Homework 5 Advanced Analytics and Metaheuristics

Group 16: Nicholas Jacob

March 5, 2024

1. Flying

   (a) To attack this problem, you will want to minimize the cost. This is the objective. There are two tricky items in the formulation, the ramp fee will only turn on if no or a limited amount of fuel is taken and extra fuel must be carried as a matter of precaution. We deal with the first as a indicator of whether we are paying the fee or not at each airport, δ_i. We identify the other variable of importance as the amount of fuel taken on at each of the airport, x_i. We also add a variable for the total fuel in the jet for ease of computation, f_i.

$$\min \sum_i \frac{x_i}{6.7} \cdot price_i + \delta_i \cdot rampfee_i \tag{1}$$
$$minfuel_i \cdot \delta_i \leq x_i \quad \forall i \in 1, \dots, 5 \tag{2}$$
$$f_0 = 7000 \tag{3}$$
$$f_1 = x_1 + f_0 \tag{4}$$
$$f_2 = x_2 + f_1 - fuelburn_1 \tag{5}$$
$$\vdots \tag{6}$$
$$2500 + fuelburn_i \leq f_i \leq 14000 \quad \forall i \in 1, \dots, 5 \tag{7}$$

   We also account for the max takeoff weight and landing weight by including passengers and tankered fuel. We see there are 2 passengers on the first leg, 4 on the second and 8 on all the rest. We also consider the refilling of the plane into final ready state (7000lbs of fuel) after the final leg.

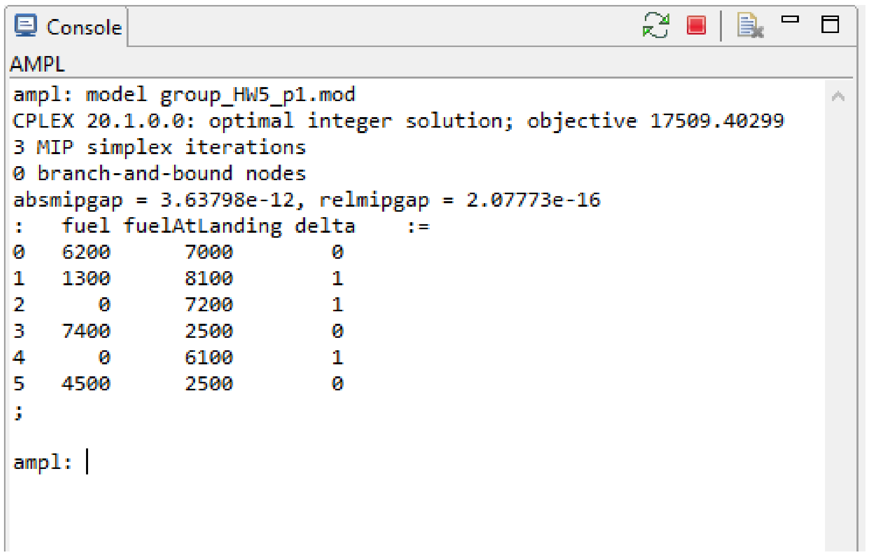
Lastly, we must turn on or off the fee for landing if we don't take the minimum fuel. We do this with the condition,

$$6.7 minGalsToWaiveFee_i - x_i \leq M\delta_i$$

Here we set $M = 140000$, the max amount of fuel we can take. Here is the code and data file:

```
reset;

option solver cplex;


set legs;

param fuelburn{legs};
param fuelprice{legs};
param minGallonsToWaiveFee{legs};
param fee{legs};
param numPass{legs};

var fuel{legs}>=0;
var delta{legs} binary;
var fuelAtLanding{legs}>= 2500;

minimize cost: sum{i in legs} (fuel[i]*fuelprice[i]/6.7 + delta[i]*fee[i]);

s.t. turnFeeOff{i in legs}: 6.7* minGallonsToWaiveFee[i] -fuel[i]<= 14000*(delta[i]);


s.t. computeFuelAtLanding0: fuelAtLanding[0] = 7000;
s.t. computeFuelAtLanding1: fuelAtLanding[1] = fuelAtLanding[0] + fuel[0] - fuelburn[1];
s.t. computeFuelAtLanding2: fuelAtLanding[2] = fuelAtLanding[1] + fuel[1] - fuelburn[2];
s.t. computeFuelAtLanding3: fuelAtLanding[3] = fuelAtLanding[2] + fuel[2] - fuelburn[3];
s.t. computeFuelAtLanding4: fuelAtLanding[4] = fuelAtLanding[3] + fuel[3] - fuelburn[4];
s.t. computeFuelAtLanding5: fuelAtLanding[5] = fuelAtLanding[4] + fuel[4] - fuelburn[5];

s.t. landingWeight{i in legs}: 22800 + 200 * numPass[i] + fuelAtLanding[i] <= 31800;

s.t. takeOffWeigth{i in legs}: 22800 + 200*numPass[i] + fuelAtLanding[i] + fuelburn[i] <= 36400;

s.t. maxFuel{i in legs}: fuelAtLanding[i] + fuel[i] <= 14000;

s.t. readyToGo: fuelAtLanding[5] + fuel[5] = 7000;

data group_HW5_p1.dat;

solve;

display fuel, fuelAtLanding, delta;


set legs:= 0        1        2        3        4        5;

param: fuelburn fuelprice minGallonsToWaiveFee        fee        numPass:=
        0        0              4                         0                                             0        0
        1        5100        8.32               600                                           800        2
        2        2200        8.99               500                                           750        4
        3        4700        6.48               650                                           600        8
        4        3800        9.27               500                                           800        8
        5        3600        4                       0                                          0        8
;
```

Lastly the output is below. We pay the fee several times even when we do take on some fuel at stops. Total cost of \$17 ,500 or so.



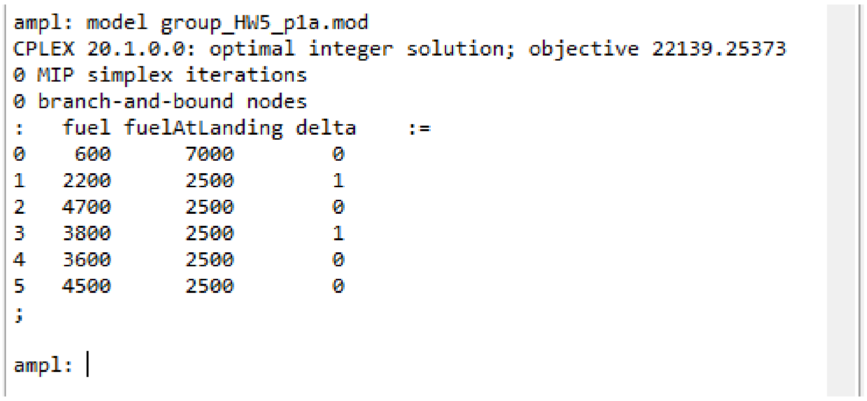
```
ampl: model group_HW5_p1.mod
CPLEX 20.1.0.0: optimal integer solution; objective 17509.40299
3 MIP simplex iterations
0 branch-and-bound nodes
absmipgap = 3.63798e-12, relmipgap = 2.07773e-16
:   fuel fuelAtLanding delta    :=
0   6200      7000        0
1   1300      8100        1
2      0      7200        1
3   7400      2500        0
4      0      6100        1
5   4500      2500        0
;

ampl: 
```

To make the model not allow tankering, we will change the model to require that fuel at landing be exactly the minimum 2500 (except on the 0th leg where I have already assumed it has been filled up)

```
reset;

option solver cplex;


set legs;

param fuelburn{legs};
param fuelprice{legs};
param minGallonsToWaiveFee{legs};
param fee{legs};
param numPass{legs};

var fuel{legs}>=0;
var delta{legs} binary;
var fuelAtLanding{legs}>= 2500;

minimize cost: sum{i in legs} (fuel[i]*fuelprice[i]/6.7 + delta[i]*fee[i]);

s.t. turnFeeOff{i in legs}: 6.7* minGallonsToWaiveFee[i] -fuel[i]<= 14000*(delta[i]);


s.t. computeFuelAtLanding0: fuelAtLanding[0] = 7000;
s.t. computeFuelAtLanding1: fuelAtLanding[1] = fuelAtLanding[0] + fuel[0] - fuelburn[1];
s.t. computeFuelAtLanding2: fuelAtLanding[2] = fuelAtLanding[1] + fuel[1] - fuelburn[2];
s.t. computeFuelAtLanding3: fuelAtLanding[3] = fuelAtLanding[2] + fuel[2] - fuelburn[3];
s.t. computeFuelAtLanding4: fuelAtLanding[4] = fuelAtLanding[3] + fuel[3] - fuelburn[4];
s.t. computeFuelAtLanding5: fuelAtLanding[5] = fuelAtLanding[4] + fuel[4] - fuelburn[5];

s.t. landingWeight{i in legs}: 22800 + 200 * numPass[i] + fuelAtLanding[i] <= 31800;

s.t. takeOffWeigth{i in legs}: 22800 + 200*numPass[i] + fuelAtLanding[i] + fuelburn[i] <= 36400;

s.t. maxFuel{i in legs}: fuelAtLanding[i] + fuel[i] <= 14000;

s.t. readyToGo: fuelAtLanding[5] + fuel[5] = 7000;

s.t. computeFuelAtLanding1e: fuelAtLanding[1] = 2500;
s.t. computeFuelAtLanding2e: fuelAtLanding[2] = 2500;
```

```
s.t. computeFuelAtLanding3e: fuelAtLanding[3] = 2500;
s.t. computeFuelAtLanding4e: fuelAtLanding[4] = 2500;
s.t. computeFuelAtLanding5e: fuelAtLanding[5] = 2500;


data group_HW5_p1.dat;

solve;

display fuel, fuelAtLanding, delta;
```

With output and new price of just above 22k. We do pay one less landing fee in this model.



```
ampl: model group_HW5_p1a.mod
CPLEX 20.1.0.0: optimal integer solution; objective 22139.25373
0 MIP simplex iterations
0 branch-and-bound nodes
:   fuel fuelAtLanding delta    :=
0    600      7000        0
1   2200      2500        1
2   4700      2500        0
3   3800      2500        1
4   3600      2500        0
5   4500      2500        0
;

ampl: |
```

(b) To add the condition that if we buy we must buy at least 200 gallons, we simply require that each x_i is a minimum of 200 * 6.7 = 1340 lbs of fuel if it is activated. We will use another binary for this purpose. We need the constraint

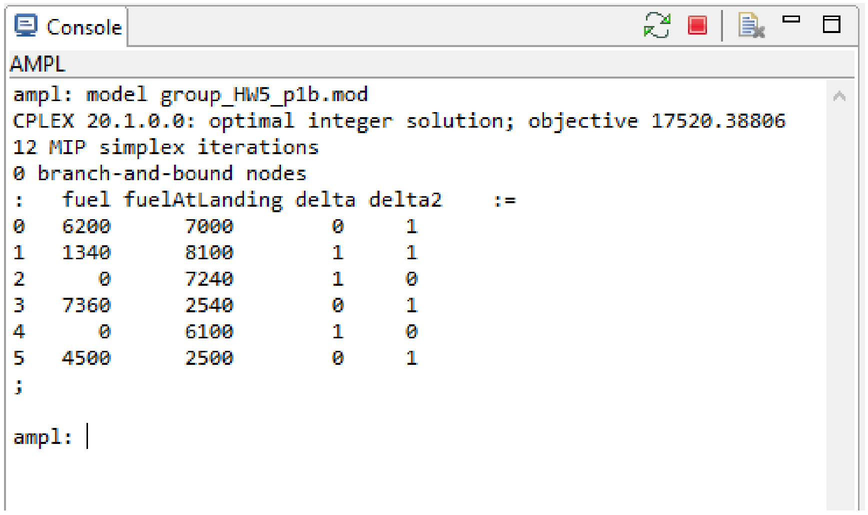
$$1340 * \Delta_i \leq x_i \leq M\Delta_i$$

Again we used 14000 for the M value. We show code next. It is very similar with the few new constraints and variables.

```
reset;

option solver cplex;


set legs;

param fuelburn{legs};
param fuelprice{legs};
param minGallonsToWaiveFee{legs};
param fee{legs};
param numPass{legs};

var fuel{legs}>=0;
var delta{legs} binary;
var fuelAtLanding{legs}>= 2500;
var delta2{legs} binary;

minimize cost: sum{i in legs} (fuel[i]*fuelprice[i]/6.7 + delta[i]*fee[i]);

s.t. turnFeeOff{i in legs}: 6.7* minGallonsToWaiveFee[i] -fuel[i]<= 14000*(delta[i]);
```

```
s.t. computeFuelAtLanding0: fuelAtLanding[0] = 7000;
s.t. computeFuelAtLanding1: fuelAtLanding[1] = fuelAtLanding[0] + fuel[0] - fuelburn[1];
s.t. computeFuelAtLanding2: fuelAtLanding[2] = fuelAtLanding[1] + fuel[1] - fuelburn[2];
s.t. computeFuelAtLanding3: fuelAtLanding[3] = fuelAtLanding[2] + fuel[2] - fuelburn[3];
s.t. computeFuelAtLanding4: fuelAtLanding[4] = fuelAtLanding[3] + fuel[3] - fuelburn[4];
s.t. computeFuelAtLanding5: fuelAtLanding[5] = fuelAtLanding[4] + fuel[4] - fuelburn[5];

s.t. landingWeight{i in legs}: 22800 + 200 * numPass[i] + fuelAtLanding[i] <= 31800;

s.t. takeOffWeigth{i in legs}: 22800 + 200*numPass[i] + fuelAtLanding[i] + fuelburn[i] <= 36400;

s.t. maxFuel{i in legs}: fuelAtLanding[i] + fuel[i] <= 14000;

s.t. readyToGo: fuelAtLanding[5] + fuel[5] = 7000;

s.t. buyAnyMoreThan200{i in legs}: 1340*delta2[i] <= fuel[i];
s.t. buyAnyMoreThan200getturnedOn{i in legs}: fuel[i]<= 14000*delta2[i];

data group_HW5_p1.dat;

solve;

display fuel, fuelAtLanding, delta, delta2;
```

We see in the output below that the plan only changes slightly with this requirement and is very near our original optimal cost, about $11 more than the original.



```
AMPL
ampl: model group_HW5_p1b.mod
CPLEX 20.1.0.0: optimal integer solution; objective 17520.38806
12 MIP simplex iterations
0 branch-and-bound nodes
:   fuel fuelAtLanding delta delta2    :=
0   6200      7000         0     1
1   1340      8100         1     1
2      0      7240         1     0
3   7360      2540         0     1
4      0      6100         1     0
5   4500      2500         0     1
;

ampl: 
```

2. Following the outline here, we let H be the total number of exhibits possibly available. Here we set $H = 10$ with the assumption that we will need much less. Define w_i means we used that particular exhibit to house animals. $x_{v,i}$ means we used i exhibit to house v animal. Then our goal is to:

$$\begin{array}{rl} minimize & \sum_{i=1}^{H} w_i \\ s.t. & \sum_{i=1}^{H} x_{i,v} = 1 \quad \forall v \\ & x_{i,v} + x_{i,u} \leq w_i \quad \forall (v,u) \in Table, \; i \leq H \\ & x_{i,v}, w_i \in \{0,1\} \end{array} \tag{8}$$

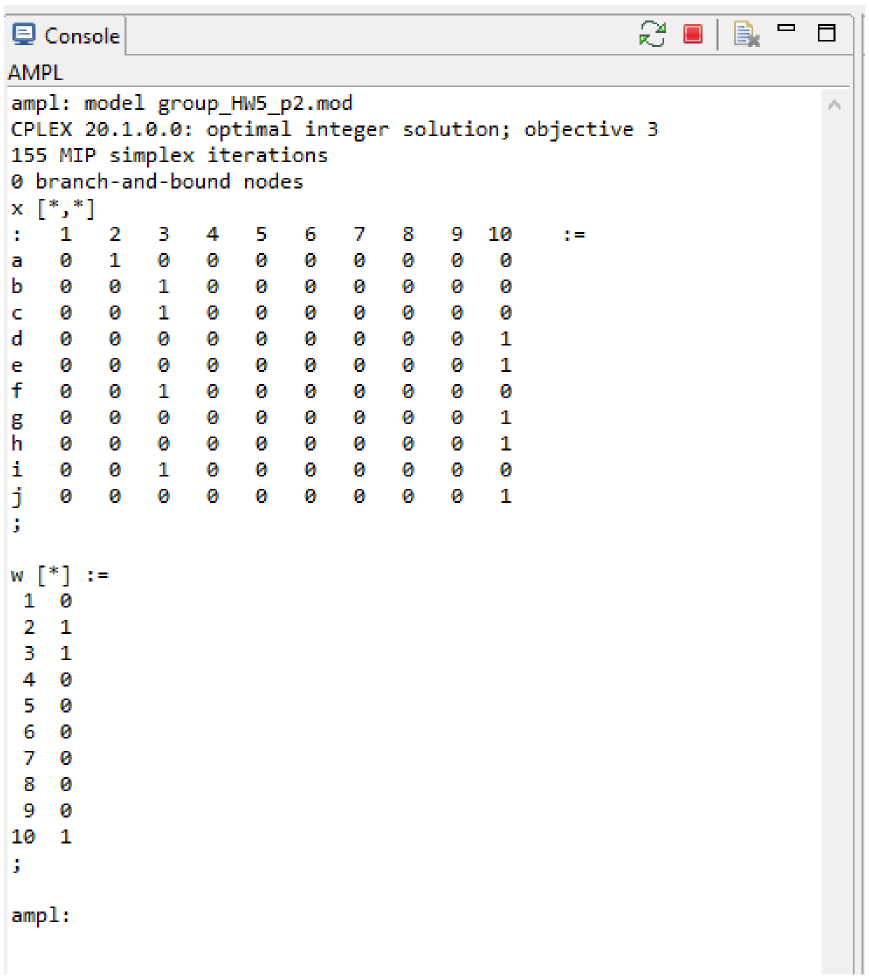
Let's take a moment to explain the constraints. The first constraint requires that each animal is only in one pen. The second constraint requires that two animals that do not get along cannot be in the same pen and forces the w binary to turn on. We code this into AMPL as follows:

```
reset;

option solver cplex;


param H > 0;
let H := 10;

set animals;
set edges within {animals,animals};



var x{animals,1..H} binary;
var w{1..H} binary;

minimize exhibits: sum{i in 1..H} w[i];
subject to homeForAll {a in animals}: sum{i in 1..H} x[a,i] = 1;
subject to edgeConstraint {(u,v) in edges, i in 1..H}:  x[u,i]+x[v,i]<= w[i];

data group_HW5_p2.dat;

solve;

display x, w;
```

With use of a data file:

```
set animals :=  a        b        c        d        e        f        g        h        i        j;
;

set edges := (a,*)        b        e        j
                       (b,*)        a        d        g
                       (c,*)        h        j
                       (d,*)        b        f
                       (e,*)        a        i
                       (f,*)        d        j
                       (g,*)        b
                       (h,*)        c        i
                       (i,*)        e        h        j
                       (j,*)        a        c        f        i
                       ;
```

We see the output here:


Console

AMPL

```
ampl: model group_HW5_p2.mod
CPLEX 20.1.0.0: optimal integer solution; objective 3
155 MIP simplex iterations
0 branch-and-bound nodes
x [*,*]
:   1   2   3   4   5   6   7   8   9   10    :=
a   0   1   0   0   0   0   0   0   0   0
b   0   0   1   0   0   0   0   0   0   0
c   0   0   1   0   0   0   0   0   0   0
d   0   0   0   0   0   0   0   0   0   1
e   0   0   0   0   0   0   0   0   0   1
f   0   0   1   0   0   0   0   0   0   0
g   0   0   0   0   0   0   0   0   0   1
h   0   0   0   0   0   0   0   0   0   1
i   0   0   1   0   0   0   0   0   0   0
j   0   0   0   0   0   0   0   0   0   1
;

w [*] :=
 1  0
 2  1
 3  1
 4  0
 5  0
 6  0
 7  0
 8  0
 9  0
10  1
;

ampl:
```

Interpreting the output, we see that we will require 3 pens. Animal a will live alone. Animals b, c, f, and i will live in the next pen. Animals d, e, g, h and j will live together in the last pen. We have no idea why AMPL selected the second third and tenth pen but that should not matter to our minimization of the total number of pens required.

3. For our approach to this problem, we create two variables, $\delta_{a,i}$ and $x_{a,i}$. a is the type of fuel and i is the tank number. Our binary δ describes which type of fuel our tank will use and the real positive$x$ describes how many 1000L will be moved of each type of fuel into that tank. Our objective is to minimize the cost of pumping the fuel into

tanks without mixing the fuel types.

$$minimize \sum_{a \in types, i \in tanks} cost(a,i) \cdot x_{a,i}$$

The conditions follow

$$\begin{array}{ll} s.t. & \sum_{a \in types} \delta_{a,i} \leq 1 \\ & \sum_{i \in tanks} capacity(i) \cdot \delta_{a,i} \geq totalAvailable(a) \\ & \sum_{a \in types} 1000 \cdot x_{a,i} \leq capacity(i) \\ & \sum_{i \in tanks} 1000 \cdot x_{a,i} = totalAvailable(a) \\ & \delta_{a,i} \leq x_{a,i} \leq M\delta_{a,i} \end{array} \tag{9}$$

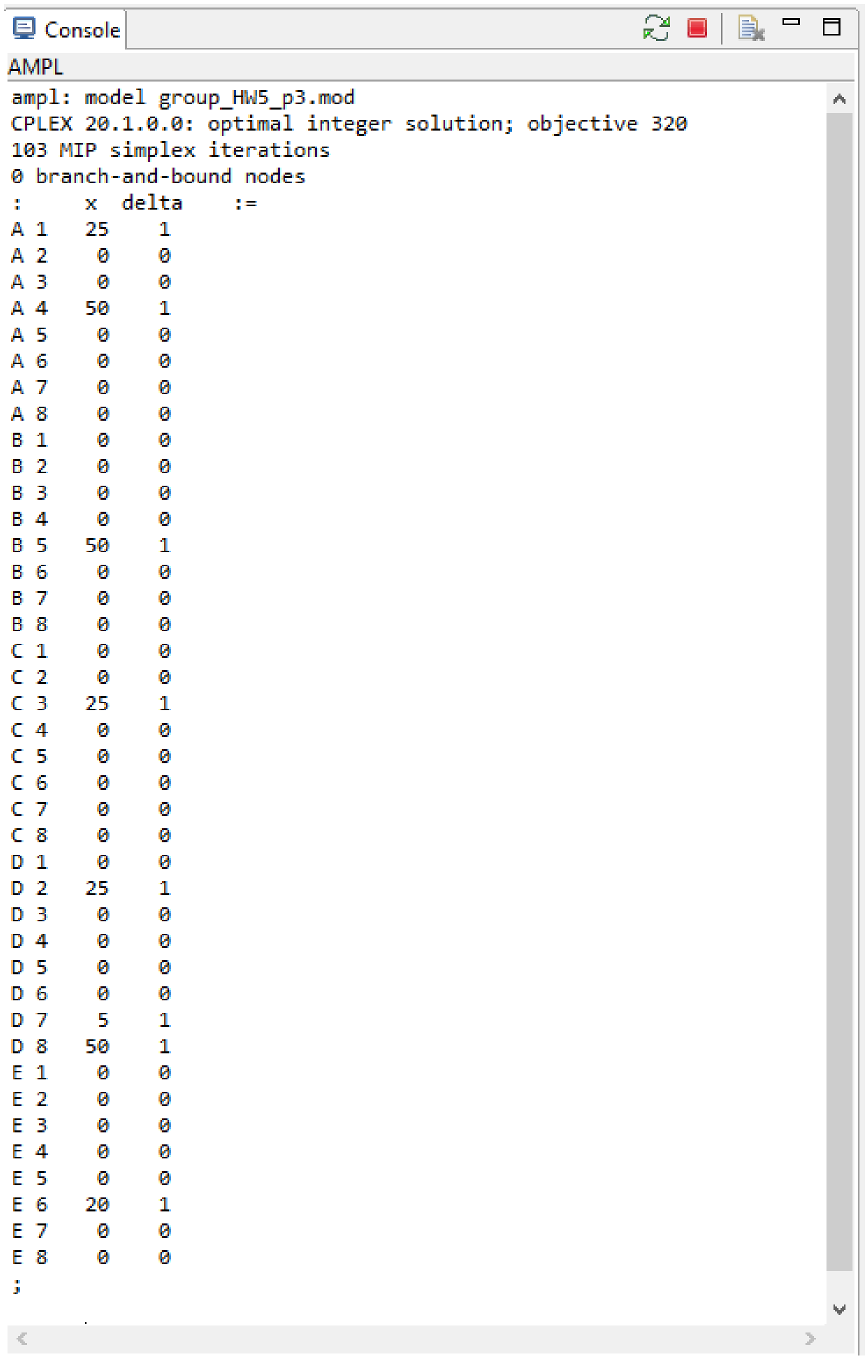
The first constraint requires only one type of oil be used in any tank. The second requires that enough capacity is reserved for each type of oil. The next requires that you don't put more oil than capacity into any one tank. The next requires that all oil find a home in one of the tanks. The last requirement activates δ. If you move any oil into the tank, the binary is turned on.

We code this into AMPL below:

```
#DSA/ISE 5113 Integer Programming
#Example IP: oil exploration

#Find least-cost selection of 5 out of 10 possible sites

reset;

#OPTIONS ----------------------------------------------------
option solver cplex;


#SETS AND PARAMETERS ----------------------------------------
set tanks = 1..8;
set types;
param cost{types,tanks};
param cap{tanks};
param g{types};

#DECISION VARIABLES -----------------------------------------
var delta{types, tanks} binary;     #pump type into tank?
var x{types, tanks} >=0; #volume moved in 1000L


#OBJECTIVE --------------------------------------------------
minimize totalCost: sum{a in types, i in tanks} cost[a,i]*x[a,i];

#CONSTRAINTS ------------------------------------------------

#one type fuel in each tank
s.t. oneTypePerTank {i in tanks}: sum {a in types} delta[a,i] <= 1;

#enough storage
s.t. storage {a in types}: sum { i in tanks} cap[i]*delta[a,i]>= g[a];

#don't overfill
s.t. overfill {i in tanks}: sum {a in types} 1000*x[a,i]<=cap[i];

#placeall fuel
s.t. placefuel {a in types}: sum{i in tanks} 1000*x[a,i] = g[a];
```

```
#announce the tank has been taken
s.t. announce {i in tanks, a in types}: x[a,i]<=g[a]*delta[a,i];
s.t. announce2 {i in tanks, a in types}: x[a,i]>= delta[a,i];


data;
set types:= A        B        C        D        E;

param cost:        1        2        3        4        5        6        7        8 :=
                A                1        2        2        1        4        4        5        3
                B                2        3        3        3        1        4        5        2
                C                3        4        1        2        1        4        5        1
                D                1        1        2        2        3        4        5        2
                E                1        1        1        1        1        1        5        5
;

param g:=
        A        75000
        B        50000
        C        25000
        D        80000
        E        20000;

param cap:=
        1        25000
        2        25000
        3        30000
        4        60000
        5        80000
        6        85000
        7        100000
        8        50000;

solve;
display x, delta;
```

We see the output here:



```
ampl: model group_HW5_p3.mod
CPLEX 20.1.0.0: optimal integer solution; objective 320
103 MIP simplex iterations
0 branch-and-bound nodes
:      x   delta    :=
A 1    25     1
A 2     0     0
A 3     0     0
A 4    50     1
A 5     0     0
A 6     0     0
A 7     0     0
A 8     0     0
B 1     0     0
B 2     0     0
B 3     0     0
B 4     0     0
B 5    50     1
B 6     0     0
B 7     0     0
B 8     0     0
C 1     0     0
C 2     0     0
C 3    25     1
C 4     0     0
C 5     0     0
C 6     0     0
C 7     0     0
C 8     0     0
D 1     0     0
D 2    25     1
D 3     0     0
D 4     0     0
D 5     0     0
D 6     0     0
D 7     5     1
D 8    50     1
E 1     0     0
E 2     0     0
E 3     0     0
E 4     0     0
E 5     0     0
E 6    20     1
E 7     0     0
E 8     0     0
;
```

We see the total cost of \$320. We use tank 1 and 4 for type A fuel, tank 5 for type B, tank 3 for type C, tanks 2, 7, and 8 for type D and tank 6 for type E. This is quite the haphazard plan. I could only imagine the field agents complaining about how the engineers have no idea what they are doing. We see an important assumption here that labour was not considered here in this plan. Someone will have to run this storage plan and then later undo it. This could cost the company more than if we had implemented a simpler plan.

4. We code the piecewise linear cost function in the method described in the modules and text. Each piece of the cost function gets a variable representing it and the total is forced by the some of each. We force them to turn on in order by careful use of binary variables associated with when each of the levels has been filled. For spacerays we do this with 4 levels (sr) and three binaries (srb). They are related as

$$\begin{aligned} s.t.\quad & 125srb1 \leq sr1 \leq 125 \\ & 100srb2 \leq sr2 \leq 100 \cdot srb1 \\ & 150srb2 \leq sr3 \leq 150 \cdot srb2 \\ & sr4 \leq 700 \cdot srb3 \end{aligned} \tag{10}$$

Each of the binaries will allow the next to turn on only after the level variable has completely filled. We also ask that $SR = sr1 + sr2 + sr3 + sr4$ be integer. We do the same thing for zappers. Details are in the AMPL model file. We also consider a plastic, labour, and total production restriction. Lastly we demand that the number of zappers plus 350 is greater that the number of space rays.

Here is the model file:

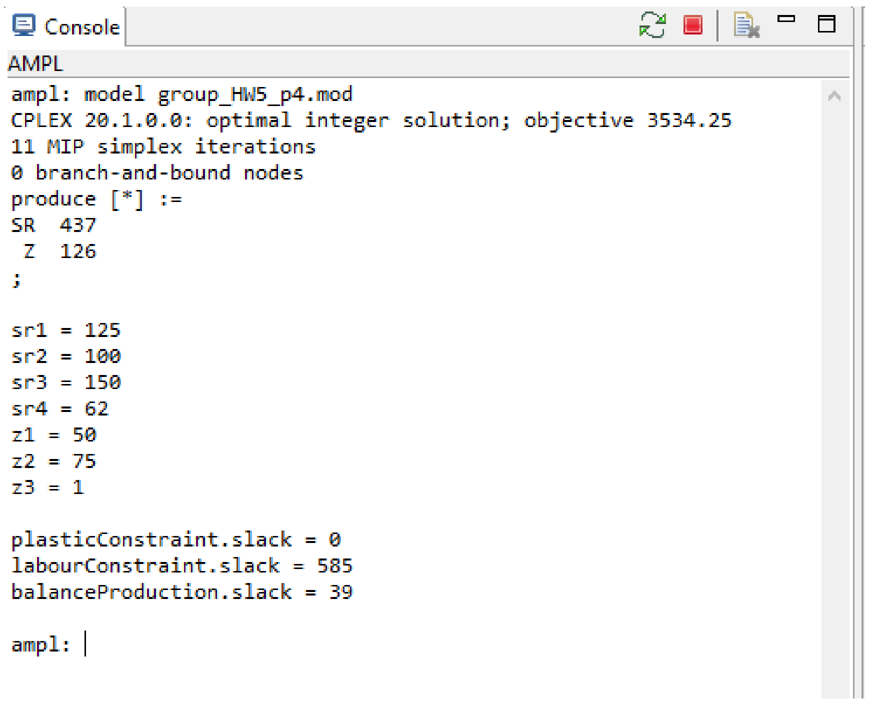
```
reset;

option solver cplex;

set toys;

param revenue{toys};
param plastic{toys};
param labor{toys};
param maxplastic;
param maxlabor;
param maxproduct;

var produce{toys}>= 0 integer;
var sr1 >=0;
var sr2 >=0;
var sr3 >=0;
var sr4 >=0;
var srb1 binary;
var srb2 binary;
var srb3 binary;
var z1 >=0;
var z2 >=0;
var z3 >=0;
var zb1 binary;
var zb2 binary;
```

```
maximize profit: sum{t in toys} revenue[t]*produce[t] - (1.5*sr1 +1.05*sr2+0.95*sr3+0.75*sr4)
                                                                                                  -(1.05*z1 +0.75*z2+1.5*z3);

s.t. plasticConstraint: sum{t in toys} produce[t]*plastic[t] <= maxplastic;
s.t. labourConstraint: sum{t in toys} produce[t]*labor[t] <= maxlabor;
s.t. totalProduction: sum{t in toys} produce[t] <= maxproduct;
s.t. balanceProduction: produce['SR']<= produce['Z'] + 350;

s.t. sr: produce['SR'] = sr1 + sr2+ sr3+sr4;

s.t. piece1aSR:  125*srb1 <= sr1;
s.t. piece1bSR: sr1 <= 125;

s.t. piece2aSR: 100*srb2 <= sr2;
s.t. piece2bSR: sr2 <= 100*srb1;

s.t. piece3aSR:  150*srb3<= sr3;
s.t. piece3bSR: sr3<= 150* srb2;

s.t. piece4SR: sr4<=srb3*700;

s.t. z: produce['Z'] = z1 + z2+ z3;

s.t. piece1aZ:  50*zb1 <= z1;
s.t. piece1bZ: z1<= 50;

s.t. piece2aZ: 75*zb2 <= z2;
s.t. piece2bZ: z2<= 75*zb1;

s.t. piece3Z: z3<= 700*zb2;

data group_HW5_p4.dat;

solve;


display produce, sr1, sr2, sr3, sr4, z1, z2, z3;

display plasticConstraint.slack, labourConstraint.slack, balanceProduction.slack;
```

Here is the data file:

```
set toys:= SR        Z;

param:         revenue        plastic        labor:=
        SR       8               2              3
        Z        5               1              4
;

param maxplastic = 1000;
param maxlabor = 2400;
param maxproduct = 700;
```

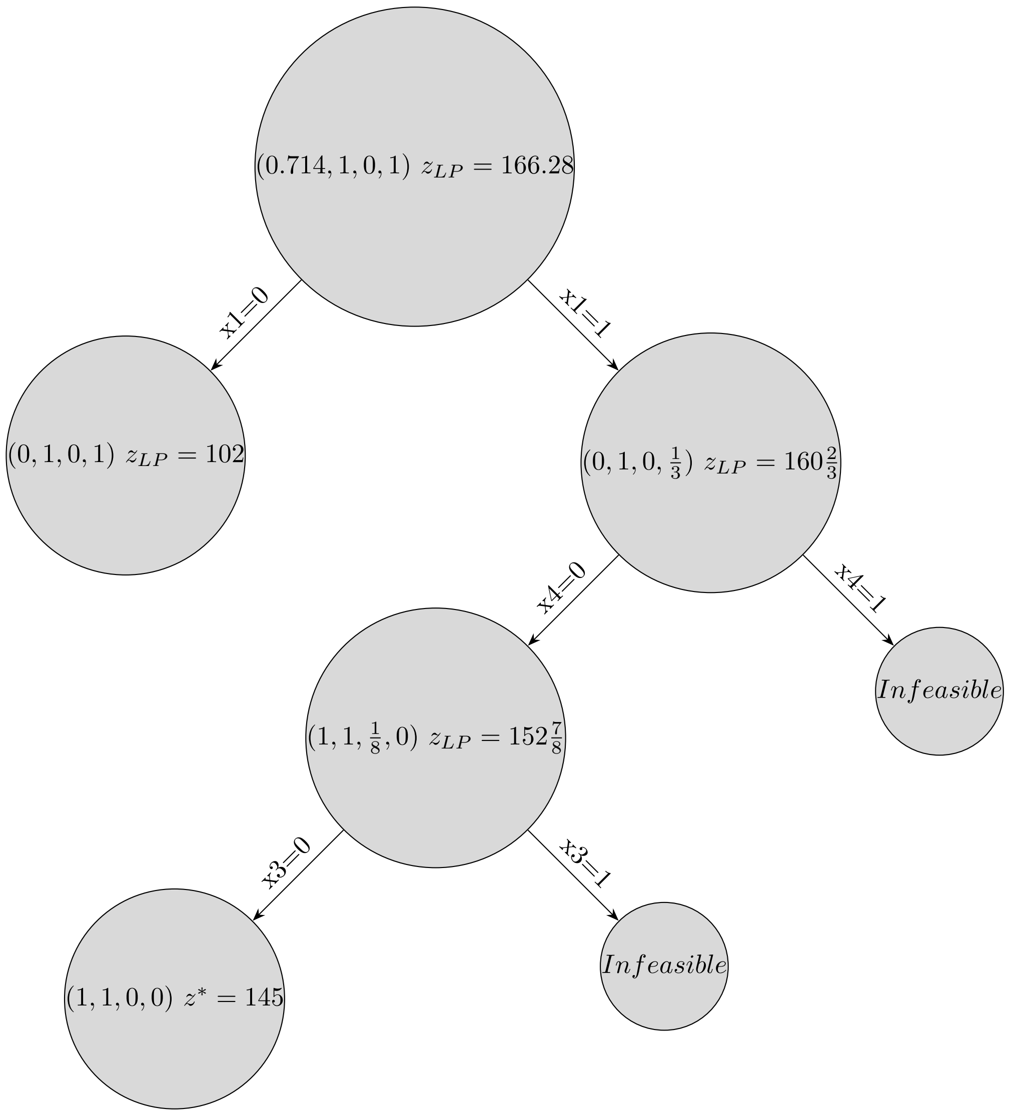
Here is the output:



```
ampl: model group_HW5_p4.mod
CPLEX 20.1.0.0: optimal integer solution; objective 3534.25
11 MIP simplex iterations
0 branch-and-bound nodes
produce [*] :=
SR  437
 Z  126
;

sr1 = 125
sr2 = 100
sr3 = 150
sr4 = 62
z1 = 50
z2 = 75
z3 = 1

plasticConstraint.slack = 0
labourConstraint.slack = 585
balanceProduction.slack = 39

ampl: 
```

We create 437 space rays, 126 zappers, making $3534.25 There is no remaining plastic but there are some more labour hours available.

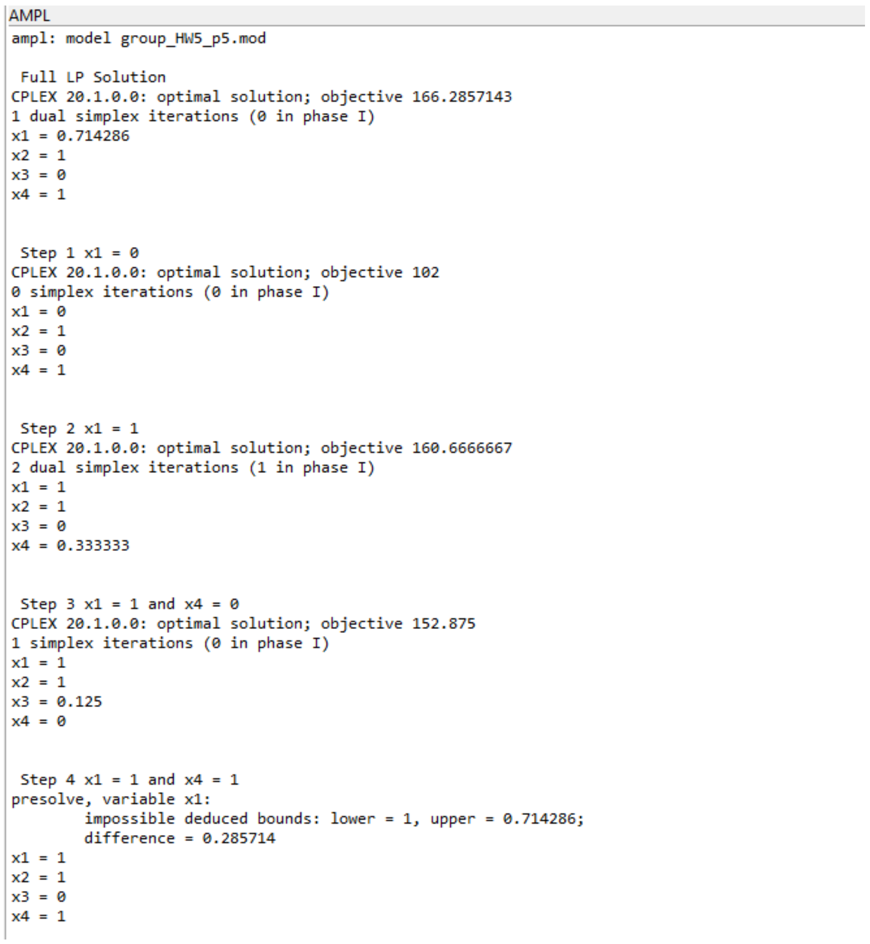
5. To approach this problem, we first coded into AMPL the LP problem without the binary restrictions on x_i. We instead insisted that $0 \leq x_i \leq 1$. We then solved the full LP problem and observed the output. In the first bubble, we see the general solution. Next we pick x_1 and force it to be binary. We do this by adding the condition that $x_1 = 0$ and then also $x_1 = 1$. We solve both of those with the full LP method. We arrive at a binary solution for $x_1 = 0$ and a non-binary solution for $x_1 = 1$. We repeat this process for x_4, picking x_4 since it was not binary in the current solution. We find a non-binary solution and an infeasible solution. We do not continue on the infeasible tree but do continue down the branch that gave an LP solution. This time we examine x_3. Again an infeasible solution and a solution that beats our original other binary solution. Having no more branches to continue on, we terminate the algorithm. We also run the algorithm in AMPL and see we did indeed find the maximum under the binary assumption.


$(0.714, 1, 0, 1)\ z_{LP} = 166.28$
x1=0
x1=1
$(0,1,0,1)\ z_{LP} = 102$
$(0,1,0,\frac{1}{3})\ z_{LP} = 160\frac{2}{3}$
x4=0
x4=1
$(1,1,\frac{1}{8},0)\ z_{LP} = 152\frac{7}{8}$
$Infeasible$
x3=0
x3=1
$(1,1,0,0)\ z^* = 145$
$Infeasible$

```
reset;

option solver cplex;

var x1>=0, <=1;
var x2>=0, <=1;
var x3>=0, <=1;
var x4>=0, <=1;

maximize objs: 90*x1+55*x2+63*x3 +47*x4;

s.t. con1: 7*x1 + 2*x2+8*x3+3*x4 <= 10;
s.t. con2: x3+x4<=1;
s.t. con3: x3-x1<=0;
s.t. con4: x4-x2<=0;
```

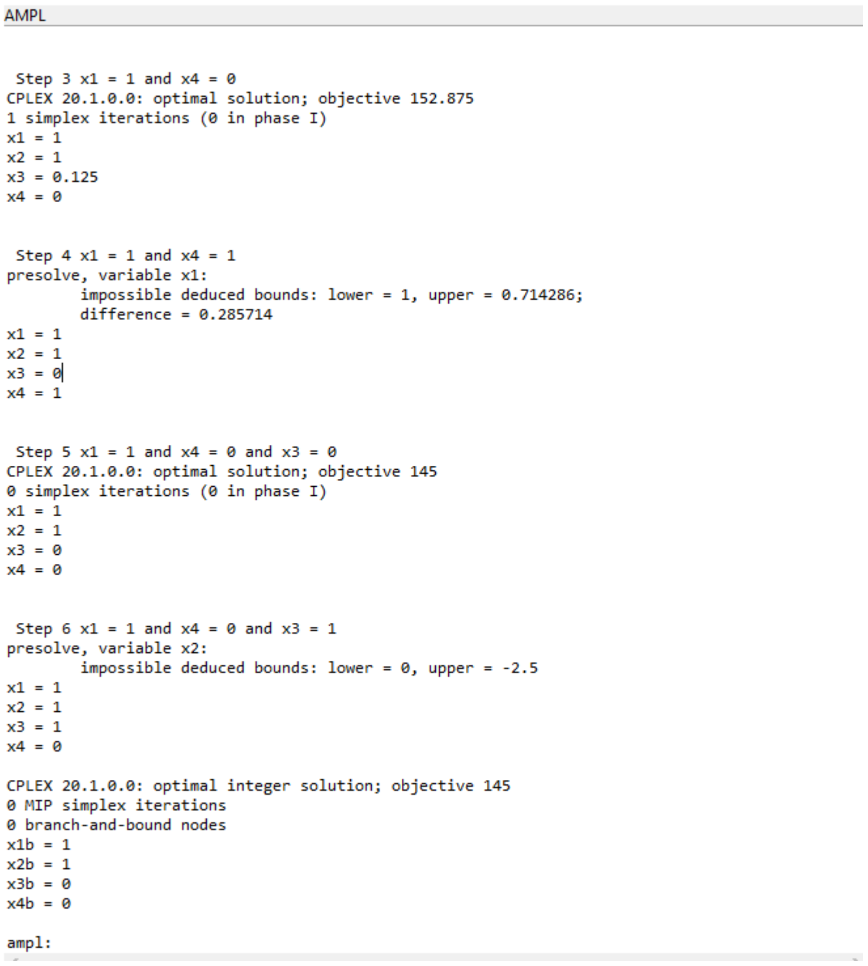
```
problem fullLP: objs, x1, x2, x3, x4, con1, con2, con3, con4;


printf "\n Full LP Solution\n";
solve fullLP;


display x1, x2, x3, x4;

s.t. x1is0: x1 = 0;
s.t. x1is1: x1 = 1;

problem step1: objs, x1, x2, x3, x4, con1, con2, con3, con4, x1is0;
printf "\n Step 1 x1 = 0\n";
solve step1;

display x1, x2, x3, x4;

problem step2: objs, x1, x2, x3, x4, con1, con2, con3, con4, x1is1;
printf "\n Step 2 x1 = 1\n";
solve step2;

display x1, x2, x3, x4;

s.t. x4is0: x4 = 0;
s.t. x4is1: x4 = 1;

problem step3: objs, x1, x2, x3, x4, con1, con2, con3, con4, x1is1, x4is0;
printf "\n Step 3 x1 = 1 and x4 = 0\n";
solve step3;

display x1, x2, x3, x4;

problem step4: objs, x1, x2, x3, x4, con1, con2, con3, con4, x1is1, x4is1;
printf "\n Step 4 x1 = 1 and x4 = 1\n";
solve step4;

display x1, x2, x3, x4;

s.t. x3is0: x3 = 0;
s.t. x3is1: x3 = 1;

problem step5: objs, x1, x2, x3, x4, con1, con2, con3, con4, x1is1, x4is0, x3is0;
printf "\n Step 5 x1 = 1 and x4 = 0 and x3 = 0\n";
solve step5;

display x1, x2, x3, x4;

problem step6: objs, x1, x2, x3, x4, con1, con2, con3, con4, x1is1, x4is0, x3is1;
printf "\n Step 6 x1 = 1 and x4 = 0 and x3 = 1\n";
solve step6;

display x1, x2, x3, x4;

var x1b binary;
var x2b binary;
var x3b binary;
var x4b binary;

maximize objsb: 90*x1b+55*x2b+63*x3b +47*x4b;

s.t. con1b: 7*x1b + 2*x2b+8*x3b+3*x4b <= 10;
s.t. con2b: x3b+x4b<=1;
s.t. con3b: x3b-x1b<=0;
s.t. con4b: x4b-x2b<=0;

problem binarySolution: objsb, x1b, x2b, x3b, x4b, con1b, con2b, con3b, con4b;
solve binarySolution;

display x1b, x2b, x3b, x4b;
```

The output is included here but is rather lengthy.



AMPL

```
ampl: model group_HW5_p5.mod

 Full LP Solution
CPLEX 20.1.0.0: optimal solution; objective 166.2857143
1 dual simplex iterations (0 in phase I)
x1 = 0.714286
x2 = 1
x3 = 0
x4 = 1


 Step 1 x1 = 0
CPLEX 20.1.0.0: optimal solution; objective 102
0 simplex iterations (0 in phase I)
x1 = 0
x2 = 1
x3 = 0
x4 = 1


 Step 2 x1 = 1
CPLEX 20.1.0.0: optimal solution; objective 160.6666667
2 dual simplex iterations (1 in phase I)
x1 = 1
x2 = 1
x3 = 0
x4 = 0.333333


 Step 3 x1 = 1 and x4 = 0
CPLEX 20.1.0.0: optimal solution; objective 152.875
1 simplex iterations (0 in phase I)
x1 = 1
x2 = 1
x3 = 0.125
x4 = 0


 Step 4 x1 = 1 and x4 = 1
presolve, variable x1:
        impossible deduced bounds: lower = 1, upper = 0.714286;
        difference = 0.285714
x1 = 1
x2 = 1
x3 = 0
x4 = 1
```


AMPL

```
 Step 3 x1 = 1 and x4 = 0
CPLEX 20.1.0.0: optimal solution; objective 152.875
1 simplex iterations (0 in phase I)
x1 = 1
x2 = 1
x3 = 0.125
x4 = 0


 Step 4 x1 = 1 and x4 = 1
presolve, variable x1:
        impossible deduced bounds: lower = 1, upper = 0.714286;
        difference = 0.285714
x1 = 1
x2 = 1
x3 = 0
x4 = 1


 Step 5 x1 = 1 and x4 = 0 and x3 = 0
CPLEX 20.1.0.0: optimal solution; objective 145
0 simplex iterations (0 in phase I)
x1 = 1
x2 = 1
x3 = 0
x4 = 0


 Step 6 x1 = 1 and x4 = 0 and x3 = 1
presolve, variable x2:
        impossible deduced bounds: lower = 0, upper = -2.5
x1 = 1
x2 = 1
x3 = 1
x4 = 0

CPLEX 20.1.0.0: optimal integer solution; objective 145
0 MIP simplex iterations
0 branch-and-bound nodes
x1b = 1
x2b = 1
x3b = 0
x4b = 0

ampl:
```