AMPL, TSP, and MINTO-AMPL

Jeff Linderoth

June 22, 2004
Outline

- AMPL
- Traveling Salesperson Problem
- Using MINTO with AMPL
  - A (fairly complicated) example – the TSP with AMPL and MINTO
• AMPL is an Algebraic Modeling Language

• In many ways, AMPL is like any other programming language.
  
  ◊ It just has special syntax that helps us create an optimization instance and interact with optimization solvers.

• AMPL is a *very* useful tool for building and solving optimization instances, but it is not too user friendly!
An engineering plant can produce five types of products: \( p_1, p_2, \ldots, p_5 \) by using two production processes: grinding and drilling. Each product requires the following number of hours of each process, and contributes the following amount (in hundreds of dollars) to the net total profit.

<table>
<thead>
<tr>
<th></th>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_3 )</th>
<th>( p_4 )</th>
<th>( p_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grinding</td>
<td>12</td>
<td>20</td>
<td>0</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>Drilling</td>
<td>10</td>
<td>8</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Profit</td>
<td>55</td>
<td>60</td>
<td>35</td>
<td>40</td>
<td>20</td>
</tr>
</tbody>
</table>
PPP – More Info

- Each unit of each product take 20 manhours for “final assembly”.

- The factory has three grinding machines and two drilling machines.

- The factory works a six day week with two shifts of 8 hours/day. Eight workers are employed in assembly, each working one shift per day.

- $x_i$: The number of product $p_i$ to make in a week.
Constraints

- Grinding...
  - 3 machines. 16 hours/day. 6 days/week.
- Get the Units right...
- 288 grinding hours available per week.
  - 3 machines * 16 grinding hours/(machine*day) * 6 days/week = 288 grinding hours/week.

\[12x_1 + 20x_2 + 0x_3 + 25x_4 + 15x_5 \leq 288\]

- LHS: Grinding hours in production plan per week
- RHS: Total grinding hours available per week.
More Constraints...

- Drilling
  - $10x_1 + 8x_2 + 16x_3 + 0x_4 + 0x_5 \leq 2 \times 16 \times 6 = 192$

- Finishing Labor
  - 8 Assembly workers, each working 48 hours/week.
  - $20x_1 + 20x_2 + 20x_3 + 20x_4 + 20x_5 \leq 8 \times 48 = 384$

- The Laws of Nature
  - $x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0$. 

June 22, 2004 
DIMACS Reconnect Conference on MIP 
Slide 7
Final Problem

maximize

\[55x_1 + 60x_2 + 350x_3 + 40x_4 + 20x_5\] (Profit/week)

subject to

\[12x_1 + 20x_2 + 0x_3 + 25x_4 + 15x_5 \leq 288\]
\[10x_1 + 8x_2 + 16x_3 + 0x_4 + 0x_5 \leq 192\]
\[0x_1 + 20x_2 + 20x_3 + 20x_4 + 20x_5 \leq 384\]

\[x_i \geq 0 \quad \forall i = 1, 2, \ldots 5\]

★ AMPL Interactive Portion
Generalizing the Model

- Suppose we want to generalize the model to more (or less) than five products.
- Suppose we wanted to have more than three resources constraining us?
- Suppose we wanted to change certain parameters associated with the model?
  - AMPL (and all “real” modeling environments) allow the model to be separated from the data.
  - This is IMPORTANT!!!
General PPP Model

- **Sets**
  - \( P \): Set of products to be made
  - \( R \): Set of resources available (constraining our production)

- **Parameters**
  - \( c_p \): Net profit of producing one unit of product \( p \) (\( \forall p \in P \))
  - \( b_r \): Amount of resource \( r \) available (\( \forall r \in R \))

- **Variables**
  - \( x_p \): Amount of product \( p \) to produce (\( \forall p \in P \))
AMPL Entities

- Data
  - Sets: lists of products, materials, etc.
  - Parameters: numerical inputs such as costs, etc.

- Model
  - Variables: The values to be decided upon.
  - Objective Function.
  - Constraints.

- These are usually stored in different files.

★ AMPL Interactive Portion
An AMPL Template

- Define Sets
- Define Parameters
- Define Variables
  - Also can define variable bound constraints in this section
- Define Objective
  - Define Constraints
Important AMPL Keywords/Syntax

- model file.mod;
- data file.mod;
- reset;
- quit;
- write mfile

- set
- param
- var
- maximize (minimize)
- subject to
Important AMPL Notes

- The # character starts a comment
- All statements must end in a semi-colon;
- Names must be unique!
  - A variable and a constraint cannot have the same name
- AMPL is case sensitive. Keywords must be in lower case.
- Even if the AMPL error message is cryptic, look at the location where it shows an error – this will often help you deduce what is wrong.
- See papers/ampl1.pdf for a short introduction to AMPL.
- I also have brought a couple AMPL books for us to use
MINTO-AMPL Interface

- In directory minto31-linux-osiclp/APPL-ampl, there is the code for the minto-ampl interface
- If you build the executable here (mintoamp), this will be a solver you can use with AMPL
- Important options include
  - option solver mintoamp
  - option mintoamp_options 'loadnames 1 deactivate_all 1'
  - option mintoamp_auxfiles rc
\[ \min \sum_{e \in E} c_e x_e \]

\[ \sum_{e \in \delta(v)} x_e \leq 2 \quad \forall v \in V \]

\[ \sum_{e \in E(S)} x_e \leq |S| - 1 \quad \forall S \subset V \text{ with } |S| \geq 3, |S| \leq |V| - 1 \]

\[ x \in \{0, 1\}^{|E|} \]

\[ \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V \text{ with } |S| \geq 3, |S| \leq |V| - 1 \]
AMPL does not (as far as I know) allow you to easily write “recursive” constraints like the subtour elimination constraints.

There are probably way, way, way too many of them anyway.

We will do an example where we solve an instance of the TSP using MINTO, where the problem (without the integrality constraints) is written in AMPL.

This will also be “an exercise” in the lab.

How do we separate?
• Given \( x^* \in \mathbb{R}^{|E|} \), does there exist \( S \subseteq V \) such that \( \sum_{e \in S} x^*_e < 2 \)

• for each \( e = (u, v) \in E \) do
  
  ◦ compute a minimum \((u, v)\) cut \( S^* \) with respect to weights \( x^*_e \) (\( u \in S^*, v \in V \setminus S^* \))

• If \( \sum_{e \in S^*} x^*_e < 2 \)

  ⇒ You have a cut.

• Add \( \sum_{e \in E(S^*)} x_e \leq |S^*| - 1 \)
The mapping of “AMPL” variables to “MINTO” variables is done through the names of the variables.

(This is why loadnames 1 and auxfies rc) are important.