CA4011 - Operations Research Assignment 2

Michael Wall
13522003
michael.wall22@mail.dcu.ie

Part B (Math Programming using AMPL)

Chapter 1 problems

Task 1-2 (a)

```
set PROD; # products
set STAGE; # stages

param rate {PROD,STAGE} > 0; # tons per hour in each stage
param avail {STAGE} >= 0; # hours available/week in each stage
param profit {PROD}; # profit per ton

param commit {PROD} >= 0; # lower limit on tons sold in week
param market {PROD} >= 0; # upper limit on tons sold in week

var Make {p in PROD} >= commit[p], <= market[p]; # tons produced

maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];

        # Objective: total profits from all products

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

        # In each stage: total of hours used by all
        # products may not exceed hours available
```

The change required is to the last line of code (excluding comments). We change the <= to a == operator. There is no change in the results because the total hours used already is equal to the total hours available in a given stage.

Task 1-2 (b)

```
set PROD; # products
set STAGE; # stages

param max_weight > 0;

param rate {PROD,STAGE} > 0; # tons per hour in each stage
param avail {STAGE} >= 0; # hours available/week in each stage
param profit {PROD}; # profit per ton

param commit {PROD} >= 0; # lower limit on tons sold in week
param market {PROD} >= 0; # upper limit on tons sold in week

var Make {p in PROD} >= commit[p], <= market[p]; # tons produced

maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];

        # Objective: total profits from all products

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

        # In each stage: total of hours used by all
        # products may not exceed hours available

subject to Weight_Lim:
    sum {p in PROD} Make[p] <= max_weight;
```

The required change is to introduce a new parameter (line 3 above), and a new condition (last 2 lines) to ensure the total tonnes produced does not exceed the max_weight. A new line in the data file must be added, as follows:

```
...
param max_weight := 6500 ;
...
```

This changes the results from:

```
:      Make.lb    Make   Make.ub    :=
bands   1000    3357.14   6000
coils    500     500      4000
plate    750    3142.86   3500
;

Total_Profit = 190071
```

To the following with weight limit of 6500:

```
:      Make.lb    Make   Make.ub    :=
bands   1000    1541.67   6000
coils    500    1458.33   4000
plate    750    3500      3500
;

Total_Profit = 183792
```

The reason for this is due to the plates being more profitable to create given time and weight constraints, so maximum output is produced for these. After this, bands and coils are created to maximize the profit within the constraints of the system.

Task 1-2 (c)

```
set PROD; # products
set STAGE; # stages

param rate {PROD,STAGE} > 0; # tons per hour in each stage
param avail {STAGE} >= 0; # hours available/week in each stage
param profit {PROD}; # profit per ton

param commit {PROD} >= 0; # lower limit on tons sold in week
param market {PROD} >= 0; # upper limit on tons sold in week

var Make {p in PROD} >= commit[p], <= market[p]; # tons produced

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

        # Objective: total profits from all products

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

        # In each stage: total of hours used by all
        # products may not exceed hours available
```

The required change is to adjust the maximize function on line 9 above to maximize the total sum of the weight of products produced. No changes are made to the data file. The effect on production is as follows, from the original:

```
:      Make.lb    Make    Make.ub    :=
bands   1000     3357.14   6000
coils    500      500      4000
plate    750     3142.86   3500
;
```

To the following productions:

```
:      Make.lb   Make  Make.ub    :=
bands   1000     5750   6000
coils    500      500   4000
plate    750      750   3500
;
```

This hits the maximum production possible of 7000 tonnes. The reason for this is that we produce the heaviest item with the fastest production time first, however we still need to produce some coils and plates, which is why bands are not produced to full market demand.

Task 1-2 (d)

```
set PROD; # products
set STAGE; # stages

param rate {PROD,STAGE} > 0; # tons per hour in each stage
param avail {STAGE} >= 0; # hours available/week in each stage
param profit {PROD}; # profit per ton
param share {PROD}; # min share of product # lower limit
param market {PROD} >= 0; # upper limit on tons sold in week

var Make {p in PROD} <= market[p]; # tons produced

maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];

        # Objective: total profits from all products

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

        # In each stage: total of hours used by all
        # products may not exceed hours available

subject to Share_Limits {p in PROD}:
    Make[p] >= share[p] * (sum {px in PROD} Make[px]);
```

The required change is to add a new param (line 6) for shares, and a new condition (last 2 lines) to ensure that the share of the item produced meets a certain percentage. In the data file we must adjust the section setting params for products to the following:

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

This gives the following changes in the results from the original:

```
:      Make.lb    Make    Make.ub    :=
bands   1000    3357.14   6000
coils    500     500      4000
plate    750    3142.86   3500
;

Total_Profit = 190071
```

To the following productions:

```
:      share   Make Make.ub    :=
bands    0.4   3500   6000
coils    0.1    700   4000
plate    0.4   2800   3500
;

Total_Profit = 189700
```

If we change the minimum shares for bands and plates to 0.5 and the minimum share for coils to 0.1, we get an infeasible problem. It is not possible to satisfy all of the constraints set forward by the model, as the share of production cannot be greater than 1.0 total.

Task 1-2 (e)

```
set PROD := bands coils plate;
set STAGE := reheat roll fin;
param rate: reheat roll fin :=
bands 200 200 .
coils 200 140 .
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 fin 20 ;
```

Above are the changes to the data file which need to be made to implement this change. We add a new finishing stage called fin, and specify a rate and availability for the stage. For bands and coils, we specify that no value is available by using a full stop "." instead of a zero or space.

Task 1-4 (a)

The data values for this problem are as follows:

- The car types to produce
- The rate of production for each car
- The profit for each car
- The minimum demand for each car
- The maximum factory time for the week of production

I would declare these as follows in AMPL:

```
set CARS; # car types
param rate {CARS} > 0; # cars per hour
param profit {CARS}; # profit per car
param commit {CARS} >= 0; # lower limit on car demands
param max_hours > 0; # max factory hours
```

The decision variables are:

- Amount of each car to produce

I would declare these as follows in AMPL

```
var Make {c in CARS} >= commit[c]; # cars produced
```

Task 1-4 (b)

I would declare objective and constraints as follows:

```
# objective function
maximize Total_Profit: sum {c in CARS} profit[c] * Make[c];

# constraints
subject to Time:
    max_hours >= sum {c in CARS} (rate[c]) * Make[c];
```

Task 1-4 (c)

The following production results were obtained for each car:

```
: Make.lb   Make   Make.ub     :=
c    20     32.5   Infinity
l    15     15     Infinity
t    10     10     Infinity
;
Total_Profit = 28750
```

Task 1-4 (d)

If you maximise car production you will create 70 cars, 12.5 more and a profit of 27500 which is 1250 less.

Task 1-4 (e)

The new expanded model is as follows:

```
set CARS; # car types
param rate {CARS} > 0; # cars per hour
param profit {CARS}; # profit per car
param commit {CARS} >= 0; # lower limit on car demands
param fuel {CARS} >= 0; # fuel efficiency of car
param max_hours > 0; # max factory hours
param min_effic > 0; # min ave fuel efficiency

var Make {c in CARS} >= commit[c]; # cars produced
var Eff == (sum {c in CARS} Make[c] * fuel[c])/(sum {cx in CARS}
Make[cx]);
# objective function
maximize Total_Profit: sum {c in CARS} profit[c] * Make[c];

# constraints
subject to Time:
max_hours >= sum {c in CARS} (rate[c]) * Make[c];

subject to Effic:
min_effic * sum {cx in CARS} Make[cx] <= sum {c in CARS} (fuel[c])
* Make[c];
```

Task 1-4 (f)

With the efficiencies specified, and average efficiency of 35, the optimal production values are as follows:

```
: Make.lb    Make     Make.ub     :=
c    20     25.7143   Infinity
l    15     15        Infinity
t    10     23.5714   Infinity
;

Total_Profit = 28071.4
Eff = 35
```

Instead of producing fractions of cars, you might decide to produce a whole number of the car which is fastest to produce.

With 10 extra hours of production time, the profit is lower by 750 than if we did not have a fuel efficiency constraint.

Task 1-4 (g)

The model could have a new set section and availability parameter to account for the different stages in producing a car, such as part manufacturing, assembly and testing.

Chapter 2 problems

Task 2-2 (a)

Given the task set out in 2-2-a, I produced the following model:

```
set EXER; # set of exercises
param cals {EXER} > 0; # calories burnt per hour exercise
param tol {EXER} > 0; # tolerance for each exercise
param cal_target > 0; # target calories to burn

var Work {e in EXER} <= tol[e], >= 0;
# perform work up to tolerance of each exercise
var Burnt {e in EXER} == cals[e] * Work[e];
# objective function
minimize Total_Time: sum {e in EXER} Work[e];
# constraints
subject to CalorieTarget:
    cal_target <= sum {e in EXER} cals[e] * Work[e];
```

And the following data:

```
set EXER := walking jogging swimming machine indoor pushback;
param: cals tol :=
walking  100 5
jogging  200 2
swimming 300 3
machine  150 3.5
indoor   300 3
pushback 500 0.5 ;
param cal_target := 2000 ;
```

This produced the following results:

```
:           Work   Work.ub Burnt    :=
indoor     2.83333   3      850
jogging    0         2        0
machine    0         3.5      0
pushback   0.5       0.5    250
swimming   3         3      900
walking    0         5        0
;
Total_Time = 6.33333
```

Task 2-2 (b)

I set up the new constraints for task b with the following model:

```
set EXER; # set of exercises
param cals {EXER} > 0; # calories burnt per hour exercise
param tol {EXER} > 0; # tolerance for each exercise
param min_time {EXER} >= 0; # minimum time for each exercise
param cal_target > 0; # target calories to burn

var Work {e in EXER} <= tol[e], >= min_time[e];
# perform work up to tolerance of each exercise
var Burnt {e in EXER} == cals[e] * Work[e];
# objective function
minimize Total_Time: sum {e in EXER} Work[e];
# constraints
subject to CalorieTarget:
    cal_target <= sum {e in EXER} cals[e] * Work[e];
subject to Variety:
    4 >= Work['walking'] + Work['jogging'] + Work['machine']
```

And the following data:

```
set EXER := walking jogging swimming machine indoor pushback;
param: cals tol min_time :=
walking  100 5 1
jogging  200 2 1
swimming 300 3 1
machine  150 3.5 1
indoor   300 3 0
pushback 500 0.5 0 ;
param cal_target := 2000 ;
```

This yielded the following results:

```
:         Work.lb    Work    Work.ub Burnt     :=
indoor       0     1.33333    3      400
jogging      1     1          2      200
machine      1     1          3.5    150
pushback     0     0.5        0.5    250
swimming     1     3          3      900
walking      1     1          5      100
;
Total_Time = 7.83333
```

Task 2-3 (a)

The following model is what I created for task a:

```
set SUPP; # set of suppliers
param cane {SUPP} >= 0, <= 100; # percentage of cane
param corn {SUPP} >= 0, <= 100; # percentage of corn
param beet {SUPP} >= 0, <= 100; # percentage of beet
param cost {SUPP} >= 0; # cost per ton of sugar mix
param cane_target >= 0; # target volume of cane sugar
param corn_target >= 0; # target volume of corn sugar
param beet_target >= 0; # target volume of beet sugar
var Buy {s in SUPP} >= 0; # amount to buy from each supplier
var Cost {s in SUPP} == cost[s] * Buy[s];
# objective function
minimize Total_Cost: sum {s in SUPP} cost[s] * Buy[s];
# constraints
subject to CaneTarget:
    cane_target == sum {s in SUPP} (cane[s]/100) * Buy[s];
subject to CornTarget:
    corn_target == sum {s in SUPP} (corn[s]/100) * Buy[s];
subject to BeetTarget:
    beet_target == sum {s in SUPP} (beet[s]/100) * Buy[s];
```

I used the following data for the task as well:

```
set SUPP := A B C D E F G;
param: cane corn beet cost :=
A 10 30 60 10
B 10 40 50 11
C 20 40 40 12
D 30 20 50 13
E 40 60 00 14
F 20 70 10 12
G 60 10 30 15 ;

param cane_target := 52 ;
param corn_target := 56 ;
param beet_target := 59 ;
```

This gave the following results achieving a 52/56/59 ton mix of sugars:

```
:   Buy    Cost     :=
A   60     600
B    0       0
C    0       0
D    0       0
E    0       0
F   45.5   546
G   61.5   922.5
;
Total_Cost = 2068.5
```

Task 2-3 (b)

Only a small change was needed to adjust the model. We introduce a new parameter to represent the minimum purchase from each supplier. We also adjust the Buy variable so that it is greater or equal to the new minimum purchase parameter:

```
...
param min_buy >= 0;
var Buy {s in SUPP} >= min_buy; # amount to buy from each supplier
...
```

For our data file, we only need to specify the new value for our minimum purchase at the end of the file:

```
...
param min_buy := 10 ;
...
```

These changes have the following results while still achieving the 52/56/59 mix of sugars:

```
:     Buy      Cost      :=
A   43.6364   436.364
B   10        110
C   10        120
D   10        130
E   10        140
F   30.9545   371.455
G   52.4091   786.136
;
Total_Cost = 2093.95
```

Task 2-3 (c)

The new model is as follows:

```
set SUPP; # set of suppliers
param cane {SUPP} >= 0, <= 100; # percentage of cane
param corn {SUPP} >= 0, <= 100; # percentage of corn
param beet {SUPP} >= 0, <= 100; # percentage of beet
param cost {SUPP} >= 0; # cost per ton of sugar mix
param low_per >= 0;
param high_per <= 100;
param max_per == 100;
var Buy {s in SUPP} >= 0; # amount to buy from each supplier
var Cost {s in SUPP} == cost[s] * Buy[s];
var Cane_per == sum {s in SUPP} (cane[s]/100) * Buy[s];
var Corn_per == sum {s in SUPP} (corn[s]/100) * Buy[s];
var Beet_per == sum {s in SUPP} (beet[s]/100) * Buy[s];

# objective function
minimize Total_Cost: sum {s in SUPP} cost[s] * Buy[s];
# constraints
subject to MaxVolume:
    1 == sum {s in SUPP} Buy[s];
subject to CorrectPercentages:
    1 == Cane_per + Corn_per + Beet_per;
subject to CaneRange:
    low_per/100 <= Cane_per <= high_per/100;
subject to CornRange:
    low_per/100 <= Corn_per <= high_per/100;
subject to BeetRange:
    low_per/100 <= Beet_per <= high_per/100;
```

The new data set is as follows:

```
set SUPP := A B C D E F G;
param: cane corn beet cost :=
A 10 30 60 10
B 10 40 50 11
C 20 40 40 12
D 30 20 50 13
E 40 60 00 14
F 20 70 10 12
G 60 10 30 15 ;
param low_per := 30;
param high_per := 37;
```

This yields the following results with the percentage of each sugar in the new mix along with the minimum cost:

```
:   Buy    Cost    :=
A   0.4    4
B   0      0
C   0      0
D   0      0
E   0      0
F   0.25   3
G   0.35   5.25
;
Total_Cost = 12.25
Cane_per = 0.3
Corn_per = 0.33
Beet_per = 0.37
```

Chapter 2 problems

Task 3-2 (a)

The data for this task was as follows, in alignment with the model specified:

```
param: ORIG: supply := # machines
M1 80
M2 30
M3 160 ;
param: DEST: demand := # parts
P1 10
P2 40
P3 60
P4 20
P5 20
P6 30 ;
param cost:
P1 P2 P3 P4 P5 P6 := # costs
M1 3 3 2 5 2 1
M2 4 1 1 2 2 1
M3 2 2 5 1 1 2 ;
```

This was not enough for the analysis to run with the original model. Some small changes had to be made, to lines 5, 11 and 13 which are shown below. The changes are due to the supply capacity of the machine being greater than the demand for parts:

```
set ORIG; # origins
set DEST; # destinations
param supply {ORIG} >= 0; # amounts available at origins
param demand {DEST} >= 0; # amounts required at destinations
check: sum {i in ORIG} supply[i] >= sum {j in DEST} demand[j];
param cost {ORIG,DEST} >= 0; # shipment costs per unit
var Trans {ORIG,DEST} >= 0; # units to be shipped
minimize Total_Cost:
sum {i in ORIG, j in DEST} cost[i,j] * Trans[i,j];
subject to Supply {i in ORIG}:
sum {j in DEST} Trans[i,j] <= supply[i];
subject to Demand {j in DEST}:
sum {i in ORIG} Trans[i,j] >= demand[j];
```

This resulted in the following transport table result, and the minimum total cost of 260:

```
Trans :=
M1 P3   50
M1 P6   30
M2 P2   20
M2 P3   10
M3 P1   10
M3 P2   20
M3 P4   20
M3 P5   20
;
Total_Cost = 260
```

Task 3-2 (b)

Increasing the capacity of machine 2 to 50 reduces the total cost by 20 by allowing the higher demand parts to be supplied by the cheapest production machine (notably parts 2 and 3). No further changes were needed for the model as these changes were already made in part (a) of this task. The results of this task for production are as follows:

```
Trans :=
M1 P3   50
M1 P6   30
M2 P2   40
M2 P3   10
M3 P1   10
M3 P4   20
M3 P5   20
;
Total_Cost = 240
```

Task 3-2 (c)

By limiting capacity in hours we need to make some adjustments to the model to accomodate this. The changes include limiting the supply constraint by multiplying the time to produce a part by the amount produced. This must be less than the supply of hours available. The new model is as follows:

```
set ORIG; # origins
set DEST; # destinations
param supply {ORIG} >= 0; # amounts available at origins
param demand {DEST} >= 0; # amounts required at destinations
check: sum {i in ORIG} supply[i] >= sum {j in DEST} demand[j];
param time {ORIG,DEST} >= 0; # time to produce a part at a machine
param cost {ORIG,DEST} >= 0; # shipment costs per unit
var Trans {ORIG,DEST} >= 0; # units to be shipped
var TimeUsed {ORIG} >= 0; # time used on each machine
minimize Total_Cost:
sum {i in ORIG, j in DEST} cost[i,j] * Trans[i,j];
subject to Timing {i in ORIG}:
sum {j in DEST} time[i,j] * Trans[i,j] == TimeUsed[i];
subject to Supply {i in ORIG}:
sum {j in DEST} Trans[i,j] * time[i,j] <= supply[i];
subject to Demand {j in DEST}:
sum {i in ORIG} Trans[i,j] >= demand[j];
```

This task also required adjusting the supply data to reflect the capacity in hours, and adding new data to specify the time to create a part, with the data for cost of a part unchanged. These changes and additions are as follows:

```
param: ORIG: supply := # defines set "ORIG" and param "supply"
M1 50
M2 90
M3 175 ;
...
...
param time:
P1 P2 P3 P4 P5 P6 :=
M1 1.3 1.3 1.2 1.5 1.2 1.1
M2 1.4 1.1 1.1 1.2 1.2 1.1
M3 1.2 1.2 1.5 1.1 1.1 1.2 ;
```

The results of limiting capacity by time result in the following productions of parts. The optimal solution has a lower cost associated, but since parts are produced in uneven time slots, percentages of parts are produced for certain machines. In reality this would mean these parts are not fully complete. Both machine 1 and 2 use all of their available time slots. The minimum cost possible is also noted below:

```
Trans :=
M1 P3   14.1667
M1 P6   30
M2 P2   35.9848
M2 P3   45.8333
M3 P1   10
M3 P2    4.01515
M3 P4   20
M3 P5   20
;
TimeUsed [*] :=
M1  50
M2  90
M3  60.8182
;
Total_Cost = 208.182
```

Task 3-2 (d)

We can change the objective function to get a minimum time as follows:

```
...
minimize Total_Time:
sum {i in ORIG, j in DEST} time[i,j] * Trans[i,j];
...
```

This gives the total time of 200.818 hours.

Task 3-3 (a)

The new model required two new params, and two new constraints as follows:

```
...
param supply_pct >= 0, <= 100;
param demand_pct >= 0, <= 100;
...
subject to SupplyPct {i in ORIG, j in DEST}:
Trans[i,j] <= (supply_pct/100) * supply[i];
subject to DemandPct {j in DEST, i in ORIG}:
Trans[i,j] <= (demand_pct/100) * demand[j];
...
```

The data required two new params as follows:

```
...
param supply_pct := 50;
param demand_pct := 85;
```

The original results are as follows:

```
Trans :=
CLEV DET   1200
CLEV LAF    400
CLEV LAN    600
CLEV WIN    400
GARY FRE   1100
GARY LAF    300
PITT FRA    900
PITT LAF    300
PITT STL   1700
;
Total_Cost = 196200
```

The new results are as follows, with an increased total cost:

```
CLEV DET   1020
CLEV FRA    135
CLEV FRE    400
CLEV LAF    195
CLEV LAN    510
CLEV WIN    340
GARY FRE    700
GARY LAF    445
GARY STL    255
PITT DET    180
PITT FRA    765
PITT LAF    360
PITT LAN     90
PITT STL   1445
PITT WIN     60
;
Total_Cost = 199210
```

Task 3-3 (b)

The new model for this task required a few changes, and the new model is as follows:

```
set BEGIN; # plant
set ORIG; # mill
set DEST; # destinations
param mat_supply {BEGIN} >= 0; # amount of material at plants
param supply {ORIG} >= 0; # amounts available at origins
param demand {DEST} >= 0; # amounts required at destinations
param supply_pct >= 0, <= 100;
param demand_pct >= 0, <= 100;
check: sum {h in BEGIN} mat_supply[h] == sum {i in ORIG} supply[i]
== sum {j in DEST} demand[j];
param mat_cost {BEGIN,ORIG} >= 0; # shipment costs per unit from
plant
param cost {ORIG,DEST} >= 0; # shipment costs per unit to dest
var Mat_Trans {BEGIN,ORIG} >= 0; # amount for the plant to ship to
factory
var Trans {ORIG,DEST} >= 0; # units to be shipped

minimize Total_Cost:
sum {h in BEGIN, i in ORIG} mat_cost[h, i] * Mat_Trans[h, i] +
sum {i in ORIG, j in DEST} cost[i,j] * Trans[i,j];

subject to MaterialsSupply {h in BEGIN}:
sum {i in ORIG} Mat_Trans[h,i] == mat_supply[h];
subject to MaterialsDemand {i in ORIG}:
sum {h in BEGIN} Mat_Trans[h,i] == supply[i];

subject to Supply {i in ORIG}:
sum {j in DEST} Trans[i,j] == supply[i];
subject to Demand {j in DEST}:
sum {i in ORIG} Trans[i,j] == demand[j];

subject to SupplyPct {h in BEGIN, i in ORIG}:
Mat_Trans[h,i] <= (supply_pct/100) * mat_supply[h];
subject to DemandPct {h in BEGIN, i in ORIG}:
Mat_Trans[h,i] <= (demand_pct/100) * supply[i];
```

The new data is specified as follows:

```
param: BEGIN: mat_supply :=
MIDTWN 2700
HAMLTN 4200 ;
param: ORIG: supply := # defines set "ORIG" and param "supply"
GARY 1400
CLEV 2600
PITT 2900 ;
param: DEST: demand := # defines "DEST" and "demand"
FRA 900
DET 1200
LAN 600
WIN 400
STL 1700
FRE 1100
LAF 1000 ;
param mat_cost:
GARY CLEV PITT :=
MIDTWN 12 8 17
HAMLTN 10 5 13 ;
param cost:
FRA DET LAN WIN STL FRE LAF :=
GARY 39 14 11 14 16 82 8
CLEV 27 9 12 9 26 95 17
PITT 24 14 17 13 28 99 20 ;
param supply_pct := 50;
param demand_pct := 85;
```

This results in the following transport tables for materials and to destinations, with the total shipping cost also:

```
Mat_Trans :=
HAMLTN CLEV   1890
HAMLTN GARY    210
HAMLTN PITT   2100
MIDTWN CLEV    710
MIDTWN GARY   1190
MIDTWN PITT    800
;
Trans :=
CLEV DET   1200
CLEV LAF    400
CLEV LAN    600
CLEV WIN    400
GARY FRE   1100
GARY STL    300
PITT FRA    900
PITT LAF    600
PITT STL   1400
;
Total_Cost = 268610
```

Task 3-3 (c)

The production costs (if fixed per factory and per mill), could be supplied as a parameter to each set individually. When specifying the objective function, the cost would be the supply sent multiplied by the production fraction (say if each ton cost X to produce at a mill, and Y to produce at a factory). The new function might look like this (where prod_cost is the parameter that specifies the cost for a factory to produce a ton of material:

```
...
param prod_cost {BEGIN} >= 0;
minimize Total_Cost:
sum {h in BEGIN, i in ORIG} prod_cost[h] * Mat_Trans[h, i] + sum
{h in BEGIN, i in ORIG} mat_cost[h, i] * Mat_Trans[h, i]......
```

The data specifying these production costs may look like this:

```
...
param: BEGIN: mat_supply, prod_cost :=
MIDTWN 2700 32
HAMLTN 4200 17 ;
```

The same procedure would follow for production costs at ORIG as well.

Task 3-3 (d)

The new model would account for scrap loss using a parameter called “scrap_pct” which is a percentage of steel lost to scrap. This percentage is used to increase the demand of the rolling mill so that they will have enough steel to meet customer demand after the scrap loss. The model accounting for scrap loss would look like this:

```
set BEGIN; # plant
set ORIG; # mill
set DEST; # destinations
param mat_supply {BEGIN} >= 0; # amount of material at plants
param supply {ORIG} >= 0; # amounts available at origins
param demand {DEST} >= 0; # amounts required at destinations
param supply_pct >= 0, <= 100;
param demand_pct >= 0, <= 100;
check: sum {h in BEGIN} mat_supply[h] >= sum {i in ORIG} supply[i]
<= sum {j in DEST} demand[j];
param mat_cost {BEGIN,ORIG} >= 0; # shipment costs per unit from
plant
param cost {ORIG,DEST} >= 0; # shipment costs per unit to dest
var Mat_Trans {BEGIN,ORIG} >= 0; # amount for the plant to ship to
factory
var Trans {ORIG,DEST} >= 0; # units to be shipped
param scrap_pct {ORIG} >= 0, <= 100;
minimize Total_Cost:
sum {h in BEGIN, i in ORIG} mat_cost[h, i] * Mat_Trans[h, i] +
sum {i in ORIG, j in DEST} cost[i,j] * Trans[i,j];
subject to MaterialsSupply {h in BEGIN}:
sum {i in ORIG} Mat_Trans[h,i] == mat_supply[h];
subject to MaterialsDemand {i in ORIG}:
sum {h in BEGIN} Mat_Trans[h,i] == supply[i] * (100 -
scrap_pct)/100;
subject to Supply {i in ORIG}:
sum {j in DEST} Trans[i,j] == supply[i] * (100 - scrap_pct)/100;
subject to Demand {j in DEST}:
sum {i in ORIG} Trans[i,j] == demand[j];
subject to SupplyPct {h in BEGIN, i in ORIG}:
Mat_Trans[h,i] <= (supply_pct/100) * mat_supply[h];
subject to DemandPct {h in BEGIN, i in ORIG}:
Mat_Trans[h,i] <= (demand_pct/100) * supply[i] * (100 -
scrap_pct)/100;
```

Task 3-3 (e)

Assuming a constant value for the recycling value of scrap, the new model to account for scrap selling only requires the addition of one new parameter and an adjustment to the objective function at the end:

```
...
param scrap_val >= 0;
minimize Total_Cost:
sum {h in BEGIN, i in ORIG} mat_cost[h, i] * Mat_Trans[h, i] +
sum {i in ORIG, j in DEST} cost[i,j] * Trans[i,j] - sum {i in
ORIG} (supply[i] * scrap_pct[i]/100 * scrap_val);
...
```

We subtract the value of the scrap percentage of a mill's total supply from our costs, and everything else can stay the same.