IE C13, Fall 1991–92
Deterministic Models in Operations Research
Prof. Coullard

ASSIGNMENT #7

Due Wednesday, November 27

Readings: *Course Notes* V, especially sections 1–4.

1 In question 2(f) of Assignment #6, you encountered a "shortest tour" or "traveling salesman" problem that didn't seem to be solvable just by linear programming. Here we investigate a "branch and cut" approach that uses a series of linear programs to solve the problem.

In order to make our series of linear programs easier to state, we revise the last model from Assignment #6 (the one you stored in `holes3.mod`) to begin like this:

```
param n > 0 integer;

set HOLES := 1..n;
param hpos {HOLES} >= 0;
param vpos {HOLES} >= 0;

set PAIRS := {i in HOLES, j in HOLES: i < j};

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

param nsub >= 0 integer;
set SUB {k in 1..nsub} within 1..n;
```

We have added a parameter `nsub` to give the number of "subtour elimination" constraints. For each `k` in `1..nsub`, we have also declared a subset `SUB[k]` of the `n` holes. A constraint `Subtour[k]` can then say that the tour must contain at least two edges that connect the holes in `SUB[k]` to the holes not in `SUB[k]`:

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

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

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

subject to Subtour {k in 1..nsub}:
   sum {i in SUB[k], j in 1..n diff SUB[k]} X[min(i,j),max(i,j)] >= 2;
```

In the linear program of `holes3.mod`, there was only one such constraint, connecting holes 1–5 to the others, so the appropriate data is as follows:

```
data;
param n := 14 ;
param:   hpos vpos :=
    1      0    6
    2      0    5
    3      0    4
    4      1    5
    5      1    4
    6      3    6
    7      3    5
    8      3    4
    9      4    6
   10      4    5
   11      0    2
   12      0    1
   13      0    0
   14      1    1
param nsub := 1;
set SUB[1] := 1 2 3 4 5 ;
```

Copy `holes3.mod` to `bb0.mod`, and change the latter so that it contains the model and data listed above. Then use AMPL to check that you get the same optimal solution as before:

```
ampl: include f91-7-1.mod
ampl: option m53_spec 'ITER LIMIT 100000';
ampl: solve;
Solution time = 0.18 seconds.
 Objective Tour_Length: status OPTIMAL SOLN  Iteration 83
 Optimal value   17.2426407
ampl: option omit_zero_rows 1;
ampl: option display_eps .000001;
ampl: display X;
X :=
1  2    1
1  4    1
2  4    1
3  5    1
3  11   1
5  8    1
6  7    0.5
6  9    1
6  10   0.5
7  8    1
7  10   0.5
9  10   1
11 12   0.5
11 14   0.5
12 13   1
12 14   0.5
13 14   1
;
```

Each variable `X[i,j]` is supposed to be 1 if the tour connects holes `i` and `j`, or 0 if the tour does not connect holes `i` and `j`. The above solution does not correspond to any useful tour, because some of the variables are at fractional values between 0 and 1.

Every time you type `solve` ***when working on the following questions, be sure to check that you subsequently get the*** `status OPTIMAL SOLN` ***message as shown above.*** Any other status message indicates an error that must be fixed before you proceed.

a: To try to do something about the fractional variables, we will "branch" on one of them—say, `X[6,7]`. We know that `X[6,7]` has to be either 0 or 1 in a valid tour, so we'll start by fixing it to 0:

```
subject to Branch_6_7: X[6,7] = 0;
```

Add this constraint to your model, store the result in `bb1.mod`, and find the new solution. You should find that now there are no fractional variables. Draw a diagram of the holes, showing all the edges corresponding to variables that are 1.

b: Your drawing in (a) should show two subtours, one through holes 1–10 and the other through holes 11–14. To prevent these subtours, we can add a second subtour elimination constraint, through a simple modification to the data:

```
param nsub := 2;
set SUB[1] := 1 2 3 4 5 ;
set SUB[2] := 1 2 3 4 5 6 7 8 9 10 ;
```

Make this change to your data, store the result in `bb2.mod`, and find the new solution. You should again find that there are no fractional variables. Draw another diagram of the solution.

c: Your diagram in (b) should show three more subtours! Modify your data again to add three more subtour elimination constraints, one for each of these. (Don't forget to change `nsub` to 5 in your data.) Store the result in `bb3.mod`, find yet another solution, and draw another diagram.

This time your diagram should contain no subtours. You have found the shortest possible tour, among all those that *do not* contain an edge from hole 6 to hole 7.

d: There might be an even shorter tour, however, that *does* contain an edge from hole 6 to hole 7. To investigate this possibility, modify your model from (c) so that it fixes `X[6,7]` to 1 instead of 0:

```
subject to Branch_6_7: X[6,7] = 1;
```

Leave in all the previous subtour elimination constraints, since they have already proved useful. Store this linear program in `bb4.mod`; solve it and diagram the solution as before.

e: Your diagram in (d) should show a new combination of three subtours. Just as you did in (c), prevent these subtours by adding three new subtour elimination constraints. (You should then have eight of these constraints in all.) Store the result in `bb5.mod`.

When you see the optimal objective function value for this linear program, you can tell immediately (without even displaying `X`) that none of the tours having `X[6,7]` fixed to 1 will be the shortest. Briefly explain why.

f: Since you have now fully analyzed both possibilities for `X[6,7]`, you are done. The shortest tour is the one you found in (c).

You might wonder why we added only a few subtour elimination constraints at a time, instead of just putting them all into the linear program to begin with. How many *different* subtour elimination constraints are there for this particular problem?

2 When Newton's method is applied to the problem of solving $x^2 - a = 0$, it yields a simple iterative formula for determining the square root of a number:

$$x_{k+1} = \tfrac{1}{2}(x_k + a/x_k).$$

Use Newton's method to derive an iterative formula for determining the nth root of a number.

3 For this question, consider applying Newton's method to find a local optimum of the following polynomial function:

$$f(x) = x^4 - 3x^2 + 2x + 2$$

For this function, the iterations are very simple. Use your favorite computer or calculator to carry them out.

a: Starting from $x_0 = -1$, to what value does Newton's method converge?

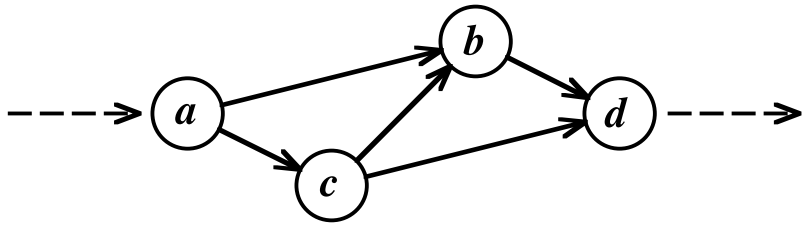
b: Starting from $x_0 = 0$, to what value does Newton's method converge?

c: Starting from $x_0 = +1$, to what value does Newton's method converge?

d: Is the value in (a) a local minimum or a local maximum?
Is the value in (b) a local minimum or a local maximum?
Is the value in (c) a local minimum or a local maximum?

e: Is there a global minimum of f? If so, what is it?

4 This problem starts out with a very simple "traffic flow" network,



and some data (to be explained) for each of the "roads" in the network:

From	To	*Time*	*Capacity*	*Sensitivity*
a	b	5.0	10	0.15
a	c	1.0	30	0.90
c	b	2.0	10	0.12
b	d	1.0	30	0.90
c	d	5.0	10	0.15

To be specific, we will imagine that the times are in minutes, the capacities are in cars per minute, and the sensitivities are in minutes per (car per minute).

The following statements can be used to represent networks of this kind in AMPL:

```
set inters;   # road intersections

param S symbolic in inters;   # origin of flow
param T symbolic in inters;   # destination of flow

set roads within {i in inters, j in inters: i != T and j != S};

param time {roads} > 0;
param cap {roads} > 0;
param sens {roads} > 0;

data;

set inters := a b c d ;

param S := a ;
param T := d ;

param: roads: time cap sens :=
       a b    5.0   10   0.15
       a c    1.0   30   0.9
       c b    2.0   10   0.12
       b d    1.0   30   0.9
       c d    5.0   10   0.15 ;
```

a: What is the shortest path, in minutes, from node `a` to node `d`?

b: What is the maximum traffic flow, in cars per minute, from `a` to `d`?

c: Question (a) above was concerned only with the speed of traffic, and question (b) only with the volume of traffic. In reality, these quantities are interrelated. As the traffic volume on a road increases from zero, the time required to travel the road also increases.

Travel time increases only moderately when there are just a few cars, but it increases very rapidly as the traffic approaches capacity. Thus a nonlinear function is needed to model this phenomenon. Let us define `t[i,j]`, the travel time on road `(i,j)`, by the following constraints:

```
var x {roads} >= 0;   # cars per minute entering road (i,j)
var t {roads} >= 0;   # travel time for road (i,j)

subject to travel_time {(i,j) in roads}:
   t[i,j] = time[i,j] + (sens[i,j]*x[i,j]) / (1-x[i,j]/cap[i,j]);

subject to capacity {(i,j) in roads}:  x[i,j] <= .999 * cap[i,j];
```

You can confirm that the travel time `t[i,j]` is close to `time[i,j]` when the road is lightly used—when `x[i,j]` is close to zero. On the other hand, the travel time goes to infinity as the use approaches the capacity—that is, as `x[i,j]` gets close to `cap[i,j]` cars per minute. The magnitude of `sens[i,j]` controls the rate at which travel time increases, as more cars try to enter the road.

(The bound in the `capacity` constraint is `.999 * cap[i,j]`, rather than just `cap[i,j]`, because `t[i,j]` would evaluate to ∞ if `x[i,j]` were allowed to equal `cap[i,j]`. This is no problem in theory, but it creates a great deal of difficulty for the nonlinear programming algorithm.)

Suppose that we want to analyze how the network can best handle some number of cars per minute. The objective is to minimize average travel time from `S` to `T`:

```
param through > 0;   # cars per minute using the network

minimize avg_time:
   (sum {(i,j) in roads} t[i,j] * x[i,j]) / through;
```

The nonlinear expression `t[i,j] * x[i,j]` is travel minutes on road `(i,j)` times cars per minute entering the road—hence, the number of cars on road `(i,j)`. The summation in the objective thus gives the total cars in the entire system. Dividing by the number of cars per minute getting through, we have the average number of minutes for each car.

To guarantee the postulated throughput, the nonlinear program also needs the following linear constraints:

```
subject to flow {i in inters: i != S and i != T}:
   sum {(i,j) in roads} x[i,j] = sum {(j,i) in roads} x[j,i];

subject to source: sum {(S,j) in roads} x[S,j] = through;
```

Type the statements above into a file `traffic1.mod`, and add a line to set `through` to 4.0. Use AMPL to find a solution. Confirm that a throughput of 4.0 cars per minute is optimally managed by sending half the cars along $a \to b \to d$ and half along $a \to c \to d$, giving an average travel time of about 8.30 minutes.

d: If you change the throughput to 0.5 cars per minute, what solution do you get? Explain how this solution is closely related to your shortest path solution from (a).

e: If you change the throughput to 29.5 cars per minute, what solution do you get? Explain how this solution is closely related to your maximum flow solution from (b).

f: The model in (c) assumes that you can make the cars' drivers take certain routes. For example, in the optimal solution for a throughput of 4.0, no drivers are allowed to "cut through" from c to b.

What would happen if instead all drivers could take whatever route they pleased? Observation has shown that, in such a case, the traffic tends to reach a *stable* solution in which no route has a travel time less than the average. The optimal solution for a throughput of 4.0 is not stable, since—as you can verify from the solution listing—the travel time on $a \to c \to b \to d$ is only about 7.86 minutes; some drivers would try to cut through if they were permitted.

In the very simple case we are considering, there are only three possible routes through the network. By adding just two constraints, you can ensure that all three routes have the same travel time:

```
subject to stable1:
   t["a","b"] + t["b","d"] = t["a","c"] + t["c","d"];

subject to stable2:
   t["a","b"] + t["b","d"] = t["a","c"] + t["c","b"]+ t["b","d"];
```

Add these to your model, and store the result in `traffic2.mod`. Apply AMPL to find the stable solution for a throughput of 4.0. What percentage of the drivers cut through, and how much does the average travel time increase? (This phenomenon is known as Braess's paradox: traffic would have been faster if the road from c to b had never been built!)

Remember to hand in your disk with the written part of the assignment.

Questions or comments to:

Prof. Robert Fourer or Prof. Collette Coullard
Department of Industrial Engineering and Management Sciences
Northwestern University
Evanston, IL 60208-3119, U.S.A.

708-491-3151
708-491-8005 (fax)

`4er@iems.nwu.edu` or `coullard@iems.nwu.edu`