# Problem 1

## R-script

require('XLConnect') # install if necessary

# also install XLConnectJars if not installed as dependency

# read dataframes from Excel spreadsheet

# you'll have to change the sheets and regions

dfFactories <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='A2:A6')

dfWarehouse <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='B2:B11')

dfStores <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='C2:C621')

dfProducts <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='D2:D36')

#dfMaxUnitsFW <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='E2')

#dfMaxUnitsWS <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='E5')

#dfMaxUnitsStored <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='E8')

dfTableRoutesFW <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='G3:J71')

dfTableRoutesWS <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='L3:O202')

dfSupply <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='Q3:S15')

dfDemand <- readWorksheetFromFile('transpStoreBigSheet.xlsx',sheet=1,region='V3:X102')

# set to your directory or use Session -> Set Working directory from the menu in RStudio to set directory

setwd('C:/myscripts/school/ds775/wk4') # set to your project directory

filename <- 'transpStore.dat' # give your file a name

cat('//--------------------------------------------------\n',file=filename)

con <- file(filename) # get connection to file to close later

cat('// transportation file produced by reading \n',file=filename,append=T)

cat('// excel file into R using XLConnect \n',file=filename,append=T)

cat('// and using R commands to output textfile \n',file=filename,append=T)

cat('//--------------------------------------------------\n\n',file=filename,append=T)

# some helper functions for writing in the right formats for .dat file

writeSet <- function( varName, df, filename ){

cat( varName, file = filename, append = T )

cat( ' = { ', file = filename, append = T )

numRows <- dim(df)[1]

for (j in 1:numRows){

cat( df[j,1], file = filename, append = T )

cat( ' ', file = filename, append = T )

}

cat('};\n', file = filename, append = T )

}

writeNumber <- function( varName, x, filename ){

cat( varName, file = filename, append = T)

cat( ' = ', file = filename, append = T)

cat( x, file = filename, append = T)

cat( ';\n', file = filename, append = T)

}

writeTuplesTable <- function( varName, df, filename){

cat( varName, file = filename, append = T)

cat( ' = {\n', file = filename, append = T)

numRows <- dim( df )[1]

for (j in 1:numRows){

cat(' < ', file = filename, append = T)

write.table( df[j,], file = filename, append = T, quote = F, sep = ', ',

row.names = F, col.names = F, eol = ' >\n' )

}

cat('};\n\n', file = filename, append = T)

}

writeTuplesArray <- function( varName, df, filename){

cat( varName, file = filename, append = T)

cat( ' = #[\n', file = filename, append = T)

numRows <- dim( df )[1]

numCols <- dim( df )[2]

tupleLen <- numCols - 1

for (j in 1:numRows){

cat(' < ', file = filename, append = T)

write.table( df[j,1:tupleLen], file = filename, append = T, quote = F, sep = ', ',

row.names = F, col.names = F, eol = ' >: ' )

cat( df[j,numCols] , file = filename, append = T )

cat( '\n', file = filename, append = T )

}

cat(']#;\n\n', file = filename, append = T)

}

# write file

writeSet('Factories', dfFactories, filename)

writeSet('Warehouse', dfWarehouse, filename)

writeSet('Stores', dfStores, filename)

writeSet('Products',dfProducts,filename)

writeNumber('MaxUnitsFW', 1000, filename)

writeNumber('MaxUnitsWS', 800, filename)

writeNumber('MaxUnitsStored', 2300, filename)

writeTuplesTable('TableRoutesFW', dfTableRoutesFW, filename)

writeTuplesTable('TableRoutesWS', dfTableRoutesWS, filename)

writeTuplesArray('Supply', dfSupply, filename)

writeTuplesArray('Demand', dfDemand, filename)

close(con) # close connection to file

## Model

{string} Factories = ...;

{string} Warehouses = ...;

{string} Stores = ...;

{string} Products = ...;

float CapacityFW = ...;

float CapacityWS = ...;

float MaxStorage = ...;

tuple tableRoutesType {

string p;

string o;

string d;

float cost;

}

{tableRoutesType} TableRoutesFW = ...;

{tableRoutesType} TableRoutesWS = ...;

tuple connection {

string o;

string d;

}

tuple route {

string p;

connection e;

}

{route} RoutesFW = { < p,<o,d> > | <p,o,d,c> in TableRoutesFW };

{route} RoutesWS = { < p,<o,d> > | <p,o,d,c> in TableRoutesWS };

{connection} ConnectionsFW = { c | <p,c> in RoutesFW };

{connection} ConnectionsWS = { c | <p,c> in RoutesWS };

tuple supply{

string p;

string o;

}

{supply} Suppliers = { <p,c.o> | <p,c> in RoutesFW };

float Supply[Suppliers] = ...;

tuple customer {

string p;

string d;

}

{customer} Customers = { <p,c.d> | <p,c> in RoutesWS };

float Demand[Customers] = ...;

float CostFW[RoutesFW] = [ <t.p,<t.o,t.d>>:t.cost | t in TableRoutesFW ];

float CostWS[RoutesWS] = [ <t.p,<t.o,t.d>>:t.cost | t in TableRoutesWS ];

{string} Orig[p in Products] = { c.o | <p,c> in RoutesFW };

{string} Ware[p in Products] = { c.d | <p,c> in RoutesFW };

{string} Dest[p in Products] = { c.d | <p,c> in RoutesWS };

{connection} CPsFW[p in Products] = { c | <p,c> in RoutesFW };

{connection} CPsWS[p in Products] = { c | <p,c> in RoutesWS };

assert forall( p in Products )

sum( o in Orig[p] )

Supply[<p,o>] == sum( d in Dest[p] ) Demand[<p,d>];

dvar float+ TransFW[RoutesFW];

dvar float+ TransWS[RoutesWS];

dexpr float StoreW[w in Warehouses] =

sum(p in Products) sum(<o,w> in CPsFW[p]) TransFW[<p,<o,w>>];

constraint ctSupply[Products][Factories];

constraint ctDemand[Products][Stores];

constraint ctConnect[Products][Warehouses];

minimize

sum( r in RoutesFW) CostFW[r] \* TransFW[r] +

sum( r in RoutesWS) CostWS[r] \* TransWS[r];

subject to {

forall( p in Products , o in Orig[p] )

ctSupply[p][o]:

sum(<o, w> in CPsFW[p])

TransFW[<p, <o, w>>] == Supply[<p, o>];

forall( p in Products , d in Dest[p] )

ctDemand[p][d]:

sum(<o, d> in CPsWS[p])

TransWS[<p, <o, d>>] == Demand[<p, d>];

forall( p in Products, w in Ware[p])

ctConnect[p][w]:

sum( <o, w> in CPsFW[p]) TransFW[<p, <o, w>>] ==

sum( <w, d> in CPsWS[p]) TransWS[<p, <w, d>>];

forall( c in ConnectionsFW )

ctCapacityFW:

sum(<p, c> in RoutesFW)

TransFW[<p, c>] <= CapacityFW;

forall( c in ConnectionsWS )

ctCapacityWC:

sum(<p, c> in RoutesWS)

TransWS[<p, c>] <= CapacityWS;

forall( w in Warehouses )

ctStorage:

sum(<p, <w, s>> in RoutesWS)

TransWS[<p, <w, s>>] <= MaxStorage;

}

execute DISPLAY {

writeln("Transport Amounts FW:");

for (var r in RoutesFW) {

writeln(" ", r.p , ":" , r.e.o , "->" , r.e.d , " " , TransFW[r] );

}

writeln("\nStorage in Warehouse:");

for (var w in Warehouses){

writeln( w," : ",StoreW[w] )

}

}

## Results

// solution (optimal) with objective 45360

Transport Amounts FW:

pA:fA->wA 240

pA:fA->wB 280

pA:fA->wC 70

pA:fA->wD 110

pA:fB->wC 0

pA:fB->wD 120

pA:fB->wE 260

pA:fB->wF 180

pA:fC->wE 110

pA:fC->wF 0

pA:fC->wG 170

pA:fC->wH 180

pA:fC->wI 120

pA:fC->wJ 220

pB:fA->wA 260

pB:fA->wB 240

pB:fA->wC 0

pB:fA->wD 0

pB:fB->wC 110

pB:fB->wD 260

pB:fB->wE 540

pB:fB->wF 90

pB:fC->wE 0

pB:fC->wF 0

pB:fC->wG 60

pB:fC->wH 160

pB:fC->wI 140

pB:fC->wJ 240

pC:fA->wA 10

pC:fA->wB 130

pC:fA->wC 40

pC:fA->wD 220

pC:fA->wE 0

pC:fA->wF 0

pC:fB->wE 180

pC:fB->wF 0

pC:fB->wG 80

pC:fB->wH 80

pC:fB->wI 70

pC:fB->wJ 260

pD:fD->wA 0

pD:fD->wB 60

pD:fD->wC 30

pD:fD->wD 70

pD:fD->wE 140

pD:fD->wF 0

pD:fD->wG 0

pD:fE->wE 0

pD:fE->wF 0

pD:fE->wG 50

pD:fE->wH 50

pD:fE->wI 40

pD:fE->wJ 140

pE:fC->wA 350

pE:fC->wB 450

pE:fC->wC 600

pE:fD->wA 0

pE:fD->wB 0

pE:fD->wC 0

pE:fD->wD 140

pE:fD->wE 860

pE:fD->wF 0

pE:fD->wG 0

pE:fE->wE 210

pE:fE->wF 10

pE:fE->wG 340

pE:fE->wH 380

pE:fE->wI 230

pE:fE->wJ 590

Storage in Warehouse:

wA : 860

wB : 1160

wC : 850

wD : 920

wE : 2300

wF : 280

wG : 700

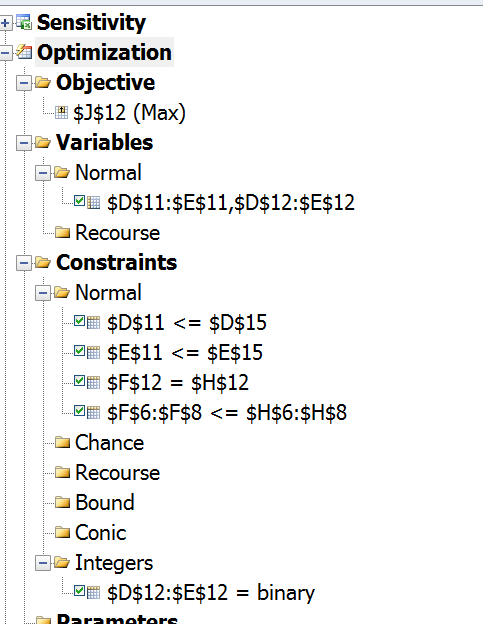
wH : 850

wI : 600

wJ : 1450

# Problem 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Doors | Windows |  |  |  |  |  |
| Profit Per Batch ($000) | 3 | 5 |  |  |  |  |  |
|  |  |  | Hours |  | Hours |  |  |
|  | Hours Used Per Batch Produced | | Used |  | Available |  |  |
| Plant 1 | 1 | 0 | 0 | <= | 4 |  |  |
| Plant 2 | 0 | 2 | 12 | <= | 12 |  |  |
| Plant 3 | 3 | 2 | 12 | <= | 18 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| Batches Produced | 0 | 6 |  |  |  |  | Total Profit ($000) |
| Produce? | 0 | 1 | 1 | == | 1 |  | 30 |
|  |  |  |  |  |  |  |  |
| M |  |  |  |  |  |  |  |
| 100000 | 0 | 100000 |  |  |  |  |  |



# Problem 3

## Model

dvar float+ x[1..4];

dvar boolean y[1..4];

dvar boolean z;

float M = 1000000;

maximize 70\*x[1] + 60\*x[2] + 90\*x[3] + 80\*x[4]

- 50000\*y[1] - 40000\*y[2] - 70000\*y[3] - 60000\*y[4];

subject to{

ct1:

x[1] <= M\*y[1];

ct2:

x[2] <= M\*y[2];

ct3:

x[3] <= M\*y[3];

ct4:

x[4] <= M\*y[4];

ct5:

y[1] + y[2] + y[3] + y[4] <= 2;

// ct6:

// x[3] <= x[1] + x[2];

//

// ct7:

// x[4] <= x[1] + x[2];

ct6:

y[3] <= y[1] + y[2];

ct7:

y[4] <= y[1] + y[2];

ct8:

5\*x[1] + 3\*x[2] + 6\*x[3] + 4\*x[4] <= 6000 + M\*z;

ct9:

4\*x[1] + 6\*x[2] + 3\*x[3] + 5\*x[4] <= 6000 + M\*(1-z);

ct10:

x[1] >= 0;

ct11:

x[2] >= 0;

ct12:

x[3] >= 0;

ct13:

x[4] >= 0;

};

## Results

// solution (optimal) with objective 80000

// Quality Incumbent solution:

// MILP objective 8.0000000000e+004

// MILP solution norm |x| (Total, Max) 2.00100e+003 2.00000e+003

// MILP solution error (Ax=b) (Total, Max) 0.00000e+000 0.00000e+000

// MILP x bound error (Total, Max) 0.00000e+000 0.00000e+000

// MILP x integrality error (Total, Max) 0.00000e+000 0.00000e+000

// MILP slack bound error (Total, Max) 0.00000e+000 0.00000e+000

//

x = [0 2000 0 0];

y = [0 1 0 0];

z = 0;