MS&E 212
Network and Integer Programming
Prof. Yinyu Ye

AMPL Handout

PART I

What is AMPL?

There are many, many different algorithms (solvers) out there that solve mathematical programs. Each of these algorithms require the input (our mathematical model) to be in a different format.

AMPL is a platform that allows us to express our mathematical models in a standardized form. Depending on what solver we choose to use, AMPL then converts our model into one that can be read by the solver. After the solver solves the model, it sends its results back to AMPL, which in turn shows it to us in a standardized form.

For our purposes, it is enough to think of AMPL as a software that solves your mathematical models.

Why AMPL instead of Excel?

A couple of reasons:
1) AMPL has the capability of solving much larger problems than Excel.
2) The input format for a math program in AMPL is more consistent with the way we express it on paper.

Wow, where can I find AMPL?

AMPL is available on the Sun workstations on the 2nd floor of Sweet Hall. These are UNIX-based workstations, and you will need to know at least some basic UNIX commands. We recommend that you consult the following webpage.

http://www.stanford.edu/group/dcg/leland-docs/unixcomm.html

Ok, I' m sitting in front of the workstation, how do I start AMPL?

Log in with your SUNet ID. Start X-Windows. Type "ampl". Enter.
You will see a prompt that looks like this:

```
ampl:
```

How does AMPL work?

Consult Part II of this handout.

PART II

Whenever we want to solve a mathematical model, we should think about the following questions –

1) What is the model?
2) What is the data for this model?
3) How do I want to solve it?
4) What is the output?

We usually put our model in a *filename.mod* file and our data in a *filename.dat* file. We will now give examples called *example.mod* and *example.dat*.

What does example.mod look like?

```
param T > 0;                        # No of periods
param M > 0;                        # Large number

param fixedcost {2..T+1} >= 0;    # Fixed cost of production in period t
param prodcost {2..T+1} >= 0;     # Production cost of production in period t
param storcost {2..T+1} >= 0;     # Storage cost of production in period t
param demand {2..T+1} >= 0;       # Demand in period t

var made {2..T+1}   >= 0;         # Units Produced in period t
var stock {1..T+1} >= 0;          # Units Stored at end of t
var decide {2..T+1} binary;       # Decision to produce

minimize total_cost:
   sum {t in 2..T+1} (prodcost[t] * made[t] +fixedcost[t] * decide[t] +
              storcost[t] * stock[t]);
subject to C:
       stock[1] = 0;

subject to A {t in 2..T+1}:
    stock[t-1] + made[t] =  demand[t] + stock[t];

subject to B {t in 2..T+1}:
    made[t]  <=  M * decide[t];
```

Ok, what' s it for?

Example.mod tells AMPL what our mathematical program is.

What does # mean?

Anything after a # is a comment and will not be read by AMPL.

What' s the ; for?

Every statement in AMPL should end with a semicolon.

How do we declare the parameters?

To tell AMPL that T, for example, is a parameter, we write

```
param T;
```

By saying

```
param T>0;
```

we are telling AMPL that T is a parameter that is greater than 0.

Two lines down in the code, we see

```
param fixedcost {2..T+1} >= 0;
```

This is telling AMPL that *fixedcost* is a parameter that is indexed from 2 to T+1, all of which are greater or equal to zero.

Note that we do not state what the values for these parameters are in *example.mod*. We are just saying "*fixedcost* is a parameter, treat it as a parameter next time you see it".

How do we declare the variables?

In the code, we had

```
var made {2..T+1}   >= 0;
```

This says "Let $made_i$ be a nonnegative variable, where i = 2, …. T+1"

Again,

```
var decide {2..T+1} binary;
```

declares $decide_i$ as a binary variable that takes on values of only 0 or 1.

And the objective function?

```
minimize total_cost:
   sum {t in 2..T+1} (prodcost[t] * made[t] +fixedcost[t] * decide[t] +
              storcost[t] * stock[t]);
```

Here, we are saying that the objective function is called total_cost, and it's defined as

$$\text{minmize} \quad \sum_{t=2}^{T+1} prod\,\text{cost}_t \cdot made_t + fixed\,\text{cost}_t \cdot decide_t + stor\,\text{cost}_t \cdot stock_t$$

Take note of how we express a summation in AMPL.

And, oh, constraints?

```
subject to A {t in 2..T+1}:
    stock[t-1] + made[t] =  demand[t] + stock[t];
```

We named this constraint A, and it is equivalent to

$$stock_{t-1} + made_t = demand_t + stock_t \qquad t = 2,....T+1$$

Note that this one line is enough to express not one, but T-1 constraints, since this constraint has to hold for all t from 2 to T+1.

What does the example.dat file look like?

```
param T:= 6;

param demand :=
2 6
3 7
4 4
5 6
6 3
7 8;
param prodcost :=
2 3
3 4
4 3
5 4
6 4
7 5;
param storcost :=
2 1
3 1
4 1
5 1
6 1
7 1;
param fixedcost :=
2 12
3 15
4 30
5 23
```

```
6 19
7 45;

param M := 10;
```

What' s this for?

This file contains the values of all the parameters that we declared in *example.mod*. Keeping the data file and mod file separate is convenient when we want to run the same model with different data.

How do I read this?

Note that instead of just a equality sign, AMPL uses a colon followed by a equal sign.

Remember that the parameters *demand, prodcost, fixedcost*, and *storcost* are all indexed from 2 to T+1. Because T was set to be 6 in the first line, the parameter *demand* etc, are indexed from 2 to 7.

In the code above, $\text{demand}_2 = 6$ and $\text{demand}_3 = 7$.

Ok, I' m set. How do I solve this?

Now that we have our example.mod and example.dat file, we can go ahead and solve our model by typing the following three lines at the ampl: prompt. Don' t forget the semicolons.

```
ampl: model example.mod;
ampl: data example.dat;
ampl: solve;
```

The first line tells AMPL that this is the model we want to solve. The second line tells AMPL "here' s the data for our model". And the third line says "go solve it".

Nice. How do I read the results?

Use the "display" command. To see the values for variable made, for example, type

```
display made;
```

How do I quit AMPL?

```
quit;
```

PART III MISC STUFF

The files *example.mod* and *example.dat* can be downloaded from the following links:-

www.stanford.edu/class/msande212/example.dat
www.stanford.edu/class/msande212/example.mod

We recommend that you try running this example yourself. You will be required to use AMPL in your homework and projects.

Is this enough?

Not quite, but it is a good start. We recommend that you download and read the following file:-

www.ampl.com/BOOK/ch1-2.pdf

Pay close attention to the use of sets and the syntax for two-dimensional variables. These are not covered in this handout, but will be useful when you are doing your homework.

Think about how to rewrite *example.mod* using sets.

How do I edit a file in UNIX?

Pico is probably the most user-friendly editor available. To edit *example.mod*, just type

```
pico example.mod
```

at the UNIX prompt. Make sure you are not in AMPL first, though.

Er... could I get some help?

If you have any questions or problems concerning UNIX, AMPL or anything else, feel free to stop by Terman 481 and talk to your TA, Charles Ng.