Lab 1

The purpose of this lab is to formulate and solve three simple problems with AMPL. Two of them have to do with graphs, so first we'll need to define a graph in AMPL. I assume you recall network flow problems and their formulation.

AMPL refresher

AMPL[1] is an algebraic modeling language. Its main use is to model Optimization problems of all sorts: linear, integer linear, nonlinear. There are a number of tutorials and manuals of AMPL on the Internet, so I won't give a detailed description — you should already know how it works if you have taken IE220. Just a quick summary of its keywords:

- `param`: define a parameter (or a vector/matrix of parameters), that is, **an input** of the problem. This is not a result of the Optimization. Examples:
  - `param n := 4; # number of nodes in the graph`
  - `param m default 7; # can be changed, initialized to 7`
  - `param k {1..n} integer >= 2; # specify type and bounds`
  - `param p {S}; # indexed by a set S (defined below)`

- `set`: defines a set of elements. Examples:
  - `set N := 1..n; # all integers between 1 and n`
  - `set cities := easton allentown bethlehem;`
  - `set N2 := N1 union N2; # AMPL has set operators`

- `var`: defines a variable (or a vector/matrix of variables). Examples:
  - `var x >= 1 <= 4; # specifying bounds`
  - `var y binary; # specify type`
  - `var z default 1; # suggest solver initial value`

- the objective function is specified by the keywords `minimize` or `maximize`, a name and an expression:
  `minimize total_cost: x1 + x2;`

- constraints can be defined on sets of indices when we need to define a class of them:
  `constraint1 {i in 1..n}: sum {j in 1..k} x [i,j] = 1;`

- text between "#" and the end of the line is ignored and can be used for comments.

Online solvers

Many good solvers can be downloaded for free, and many of them can be accessed on line. The Neos Server for Optimization

`http://neos.mcs.anl.gov/neos/solvers/index.html`

allows to submit optimization problem in AMPL for several optimizers, including many we will use in the course. Our problems will usually be Mixed-Integer Linear Programming problems, and the NEOS solvers available for them are SCIP, MINTO, and CBC. To use them, we need to have a model file, a data file, and a run file. The first is necessary, the other two can be useful but they are not mandatory.

Click on the desired solver, fill in the files, and click "submit." The subsequent page is the output of the solver: it is updated whenever the solver generates an output. Once the solver is done (that is, the problem is

[1] See `http://www.ampl.com`

solved or the time limit is reached), you will also receive an email if you specified your email address before submitting the job.

There is no need to keep that window open: simply store the job-ID and the password, and you can access the status of the optimization, or stop it, later by typing ID/password at the NEOS prompt. **Important:** once a job is submitted, it occupies computational resources (some of which are here at Lehigh) until it is done. If you are not interested in the results of a certain job, please interrupt it.

Exercise 1: knapsack problem

Solve on NEOS, using one of the solvers for Mixed-Integer Linear Programming (SCIP, MINTO, or CBC), the following Knapsack problem:

$$\begin{aligned} \min\ & 5x_1 + 6x_2 + 4x_3 + 2x_4 + 9x_5 \\ & 3x_1 + 4x_2 + 6x_3 + 5x_4 + 7x_5 \geq 5 \\ & x_i \in \{0,1\} \qquad i = 1,2,3,4,5 \end{aligned}$$

Report the results and the output from NEOS.

Graphs and AMPL

First of all, some info on how to define a graph, directed or nondirected, in AMPL. An **undirected** graph $G = (V, E)$ such as that in Figure 1a can be implemented as follows:

```
param n;
set V := 1..n;
set E within {i in V, j in V: i < j};
# variables, obj and constraints...
data;
param n := 6;
set E := (1,2) (1,4) (1,5) (2,3) (2,4) (3,4) (3,6) (4,5) (5,6);
```

Why the "`i < j`" constraint in defining the edge set E? Because E is formally a set of *subsets*, more specifically a set of subsets of two elements. AMPL may very well handle a set of sets, but we just want to keep it simple: we just assume that a set of two elements can be defined as a *pair* of elements. With "`{i in V, j in V}`", AMPL will consider any pair of elements of V.

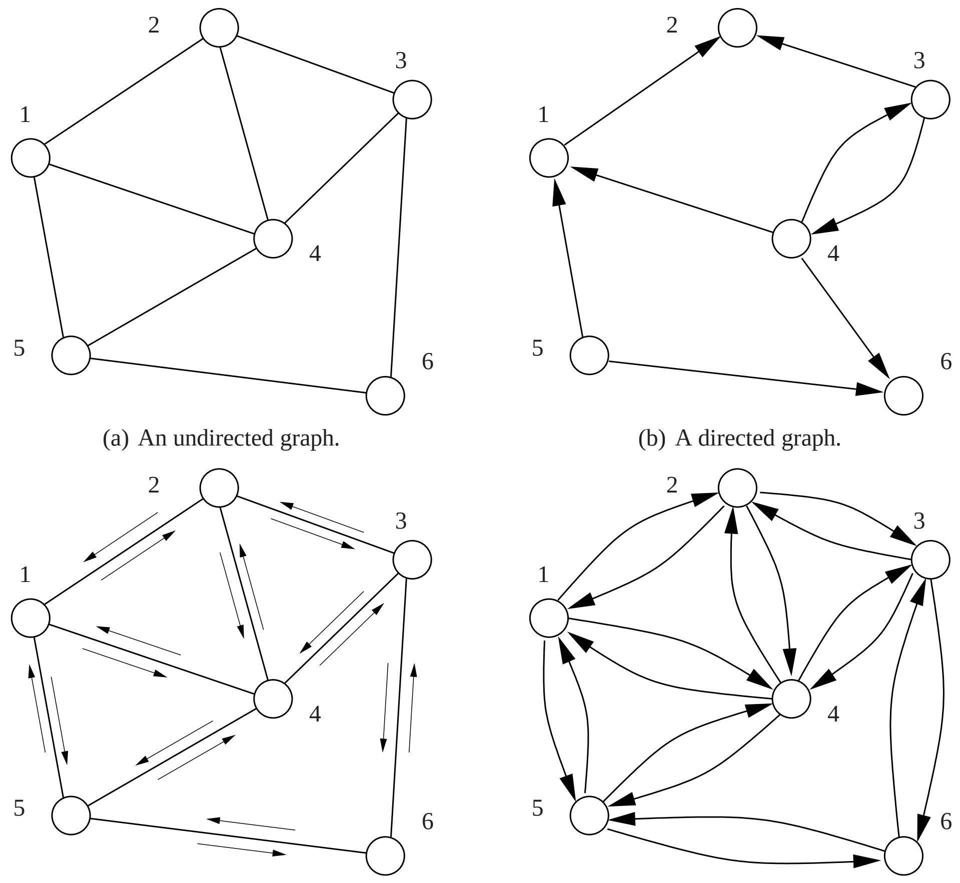
Now, a set of two elements and a pair are very similar, but while the two sets $\{i,j\}$ and $\{j,i\}$ are the same thing, the two pairs (i,j) and (j,i) are **not**. Thus, we assume that a set of two elements is equivalent to a pair with the first element smaller than the second one (a set, for instance, doesn't have a "first" and a "second" element). That is, we impose the constraint "`i < j`" when defining these pairs. When we want to consider edge $\{2,5\}$, we tell AMPL to consider the *pair* `(2,5)`. AMPL will tell you that the pair `(5,2)` doesn't belong to `E`, but that's because we assume that the pair `(5,2)` does *not* correspond to the set $\{2,5\} = \{5,2\}$, while `(2,5)` does.

This tricks AMPL into thinking that `E` is a set of pairs, but it should not trick us. We know that what AMPL sees as the pair "`(2,5)`" is indeed a *set*, $\{2,5\}$, in our paper world. AMPL thinks a subset of E is simply a set of pairs, and that an element of E is a pair. Now, what is the subset F of edges incident[2] to node 3, for example? On a piece of paper, we would write

$$F = \{\{i,j\} \in E : i = 3\},$$

while in AMPL we need to identify the pairs where 3 is either the first or the second element: we can do so by either telling AMPL to take those pairs (i,j) where $i = 3$ or $j = 3$. The following two lines are equivalent:

[2] "incident in i" = "with i as an endnode".



(a) An undirected graph.
(b) A directed graph.
(c) Directions on an undirected graph.
(d) A directed graph.

Fig. 1.

```
set F := {i in V, j in V: (i,j) in E and (i=3 or j=3)};
set F := {(i,j) in E: i=3 or j=3};
```

and AMPL will display it as

```
ampl: display F;
set F := (2,3) (3,4) (3,6);
```

where we know that the first condition (`i=3`) matched the second and third pair, and the second condition (`j=3`) matched the first pair.

Directed graphs

When we have a directed graph, we are more at ease as we don't need to trick AMPL into anything. A directed graph (or digraph) $G = (V, A)$ has a set of nodes and a set A of *arcs*. Each arc is defined as an ordered pair (i, j) where i is the "tail" and j is the "head" of the arc. A digraph $G = (V, A)$ such as the one in Figure 1b can be defined in AMPL similarly to the nondirected graph:

```
param n;
set V := 1..n;
set A within {i in V, j in V: i != j};
# variables, obj and constraints...
```

```
data;
param n := 6;
set A := (1,2) (3,2) (3,4) (4,3) (4,1) (4,6) (5,1) (5,6);
```

Notice that the condition on the pairs changed: it is now "`i != j`", which means i is only required to be different, not smaller, than j. How do we get all nodes such that there is an arc *from* node 3 to them?

```
ampl: set H = {(i,j) in A: i=3};
ampl: display H;
set H := (3,2) (3,4);
```

How about all nodes that have an arc *to* node 3?

```
ampl: set H2 = {(i,j) in A: j=3};
ampl: display H2;
set H2 := (4,3);
```

Directions on an undirected graph

What happens when we have an undirected graph $G = (V, E)$, but we have both variables defined on the edges of E and variables that use the two directions on each edge? That is, we have a graph with edges $\{i, j\}$, but we may define a quantity in the direction $i \to j$ and another in the direction $j \to i$. On paper, that is, from the modeling standpoint, that amounts to defining, for a set E of subsets, a set of pairs A as follows: $A = \{(i, j) : i \in V, j \in V, \{i, j\} \in E\}$. Strange: it seems we have one pair for each edge, and we should have two! Well, for the graph in 1c, A contains pair $(1, 4)$ and $(4, 1)$, because both $\{1, 4\}$ and $\{4, 1\}$ are in E – they are actually the same subset.

We use a similar trick with AMPL. After we define the set of edges with

```
set E within {i in V, j in V: i < j};
```

we define a set `ED` with a pair `(i,j)` and a pair `(j,i)` for each edge `(i,j)`:

```
set ED := {i in V, j in V: ((i,j) in E) or ((j,i) in E)};
```

What's this? It is a set defined on E, so that in the data file we only need to specify E, not `ED`. Also, it contains all pairs `(i,j)` such that either `(i,j)` or `(j,i)` are, according to AMPL, pairs of the set of edges. If we define `ED` after defining `E` in the AMPL code at the beginning of the document, we get:

```
ampl: display ED;
set ED :=
(1,2)  (1,5)  (2,3)  (3,2)  (3,6)  (4,2)  (4,5)  (5,4)  (6,3)
(1,4)  (2,1)  (2,4)  (3,4)  (4,1)  (4,3)  (5,1)  (5,6)  (6,5);
```

which looks like the directed graph in Figure 1d, where each edge $\{i, j\}$ of graph in Fig. 1a is turned into two arcs, (i, j) and (j, i).

The Minimum Spanning Tree (MST) problem

Problem. Given a graph $G = (V, E)$ and a function $c : E \to \mathbb{R}_+$, find the subset S of E that minimizes $\sum_{\{i,j\}\in S} c_{ij}$ and such that for any two nodes k and l there is a path connecting k and l using only nodes in S.

Solution. There are two ways to approach this, as we've seen in class. The first one uses flow variables, the second uses the concept of cut. I will go over the first model only. Let us define binary variables y_{ij}, equal to 1 if $\{i, j\}$ is included in the solution, 0 otherwise. Now we can already write the objective function,

$$\sum_{\{i,j\}\in E} c_{ij} y_{ij},$$

to be minimized. What constraints do we need? We want to make sure that there is, for each pair of nodes k and l, a path from k to l that uses the edges of E for which $y_{ij} = 1$. That is, we could look for that path, for any pair (k, l), and impose that it uses the edges for which $y_{ij} = 1$.

Consider again the graph in Fig. 1, and node pair $(1, 6)$. We don't know what edges will be included in the solution S, but it is necessary that the path from 1 to 6 uses the edges of S. Note that "a path from 1 to 6" and "a path from 6 to 1" are equivalent here, as an edge included in S can be crossed in the two directions on a path between 1 and 6, and therefore on a path from 6 to 1, too.

That allows us to simplify things a little: instead of ensuring, **for any pair** (k, l) of nodes, that a path exists between k and l, we pick a node r (say node 2), that we call *root*, and ensure that, **for any node** k, a path exists between r and k – and that implies that a path also exists between k and r, which is the same from r to k, only in the opposite direction. If we can ensure a path between the root node r and any k, then we can ensure a path between any pair of nodes k and l: simply concatenate the paths $k \to r$ and $r \to l$.

OK, but we still have to ensure that condition. Suppose we need a path from our root node, $r = 2$, to $k = 6$. We could define *flow* variables x_{ij} and x_{ji} for the two directions of an edge $\{i, j\}$ of E. These flow variables are binary as $x_{ij} = 1$ if the path from 2 to 6 uses edge $\{i, j\}$ from i to j, 0 otherwise. Also, since we want it to be a flow from 2 to 6, it should satisfy the flow constraints at 2, i.e., there should be a positive flow balance,

$$\text{(node 2)} \qquad x_{21} + x_{23} + x_{24} - x_{12} - x_{32} - x_{42} = 1$$

and at all intermediate nodes:

$$\begin{array}{lllr}
\text{(node 1)} & x_{12} + x_{14} + x_{15} & -x_{21} - x_{41} - x_{51} & = 0 \\
\text{(node 3)} & x_{32} + x_{34} + x_{36} & -x_{23} - x_{43} - x_{63} & = 0 \\
\text{(node 4)} & x_{41} + x_{42} + x_{43} + x_{45} & -x_{14} - x_{24} - x_{34} - x_{54} & = 0 \\
\text{(node 5)} & x_{51} + x_{54} + x_{56} & -x_{15} - x_{45} - x_{65} & = 0.
\end{array}$$

We don't need to ensure any flow condition at the destination node, 6, because we know that if a flow enters the network at node 2 and conserves at all nodes but 6, it must exit at 6. We can re-write the above constraints as follows:

$$\begin{array}{ll}
\text{(node 2)} & \sum_{j\in V:\{2,j\}\in E} x_{2j} - \sum_{j\in V:\{2,j\}\in E} x_{j2} = 1 \\
\text{(node 1)} & \sum_{j\in V:\{1,j\}\in E} x_{1j} - \sum_{j\in V:\{1,j\}\in E} x_{j1} = 0 \\
\text{(node 3)} & \sum_{j\in V:\{3,j\}\in E} x_{3j} - \sum_{j\in V:\{3,j\}\in E} x_{j3} = 0 \\
\text{(node 4)} & \sum_{j\in V:\{4,j\}\in E} x_{4j} - \sum_{j\in V:\{4,j\}\in E} x_{j4} = 0 \\
\text{(node 5)} & \sum_{j\in V:\{5,j\}\in E} x_{5j} - \sum_{j\in V:\{5,j\}\in E} x_{j5} = 0
\end{array}$$

or even more succintly

$$\begin{array}{llll}
\text{(node 2)} & \sum_{j\in V:\{2,j\}\in E} x_{2j} - \sum_{j\in V:\{2,j\}\in E} x_{j2} & = 1 & \\
\text{(nodes 1,3,4,5)} & \sum_{j\in V:\{i,j\}\in E} x_{ij} - \sum_{j\in V:\{i,j\}\in E} x_{ji} & = 0 & \forall i \in \{1, 3, 4, 5\}
\end{array}$$

This gives a flow from the root node to node 6. We should write similar constraints for a path from the root node to node 5, and then from 2 to 4, from 2 to 3, and from 2 to 1. Let's rewrite the above constraints to reflect that those variables are used for the path to node 6, and not for the other paths:

$$\begin{array}{llll}
\text{(node 2)} & \sum_{j\in V:\{2,j\}\in E} x^6_{2j} - \sum_{j\in V:\{2,j\}\in E} x^6_{j2} & = 1 & \\
\text{(nodes 1,3,4,5)} & \sum_{j\in V:\{i,j\}\in E} x^6_{ij} - \sum_{j\in V:\{i,j\}\in E} x^6_{ji} & = 0 & \forall i \in \{1, 3, 4, 5\}
\end{array}$$

Now let's write constraints for the path from the root node, 2, to node 4:

$$\begin{array}{llll}
\text{(node 2)} & \sum_{j\in V:\{2,j\}\in E} x^4_{2j} - \sum_{j\in V:\{2,j\}\in E} x^4_{j2} & = 1 & \\
\text{(nodes 1,3,5,6)} & \sum_{j\in V:\{i,j\}\in E} x^4_{ij} - \sum_{j\in V:\{i,j\}\in E} x^4_{ji} & = 0 & \forall i \in \{1, 3, 5, 6\}
\end{array}$$

and for the remaining nodes other than the root, i.e., 1, 3, and 5:

$$\begin{array}{lll}
\text{(node 2)} & \sum_{j\in V:\{2,j\}\in E} x^1_{2j} - \sum_{j\in V:\{2,j\}\in E} x^1_{j2} = 1 & \\
\text{(nodes 3,4,5,6)} & \sum_{j\in V:\{i,j\}\in E} x^1_{ij} - \sum_{j\in V:\{i,j\}\in E} x^1_{ji} = 0 & \forall i \in \{3,4,5,6\} \\
\\
\text{(node 2)} & \sum_{j\in V:\{2,j\}\in E} x^3_{2j} - \sum_{j\in V:\{2,j\}\in E} x^3_{j2} = 1 & \\
\text{(nodes 1,4,5,6)} & \sum_{j\in V:\{i,j\}\in E} x^3_{ij} - \sum_{j\in V:\{i,j\}\in E} x^3_{ji} = 0 & \forall i \in \{1,4,5,6\} \\
\\
\text{(node 2)} & \sum_{j\in V:\{2,j\}\in E} x^5_{2j} - \sum_{j\in V:\{2,j\}\in E} x^5_{j2} = 1 & \\
\text{(nodes 1,3,4,5)} & \sum_{j\in V:\{i,j\}\in E} x^5_{ij} - \sum_{j\in V:\{i,j\}\in E} x^5_{ji} = 0 & \forall i \in \{1,3,4,5\}
\end{array}$$

It actually makes sense to rewrite all constraints above as follows:

$$\begin{array}{lll}
\text{(node 2)} & \sum_{j\in V:\{2,j\}\in E} x^k_{2j} - \sum_{j\in V:\{2,j\}\in E} x^k_{j2} = 1 & \forall k \in \{1,3,4,5,6\} = V \setminus \{2\} \\
\text{(nodes} \neq k) & \sum_{j\in V:\{i,j\}\in E} x^k_{ij} - \sum_{j\in V:\{i,j\}\in E} x^k_{ji} = 0 & \forall i \in V : 2 \neq i \neq k,\ \ \forall k \in V \setminus \{2\}
\end{array}$$

Well, now that we have a ton of x^k_{ij} what do we do with them? We said that there can be a flow x^k_{ij} on edge $\{i,j\}$ from i to j on its path to k only if the relative $y_{ij} = 1$. That means

$$\begin{array}{lll}
x^k_{ij} \leq y_{ij} & \forall \{i,j\} \in E, \forall k \in V \setminus \{k\} & \text{(direction from } i \text{ to } j) \\
x^k_{ji} \leq y_{ij} & \forall \{i,j\} \in E, \forall k \in V \setminus \{k\} & \text{(direction from } i \text{ to } j)
\end{array}$$

Phew! The final model is (let's replace node 2 with the more general r):

$$\begin{array}{ll}
\min \sum_{\{i,j\}\in E} c_{ij} y_{ij} & \\
\sum_{j\in V:\{r,j\}\in E} x^k_{rj} - \sum_{j\in V:\{r,j\}\in E} x^k_{jr} = 1 & \forall k \in V \setminus \{r\} \\
\sum_{j\in V:\{i,j\}\in E} x^k_{ij} - \sum_{j\in V:\{i,j\}\in E} x^k_{ji} = 0 & \forall i \in V : r \neq i \neq k,\ \ \forall k \in V \setminus \{r\} \\
x^k_{ij} \leq y_{ij} & \forall \{i,j\} \in E, \forall k \in V \setminus \{r\} \\
x^k_{ji} \leq y_{ij} & \forall \{i,j\} \in E, \forall k \in V \setminus \{r\} \\
x^k_{ij}, x^k_{ji} \in \{0,1\} & \forall \{i,j\} \in E, \forall k \in V \setminus \{r\} \\
y_{ij} \in \{0,1\} & \forall \{i,j\} \in E.
\end{array}$$

How do we implement this stuff in AMPL? Easy, Section 1 gives some hints on all of this. Adding an index to a formula is as easy as adding a set to the definition of a variable in AMPL.

Exercise 2: the shortest path problem

Model and solve the problem of finding the shortest path on the graph of Figure 1a from node 1 to node 6. Solve it on Neos using one of the solvers SCIP, MINTO, or CBC. Use random (positive) weights for the edges.

Exercise 3: the minimum spanning tree problem

Model and solve the problem of finding the minimum spanning tree on the graph of Figure 1a. Solve it on Neos using one of the solvers SCIP, MINTO, or CBC. Use random (positive) weights for the edges.

Lab deliverables: email all files for exercises 1,2,3, and their results, to `belotti@lehigh.edu`. Exercises 1 and 2 should be sent by the end of the lab, and exercise 3 by next Tuesday afternoon (6pm).