IMSE 500
Models of Operations Research

ASSIGNMENT #3

Readings: *AMPL: A Modeling Language for Mathematical Programming,* chapters 3–4.

1 Consider how you would formulate a linear programming model for a distribution problem. Using AMPL, the model could begin by declaring the relevant sets,

```
set FACT;   # factories
set CUST;   # customers
```

and could continue by declaring the operating data indexed over these sets:

```
param supply {FACT} >= 0;       # supplies
param demand {CUST} >= 0;       # demands
param limit {FACT,CUST} >= 0;  # shipment limits
```

The costs and decision variables would be

```
param cost {FACT,CUST} >= 0;   # shipment cost per unit
var Ship {FACT,CUST} >= 0;     # units to be shipped
```

In terms of the sets, parameters and variables declared above, write out AMPL formulations for each of the following.

a: The objective of minimizing total shipping costs:

```
minimize Total_Cost: sum {f in FACT, c in CUST} ...  ;
```

b: The constraints that the total tons of the product shipped out of each factory *must not exceed* the tons of the product available at that factory:

```
subject to Supply {f in FACT}: ...  ;
```

c: The constraints that the total tons of the product shipped to each customer *must equal* the tons of the product required by that customer:

```
subject to Demand ...  ;
```

d: The constraints that the total tons shipped from each factory to each customer *must not exceed* the capacity of the route from that factory to that customer:

```
subject to Capacity ...  ;
```

e: If there are 2 factories and 3 customers, then how many constraints are formulated in (**b**), in (**c**), and in (**d**)?

If there are 4 factories and 27 customers, then what is the *total* number of constraints formulated?

f: Here's a simple collection of data for this model, in AMPL format. The indexing sets are:

```
set FACT := F1, F2 ;
set CUST := C1, C2, C3 ;
```

The costs can be given in a table like this,

```
param cost:  C1   C2   C3 :=
   F1        40   10   12
   F2        30    5    8 ;
```

and the operational data can be written as

```
param supply :=
   F1   525
   F2   775 ;

param limit default 525;
```

All that's missing here is the table for the demands. Customers `C1`, `C2`, and `C3` have requirements of 450, 575, and 250 tons, respectively. How should the `demand` table be written in AMPL?

g: Type your completed model into a file called `dist1.mod`, and your completed data into a file called `dist1.dat`.

Use `model` and `data` statements to read these files into AMPL, and then a `solve` statement to compute the optimal solution. Use `display` statements to help you answer the following questions:

- How much should be shipped from each factory to each customer?
- How much is left unshipped at each factory?

2

Now consider a *multicommodity* version of the transportation model in problem 1. Suppose that rather than one product, there are three different products: *bands*, *coils*, and *plate*. Each factory has available, and each customer requires, the following tonnages of each product:

		bands	*coils*	*plate*
Supply:	$F1$	250	200	75
	$F2$	100	625	50
Demand:	$C1$	150	250	50
	$C2$	150	375	50
	$C3$	50	200	0

The problem is now to determine how much *of each product* to ship from each factory to each customer. The shipping limit on each route now means that at most 525 *total* tons of product can be shipped from any one factory to any one customer. All products sent from a given factory to a given customer continue to have the same shipping cost per ton, as given in problem 1.

To make a new model file `dist2.mod` that will handle this multicommodity problem, start with the model we called `dist1.mod` in problem 1. Declare a new set

```
set PROD;   # products
```

and alter the `param supply` and `param demand` declarations so that these parameters are also indexed over the set `PROD`:

```
param supply {FACT,PROD} >= 0;
param demand {CUST,PROD} >= 0;
```

Similarly, alter the `var Ship` declaration so that the variables are indexed over `FACT`, `CUST`, and `PROD`. Now write the objective and constraints of the AMPL formulation:

a: The objective of minimizing total shipping costs:

```
minimize Total_Cost:
   sum {f in FACT, c in CUST, p in PROD} ...  ;
```

b: The constraints that the total tons *of each product* shipped out of each factory must not exceed the tons of that product available at that factory:

```
subject to Supply {f in FACT, p in PROD}: ...  ;
```

c: The constraints that the total tons *of each product* shipped to each customer must equal the tons of that product required by that customer:

```
subject to Demand ...  ;
```

d: The constraints that the total tons *of all products* shipped from each factory to each customer must not exceed the capacity of the route from that factory to that customer:

```
subject to Capacity {f in FACT, c in CUST}:
   sum {p in PROD} ...  ;
```

e: If there are 2 factories, 3 customers, and 5 products, then how many constraints are formulated in (**b**), in (**c**), and in (**d**)?

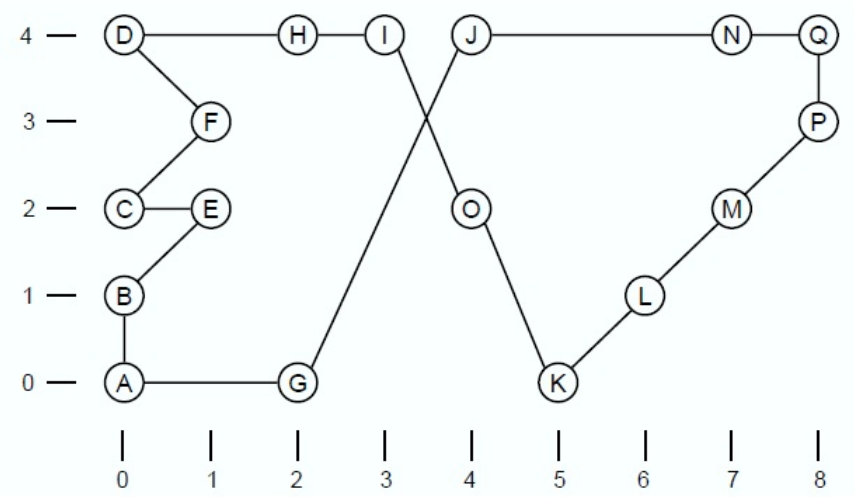
If there are 4 factories, 27 customers, and 50 products, then what is the *total* number of constraints formulated?

f: Create the data file `dist2.dat` for this multicommodity model. Use AMPL to find the optimal solution. What is the optimal number of tons *of each product* to ship from each factory to each customer?

How much *of each product* is left unshipped at each factory?

g: In the optimal solutions, the total multicommodity shipping cost from 2(**f**) is higher than the total single-product shipping cost from 1(**g**), even though the total product supply at each factory and the total product demand of each customer are the same in both cases. Explain in a few sentences why the optimal cost is now higher.

3 Consider the programming of a machine to repeatedly drill a set of holes in the following pattern: To complete its work, the machine follows a path that visits



each hole once before returning to the starting point. The lines in our diagram show one such "tour"; but the machine could work faster if a shorter tour could be found.

A copy of this hole diagram is posted for use in making the diagrams requested by this and the next problem.

a: What is the length of the tour shown in the diagram above?

b: Using pencil and paper, try to find as short a tour as you can. Draw a diagram of your tour (like the one above), and compute your tour's length.

c: The *nearest neighbor method* is a simple heuristic for generating (hopefully) reasonable tours. It starts at any one hole, and traces out a tour by traveling from each hole to its nearest neighbor not yet visited. If a tie for nearest neighbor is encountered, one of the nearest neighbors is chosen by some arbitrary rule. When all holes have been visited, the tour returns to its start.

Find the tours that are generated by the nearest neighbor method, starting from

- hole O
- hole E

Break ties by picking the neighbor whose label is *latest* in the alphabet. Draw a diagram of each of these tours, and compute their lengths.

d: The *2-opt procedure* improves tours found by any other method, by exchanging pairs of tour segments. Specifically, for any pair of moves i_1–i_2 and j_1–j_2 on the current tour, it replaces them with i_1–j_2 and j_1–i_2 if the result is a shorter tour. For example, in the diagram above, moves G–J and I–O can be replaced with G–O and I–J to give a shorter tour.

Find a second 2-opt improvement that can be made in the diagram above.

4 We can do better by setting up and solving a linear program. We start with a set of holes, and the horizontal and vertical coordinates of each:

```
set HOLES ordered;
param hpos {HOLES} >= 0;
param vpos {HOLES} >= 0;
```

Given this information, we can define the set of all unique pairs of holes, and the distance between each pair:

```
set PAIRS := {i in HOLES, j in HOLES: ord(i) < ord(j)};

param dist {(i,j) in PAIRS}
   := sqrt((hpos[j]-hpos[i])**2 + (vpos[j]-vpos[i])**2);
```

Let us define variables `Use`, indexed over the pairs of holes, with the idea that `Use[i,j]` will be 1 if the move between `i` to `j` is used as part of the tour, and that `Use[i,j]` will be zero otherwise. Then it is easy to see that the total length of the tour is the sum of `dist[i,j] * Use[i,j]` over all pairs of holes `i` and `j`. Since we want the tour to be as short as possible, we have:

```
var Use {PAIRS} >= 0, <= 1;

minimize Tour_Length: sum {(i,j) in PAIRS} dist[i,j] * Use[i,j];
```

a: To force the variables `Use[i,j]` to correspond to a tour, we propose to add the following constraints:

```
subject to Visit_All {i in HOLES}:
   sum {(i,j) in PAIRS} Use[i,j] + sum {(j,i) in PAIRS} Use[j,i] = 2;
```

Briefly explain why, if the values of the `Use[i,j]` variables do not satisfy these constraints, then they cannot possibly describe a tour.

b: For the particular holes shown in our diagram, the data values are as follows:

```
param: HOLES: hpos vpos :=
         A   0  0      I   3  4
         B   0  1      J   4  4
         C   0  2      K   5  0
         D   0  4      L   6  1
         E   1  2      M   7  2
         F   1  3      N   7  4
         G   2  0      O   4  2
         H   2  4      P   8  3
                       Q   8  4 ;
```

The model from (**a**) together with this data are posted in files `holes.mod` and `holes.dat`. Solve the resulting linear program. (Be sure you get the `optimal solution` message!) Use the following commands to get a concise listing of the solution:

```
option display_eps .000001;
option omit_zero_rows 1;
display Use;
```

Draw a diagram of the solution, with a solid line showing each variable that is 1, and a dashed line showing each variable that is ½. Why is this solution *not* useful for the hole-drilling problem?

c: To try to make the linear program more useful, you can add "subtour elimination" constraints, which say that at least two moves must connect the nodes in any subset to the nodes not in the subset. Constraints of this kind can be written in AMPL as follows:

```
param nSubtours >= 0 integer;
set SUB {1..nSubtours} within HOLES;

subject to Subtour_Elimination {k in 1..nSubtours}:
   sum {i in SUB[k], j in HOLES diff SUB[k]}
      if (i,j) in PAIRS then Use[i,j] else Use[j,i] >= 2;
```

You can see that in your diagram there is only one move connecting the nodes in subset $-L, M, N, P, Q''$ to the other nodes. Here is data for the subtour elimination constraint for this set:

```
param nSubtours := 1 ;
set SUB[1] := L M N P Q ;
```

Add these lines to the given model and data files, producing files `holes2.mod` and `holes2.dat`.

Solve and diagram the result. You should find that the variables are now all 0 or 1, but that the solution is composed of three separate tours, through three separate subsets of nodes. What are these subsets?

d: Add three more subtour elimination constraints to `holes2.dat`, corresponding to the three subtours in the solution, and solve again. What subtours do you find now?

e: Continue adding subtour elimination constraints to `holes2.dat` until you find a solution that has no subtours or fractional variables. Diagram this optimal tour. Were there any optimal tours among those that you found earlier in this assignment?