in the previous exercise. In the vertex disjoint 1+1 protection case, however, each commodity must be routed twice via exactly two node disjoint paths (i.e., each of the two paths carries the full demand of the commodity such that the total demand is routed twice).

Hint: Use the model presented in the lecture and see the first hint in the previous exercise.

Attention: In contrast to the previous exercise, you cannot aggregate commodities with a common source this time!

Solutions:

atlanta	2.8388000000e+08
di-yuan	5.3718000000e+06
france	infeasible
nobel-eu	2.5240200000e+06 (<0.5% gap)
nobel-germany	4.8228000000e+05 (<0.5% gap)
nobel-us	6.8361500000e+06 (<0.5% gap)
pdh	2.8227392000e+07 (<3.5% gap)
polska	6.7808000000e+04 (<0.7% gap)

Challenge: Compute the best possible lower bound for the `cost266.txt` test instance.

Technical Information

In the directory `COatWork-Data/1005` you will find a collection of real world problem instances and utilities to read, solve, and visualize these.

Data/*.txt	Test instances
Data/*.eps	The corresponding Network pictures
SNDParser.zpl	ZIMPL parser for the test instances
scripts/solve.sh	Script to solve the created MILPs and write a special solution file, which can be used to create images
scripts/visualize.sh	Script to create images from the solution file

If you want to "see" the network use the *.eps pictures in the Data folder. These pictures show you the networks, the terminal nodes (big nodes) and the edge-cost $l(e)$. If you want to visualize the size of the emanating demand of every node (the bigger the node, the bigger the demand of the node) instead of the terminals, type:

```
> visualize.sh -o MyImage.eps --show-demands Data/atlanta.txt
```

Use the parser `SNDParser.zpl` to read the test instances as follows:

```
> zimpl -o MyMILP -D file=Data/instance.txt SNDParser.zpl MyModel.zpl
```

The parser `SNDParser.zpl` reads the test instance file and provides sets and functions that you can use in `MyModel.zpl` to formulate the actual integer linear programming model. A documentation of these sets and functions can be found in the beginning of the file `SNDParser.zpl`. The above zimpl command will construct a `MyMILP.lp` file which contains the MIP formulation that can be passed to SCIP to solve the problem.

If you wish to create images of your solutions, **your model must contain a set of variables `x[E]` for the undirected edge set `E` provided by the parser `SNDParser.zpl`**. The values of these variables are used to determine whether an edge is drawn in the solution image or not and also how it is drawn (thick lines correspond to large capacity). In the Steiner Tree and Survivable Network Design Problem, the `x[E]` variables should indicate if an edge is contained in the solution or not. In the Capacitated Network Design Problems, these variables should correspond to the total edge capacity. It does not matter if these variables are binary, general integer, or continuous or if they are primary decision variables or artificial variables used only to represent the total edge capacities.

Note that if your model does NOT contain `x[E]` variables, the only thing you can do is to pass the `MyMILP.lp` file to SCIP

```
> scip -f MyImage.lp
```

and look at the output.

If your model contains `x[E]` variables, you can solve the MILP in `MyMILP.lp` and generate a postscript image of the solution using the scripts `solve.sh` and `visualize.sh` as follows:

```
> solve.sh -o MySol.sol MyMILP.lp -t 50
> visualize.sh -o MyImage.eps Data/instance.txt MySol.sol
```

The script `solve.sh` reads and solves the MILP file `MyMILP.lp` and writes the solution values of variables `x[E]` into the file `MySol.sol`. The option `-t` specifies the time limit in seconds. The script `visualize.sh` reads the original test instance file `Data/instance.txt` together with the solution file `MySol.sol` and creates an EPS image file `MyImage.eps` of the solution. See above on how to visualize the size of the demand at every node.