LP Homework Collection 1

September 24, 2025

1 Problem 1-2

a The best way to change the products that must equal the total hours available each week is to add a slack variable. As noticed in the question, we saw that the production plan stays the same, with the same amount of goods being produced. Once we extract the slack variables, we see that they are actually all zeros. This is true because, even in the original formulation, we noticed that our production optimum was driven mostly by our first constraint of maximizing profit, choosing according to which product was more efficient in terms of production but also profit. Next is the line of code that I changed and my output
Change:

```
var Slack {s in STAGE} >= 0;  # slack variable for each stage
subject to Time {s in STAGE}:
   sum {p in PROD} (1/rate[p,s]) * Make[p] + Slack[s] = avail[s]; #KEY CHANGE
```

Using the function from the appendix section.
Output:

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 190071.4286
2 simplex iterations
Total Profit: 190071.42857142858

Production plan (tons per product):
bands: 3357.142857142858 tons
coils: 500.0 tons
plate: 3142.857142857143 tons

Production plan as table:
          Make.val
bands   3357.142857
coils    500.000000
plate   3142.857143

Slack per stage:
        Slack.val
reheat          0
roll            0
```

b In order to represent this change you add in a parameter of max weight that is not indexed. Then you create a restriction while adding your wanted weight in your .dat file. Attached are the lines added.

```
param max_weight >= 0;
subject to Weight_Limit:
   sum {p in PROD} Make[p] <= max_weight;
param max_weight = 6500;
```

Again, using our defined function from the start

```
ampl = solve_ampl_model('steel4_b.mod', 'steel4_b.dat')
```

With the output: Output:

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 183791.6667
3 simplex iterations
Total Profit: 183791.66666666666

Production plan (tons per product):
bands: 1541.666666666666 tons
coils: 1458.333333333334 tons
plate: 3500.0 tons

Production plan as table:
          Make.val
bands  1541.666667
coils  1458.333333
plate  3500.000000
```

This new restriction actually changes how the production plan looks because now we have to think about the maximum amount we can produce of each product, not just how fast we can make them. Before, the optimizer only cared about speed, so it would focus on producing the fastest products first. But with the max weight limits, we have to balance speed with how much we can actually produce and sell. So now the production mix takes into account both how fast we can make a product and how many tons we're allowed to produce, which changes the "value-to-weight" tradeoff in the optimal plan.

c Now for this we are looking to

$$\max \sum x_i$$
$$\text{where } x_i = \text{tons produced per product i}$$

Thus, in our code for the model, we change the following only

```
maximize Total_Tons: sum {p in PROD} Make[p];
```

This results in a different optimal solution since we no longer care how much profit we can get from each. But instead, it just picks which products are most likely to maximize our production, which means it will pick the products that are the fastest to complete. The results from the code are as follows.

```
ampl = solve_ampl_model('steel4_b.mod', 'steel4_b.dat', maximize = 'Tons')
```

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 7000
0 simplex iterations
Total Tons: 7000.0

Production plan (tons per product):
bands: 5750 tons
coils: 500 tons
plate: 750 tons

Production plan as table:
       Make.val
bands      5750
coils       500
plate       750
```

d For this, I had to make changes both in the definition of our variable and what our objective function is subject to, but also changes in the data by adding an extra column. Changes:

```
.mod file
var Make {p in PROD} <= market[p];

param max_weight >= 0;
subject to Market_Share {p in PROD}:
   Make[p] >= share[p] * sum {q in PROD} Make[q];

.dat file
param:    profit  market  share:=
  bands     25     6000    0.4
  coils     30     4000    0.4
  plate     29     3500    0.1;
```

Following are the outputs

```
ampl = solve_ampl_model('steel4_d.mod', 'steel4_d.dat')

ampl = solve_ampl_model('steel4_d.mod', 'steel4_d_2.dat')
```

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 183211.9403
5 simplex iterations
Total Profit: 183211.94029850746

Production plan (tons per product):
bands: 3343.283582089552 tons
coils: 2674.626865671642 tons
plate: 668.6567164179105 tons

Production plan as table:
          Make.val
bands  3343.283582
coils  2674.626866
plate   668.656716

-----------------------------------------------------------------
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective -0
3 simplex iterations
Total Profit: 0.0
```

```

Production plan (tons per product):
bands: 0 tons
coils: 0 tons
plate: 0 tons

Production plan as table:
       Make.val
bands         0
coils         0
plate         0
```

Now, the way this changes the constraint is that we have to make different productions for our lower bounds, which in turn makes it impossible to optimize using a strategy where one certain product dominates. Now the program gives an optimal solution of zeros because if we have minimum shares that add up to 1.1, it makes no logical sense since we are trying to produce more than the maximum. Thus, if we can't produce enough, we should simply not produce, since this would satisfy the condition that the shares stay between 0 and 1.

e The following changes were made to the data: I added a finishing column in the rate parameter with very high rates for bands, which could make it essentially zero when calculating the rate at which they are needed, with a rate that only truly affects plates.

```
data;

set PROD := bands coils plate;
set STAGE := reheat roll finishing;

param rate:  reheat  roll finishing:=
  bands        200    200   1000000000000000
  coils        200    140   1000000000000000
  plate        200    160   150;


param:    profit  commit  market  :=
  bands     25     1000    6000
  coils     30      500    4000
  plate     29      750    3500  ;

param avail :=  reheat 35   roll   40   finishing 20
```

Results:

```
ampl = solve_ampl_model('steel4_e.mod', 'steel4_e.dat')
```

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 189916.6667
3 simplex iterations
Total Profit: 189916.6666666667

Production plan (tons per product):
bands: 3416.666666666667 tons
coils: 583.3333333333335 tons
plate: 3000.0 tons

Production plan as table:
```

```
          Make.val
bands  3416.666667
coils   583.333333
plate  3000.000000
```

2 Problem 1-3

a If we run the model from 1-5 we see the following

```
model = solve_ampl_model('steel3.mod', 'steel3.dat')
Time = model.getConstraint('Time')
t = Time.getValues().toPandas()
Make = model.getVariable('Make')
m = Make.getValues('rc').toPandas()
print(t)
print(m)
```

We get the following output

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 194828.5714
1 simplex iteration
Total Profit: 194828.57142857142

Production plan (tons per product):
bands: 6000.0 tons
coils: 500.0 tons
plate: 1028.571428571428 tons

Production plan as table:
          Make.val
bands  6000.000000
coils   500.000000
plate  1028.571429
   Time.dual
0       4640
        Make.rc
bands  1.800000
coils -3.142857
plate  0.000000
```

The meaning of these numbers is as follows. The marginal values associated with the Time.dual show how much our value would increase if we relaxed the assumption by a small amount; for example, if we relaxed it by one hour, we could see an increase of 4640 in profit, which means that we are binding thanks to the constraint. When it comes to our reduced costs, this is the same as the upper and lower bounds but for our marginal values with respect to the constraints. If, for example, we increase a ton in the lower bound for bands, we would increase profits by 1.8 per ton; similarly, for coils, we would see that an increase in the lower bounds would actually decrease our profits by 3.14 per ton. Strangely, for plates, it wouldn't increase or decrease. This tells us that our constraints are only binding for coils and bands, whereas plates are left without a constraint.

b This is due to the marginal cost of production changing. If we see the added time needed to complete each item as an added cost, then, due to that cost being higher for bands than plates, we substituted them in terms of our production, thus shifting our production from bands to plates.

c I ran two different versions of the same concept with two different codes. Following is the code for both parts!

```
CODE #1
model = solve_ampl_model('steel4.mod',
'steel4.dat', reheat_hours=35)
model1 = solve_ampl_model('steel4.mod', 'steel4.dat', reheat_hours=36)
model2 = solve_ampl_model('steel4.mod', 'steel4.dat', reheat_hours=37)
model3 = solve_ampl_model('steel4.mod', 'steel4.dat', reheat_hours=38)
model4 = solve_ampl_model('steel4.mod', 'steel4.dat', reheat_hours=39)
P0 = model.getObjective('Total_Profit')
P1 = model1.getObjective('Total_Profit')
P2 = model2.getObjective('Total_Profit')
P3 = model3.getObjective('Total_Profit')
P4 = model4.getObjective('Total_Profit')
P0val = P0.value()
P1val = P1.value()
P2val = P2.value()
P3val = P3.value()
P4val = P4.value()
P1_P0 = P1val - P0val
P2_P1 = P2val - P1val
P3_P2 = P3val - P2val
P4_P3 = P4val - P3val
print(f"Profit P1 - P0: {P1_P0}")
print(f"Profit P2 - P1: {P2_P1}")
print(f"Profit P3 - P2: {P3_P2}")
print(f"Profit P4 - P3: {P4_P3}")
reheat_hours = [36, 37, 38, 39]
marginal_profit = [P1_P0, P2_P1, P3_P2, P4_P3]
plt.figure(figsize=(8,5))
plt.plot(reheat_hours, marginal_profit, marker='o', linestyle='-', color='blue'
    )
plt.title("Marginal Profit vs Reheat Hours")
plt.xlabel("Reheat Hours")
plt.ylabel("Marginal Profit Increase")
plt.grid(True)
plt.show()

CODE #2
reheat_hours_values = np.arange(34, 45, .5)

profits = []
for h in reheat_hours_values:
    model = solve_ampl_model('steel4.mod', 'steel4.dat', reheat_hours=h)
    P = model.getObjective('Total_Profit')
    profits.append(P.value())

marginal_profit = np.diff(profits)
marginal_hours = reheat_hours_values[1:]


plt.figure(figsize=(8,5))
plt.plot(marginal_hours, marginal_profit, marker='o', linestyle='-', color='
    blue')
```

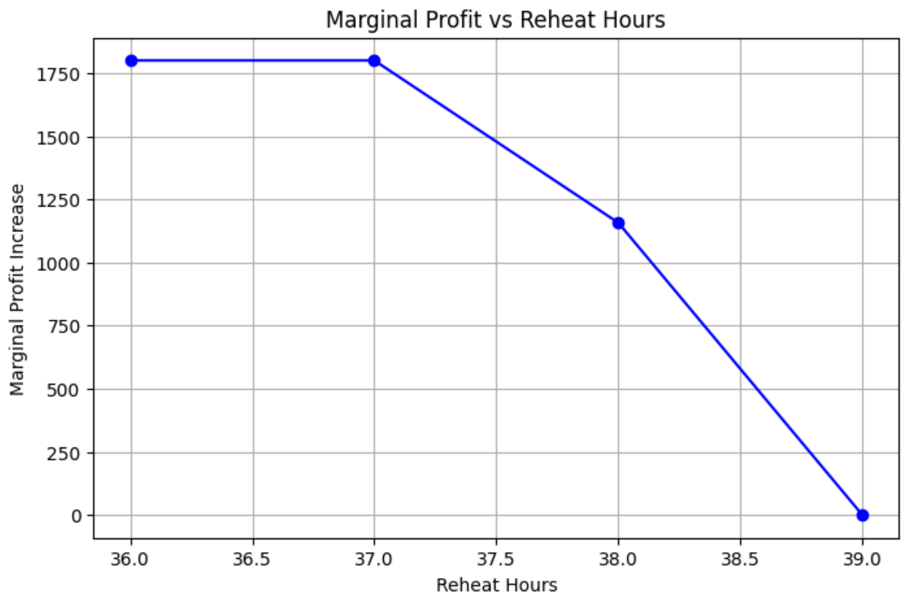

Figure 1: Smaller Interval

```
plt.title("Marginal Profit vs Reheat Hours")
plt.xlabel("Reheat Hours")
plt.ylabel("Marginal Profit Increase")
plt.grid(True)
plt.axvline(x=37.643, color='red', linestyle='--', label='x = 37 9/4')

plt.legend()
plt.show()
```

The reason I am including both codes is that the first one is the one I coded, while the second one was given to me by ChatGPT after I asked it to provide a function to plot what I did but with higher intervals so the graphs would come out clearer. The first code gives me the following output for answering the first part of (c).

```
Profit P1 - P0: 1799.999999999971
Profit P2 - P1: 1799.999999999971
Profit P3 - P2: 1157.1428571428987
Profit P4 - P3: 0.0
```

This confirms that as we go past 38, there is a smaller increase in profit, and past 39 there is no change in profit. I was also able to generate the following graphs.

d As you can see below, when the reheat time drops to 11 hours, we are simply in a different region of our polyhedron and now at a different solution set. The slope of Figure 3 is also that of the shadow price shown in Figure 4, where the slope changes as the shadow price changes. This reminds me a lot of marginal cost and its implication on time, which corresponds to this shadow cost.

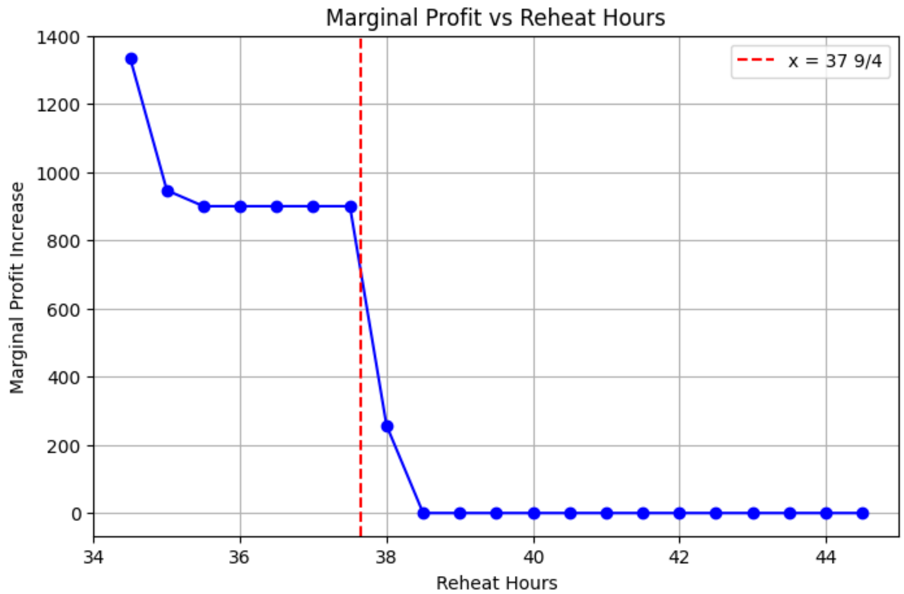

Figure 2: Higher Interval

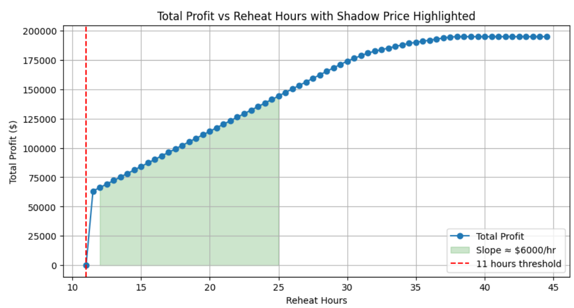

Figure 3: Shadow Price

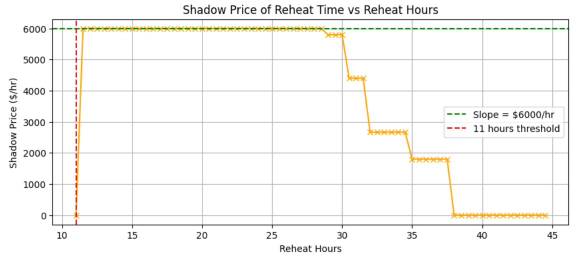

Figure 4: Reheat Hours

3 Problem 2-6

For this section, I modified my previous function slightly to allow it to be used in this situation.

```python
from amplpy import AMPL
import pandas as pd
import os

def solve_ampl_model_min(model_file, data_file, solver='gurobi', minimize='Profit',
    base_path=None):
    ampl = AMPL()
    if base_path is not None:
        model_file = os.path.join(base_path, model_file)
        data_file = os.path.join(base_path, data_file)

    # Load model and data
    ampl.eval(f"model '{model_file}';")
    ampl.eval(f"data '{data_file}';")

    # Set solver
    ampl.eval(f"option solver {solver};")

    # Solve model
    ampl.solve()

    # Print objective value
    total_obj = ampl.get_objective(f"Total_{minimize}")
    print(f"Total {minimize}: {total_obj.value()}")

    # Print variable values
    make = ampl.get_variable('Make')
    make_df = make.get_values().to_pandas()

    print("\nProduction plan:")
    print(make_df)
    return ampl
```

a For the Model

```
set W; # widths
set P; # patterns

param order {W} > 0;  # order for each width
param a {W,P} >= 0;  # number of rolls of width w in pattern p

var Make {P} >= 0; # tons produced

minimize Total_Number: sum {p in P} Make[p];
   subject to Order {w in W}:
   sum {p in P} a[w,p] * Make[p] >= order[w];
```

For the Data

```
# Data for paper mill cutting stock problem
# From Example 2-6 in AMPL Book
set W := 20 45 50 55 75;

```

```
# Orders for each width
param order :=
20 48
45 35
50 24
55 10
75 8;

# Set of cutting patterns
set P := 1 2 3 4 5 6;

# Pattern matrix: how many rolls of each width in each pattern
param a :
      1   2   3   4   5   6 :=
20    3   1   0   2   1   3
45    0   2   0   0   0   1
50    1   0   1   0   0   0
55    0   0   1   1   0   0
75    0   0   0   0   1   0 ;
```

Using this line in python

```
    model = solve_ampl_model_min('paper_mill.mod', 'paper_mill.dat', base_path=
        r'Problem 2-6 AMPL Book', minimize='Number')
```

We get the following output

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 49.5
3 simplex iterations
Total Number: 49.5

Production plan:
   Make.val
1       14.0
2       17.5
3       10.0
4        0.0
5        8.0
6        0.0
```

b I made the following changes only to the model file. Basically, I modified our restriction by changing how much we are making with these new boundaries.

The change to the model file.

```
set W; # widths
set P; # patterns

param order {W} > 0;  # order for each width
param a {W,P} >= 0;  # number of rolls of width w in pattern p

var Make {P} >= 0;

minimize Total_Number: sum {p in P} Make[p];

subject to Order_Lower {w in W}:
   sum {p in P} a[w,p] * Make[p] >= 0.9 * order[w];
```

```
subject to Order_Upper {w in W}:
sum {p in P} a[w,p] * Make[p] <= 1.4 * order[w];
```

Python:

```
model = solve_ampl_model_min('paper_mill2.mod', 'paper_mill2.dat', base_path=r'
    Problem 2-6 AMPL Book', minimize='Number')
```

Output:

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 44.55
0 simplex iterations
Total Number: 44.550000000000004

Production plan:
   Make.val
1      7.60
2     15.75
3     14.00
4      0.00
5      7.20
6      0.00
```

c In this part, I decided to add two extra patterns where I am only producing 50-inch cutouts. This is due to the high demand for these, but there is only one cutting pattern, or at most two, in each case where they are not the priority. By doing this, the solution then implemented these patterns into the optimal production strategy.

Changes were made to my .dat file while leaving the .mod file the same.

```
# Data for paper mill cutting stock problem
# From Example 2-6 in AMPL Book
set W := 20 45 50 55 75;
set P := 1 2 3 4 5 6 7;

param width_roll := 110;

# Orders for each width
param order :=
20 48
45 35
50 24
55 10
75 8;

param a :
      1   2   3   4   5   6   7 :=
20    3   1   0   2   1   3   0
45    0   2   0   0   0   1   0
50    1   0   1   0   0   0   2
55    0   0   1   1   0   0   0
75    0   0   0   0   1   0   0;
```

Then in python:

```python
og_model = solve_ampl_model_min('paper_mill.mod', 'paper_mill.dat', base_path=r'Problem 2-6 AMPL Book', minimize='Number')
model = solve_ampl_model_min('paper_mill3.mod', 'paper_mill3.dat', base_path=r'Problem 2-6 AMPL Book', minimize='Number')


total_obj_og_val = total_obj_og.value()
total_obj_model_val = total_obj_model.value()

# Compare or subtract
diff_obj = total_obj_model_val - total_obj_og_val

print(f"Original model objective: {total_obj_og_val}")
print(f"New model objective: {total_obj_model_val}")
print(f"Difference: {diff_obj}")
```

With the output

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 49.5
3 simplex iterations
Total Number: 49.5

Production plan:
   Make.val
1      14.0
2      17.5
3      10.0
4       0.0
5       8.0
6       0.0
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 46.25
4 simplex iterations
Total Number: 46.25

Production plan:
   Make.val
1      7.50
2     17.50
3     10.00
4      0.00
5      8.00
6      0.00
7      3.25
Original model objective: 49.5
New model objective: 46.25
Difference: -3.25
```

As we see in the last part, there is a difference of -3.25 rolls being used, which is a good improvement since we cut down on some waste by simply adding an extra pattern. This is because previously we were wasting a lot of material in a cutting pattern that wasn't optimal for our actual 50-inch cutouts, as we would either have a waste of 60 if we didn't need the rest of the other pattern of 20s or 50. This way, we reduced that waste to 10.

d Now, by changing our .mod file to include the constraint that the patterns made need to be integers, we get the following outputs.

Running the following code in Python for each iteration, just changing “paper mill” to whichever problem we are on.

```
#Lets run this for part a
og_model = solve_ampl_model_min('paper_mill.mod', 'paper_mill.dat', base_path=r
    'Problem 2-6 AMPL Book', minimize='Number', show_plan=False)
model = solve_ampl_model_min('paper_mill_int.mod', 'paper_mill.dat', base_path=
    r'Problem 2-6 AMPL Book', minimize='Number', show_plan=False)

total_obj_og = og_model.get_objective("Total_Number")
total_obj_model = model.get_objective("Total_Number")

total_obj_og_val = total_obj_og.value()
total_obj_model_val = total_obj_model.value()

# Compare or subtract
diff_obj = total_obj_model_val - total_obj_og_val

print(f"Original model objective: {total_obj_og_val}")
print(f"New model objective: {total_obj_model_val}")
print(f"Difference: {diff_obj}")
```

The results we get are as following.

```
#Problem a
Original model objective: 49.5
New model objective: 50.0
Difference: 0.5

#Problem b
Original model objective: 44.550000000000004
New model objective: 46.0
Difference: 1.4499999999999957

#Problem c
Original model objective: 46.25
New model objective: 47.0
Difference: 0.75
```

As we noticed, the differences aren’t massive and could be taken care of by simply rounding to the nearest value. However, in Problem B, we see the value exceed 1, which means simple rounding couldn’t fully address this issue; still, the difference might be minimal.

4 Problem 4-5

For this whole problem I still used $mysolve_a mpl_m odel$ function that I created for other problems.

a Using the following ampl model and data.

```
data;

param T := 4;
set PROD := bands coils;

param avail :=  1 40   2 40   3 32   4 40 ;
```

```

param rate :=  bands 200   coils 140 ;
param inv0 :=  bands  10   coils   0 ;

param prodcost :=  bands 10    coils  11 ;
param invcost  :=  bands  2.5  coils   3 ;

param revenue:   1     2     3     4 :=
       bands    25    26    27    27
       coils    30    35    37    39 ;

param market:    1     2     3     4 :=
       bands  6000  6000  4000  6500
       coils  4000  2500  3500  4200 ;

set PROD;     # products
param T > 0;  # number of weeks

param rate {PROD} > 0;           # tons per hour produced
param inv0 {PROD} >= 0;          # initial inventory
param avail {1..T} >= 0;         # hours available in week
param market {PROD,1..T} >= 0;  # limit on tons sold in week

param prodcost {PROD} >= 0;      # cost per ton produced
param invcost {PROD} >= 0;       # carrying cost/ton of inventory
param revenue {PROD,1..T} >= 0; # revenue per ton sold

var Make {PROD,1..T} >= 0;      # tons produced
var Inv {PROD,0..T} >= 0;       # tons inventoried
var Sell {p in PROD, t in 1..T} >= 0, <= market[p,t]; # tons sold

maximize Total_Profit:
   sum {p in PROD, t in 1..T} (revenue[p,t]*Sell[p,t] -
      prodcost[p]*Make[p,t] - invcost[p]*Inv[p,t]);

               # Total revenue less costs in all weeks

subject to Time {t in 1..T}:
   sum {p in PROD} (1/rate[p]) * Make[p,t] <= avail[t];

               # Total of hours used by all products
               # may not exceed hours available, in each week

subject to Init_Inv {p in PROD}:  Inv[p,0] = inv0[p];

               # Initial inventory must equal given value

subject to Balance {p in PROD, t in 1..T}:
   Make[p,t] + Inv[p,t-1] = Sell[p,t] + Inv[p,t];

               # Tons produced and taken from inventory
               # must equal tons sold and put into inventory
```

I ran this just with a different data file three times and attached are the outputs showing they are different.

```
Production plan as table:
               Make.val
```

```
      index0        index1
bands  1            5990
       2            6000
       3            1400
       4            2000
coils  1            1407
       2            1400
       3            3500
       4            4200

Production plan as table:
                  Make.val
index0 index1
bands  1       2285.714286
       2       4428.571429
       3       1400.000000
       4       2000.000000
coils  1       4000.000000
       2       2500.000000
       3       3500.000000
       4       4200.000000

Production plan as table:
               Make.val
index0 index1
bands  1               0
       2            6000
       3            4000
       4            6500
coils  1            5600
       2            1400
       3            1680
       4            1050
```

As we can see all of these will be different depending on the data used which changes our revenue for each item.

b Now in order to make this a stochastic model I had to change the code where you can see the following code below for the .mod

```
data;
param T := 4;
set PROD := bands coils;
param S = 3;

param avail :=  1 40  2 40  3 32  4 40 ;

param rate :=  bands 200   coils 140 ;

param inv0 :=
    bands 10
    coils 0;

param prob :=
    1 0.45
    2 0.35
    3 0.20;
```

```
param prodcost :=  bands 10    coils  11 ;
param invcost  :=  bands  2.5  coils   3 ;

param revenue :=
  [bands, 1, 1] 25
  [bands, 2, 1] 26
  [bands, 3, 1] 27
  [bands, 4, 1] 27
  [coils, 1, 1] 30
  [coils, 2, 1] 35
  [coils, 3, 1] 37
  [coils, 4, 1] 39
  [bands, 1, 2] 23
  [bands, 2, 2] 24
  [bands, 3, 2] 25
  [bands, 4, 2] 25
  [coils, 1, 2] 30
  [coils, 2, 2] 33
  [coils, 3, 2] 35
  [coils, 4, 2] 36
  [bands, 1, 3] 21
  [bands, 2, 3] 27
  [bands, 3, 3] 33
  [bands, 4, 3] 35
  [coils, 1, 3] 30
  [coils, 2, 3] 32
  [coils, 3, 3] 33
  [coils, 4, 3] 33
;

param market:    1     2     3     4 :=
       bands  6000  6000  4000  6500
       coils  4000  2500  3500  4200 ;

set PROD;      # products

param T > 0;  # number of weeks
param S > 0;

param rate {PROD} > 0;           # tons per hour produced
param inv0 {PROD} >= 0;          # initial inventory
param avail {1..T} >= 0;         # hours available in week
param market {PROD,1..T} >= 0;   # limit on tons sold in week

param prodcost {PROD} >= 0;      # cost per ton produced
param invcost {PROD} >= 0;       # carrying cost/ton of inventory
param revenue {PROD,1..T,s in 1..S} >= 0;

param prob {1..S} >= 0, <= 1;
check: 0.99999 < sum {s in 1..S} prob[s] < 1.00001;


var Make {PROD,1..T,s in 1..S} >= 0;      # tons produced
var Inv {PROD,0..T,s in 1..S} >= 0;       # tons inventoried
var Sell {p in PROD, t in 1..T, s in 1..S} >= 0, <= market[p,t]; # tons sold

maximize Total_Profit:
```

```
   sum {s in 1..S} prob[s] *
      sum {p in PROD, t in 1..T} (
         revenue[p,t,s] * Sell[p,t,s]
         -prodcost[p] * Make[p,t,s]
         - invcost[p] * Inv[p,t,s]);

subject to Time {t in 1..T,s in 1..S}:
   sum {p in PROD} (1/rate[p]) * Make[p,t,s] <= avail[t];

               # Total of hours used by all products
               # may not exceed hours available, in each week

subject to Init_Inv {p in PROD,s in 1..S}:  Inv[p,0,s] = inv0[p];

               # Initial inventory must equal given value

subject to Balance {p in PROD, t in 1..T,s in 1..S}:
   Make[p,t,s] + Inv[p,t-1, s] = Sell[p,t,s] + Inv[p,t,s];
               # Tons produced and taken from inventory
               # must equal tons sold and put into inventory
```

We then get the following output:

```
Gurobi 12.0.3:Gurobi 12.0.3: optimal solution; objective 503789.35
43 simplex iterations
Total Profit: 503789.35

Production plan (tons per product):
('bands', 1, 1): 5990.0 tons
('bands', 1, 2): 2285.714285714286 tons
('bands', 1, 3): 0.0 tons
('bands', 2, 1): 6000.0 tons
('bands', 2, 2): 4428.571428571428 tons
('bands', 2, 3): 6000.0 tons
('bands', 3, 1): 1400.0 tons
('bands', 3, 2): 1400.0 tons
('bands', 3, 3): 4000.0 tons
('bands', 4, 1): 2000.0 tons
('bands', 4, 2): 2000.0 tons
('bands', 4, 3): 6500.0 tons
('coils', 1, 1): 1407.0 tons
('coils', 1, 2): 4000.0 tons
('coils', 1, 3): 5600.0 tons
('coils', 2, 1): 1400.0 tons
('coils', 2, 2): 2500.0 tons
('coils', 2, 3): 1400.0 tons
('coils', 3, 1): 3500.0 tons
('coils', 3, 2): 3500.0 tons
...
              3       1680.000000
       4      1       4200.000000
              2       4200.000000
              3       1050.000000
```

As we can see from each product, time, ans scenario they all match up with the results we found from the previous part in our problem.

c Now adding in our nonantipacity constraints for how much we make, our inventory and the amount we sell we change just the .mod file by adding in our new constraints with the subject to argument.

```
subject to Make_na{p in PROD, s in 1..S-1}:
   Make[p,1,s] = Make[p,1,s+1];

subject to Inv_na{p in PROD, s in 1..S-1}:
   Inv[p,1,s] = Inv[p,1,s+1];

subject to Sell_na{p in PROD, s in 1..S-1}:
   Sell[p,1,s] = Sell[p,1,s+1];
```

We get the following output when runing the new code

```
Total Profit: 500740.71428571426

Production plan (tons per product):
('bands', 1, 1): 5990.0 tons
('bands', 1, 2): 5990.0 tons
('bands', 1, 3): 5990.0 tons
('bands', 2, 1): 4428.571428571428 tons
('bands', 2, 2): 4428.571428571428 tons
('bands', 2, 3): 6000.0 tons
('bands', 3, 1): 1400.0 tons
('bands', 3, 2): 1400.0 tons
('bands', 3, 3): 4000.0 tons
('bands', 4, 1): 2000.0 tons
('bands', 4, 2): 2000.0 tons
('bands', 4, 3): 6500.0 tons
('coils', 1, 1): 1407.0 tons
('coils', 1, 2): 1407.0 tons
('coils', 1, 3): 1407.0 tons
('coils', 2, 1): 2500.0 tons
('coils', 2, 2): 2500.0 tons
('coils', 2, 3): 1400.0 tons
('coils', 3, 1): 3500.0 tons
('coils', 3, 2): 3500.0 tons
...
             3       1680.000000
      4      1       4200.000000
             2       4200.000000
             3       1050.000000
```

As you can see when our, product and T = 1 we get the same amount of tons no matter the scenario. Again, this is due to us forcing this to happen with our constraints but this helps us actually have a true model since they all have the same starting point and we don't just get three different pathways since the final production is based off different initial productions stages.

d So using the following code in python I was able to estimate the expected profit made at each scenario. It is important to notice that the reason why 1 and 2 went down while 3 goes up once we changed the probabilities to make 3 almost imminent. Since we know that our profit is affected by the portability for each and since our probability only went up for 3 and down for the others we see the change, this is since it now optimizes for 3 a lot more than the other two since its more likely to happen we want to optimize scenario 3 more than anything.

```
um{p in PROD, t in 1 .. T} (revenue[p,t,s]*Sell[p,t,s] - prodcost[p]*Make[p,t,s
    ] - invcost[p]*Inv[p,t,s]) [*]
 :=
1  514090
2  461833
3  538793
;

sum{p in PROD, t in 1 .. T} (revenue[p,t,s]*Sell[p,t,s] - prodcost[p]*Make[p,t,
    s] - invcost[p]*Inv[p,t,s]) [*]
 :=
1  504493
2  459644
3  549970
;
```

5 Appendix

First, I created a function in python to load in my model changes quickly.

```python
def solve_ampl_model(model_file, data_file, solver='gurobi', maximize='Profit',
    reheat_hours=None):
    ampl = AMPL()

    # Load model and data
    ampl.eval(f"model '{model_file}';")
    ampl.eval(f"data '{data_file}';")

    # Update reheat hours if specified
    if reheat_hours is not None:
        ampl.eval(f'let avail["reheat"] := {reheat_hours};')

    # Set solver
    ampl.eval(f"option solver {solver};")

    # Solve model
    ampl.solve()

    # Print objective value
    total_obj = ampl.get_objective(f"Total_{maximize}")
    print(f"Total {maximize}: {total_obj.value()}")

    # Print variable values
    make = ampl.get_variable('Make')
    make_df = make.get_values().to_pandas()

    print("\nProduction plan (tons per product):")
    for product, value in make_df['Make.val'].items():
        print(f"{product}: {value} tons")

    print("\nProduction plan as table:")
    print(make_df)
    return ampl
```