Advanced Operations Research Techniques
IE316

Lecture 10

Dr. Ted Ralphs

Reading for This Lecture

- AMPL Book: Chapter 1
- AMPL: A Mathematical Programming Language

Software for Mathematical Programs

- So far, we have seen how to solve linear programs by hand.
- In practice, most people use commercial software.
- All commercial solvers have some version of the simplex method, along with some other methods.
- Some commercial solvers
  - CPLEX
  - XPRESS-MP
  - MINOS
- Question: How do we tell the solver what the linear program is?

Inputting the Model Directly

- One possible approach:
  - Formulate the model.
  - Generate the constraint matrix for your instance and data.
  - Type the entire constraint matrix into a file using a standard format.
  - Pass the file to a solver.
  - Get the answer and interpret it in terms of the original model.
- Problems with this approach:
  - The constraint matrices can be huge.
  - It is extremely tedious to type them in.
  - If you want to modify the model parameters or data, you have to retype the entire matrix.
  - Different solvers accept different file formats.
- Ugly!

A Better Approach: Modeling Languages

- Modeling languages provide an interface between the user and the solver.
- They allow the user to input the model in a "natural" format.
- They make it easy to modify parameters and data.
- They can work with multiple solvers.
- Much better!
- Modeling languages
  - GAMS
  - LINGO
  - MPL
  - AMPL

AMPL and CPLEX

- The most pervasive modeling language is AMPL and the most pervasive solver is CPLEX.
- For the remainder of the course, we will use these two software packages frequently.
- Student versions can be downloaded from `www.ampl.com`.
- They can also be used over the Web.

AMPL Concepts

- In many ways, AMPL is like any other programming language.
- Example: A simple product mix problem.

```
ampl: option solver cplex;
ampl: var X1;
ampl: var X2;
ampl: maximize profit: 3*X1 + 3*X2;
ampl: subject to hours: 3*X1 + 4*X2 <= 120000;
ampl: subject to cash: 3*X1 + 2*X2 <= 90000;
ampl: subject to X1_limit: X1 >= 0;
ampl: subject to X2_limit: X2 >= 0;
ampl: solve;
CPLEX 7.1.0: optimal solution; objective 105000
2 simplex iterations (0 in phase I)
ampl: display X1;
X1 = 20000
ampl: display X2;
X2 = 15000
```

Storing Commands in a File

- You can type the commands into a file and then load them.
- This makes it easy to modify your model later.
- Example:

```
ampl: option solver cplex;
ampl: model simple.mod;
ampl: solve;
CPLEX 7.1.0: optimal solution; objective 105000
2 simplex iterations (0 in phase I)
ampl: display X1;
X1 = 20000
ampl: display X2;
X2 = 15000
```

Generalizing the Model

- Suppose we want to generalize this production model to more than two products.
- AMPL allows the model to be separated from the data.
- Components of a linear program in AMPL
  - Data
    - Sets: lists of products, raw materials, etc.
    - Parameters: numerical inputs such as costs, production rates, etc.
  - Model
    - Variables: The values to be decided upon.
    - Objective Function: A function to maximized or minimized.
    - Constraints: Functions that must lie within given bounds.

Example: Production Model

```
set prd;                          # products

param price {prd};                # selling price
param cost {prd};                 # cost per unit for raw material
param hours {prd};                # hours of machine to produce
param max_cash;                   # total cash available
param max_prd;                    # total production hours available

var make {prd} >= 0;              # number of units to manufacture

maximize profit: sum{i in prd} (price[i]-cost[i])*make[i];

subject to hours: sum{i in prd} hours[i]*make[i] <= max_prd;

subject to cash: sum{i in prd} cost[i]*make[i] <= max_cash;
```

Example: Production Model Data

```
set prd := widgets gadgets;

param max_prd := 120000;
param max_cash := 90000;

param:          price    cost    hours :=
      widgets     6        3       3
      gadgets     5        2       4;
```

Solving the Production Model

```
ampl: option solver cplex;
ampl: model prod.mod;
ampl: data prod.dat;
ampl: solve;
ampl: solve;
CPLEX 7.1.0: optimal solution; objective 105000
2 simplex iterations (0 in phase I)
ampl: display make;
make [*] :=
gadgets  15000
widgets  20000
;
```

Changing the Parameters

- Suppose we want to increase available production hours by 2000.
- To resolve from scratch, simply modify the data file and reload.

```
ampl: reset data;
ampl: data prod.dat;
ampl: solve;
ampl: solve;
CPLEX 7.1.0: optimal solution; objective 106000
2 simplex iterations (0 in phase I)
ampl: display make;
make [*] :=
gadgets  16000
widgets  19333.3
;
```

Retaining the Current Basis

- Instead of resetting all the data, you can modify one element.

```
ampl: reset data max_prd;
ampl: data;
ampl data: param max_prd := 122000;
ampl data: solve;
CPLEX 7.1.0: optimal solution; objective 106000
0 simplex iterations (0 in phase I)
ampl: display make;
make [*] :=
gadgets  16000
widgets  19333.3
;
```

- Notice that the basis was retained.

Extending the Model

- Now suppose we want to add another product.

```
set prd := widgets gadgets watchamacallits;

param max_prd := 120000;
param max_cash := 90000;

param:              price    cost    hours :=
      widgets         6        3       3
      gadgets         5        2       4
  watchamacallits     4        1       3;
```

Solving the Extended Model

```
ampl: reset data;
ampl: data prod2.dat;
ampl: solve;
CPLEX 7.1.0: optimal solution; objective 120000
2 simplex iterations (0 in phase I)
ampl: display make;
make [*] :=
        gadgets      0
watchamacallits  15000
        widgets  25000
;
```

Indexing Constraints

Now we're going to add multiple machine types.

```
set prd;                              # products
set mach;                             # machine types
param price {prd};                    # selling price
param cost {prd};                     # cost of raw materials
param hours {prd, mach};              # hours by product and machine type
param max_cash;                       # total cash available
param max_prd {mach};                 # total production hours by machine
var make {prd} >= 0;                  # number of units to manufacture

maximize profit : sum {i in prd} (price[i] - cost[i]) * make[i];

subject to hours_limit {j in mach} :
sum {i in prd} hours[i,j]*make[i] <= max_prd[j];

subject to cash_limit :
sum {i in prd} cost[i]*make[i] <= max_cash;
```

Solving the New Model

```
ampl: model mmprod.mod;
ampl: data mmprod.dat
ampl: solve;
CPLEX 7.1.0: optimal solution; objective 90000
2 simplex iterations (0 in phase I)
ampl: display make
ampl? ;
make [*] :=
gadgets  10000
widgets  20000
;
```