Integer Programming Summary

The University of Queensland

MATH3202

Matthew Rea
44304533
June 15, 2020

Summary

Notes

• When using the constraint $y_i \geq \frac{x_i}{M_i}$, for some constant M_i, such that $\frac{x_i}{M_i} \in [0,1]$, $\forall i \in I$, one should pick M_i to be the smallest it can be to avoid numerical error

• **Definition:** A unimodular matrix is a square matrix of integers with a determinant of -1, 0, or +1.

• **Definition:** A totally unimodular matrix is a matrix for which every square non-singular sub-matrix is unimodular. This means that every square sub-matrix has a determinant of -1, 0, or +1, thus, every element is -1, 0, or +1.

• **Theorem:** Let A be an $m \times n$ matrix whose rows can be partitioned into disjoint sets B and C such that:

◇ every column of A contains at most two non-zero entries
◇ every entry of A is -1, 0, or +1
◇ if 2 non-zero entries of a column in A have the same sign, then one is in B and the other in C
◇ if 2 non-zero entries of a column in A have opposite signs, then both are in B or both are in C

then A is totally unimodular.

• If a problem contains an **absolute value**, then if the objective is of the form $\min |x|$ or if a constraint is of the form $|x| \leq b$, then we replace instances of x and $|x|$ with the following:

$$|x| = x^+ + x^-$$

$$x = x^+ - x^-$$

Note that if we have a different problem where the objective is of the form $\max |x|$ or if a constraint is of the form $|x| \geq b$, then we replace instances of x and $|x|$ with the following:

$$|x| = x^+ + x^-$$

$$x = x^+ - x^-$$

However, we must also add the following constraints:

$$x^* \in \{0,1\}$$

$$x^+ \leq x^* M$$

$$x^- \leq (1 - x^*)M$$

• In Python, avoid using the logic

```
X[p,i] == 1
```

as floating point numbers are used. Instead, use

```
X[p,i] > 0.9
```

• Suppose we want to know all possible solutions to a problem, a way to cut off a particular solution, x^*, and only that solution, is:
If we are solving

$$\min c^T x, \text{ such that } Ax \leq b, \ x \in \{0,1\}, \ i \in I$$

and we get the solution x^*, then to eliminate this solution and only this solution, we can add the following constraint:

$$\sum_{i|x_i^*=1} x_i + \sum_{i|x_i^*=0} (1 - x_i) \leq |I| - 1$$

Telfa

Problem

Telfa Pty Ltd makes tables and chairs. Each of these items needs labour and wood. The unit profit and resources needed for each item are:

	Table	**Chairs**	**Available**
Profit ($)	8	5	-
Labour (hrs)	1	1	6
Wood	9	5	45

Telfa need to decide how many tables and how many chairs to make.

Solution

Sets

I, items {chair, table}
R, resources required to build items {labour, wood}

Data

p_i, profit ($) on item $i \in I$
u_{ir}, amount of resource $r \in R$ required to build item $i \in I$
a_r, amount of available resource $r \in R$

Variables

x_i, number of items $i \in I$ to make (x_1 = tables to make; x_2 = chairs to make)

Objective

$$\max Z = 8x_1 + 5x_2$$

OR

$$\max Z = \sum_{i \in I} p_i x_i$$

Constraints

Labour: $x_1 + x_2 \leq 6$
Wood: $9x_1 + 5x_2 \leq 45$
Non-Negative Items: $x_1, x_2 \geq 0$
Integer Items: $x_1, x_2 \in \mathbb{Z}$

OR
Available Resources:

$$\sum_{i \in I} u_{ir} x_i \leq a_r, \ \forall r \in R$$

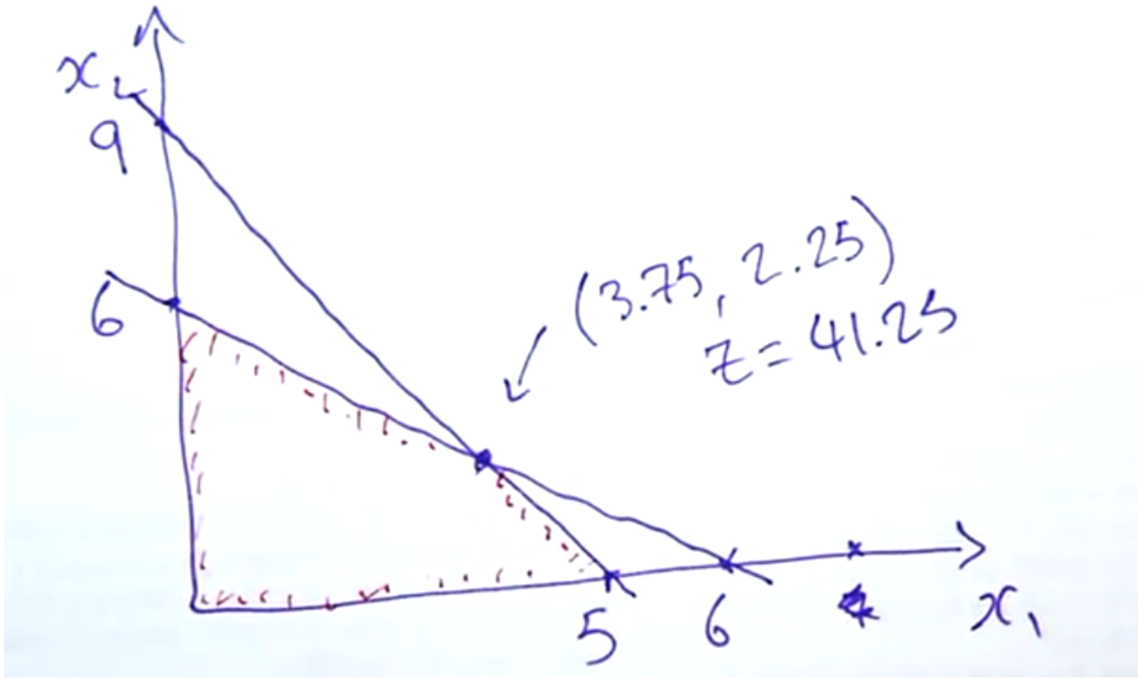
Non-Negative Items:

$$x_i \geq 0, \ \forall i \in I$$

Integer Items:

$$x_i \in \mathbb{Z}, \ \forall i \in I$$

This two-dimensional problem can be shown graphically, as seen below. We begin by solving



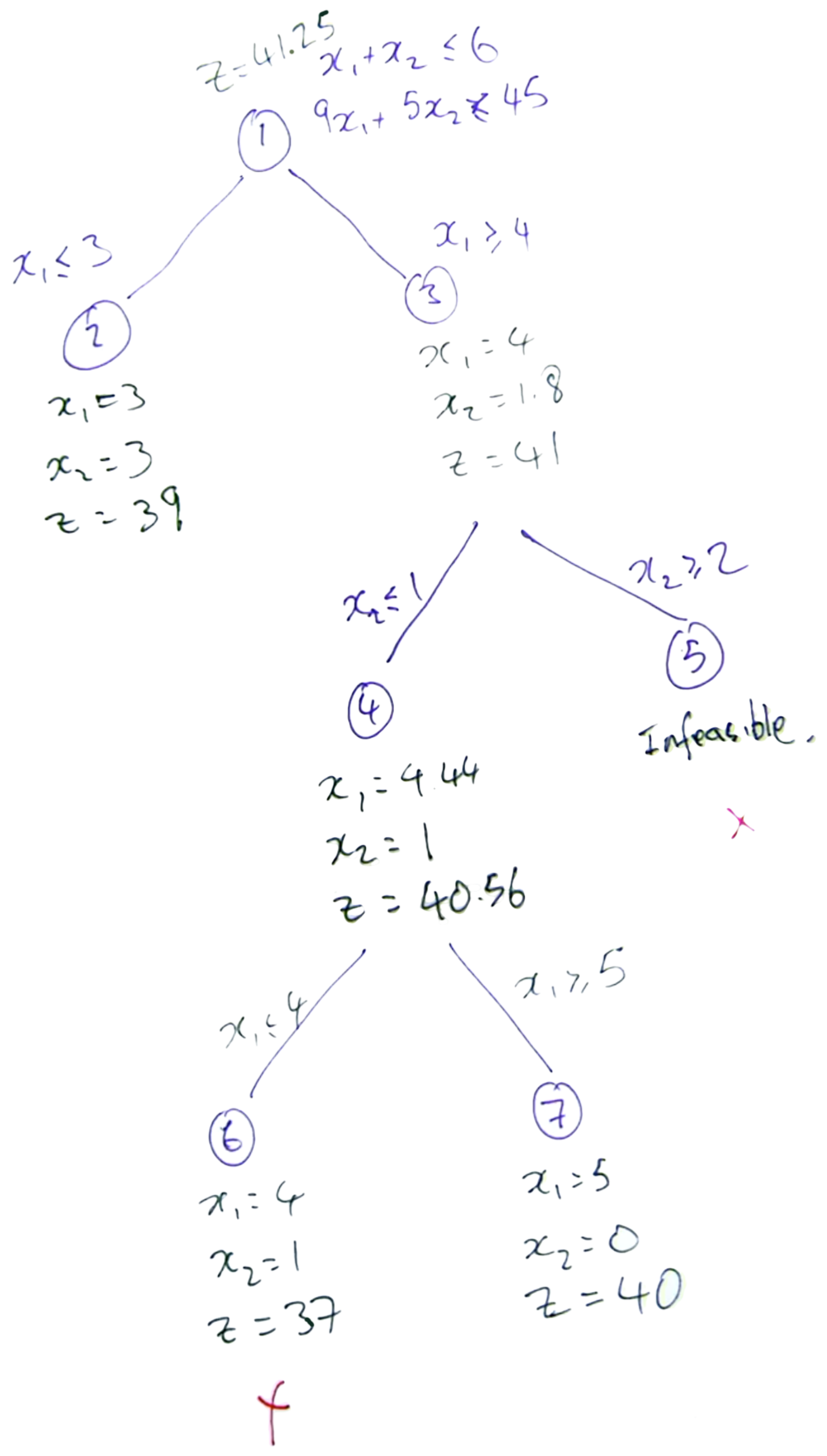
this directly as a linear programming problem, that is, ignoring the constraint that the variables must be integers. Doing this yields the optimal solution

$$x_1 = 3.75 \text{ tables}, \; x_2 = 2.25 \text{ chairs} \implies z = \$41.25$$

We now need to determine the optimal integer solution. To do this, there are a number of algorithms. The one used to solve this problem is called the *Branch and Bound* algorithm. It works in the following way.

1. Find the linear programming solution.
2. Split the problem into two parts: (a) where $x_1 \leq$ the floor of the LP solution, and (b) where $x_1 \geq$ the ceiling of the LP solution.
3. Evaluate these two problems under these new constraints.
4. If the solution gives variables that are not integers, repeat from step **(2)**.
5. Choose the optimal solution as the solution that has the most optimal objective value, with variables that are all integers.

The following tree diagram helps illustrate this process for this particular example.



We thus find the optimal value to be

$$x_1 = 5 \text{ tables}, \; x_2 = 0 \text{ chairs} \implies z = \$40$$

Code

The code used to find the objective value for each ”node” on the tree diagram above can be found below. Note that the bounds were changed for each problem (lines 26-27).

```
 1  from gurobipy import *
 2
 3  #Sets
 4  Items = ["Chair", "Table"]
 5  Resources = ["Labour", "Wood"]
 6
 7  I = range(len(Items))
 8  R = range(len(Resources))
 9
10
11  #Data
12      #Profit for each item
13  p = [5, 8]
14  pRange = range(len(p))
15
16      #Required resources for each item
17  u = [[1,5],
18       [1,9]]
19
20      #Available resources
21  a = [6, 45]
22  aRange = range(len(a))
23
24
25  #Change the bounds to do a Branch & Bound algorithm
26  LB = [0, 0]
27  UB = [1000, 1000]
28
29
30  #Model
31  m = Model("Telfa")
32
33
34  #Variables
35  X = {pr: m.addVar(lb = LB[pr], ub = UB[pr],
36                    name = Items[pr]) for pr in pRange}
37
38  m.update()
39
40
41  #Objective
42  m.setObjective(quicksum(p[pr] * X[pr] for pr in pRange),
43                  GRB.MAXIMIZE)
44
45
46  #Constraints
47      #Available resources
48  for ar in aRange:
49      m.addConstr(quicksum(X[pr] * u[pr][ar]
50      for pr in pRange) <= a[ar])
```

```


#Optimise
m.optimize()


#Print Results
if m.status == GRB.INFEASIBLE:
    print("Infeasible")
else:
    print("Total Profit:", m.objVal)
```

Cloth Co

Problem

Cloth Co manufactures shirts, shorts, and pants. Each of these lines needs a special machine, which has to be rented. Of course the machine need not be rented if none of the line is produced. In addition, each line needs labour and cloth as shown below.

Item	**Labour (hrs)**	**Cloth (m^2)**	**Price ($)**	**Cost ($)**	**Rental Cost ($/week)**
Shirts	3	4	12	6	200
Shorts	2	3	8	4	150
Pants	6	4	15	8	100
Available	150	160	-	-	-

What is the optimal production per week?

Solution

Sets

I, items {shirts, shorts, pants}
R, resources required to build items {labour, cloth}

Data

p_i, profit ($) on item $i \in I$
c_i, rental cost ($/week) of producing item $i \in I$
u_{ir}, amount of resource $r \in R$ required to build item $i \in I$
a_r, amount of available resource $r \in R$

Variables

$$x_i, \text{ number of items } i \in I \text{ to make}$$

$$y_i = \begin{cases} 1, & \text{if any of item } i \in I \text{ is made} \\ 0, & \text{otherwise} \end{cases}$$

Objective

$$\max Z = 6x_1 + 4x_2 + 7x_3 - 200y_1 - 150y_2 - 100y_3$$

OR

$$\max Z = \sum_{i \in I} (p_i x_i - c_i y_i)$$

Constraints

Available Resources:

$$\sum_{i \in I} u_{ir} x_i \leq a_r, \ \forall r \in R$$

Non-Negative Items:

$$x_i \geq 0, \ \forall i \in I$$

Integer Items:

$$x_i \in \mathbb{Z}, \ \forall i \in I$$

Constraint on y to make it boolean:

$$y_i \in \{0, 1\},\ \forall i \in I$$

Constraint to include y in the model, so that some items are actually made if it is profitable:

$$y_i \geq \frac{x_i}{M_i},\ \text{for some constant } M_i,\ \text{such that } \frac{x_i}{M_i} \in [0, 1],\ \forall i \in I$$

Note that for implementing this model on a computer, we should pick M_i to be the smallest it can be (i.e. $M_1 = 40,\ M_2 = 54,\ M_3 = 25$) to avoid numerical error.

We should also note that for the above constraint, the following would not work:

- $y_1 + y_2 + y_3 \geq 1$, as if there was negative profit on all items, this would force one item to be made, which is not optimal

- $y_i \geq \frac{x_i}{x_i+1}$, as this is non-linear

Set Covering

Problem

Kilroy council needs to build fire stations to service its six towns. It wants to build the minimum number of fire stations, yet ensure that each town is within 15 minutes of a fire station. For each city, they know all the cities that are within 15 minutes:

Town	**Towns within 15 mins**
1	1, 2
2	1, 2, 6
3	3, 4
4	3, 4, 5
5	4, 5, 6
6	2, 5, 6

Which fire stations should be built?

Solution

Sets

T, towns $\{1, \ldots, 6\}$

Data

w_t, towns within 15 mins from town $t \in T$

Variables

$$x_t = \begin{cases} 1, & \text{if we build a fire station in town } t \in T \\ 0, & \text{otherwise} \end{cases}$$

Objective

$$\min Z = \sum_{t \in T} x_t$$

Constraints

Must have 1 fire station within 15 mins of town 1 and town 2, . . .:

$$x_1 + x_2 \geq 1;\ x_1 + x_2 + x_6 \geq 1;\ \ldots$$

OR

Must have 1 fire station within 15 mins of all towns:

$$\sum_{q \in w_t} x_q \geq 1,\ \forall t \in T$$

Concrete (Linear Assignment Problem)

Problem

A ready mix concrete company has 50 large batching plants and 50 large jobs. It knows how far away each job is from each plant. How should it assign each job to a plant so as to minimise the total distance between jobs and plants?

Solution

Sets

N, set of indices $\{0, \ldots, 49\}$

Data

d_{ij}, distance from job $i \in N$ to plant $j \in N$

Variables

$$x_{ij} = \begin{cases} 1, & \text{if job } i \in N \text{ is assigned to plant } j \in N \\ 0, & \text{otherwise} \end{cases}$$

Objective

$$\min Z = \sum_{i \in N} \sum_{j \in N} d_{ij} x_{ij}$$

Constraints

Every job must be assigned to a plant (and every plant to a job):

$$\sum_{j \in N} x_{ij} = 1, \ \forall i \in N$$

$$\sum_{i \in N} x_{ij} = 1, \ \forall j \in N$$

Code

```
import math
import random
from gurobipy import *
import pylab

def Distance(p1, p2):
    return math.hypot(p1[0] - p2[0], p1[1] - p2[1])

#Sets
Num = 50
N = range(Num)

```

```
    #Set up plants and jobs
Square = 1000
random.seed(Num)
Plant = [(random.randint(0, Square), random.randint(0, Square))
         for j in N]
Job = [(random.randint(0, Square), random.randint(0, Square))
       for i in N]


#Data
d = [[Distance(Plant[i], Job[j]) for j in N] for i in N]


#Model
m = Model("Concrete")


#Variables
X = [[m.addVar(vtype = GRB.BINARY) for j in N] for i in N]

m.update


#Objective
m.setObjective(quicksum(d[i][j] * X[i][j]
               for j in N for i in N))


#Constraints
c1 = [m.addConstr(quicksum(X[i][j] for j in N) == 1) for i in N]
c2 = [m.addConstr(quicksum(X[i][j] for i in N) == 1) for j in N]


#Optimise
m.optimize()


#Print results
[pylab.plot(Plant[i][0], Plant[i][1], color = 'black',
            marker = '*') for i in N]
[pylab.plot([Plant[i][0], Job[j][0]], [Plant[i][1], Job[j][1]],
            color = 'black') for i in N for j in N
            if X[i][j].x > 0.99]

pylab.show()
```

This gave the objective value of 7481.88.

Linear Assignment Problem

Problem

Consider the matrix of assignment costs:

	1	**2**	**3**	**4**	**5**
1	48	56	31	59	34
2	47	54	94	74	36
3	67	12	67	31	43
4	74	31	37	23	55
5	60	41	27	39	44

How should the rows be assigned to the columns?

Solution

We begin by subtracting a constant from each row. The aim is to subtract a constant from each row such that at least one value becomes zero and there are no negative values. Doing this, we get

	1	**2**	**3**	**4**	**5**	**Subtracted Constant**
1	17	25	0	28	3	31
2	11	18	58	38	0	36
3	55	0	55	19	31	12
4	51	8	14	0	32	23
5	33	14	0	12	17	27

Now that a constant has been subtracted from each row, to get a zero in each row, the same must be done to the columns to ensure a zero is present in each column (with no negative values).

	1	**2**	**3**	**4**	**5**	**Subtracted Constant**
1	6	25	0	28	3	31
2	0	18	58	38	0	36
3	44	0	55	19	31	12
4	40	8	14	0	32	23
5	22	14	0	12	17	27
Subtracted Constant	11	0	0	0	0	140

If we sum the constants that were subtracted, we know that the objective value is at least this, that is, the objective value is at least $31 + 36 + \ldots + 27 + 11 + 0 + \ldots + 0 = 140$.

Note that if there were zeros such that each row could be mapped to a column, with no rows or columns having no mappings, then we would know that this was the optimal solution. We see that this fails in this case as:

$$1 \mapsto 3,\ 2 \mapsto 1 \text{ or } 5,\ 3 \mapsto 2,\ 4 \mapsto 4,\ 5 \mapsto 3$$

meaning that there will be either 1 or 5 that has no mapping to it, depending on what we choose 2 to map to (either 1 or 5). This is because both 1 and 5 both map to 3 <u>only</u>.

The next step is to look at the rows of either 1 or 5 and subtract a second constant from the row to create a new zero. We choose to reduce row 1.

	1	**2**	**3**	**4**	**5**	**Subtracted Constant**
1	3	22	-3	25	0	34
2	0	18	58	38	0	36
3	44	0	55	19	31	12
4	40	8	14	0	32	23
5	22	14	0	12	17	27
Subtracted Constant	11	0	0	0	0	143

The new objective value would be 143 as an extra 3 was subtracted. However, this solution is infeasible as there is now a new zero in row 1, but there is also a negative value. The next step is to subtract a (negative) constant to the column to make every value non-negative.

	1	**2**	**3**	**4**	**5**	**Subtracted Constant**
1	3	22	0	25	0	34
2	0	18	61	38	0	36
3	44	0	58	19	31	12
4	40	8	17	0	32	23
5	22	14	3	12	17	27
Subtracted Constant	11	0	-3	0	0	140

Now we see that row 5 does not have a zero. We subtract a further constant to obtain at least one zero.

	1	**2**	**3**	**4**	**5**	**Subtracted Constant**
1	3	22	0	25	0	34
2	0	18	61	38	0	36
3	44	0	58	19	31	12
4	40	8	17	0	32	23
5	19	11	0	9	14	30
Subtracted Constant	11	0	-3	0	0	143

This solution gives an adequate mapping:

$$2 \mapsto 1,\ 3 \mapsto 2,\ 5 \mapsto 3,\ 4 \mapsto 4,\ 1 \mapsto 5$$

to give the (minimised) objective value of 143.

Interpretation: The numbers in the matrix are the reduced costs. The numbers on the outside (subtracted constants for rows and columns) are the dual variables. This process is a dual algorithm, as opposed to a primal algorithm as it is starting out with a solution that is below the optimal objective value and then builds up to the feasible optimal objective value to minimise the objective function.

Noughts and Crosses

Problem

How can you fill in a noughts and crosses grid with a specified number of X's (and the balance O's) so that the number of lines completely formed of X's or O's is minimised?

Solution

Define the positions on the noughts and crosses board as follows

0	1	2
3	4	5
6	7	8

Sets

N, set of positions $\{0, \ldots, 8\}$
L, set of lines on the board $\{(0, 1, 2), (0, 4, 8), \ldots\}$

Data

$numcrosses$, number of crosses

Variables

$$x_n = \begin{cases} 0, & \text{if square } n \in N \text{ is a nought} \\ 1, & \text{if square } n \in N \text{ is a cross} \end{cases}$$

$$y_l = \begin{cases} 0, & \text{if line } l \in L \text{ is not all the same} \\ 1, & \text{if line } l \in L \text{ is all the same} \end{cases}$$

Objective

$$\min Z = \sum_{l \in L} y_l$$

Constraints

The number of crosses is the specified number of crosses:

$$\sum_{n \in N} x_n = numcrosses$$

The line is made up of all noughts / crosses or varies:

$$y_l \geq x_{n_1|l} + x_{n_2|l} + x_{n_3|l} - 2, \ \forall l \in L$$

$$y_l \geq (1 - x_{n_1|l}) + (1 - x_{n_2|l}) + (1 - x_{n_3|l}) - 2$$
$$\implies y_l \geq 1 - x_{n_1|l} - x_{n_2|l} - x_{n_3|l}, \ \forall l \in L$$

Code

```python
import math
import random
from gurobipy import *

#Sets
N = range(9)
Lines = [[0,1,2],
         [3,4,5],
         [6,7,8],
         [0,3,6],
         [1,4,7],
         [2,5,8],
         [0,4,8],
         [2,4,6]]
L = range(len(Lines))


#Data
numcrosses = 1


#Model
m = Model("Noughts and Crosses")


#Variables
X = [m.addVar(vtype = GRB.BINARY) for i in N]
Y = [m.addVar(vtype = GRB.BINARY) for l in L]

m.update()


#Objective
m.setObjective(quicksum(Y))


#Constraints
[m.addConstr(quicksum(X[i] for i in Lines[l]) + Y[l] >= 1) for l in L]
[m.addConstr(quicksum(X[i] for i in Lines[l]) - Y[l] <= 2) for l in L]
[m.addConstr(quicksum(X) == numcrosses)]


#Optimise
m.optimize()


#Print results
for i in range(3):
    lstr = ''
    for j in range(3):
        lstr += "X" if X[i*3+j].x > 0.99 else "O"
    print(lstr)
```

Euing Oil - Piecewise Functions

Problem

Euing Petroleum produces two types of petrol (blend 1 and blend 2) from two types of oil (oil 1 and oil 2). Each litre of blend 1 must contain at least 50% oil 1 and each litre of blend 2 must contain at least 60% of oil 1. Each litre of blend 1 is sold for \$1.20 and each litre of blend 2 is sold for \$1.40. Currently 500 litres of oil 1 and 1,000 litres of oil 2 are available. As many as 1,500 litres more of oil 1 can be purchased, at the following prices:

- First 500 litres - \$2.50 per litre
- Next 500 litres - \$2.00 per litre
- Next 500 litres - \$1.50 per litre

Formulate an IP that will maximise Euing's profit.

Solution

Sets

J, petrols {1, 2}
I, oils {1, 2}
B, cost brackets of buying more oils {1, 2, 3}

Data

r_j, revenue (\$/litre) for petrol $j \in J$
c_b, cost (\$/litre) of buying more oil 1 for cost bracket $b \in B$

Variables

x_{ji}, amount of oil $i \in I$ used in petrol blend $j \in J$
y_b, amount of oil 1 purchased in bracket $b \in B$

$$z_b = \begin{cases} 1, & \text{if the fuel purchased from bracket } b \in B \text{ is the max allowed} \\ 0, & \text{if the fuel purchased from bracket } b \in B \text{ is the less than the max allowed} \end{cases}$$

Objective

$$\max Z = 1.2(x_{1,1} + x_{1,2}) + 1.4(x_{2,1} + x_{2,2}) - 2.5y_1 - 2y_2 - 1.5y_3$$

OR

$$\max Z = \sum_{j \in J} \sum_{i \in I} r_j x_{ji} - \sum_{b \in B} c_b y_b$$

Constraints

Each litre of blend 1 must contain at least 50% of oil 1:

$$x_{1,1} \geq 0.5 \sum_{i \in I} x_{1,i} \implies x_{1,1} \geq x_{1,2}$$

Each litre of blend 2 must contain at least 60% of oil 1:

$$x_{2,1} \geq 0.6 \sum_{i \in I} x_{2,i} \implies x_{2,1} \geq 1.5 x_{2,2}$$

The amount of oil 1 used must be less than what we have plus what we buy:

$$\sum_{j \in J} x_{j,1} \leq 500 + \sum_{b \in B} y_b$$

The amount of oil 2 used must be less than what we have:

$$\sum_{j \in J} x_{j,2} \leq 1000$$

The amount of oil 1 we can buy in each bracket is less than or equal to 500:

$$y_b \leq 500, \ \forall b \in B$$

The amount of oil used and oil 1 bought must be non-negative:

$$x_{ji} \geq 0, \ \forall i \in I, j \in J$$

$$y_b \geq 0, \ \forall b \in B$$

The amount of oil 1 purchased from each bracket is:

$$y_1 \geq 500 z_1$$

$$y_2 \leq 500 z_1$$

$$y_2 \geq 500 z_2$$

$$y_3 \leq 500 z_2$$

$$y_3 \geq 500 z_3$$

Code

```
from gurobipy import *

#Sets
J = [0,1]
I = [0,1]
B = [1,2,3]


#Data
r = [1.20, 1.40]
c = [2.50, 2.00, 1.50]


#Model
m = Model("Euing Oil")

```

```

#Variables
X11 = m.addVar(name = "X11")
X12 = m.addVar(name = "X12")
X21 = m.addVar(name = "X21")
X22 = m.addVar(name = "X22")

Y1 = m.addVar(ub = 500, name = "Y1")
Y2 = m.addVar(ub = 500, name = "Y2")
Y3 = m.addVar(ub = 500, name = "Y3")

Z1 = m.addVar(vtype = GRB.BINARY, name = "Z1")
Z2 = m.addVar(vtype = GRB.BINARY, name = "Z2")
Z3 = m.addVar(vtype = GRB.BINARY, name = "Z3")


#Objective
m.setObjective(r[0] * (X11 + X12) + r[1] * (X21 + X22) -
               c[0] * Y1 - c[1] * Y2 - c[2] * Y3, GRB.MAXIMIZE)


#Constraints
m.addConstr(X11 >= 0.5 * (X11 + X12))
m.addConstr(X21 >= 0.6 * (X21 + X22))
m.addConstr(X11 + X21 <= 500 + Y1 + Y2 + Y3)
m.addConstr(X12 + X22 <= 1000)
m.addConstr(Y1 >= 500 * Z1)
m.addConstr(Y2 <= 500 * Z1)
m.addConstr(Y2 >= 500 * Z2)
m.addConstr(Y3 <= 500 * Z2)
m.addConstr(Y3 >= 500 * Z3)


#Optimise
m.optimize()


#Print results
print("Objective Value:", m.objVal)
for v in m.getVars():
    print(v.VarName, round(v.x, 2))
```

This gave the optimal profit of \$1250.00, where 2500 litres of petrol 2 was made (with 1500 litres of oil 1 and 1000 litres of oil 2, meaning that 1000 litres of oil 1 was purchased).

Factory Planning

Problem

An engineering factory makes seven products on the following machines: four grinders, two vertical drills, three horizontal drills, one borer and one planer. Each product yields a per unit profit and has a required per unit production time (in hours) on each of the machines, as given in the following tables (a dash indicates a product does not require that process):

Product	**Grinding**	**VDrilling**	**HDrilling**	**Boring**	**Planing**	**Profit ($)**
1	0.5	0.1	0.2	0.05	-	10
2	0.7	0.2	-	0.03	-	6
3	-	-	0.8	-	0.01	8
4	-	0.3	-	0.07	-	4
5	0.3	-	-	0.1	0.05	11
6	0.2	0.6	-	-	-	9
7	0.5	-	0.6	0.08	0.05	3

Over the production horizon certain machines will be down for maintenance as follows:

- January: 1 Grinder
- February: 2 Horizon Drills
- March: 1 Borer
- April: 1 Vertical Drill
- May: 1 Grinder and 1 Vertical Drill
- June: 1 Planer and 1 Horizontal Drill

There are also marketing limitations on each product in each month as follows:

Product	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**
1	500	600	300	200	0	500
2	1000	500	600	300	100	500
3	300	200	0	400	500	100
4	300	0	0	500	100	300
5	800	400	500	200	1000	1100
6	200	300	400	0	300	500
7	100	150	100	100	0	60

It is possible to store up to 100 of each product at a time at a cost of $0.5 per unit per month. There are no starting stocks, but it is desired to have 50 units of each product in stock at the end of June. We can assume the factory works 16 hours a day for 24 days a month and that there are no sequencing problems.

Q1 When and what should the factory make in order to maximise the total profit?

Q2 Instead of the fixed maintenance schedule given, suppose you can choose which months the machines are to be maintained. Adjust your model to calculate the optimal maintenance schedule.

Solution - Q1

Sets

P, products $\{0, \dots, 6\}$
M, machines {Grinder, Vertical Drill, ...}
T, months $\{0, \dots, 5\}$

Data

$profit_p$, the profit on selling product $p \in P$
n_m, number of available machines $m \in M$
$usage_{pm}$, required per unit production time for machine $m \in M$ for product $p \in P$
$market_{pt}$, market limitations on product $p \in P$ for month $t \in T$
$maint_{tm}$ machines of type $m \in M$ unavailable in month $t \in T$
$storecost$, cost per unit per month
$maxstore$, maximum storage per product per month
$initialstore$, initial amount of each product in storage
$endstore$, final amount of each product in storage
$monthhours$, hours per month available on each machine

Variables

x_{pt}, units of product $p \in P$ to produce in month $t \in T$
s_{pt}, units of product $p \in P$ to store at the end of month $t \in T$
y_{pt}, units of product $p \in P$ to sell in month $t \in T$

Objective

$$\max Z = \sum_{t \in T} \sum_{p \in P} profit_p \times y_{pt} - \sum_{t \in T} \sum_{p \in P} storecost \times s_{pt}$$

Constraints

Market limitations:

$$y_{pt} \leq market_{pt}, \ \forall p \in P, t \in T$$

Sum of the time spent to make each product:

$$\sum_{p \in P} usage_{pm} \times x_{pt} \leq monthhours \times (n_m - maint_{tm}), \ \forall m \in M, t \in T$$

Inventory must be less than or equal to its maximum storage:

$$s_{pt} \leq maxstore, \ \forall p \in P, t \in T$$

Inventory in last month must be equal to the final month:

$$s_{p5} = endstore, \ \forall p \in P$$

Inventory for each month must be the amount not produced, but not sold for the month, plus the existing storage:

$$s_{pt} = x_{pt} - y_{pt} + s_{p(t-1)}, \ \forall p \in P, t \in T$$

Inventory for first month:

$$s_{p0} = x_{p0} - y_{p0} + initialstore, \ \forall p \in P$$

Variables must be non-negative integers:

$$x_{pt}, s_{pt}, y_{pt} \in \{0, \mathbb{Z}^+\}$$

Code - Q1

```python
from gurobipy import *

#Sets
P = range(7)

Machines = ["Grinder", "VDrill", "HDrill", "Borer", "Planer"]
M = range(len(Machines))

T = range(6)


#Data
profit = [10, 6, 8, 4, 11, 9, 3]

n = [4, 2, 3, 1, 1]

usage = [[0.5, 0.1, 0.2, 0.05, 0.00],
         [0.7, 0.2, 0.0, 0.03, 0.00],
         [0.0, 0.0, 0.8, 0.00, 0.01],
         [0.0, 0.3, 0.0, 0.07, 0.00],
         [0.3, 0.0, 0.0, 0.10, 0.05],
         [0.2, 0.6, 0.0, 0.00, 0.00],
         [0.5, 0.0, 0.6, 0.08, 0.05]]

market = [[ 500, 600, 300, 200,   0, 500],
          [1000, 500, 600, 300, 100, 500],
          [ 300, 200,   0, 400, 500, 100],
          [ 300,   0,   0, 500, 100, 300],
          [ 800, 400, 500, 200,1000,1100],
          [ 200, 300, 400,   0, 300, 500],
          [ 100, 150, 100, 100,   0,  60]]

maint = [[1, 0, 0, 0, 0],
         [0, 0, 2, 0, 0],
         [0, 0, 0, 1, 0],
         [0, 1, 0, 0, 0],
         [1, 1, 0, 0, 0],
         [0, 0, 1, 0, 1]]

storecost = 0.5

maxstore = 100

initialstore = 0

endstore = 50

monthhours = 16 * 24


#Model
fp = Model("Factory Planning")
```

```


#Variables
X = {(p,t): fp.addVar(vtype = GRB.INTEGER) for p in P for t in T}
S = {(p,t): fp.addVar(vtype = GRB.INTEGER) for p in P for t in T}
Y = {(p,t): fp.addVar(vtype = GRB.INTEGER) for p in P for t in T}


#Objective
fp.setObjective(quicksum(profit[p] * Y[p,t] for p in P for t in T) -
                quicksum(storecost * S[p,t] for p in P for t in T),
                GRB.MAXIMIZE)


#Constraints
[fp.addConstr(Y[p,t] <= market[p][t]) for p in P for t in T]

[fp.addConstr(quicksum(usage[p][m] * X[p,t] for p in P) <=
                       monthhours * (n[m] - maint[t][m]))
 for m in M for t in T]

[fp.addConstr(S[p,t] <= maxstore) for p in P for t in T]

[fp.addConstr(S[p,5] == endstore) for p in P]

[fp.addConstr(S[p,t] == X[p,t] - Y[p,t] + S[p,t-1])
 for t in T if t > 0 for p in P]

[fp.addConstr(S[p,0] == X[p,0] - Y[p,0] + initialstore) for p in P]


#Optimise
fp.optimize()


#Print results
print("\n Profit = $", fp.objVal)

print("\n Sales")
for p in P:
    print(p, [int(Y[p,t].x) for t in T])

print("\n Production")
for p in P:
    print(p, [int(X[p,t].x) for t in T])

print("\n Storage")
for p in P:
    print(p, [int(S[p,t].x) for t in T])
```

This gave the maximised profit of $93,709.50.

Solution - Q2

The following additions were made to the model from Q1.

Variables

z_{tm}, a binary variable indicating whether machine $m \in M$ should be maintained in $t \in T$

Constraints

The previous constraint for the sum of time spent on each product was made:

$$\sum_{p \in P} usage_{pm} \times x_{pt} \leq monthhours \times (n_m - z_{tm}),\ \forall m \in M, t \in T$$

The amount of maintenance done is the same as what was being done before:

$$\sum_{t \in T} z_{tm} = \sum_{t \in T} maint_{tm},\ \forall m \in M$$

The number of machines being maintained in each month must be less than or equal to 2:

$$\sum_{m \in M} z_{tm} \leq 2,\ \forall t \in T$$

Code - Q2

```
from gurobipy import *

#Sets
P = range(7)

Machines = ["Grinder", "VDrill", "HDrill", "Borer", "Planer"]
M = range(len(Machines))

T = range(6)


#Data
profit = [10, 6, 8, 4, 11, 9, 3]

n = [4, 2, 3, 1, 1]

usage = [[0.5, 0.1, 0.2, 0.05, 0.00],
         [0.7, 0.2, 0.0, 0.03, 0.00],
         [0.0, 0.0, 0.8, 0.00, 0.01],
         [0.0, 0.3, 0.0, 0.07, 0.00],
         [0.3, 0.0, 0.0, 0.10, 0.05],
         [0.2, 0.6, 0.0, 0.00, 0.00],
         [0.5, 0.0, 0.6, 0.08, 0.05]]

market = [[ 500, 600, 300, 200,   0, 500],
          [1000, 500, 600, 300, 100, 500],
          [ 300, 200,   0, 400, 500, 100],
          [ 300,   0,   0, 500, 100, 300],
          [ 800, 400, 500, 200,1000,1100],
          [ 200, 300, 400,   0, 300, 500],
          [ 100, 150, 100, 100,   0,  60]]

maint = [[1, 0, 0, 0, 0],
         [0, 0, 2, 0, 0],
         [0, 0, 0, 1, 0],
         [0, 1, 0, 0, 0],
         [1, 1, 0, 0, 0],
         [0, 0, 1, 0, 1]]

storecost = 0.5

maxstore = 100

initialstore = 0

endstore = 50

monthhours = 16 * 24


#Model
fp = Model("Factory Planning")
```

```python


#Variables
X = {(p,t): fp.addVar(vtype = GRB.INTEGER) for p in P for t in T}
S = {(p,t): fp.addVar(vtype = GRB.INTEGER) for p in P for t in T}
Y = {(p,t): fp.addVar(vtype = GRB.INTEGER) for p in P for t in T}
Z = {(p,t): fp.addVar(vtype = GRB.BINARY) for p in P for t in T}


#Objective
fp.setObjective(quicksum(profit[p] * Y[p,t] for p in P for t in T) -
                quicksum(storecost * S[p,t] for p in P for t in T),
                GRB.MAXIMIZE)


#Constraints
    #Example of defining constraints with dictionary comprehension
#maxstorecons = {(p,t): fp.addConstr(S[p,t] <= maxstore)
#                for p in P for t in T}

[fp.addConstr(Y[p,t] <= market[p][t]) for p in P for t in T]

[fp.addConstr(quicksum(usage[p][m] * X[p,t] for p in P) <=
                       monthhours * (n[m] - Z[t,m]))
 for m in M for t in T]

[fp.addConstr(S[p,t] <= maxstore) for p in P for t in T]

[fp.addConstr(S[p,5] == endstore) for p in P]

[fp.addConstr(S[p,t] == X[p,t] - Y[p,t] + S[p,t-1])
 for t in T if t > 0 for p in P]

[fp.addConstr(S[p,0] == X[p,0] - Y[p,0] + initialstore) for p in P]

[fp.addConstr(quicksum(Z[t,m] for t in T) == sum(maint[t][m]
 for t in T)) for m in M]

[fp.addConstr(quicksum(Z[t,m] for m in M) <= 2) for t in T]


#Optimise
fp.optimize()


#Print results
print("\n Profit = $", fp.objVal)

print("\n Sales")
for p in P:
    print(p, [int(Y[p,t].x) for t in T])

print("\n Production")
for p in P:
    print(p, [int(X[p,t].x) for t in T])

print("\n Storage")
for p in P:
    print(p, [int(S[p,t].x) for t in T])

print("\n Maintenance")
for m in M:
    print(m, [int(Z[t,m].x) for t in T])
```

This gave the maximised profit of $108,855.00.

Minimax

Problem

Suppose we want to find the solution that minimises

$$\max_i \sum_j c_{ij} x_j$$

How can we do this using linear programming?

Solution

We define a new variable, z, and add constraints of the form:

$$z \geq \sum_j c_{ij} x_j, \ \forall i$$

Then, we simply minimise z.

Minimising Absolute Error

Problem

In regular least squares fitting for data, we choose a line to minimise the sum of the squared vertical distances of points from the line. This has a nice solution, but can be badly affected by large deviations.

A more robust approach is to minimise the absolute error between points and the line. How can we do this using linear / integer programming?

Solution

To find the least squares fit for data, the general linear model for the data is

$$b_i = a_i x + c + e_i$$

where a_i is the gradient, c is the intercept, and e_i is the error. The least squares method solves this by minimising $\sum_i e_i^2$. Note that x, c, e_i are all unconstrained. This cannot be solved using linear programming, however, as it is non-linear.

To minimise the absolute error, we want to minimise $\sum_i |e_i|$, which can be done via linear / integer programming. We then introduce the following constraints:

$$e_i^* \geq e_i$$

$$e_i^* \geq -e_i$$

Then, we want $\min \sum_i e_i^*$. We then replace the general linear model with

$$b_i = a_i x + c + e_i^+ - e_i^-$$

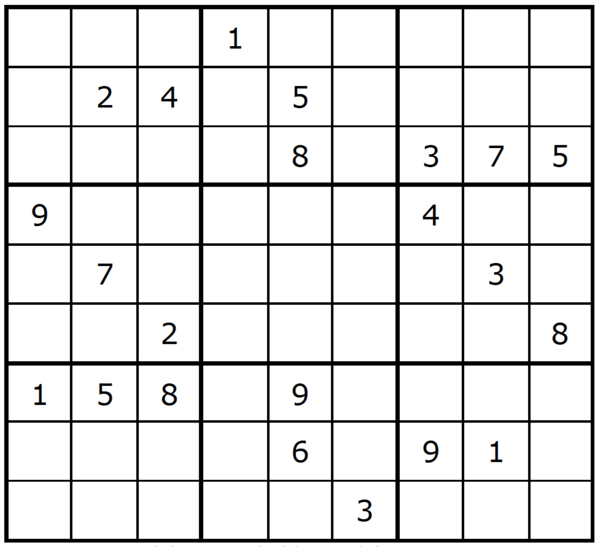
with the constraints

$$e_i^+, e_i^- \geq 0$$

Sudoku

Problem

Consider the following puzzle (the 'Tough Sudoku' for March 28th 2018 from Sudoku.com.au):



			1					
	2	4		5				
				8		3	7	5
9						4		
	7						3	
		2						8
1	5	8		9				
				6		9	1	
					3			

Sudoku provided by Sudoku.com.au

Formulate an integer programming problem that will solve this Sudoku puzzle.

Solution

Sets

I, rows of table $\{0, \ldots, 8\}$
J, columns of table $\{0, \ldots, 8\}$
K, numbers each cell can take $\{1, \ldots, 9\}$
α, sub-tables in overall table going horizontally $\{0, \ldots, 2\}$
β, sub-tables in overall table going vertically $\{0, \ldots, 2\}$

Data

p_{ij}, pre-assigned number for $i \in I, j \in J$ (0 if not given)

Variables

$$x_{ijk} = \begin{cases} 1, & \text{if cell } i \in I, j \in J \text{ has number } k \in K \\ 0, & \text{otherwise} \end{cases}$$

Objective

No objective needed for this model.

Constraints

Pre-assigned numbers on table:

$$x_{ijk} = 1, \ \forall i \in I, j \in J, k \in K$$

One number in each cell:

$$\sum_{k \in K} x_{ijk} = 1, \ \forall i \in I, j \in J$$

Each number is present in each row:

$$\sum_{j \in J} x_{ijk} = 1, \ \forall i \in I, k \in K$$

Each number is present in each column:

$$\sum_{i \in I} x_{ijk} = 1, \ \forall j \in J, k \in K$$

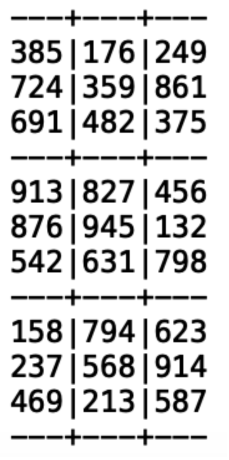
One number in each sub-table in the overall table:

$$\sum_{i,j \in \{0,1,2\}} x_{(3a+i)(3b+j)(k)} = 1, \ \forall a \in \alpha, b \in \beta, k \in K$$

Code

```
from gurobipy import *

#Sets
I = range(9)
J = range(9)
K = range(1,10)
alpha = range(3)
beta = range(3)

#Data
p = [[0,0,0, 1,0,0, 0,0,0],
     [0,2,4, 0,5,0, 0,0,0],
     [0,0,0, 0,8,0, 3,7,5],

     [9,0,0, 0,0,0, 4,0,0],
     [0,7,0, 0,0,0, 0,3,0],
     [0,0,2, 0,0,0, 0,0,8],

     [1,5,8, 0,9,0, 0,0,0],
     [0,0,0, 0,6,0, 9,1,0],
     [0,0,0, 0,0,3, 0,0,0]]

#Model
m = Model("Sudoku")

#Variables
X = {(i,j,k): m.addVar(vtype = GRB.BINARY)
              for i in I for j in J for k in K}

#Objective
#None required

#Constraints
PreAssign = {
        (i,j): m.addConstr(X[i,j,p[i][j]] == 1)
        for i in I for j in J if p[i][j] > 0}

OnePerSquare = {
        (i,j): m.addConstr(quicksum(X[i,j,k] for k in K) == 1)
        for i in I for j in J}

EachValueInRow = {
        (i,k): m.addConstr(quicksum(X[i,j,k] for j in J) == 1)
        for i in I for k in K}

EachValueInCol = {
        (j,k): m.addConstr(quicksum(X[i,j,k] for i in I) == 1)
        for j in J for k in K}

EachValueInSubTable = {
        (a,b,k): m.addConstr(quicksum(X[3*a+i,3*b+j,k]
                   for i in range(3) for j in range(3)) == 1)
        for a in alpha for b in beta for k in K}

#Optimise
m.optimize()

#Print results
print("---+---+---")
for i in I:
    if i == 3 or i == 6:
        print("---+---+---")
    for j in J:
        if j == 3 or j == 6:
            print("|", end = "")
        for k in K:
            if X[i,j,k].x > 0.9:
                print(k, sep = '', end = "")
    print("")
print("---+---+---")
```

This gave the result of



```
---+---+---
385|176|249
724|359|861
691|482|375
---+---+---
913|827|456
876|945|132
542|631|798
---+---+---
158|794|623
237|568|914
469|213|587
---+---+---
```

English-14

Problem

Suppose you want to write in English but have to restrict yourself to just 14 letters. You can choose any 14 but then you can only use words made up of those letters. Which 14 letters would you choose?

Solution

Sets

A, the alphabet {a, b, ...}
W, set of words

Data

f_w, frequency of word $w \in W$
$\delta_{wa} = 1,$ if letter $a \in A$ is used in word $w \in W;$ $\delta_{wa} = 0,$ otherwise
k, the number of letters that can be chosen (14 in this case)

Variables

$$x_a = \begin{cases} 1, & \text{if } a \in A \text{ is in our restricted alphabet} \\ 0, & \text{otherwise} \end{cases}$$

$$y_w = \begin{cases} 1, & \text{if word } w \in W \text{ can be spelt} \\ 0, & \text{otherwise} \end{cases}$$

Objective

$$\max Z = \sum_{w \in W} f_w y_w$$

Constraints

The maximum number of letters that can be used:

$$\sum_{a \in A} x_a = k$$

The word can only be spelt if all letters in it are available (second will run much faster):

$$\sum_{a \in A} \delta_{wa} y_w \leq \sum_{a \in A} \delta_{wa} x_a, \ \forall w \in W$$

OR

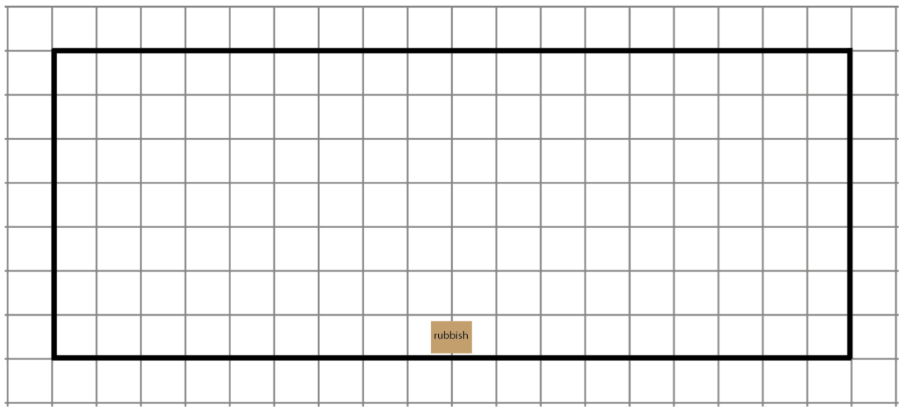
$$y_w \leq x_a, \ \forall w \in W, a \in A \text{ if } \delta_{wa} = 1$$

Tables and Chairs

Problem

A food court would like to place tables and chairs so that they can maximise the number of chairs in the area below, while ensuring that

- there is at most one chair or table per square, but nothing on the rubbish
- no tables touch (even diagonally)
- a chair must have a table beside it



Solution

Sets

I, rows of grid $\{0, 1, \dots\}$
J, columns of grid $\{0, 1, \dots\}$

Data

$$r_{ij} = \begin{cases} 1, & \text{if there is rubbish at } i \in I, j \in J \\ 0, & \text{otherwise} \end{cases}$$

Variables

$$x_{ij} = \begin{cases} 1, & \text{if there is a table at } i \in I, j \in J \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1, & \text{if there is a chair at } i \in I, j \in J \\ 0, & \text{otherwise} \end{cases}$$

Objective

$$\max Z = \sum_{i \in I} \sum_{j \in J} y_{ij}$$

Constraints

There must be at most one table in every 2x2 area of the grid so no tables touch:

$$x_{ij} + x_{i(j+1)} + x_{(i+1)}j + x_{(i+1)(j+1)} \leq 1,\ \forall i \in I, j \in J$$

There must only be at most one table or one chair in each 1x1 area of the grid:

$$x_{ij} + y_{ij} \leq 1,\ \forall i \in I, j \in J$$

There must be no tables or chairs in the 1x1 area where the rubbish is:

$$x_{ij} + y_{ij} = 0,\ \forall i \in I, j \in J \text{ if } r_{ij} = 1$$

A chair must have at least one table beside it:

$$y_{ij} \leq x_{(i-1)j} + x_{(i+1)j} + x_{i(j-1)} + x_{i(j+1)},\ \forall i \in I, j \in J$$

Code

```
from gurobipy import *

#Sets
I = range(7) #7 rows
J = range(18) #18 columns

#Data
    #Locations of rubbish
r = [(6,8), (6,9)]

    #Are tables allowed to touch or not
TablesTouching = False

    #Return list of neighbours of (i,j)
def neighbours(i,j):
    nList = []
    if i > 0:
        nList.append((i-1,j))
    if i < I[-1]:
        nList.append((i+1,j))
    if j > 0:
        nList.append((i,j-1))
    if j < J[-1]:
        nList.append((i,j+1))
    return nList

#Model
m = Model("Tables and Chairs")

```

```
#Variables
X = {(i,j): m.addVar(vtype = GRB.BINARY)
            for i in I for j in J}

Y = {(i,j): m.addVar(vtype = GRB.BINARY)
            for i in I for j in J}

#Objective
m.setObjective(quicksum(Y[i,j] for i in I for j in J),
               GRB.MAXIMIZE)

#Constraints
ChairMustBeBesideTable = {
        (i,j): m.addConstr(Y[i,j] <= quicksum(X[i1,j1]
        for (i1,j1) in neighbours(i,j)))
        for i in I for j in J}

ChairCantBeOnTable = {
        (i,j): m.addConstr(X[i,j] + Y[i,j] <= 1)
        for i in I for j in J}

RubbishSquaresEmpty = {
        (i,j): m.addConstr(X[i,j] + Y[i,j] == 0)
        for (i,j) in r}

if not TablesTouching:
    TablesCantTouch = {
            (i,j): m.addConstr(X[i,j] + X[i+1,j] +
            X[i,j+1] + X[i+1,j+1] <= 1)
            for i in I[:-1] for j in J[:-1]}

#Optimise
m.optimize()

#Print results
for i in I:
    Line = ""
    for j in J:
        if X[i,j].x > 0.9:
            Line += "t"
        elif Y[i,j].x > 0.9:
            Line += "c"
        else:
            Line += "e"
    print(Line)
```

Heist Puzzle

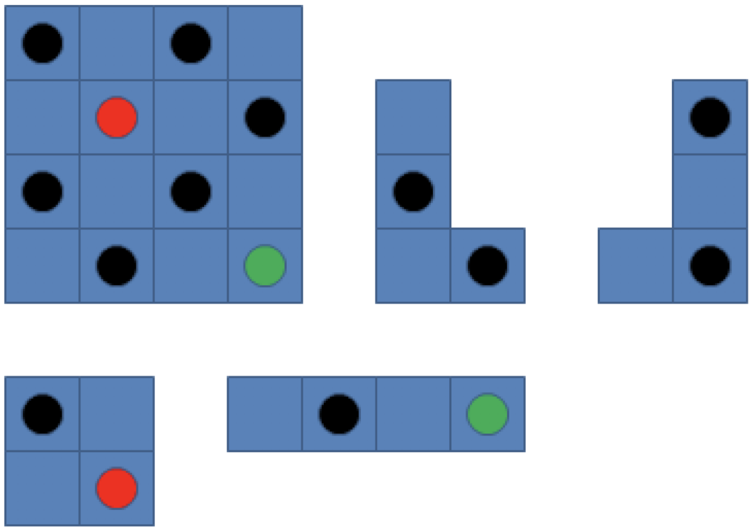
Problem

Many puzzles can be formulated as Integer Programming problems. Today we consider the Heist puzzle:

http://www.youtube.com/watch?v=q-roBTkYC3I

The full Heist has 13 pieces, 12 of them comprising 5 squares and 1 comprising 4 squares. The pieces are double sided with different patterns of "locks" and "gems" on each side. The challenge is to use the pieces to tile a specific 8 by 8 pattern. There are a number of patterns of increasing difficulty.

In order to make the data manageable, we will consider a smaller version of the puzzle with a 4 by 4 board and one-sided pieces, as shown below. The pieces can be rotated, but not flipped. This is easy to solve by hand but we will endeavour to come up with a general-purpose integer programming formulation that can be extended to solve larger problems. Construct an integer



programming formulation to solve this problem. How can you use an IP solver to find all solutions?

Solution

Sets

S, set of squares $\{0, \ldots, 15\}$
P, set of pieces $\{0, \ldots, 3\}$
I_p, set of placements for piece $p \in P$
$Squares_{pi}$, set of squares covered by piece $p \in P$ in placement $i \in I_p$

Data

There is no data for this problem.

Variables

$$x_{pi} = \begin{cases} 1, & \text{if piece } p \in P \text{ uses placement } i \in I_p \\ 0, & \text{otherwise} \end{cases}$$

Objective

There is no objective function for this problem, as is the case with many puzzles.

Constraints

Every square is covered exactly once:

$$\sum_{p \in P,\ i \in I_p,\ \text{if } s \in Squares_{pi}} x_{pi} = 1,\ \forall s \in S$$

Each piece is used only once:

$$\sum_{i \in I_p} x_{pi} = 1,\ \forall p \in P$$

Eliminate the previous solution to determine the number of solutions:

$$\sum_{(p,i) \in \text{solution}} x_{pi} \leq 3$$

Code

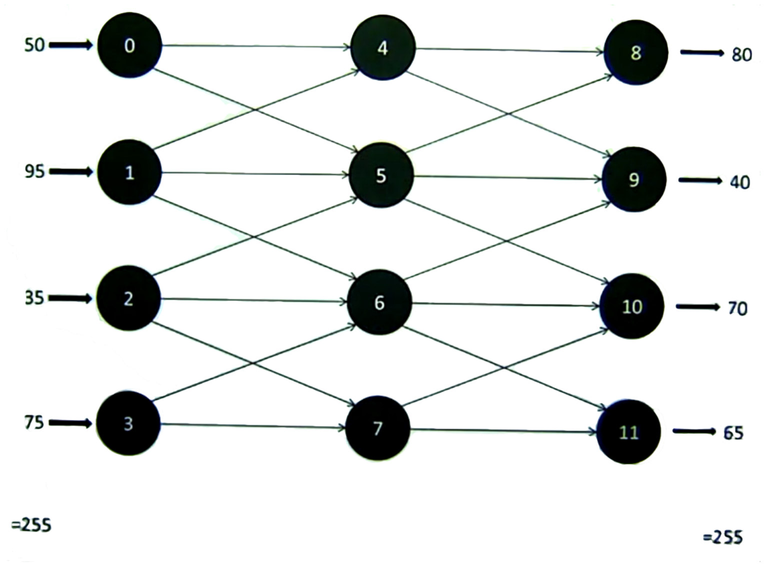
```
from gurobipy import *

#Sets
S = range(16)
P = range(4)

Squares = [
    [(4,8,12,13),(9,10,11,13),(2,3,7,11),(7,9,10,11),(10,12,13,14)],
    [(2,6,9,10),(4,8,9,10),(0,1,4,8),(2,3,6,10),(0,1,2,6),(8,9,10,14)],
    [(0,1,4,5),(1,2,5,6),(5,6,9,10),(4,5,8,9)],
    [(3,7,11,15),(12,13,14,15)]]

I = [range(len(Squares[p])) for p in P]

#Data
#None for this problem

#Model
m = Model("Heist Puzzle")

#Variables
X = {(p,i): m.addVar(vtype = GRB.BINARY) for p in P for i in I[p]}

#Objective
#None for this problem

#Constraints
EverySquare = {m.addConstr(quicksum(X[p,i] for p in P for i in I[p]
                                    if s in Squares[p][i]) == 1)
               for s in S}

EachPiece = {m.addConstr(quicksum(X[p,i] for i in I[p]) == 1)
             for p in P}

#Optimise
    #Note that the following code will determine all of the feasible
    #solutions by adding in a constraint that the sum of the variables
    #must be less than 4 if those variables were 1 in the solution
while True:
    m.optimize()
    if m.status == GRB.INFEASIBLE:
        break

    #Print results
    for p in P:
        for i in I[p]:
            if X[p,i].x > 0.9:
                print(Squares[p][i])
    m.addConstr(quicksum(X[p,i] for (p,i) in X if X[p,i].x > 0.9) <= 3)
```

Network Model

Problem

The following is an example of a (Minimum Cost) Network Flow problem (MCNF).

There is a layered network, seen below, with units of flow being injected in the left-hand side and being taken out at the right-hand side. Each arc (line connecting two nodes) may have a cost per unit flow and we want to move our product through the network in an optimal way.



Note that in models like this,
- the nodes on the left {0, . . . , 3} are known as <u>source nodes</u> and are <u>positive</u>
- the nodes on the right {8, . . . , 11} are known as <u>sink nodes</u> and are <u>negative</u>
- the nodes in the middle {4, . . . , 7} are known as <u>transshipment nodes</u> and have a value of 0

Solution

Sets

N, set of nodes
A, set of arcs

Data

s_n, supply at node $n \in N$
c_a, cost of going along arc $a \in A$
f_a, node going from on arc $a \in A$
t_a, node going to on arc $a \in A$
u_a, upper bound for arc $a \in A$

Variables

x_a, amount that flows through arc $a \in A$

Objective

$$\min Z = \sum_{a \in A} c_a x_a$$

Constraints

Amount that flows through each arc is non-negative:

$$x_a \geq 0$$

Amount that flows through each arc is less than the upper bound:

$$x_a \leq u_a$$

The in-flow is equal to the out-flow:

$$\sum_{a \in A | f_a = n} x_a - \sum_{a \in A | t_a = n} x_a = s_n, \ \forall n \in N$$

Code

```
import math
import random
from gurobipy import *

#Set random seed
random.seed(0)

#Sets
N = range(12)
Arcs = [(0,4), (0,5),
        (1,4), (1,5), (1,6),
        (2,5), (2,6), (2,7),
        (3,6), (3,7),
        (4,8), (4,9),
        (5,8), (5,9), (5,10),
        (6,9), (6,10), (6,11),
        (7,10), (7,11)]
A = range(len(Arcs))

#Data
s = [50,95,35,75,0,0,0,0,-80,-40,-70,-65]
    #random cost data
c = [random.randint(1,100) for a in A]

#Model
m = Model("Network Model")

#Variables
X = [m.addVar() for a in A]
m.update()

#Objective
m.setObjective(quicksum(c[a] * X[a] for a in A))

#Constraints
for n in N:
    m.addConstr(quicksum(X[a] for a in A if Arcs[a][0] == n) -
                quicksum(X[a] for a in A if Arcs[a][1] == n)
                == s[n])

#Optimise
m.optimize()

#Print results
for a in A:
    print(Arcs[a], c[a], X[a].x)
```

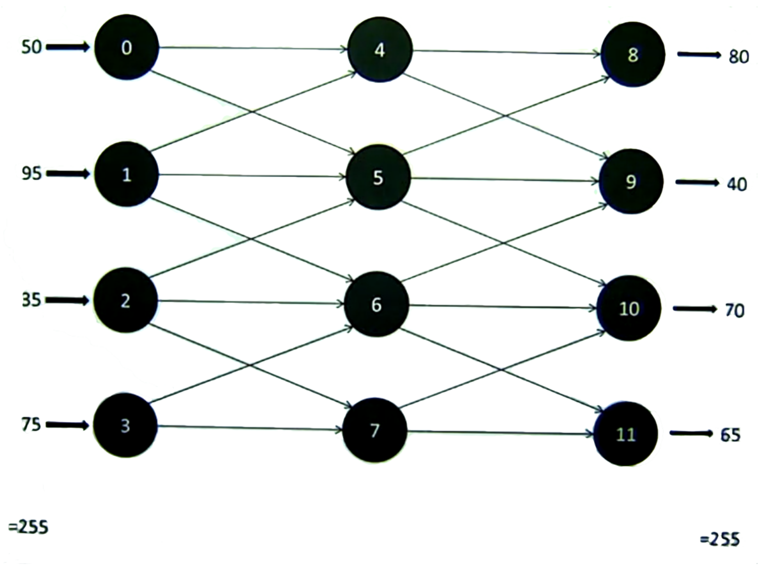
Time Network Model

Problem

This problem is similar to the *Network Model* above, except it now incorporates time into the model. The network can now change between time periods. It also allows for units to be stored at each node.

The following is an example of a multi-period (Minimum Cost) Network Flow problem (MCNF).

There is a layered network, seen below, with units of flow being injected in the left-hand side and being taken out at the right-hand side. Each arc (line connecting two nodes) may have a cost per unit flow and we want to move our product through the network in an optimal way.



Note that in models like this,
• the nodes on the left {0, . . . , 3} are known as <u>source nodes</u> and are <u>positive</u>
• the nodes on the right {8, . . . , 11} are known as <u>sink nodes</u> and are <u>negative</u>
• the nodes in the middle {4, . . . , 7} are known as <u>transshipment nodes</u> and have a value of 0

Solution

Sets

N, set of nodes
A, set of arcs
T, set of time periods

Data

s_{nt}, supply at node $n \in N$ at time $t \in T$
c_a, cost of going along arc $a \in A$
f_a, node going from on arc $a \in A$
t_a, node going to on arc $a \in A$
u_a, upper bound for arc $a \in A$
d_n, holding cost at node $n \in N$

Variables

x_{at}, amount that flows through arc $a \in A$ at time $t \in T$
h_{nt}, amount stored at node $n \in N$ at time $t \in T$

Objective

$$\min Z = \sum_{a \in A} \sum_{t \in T} c_a x_{at} + \sum_{n \in N} \sum_{t \in T} d_n h_{nt}$$

Constraints

Amount that flows through each arc is non-negative:

$$x_{at} \geq 0$$

Amount that flows through each arc is less than the upper bound:

$$\sum_{t \in T} x_{at} \leq u_a, \ \forall t \in T$$

The in-flow is equal to the out-flow:

$$\sum_{a \in A | f_a = n} (x_{at} + h_{nt}) - \sum_{a \in A | t_a = n} \left(x_{a(t-1)} - h_{n(t-1)} \right) = s_{nt}, \ \forall n \in N, \ t \in T$$

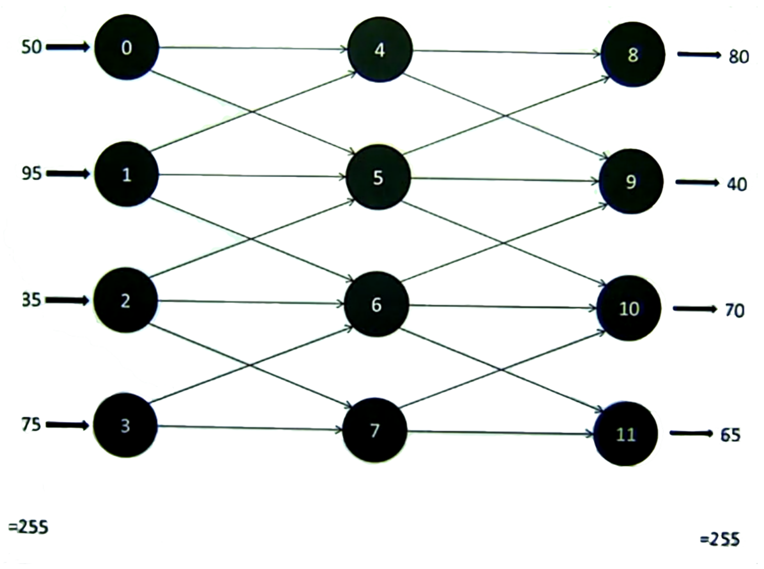
Note that we also need some boundary conditions for this constraint.

Multi-Commodity Network Model

Problem

The following is an example of a Multi-Commodity (Minimum Cost) Network Flow problem (MCNF). It is similar to the *Network Model* problem, except it incorporates different commodities. For example, assuming the network represents a mail service, it does not make sense to assume that all mail is the same, as each mail item must go to a specific location.

There is a layered network, seen below, with units of flow being injected in the left-hand side and being taken out at the right-hand side. Each arc (line connecting two nodes) may have a cost per unit flow and we want to move our product through the network in an optimal way.



Note that in models like this,
• the nodes on the left {0, . . . , 3} are known as <u>source nodes</u> and are <u>positive</u>
• the nodes on the right {8, . . . , 11} are known as <u>sink nodes</u> and are <u>negative</u>
• the nodes in the middle {4, . . . , 7} are known as <u>transshipment nodes</u> and have a value of 0

Solution

Sets

N, set of nodes
A, set of arcs
P, set of commodities

Data

s_n, supply at node $n \in N$
c_a, cost of going along arc $a \in A$
f_a, node going from on arc $a \in A$
t_a, node going to on arc $a \in A$
u_a, upper bound for arc $a \in A$

Variables

x_{ap}, amount that flows through arc $a \in A$ of commodity $p \in P$

Objective

$$\min Z = \sum_{a \in A} \sum_{p \in P} c_{ap} x_{ap}$$

Constraints

Amount that flows through each arc is non-negative:

$$x_{ap} \geq 0$$

Amount that flows through each arc is less than the upper bound:

$$\sum_{p \in P} x_{ap} \leq u_a, \ \forall a \in A$$

The in-flow is equal to the out-flow:

$$\sum_{a \in A | f_a = n} x_{ap} - \sum_{a \in A | t_a = n} x_{ap} = s_{np}, \ \forall n \in N, \ p \in P$$

Generalised Network Model

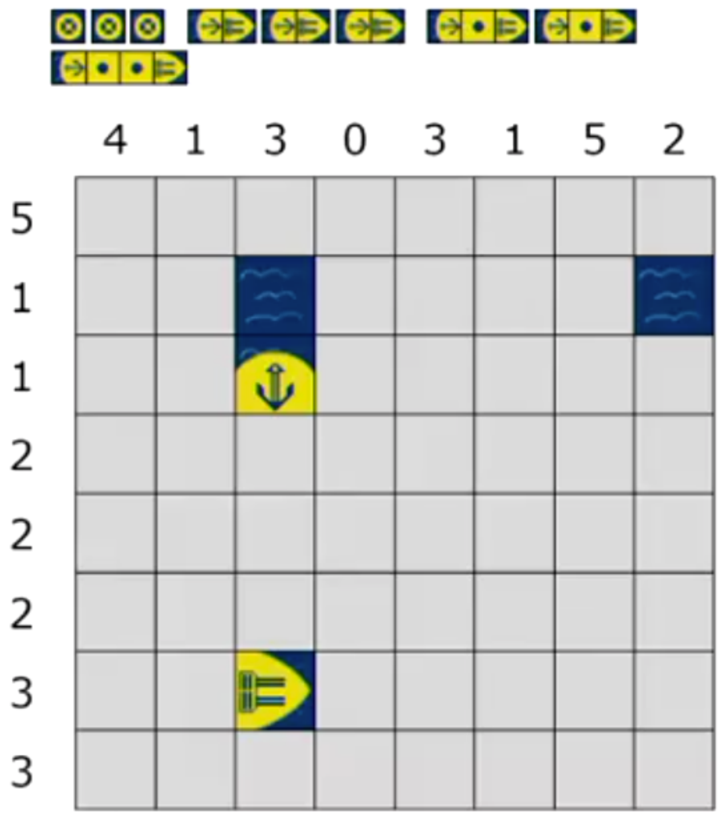
Note that there is another type of network model called the Generalised Network Flow Model. In this network, the in-flow does not necessarily have to equal the out-flow from each note. In these situations, there is a gain factor on each arc:

$$\lambda_a > 0$$

Battleships

Problem

A game of Battleships works where you are given a starting grid, as seen in the image below. You have to place battleships of different length (seen in the top of the image) on the grid so that each battleship is surrounded by water and so that the sum of squares with a battleship for each row equals the number of that row and similarly for each column.



Solution

Sets

I, rows on grid $\{0, \ldots, 7\}$ (going from top to bottom on grid)
J, columns on grid $\{0, \ldots, 7\}$ (going from left to right on grid)
S, length of ships $\{1, \ldots, 4\}$

Data

n_s, number of ships available of length $s \in S$
$ncol_j$, number of ships required in column $j \in J$
$nrow_i$, number of ships required in row $i \in I$

Variables

x^V_{ijs}, vertical ship of length $s \in S | s > 2$ at square of row $i \in I,\ j \in J$
x^H_{ijs}, horizontal ship of length $s \in S | s > 2$ at square of row $i \in I,\ j \in J$

Objective

No objective exists for this problem

Constraints

Number of ships of each length must be number of ships available:

$$\sum_{j \in J} \sum_{i \in I} x^V_{ijs} + \sum_{j \in J} \sum_{i \in I} x^H_{ijs} = n_s,\ \forall s \in S$$

Sum of ships in each column must equal ships required for column:

$$\sum_{i \in I} \sum_{s \in S} s \times x^V_{ijs} + \sum_{i \in I} \sum_{j' \in J | \text{ship intersects } j \in J} \sum_{s \in S} x^H_{ij's} = ncol_j,\ \forall j \in J$$

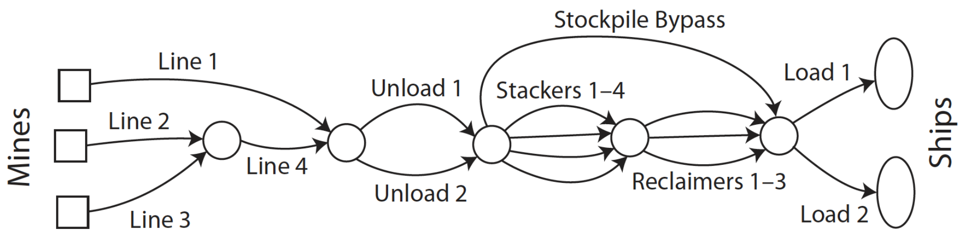
Ship position only exists if it doesn't violate any pre-assignments or requirements (i.e. run off the board or on any squares that must be water, etc.):

$$x^H_{ijs},\ x^V_{ijs} \text{ do not exist if they violate requirements}$$

Coal Line Maintenance

Problem

Throughput of coal from mine to port to ship is a critical issue for Australia's largest coal systems. Press reports of huge queues of ships waiting to be loaded are common and these queues are expensive for the industry. A typical but simple diagram for a coal system is shown below.



Coal is moved by train from the coal mines along the lines until it meets the shared unloading facility, where there are two unloaders. From the unloaders it is moved via conveyer belt into the stockpile, using stackers. From the stackers it is removed using reclaimers, and again moved by conveyer belt through the loaders on to ships. There is an option to bypass the stockpile completely.

Problem - Part A

More generally, we can describe a coal system using a collection of nodes and arcs. Some of the nodes are source nodes (mines) and some are sink nodes (ships). For each arc we know the origin and destination nodes and the maximum weekly throughput of the arc.

(i) Develop a linear programming model of a general coal system that determines how much coal to move on each arc so as to maximise the total throughput. This throughput is the total amount of coal moved out of the source nodes, which will be equal to the total amount of coal moved into the sink nodes. For all other nodes the total amount of coal moved into the node will be same as the total amount of coal moved out of the node. For the purposes of this model you can ignore the time lag of coal moving through the system with respect to the weekly schedule.

(ii) In order to keep the system running smoothly, it needs to be maintained. Assume we are given a set of maintenance tasks applying to the arcs, with at most one task for each arc. For each arc we know whether or not it has a maintenance task and the effort (in man days) for the maintenance task.

We wish to schedule all the known maintenance tasks over the next T weeks. For each week we know the maximum man days available for maintenance, which may vary from week to week.

Assume that each maintenance task must be started and finished in the same week, and that when an arc is being maintained its throughput goes down to 0 for the whole week.

Develop a mixed integer programming model to produce a maintenance schedule for the next T weeks so as to maximise the total throughput.

Solution - Part A

Sets

N, set of nodes
A, set of arcs
T, weeks

Data

cap_a, capacity for arc $a \in A$
f_a, node going from on arc $a \in A$
t_a, node going to on arc $a \in A$
δ_a, 1 if arc $a \in A$ has maintenance; 0 otherwise
d_a, man days for maintaining arc $a \in A$
$days_t$, maximum man days for week $t \in T$

Variables

x_{at}, amount flowing on arc $a \in A$ in week $t \in T$
y_{at}, 1 if maintain arc $a \in A$ in week $t \in T$; 0 otherwise

Objective

$$\max Z = \sum_{t \in T} x_{back,\ t}$$

Constraints

$$x_{at} \leq cap_a(1 - y_{at}),\ \forall a \in A,\ t \in T$$

$$\sum_{a \in A | t_a = n} x_{at} = \sum_{a \in A | f_a = n} x_{at},\ \forall n \in N,\ t \in T$$

$$\sum_{a \in A} y_{at} \times d_a \leq days_t,\ \forall t \in T$$

$$\sum_{t \in T} y_{at} = \delta_a,\ \forall a \in A$$

$$x_{at} \geq 0,\ \forall a \in A,\ t \in T$$

$$y_{at} \in \{0, 1\},\ \forall a \in A,\ t \in T$$

Problem - Part B

Consider this model for particular data where each of the segments has the following maximum daily throughput (in thousands of tonnes per day):

Section	Throughput	Section	Throughput
Line 1	100	Stacker 1	40
Line 2	60	Stacker 2	40
Line 3	60	Stacker 3	40
Line 4	100	Stacker 4	40
Unload 1	80	Reclaim 1	50
Unload 2	80	Reclaim 2	50
Stockpile Bypass	20	Reclaim 3	50
		Load 1	75
		Load 2	75

The company needs to plan maintenance activities for the next four weeks and have the following sections requiring work:

Section	Effort
Line 3	50
Unload 2	15
Stockpile Bypass	55
Stacker 1	30
Stacker 2	20
Stacker 3	70
Stacker 4	20
Reclaim 1	35
Reclaim 2	35
Load 1	45

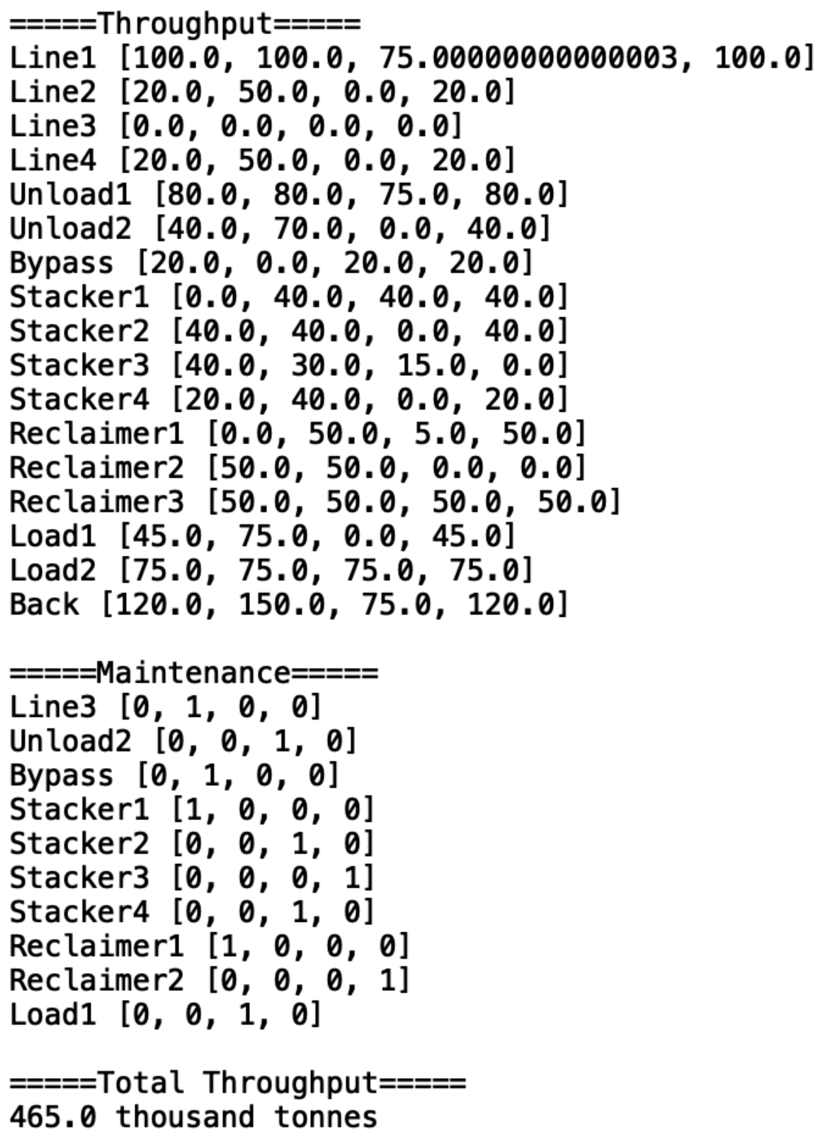
The effort estimates are given in person-days. Each week they can carry out a maximum of 110 person-days of maintenance. Each maintenance activity must be carried out completely within a week. What is the optimal maintenance schedule that will give the maximum throughput in those four weeks?

Solution - Part B

Code

```
from gurobipy import *

#Sets
N = [0,3,4,5,6,7,8]

A = {'Line1': (0,4),
     'Line2': (0,3),
     'Line3': (0,3),
     'Line4': (3,4),
     'Unload1': (4,5),
     'Unload2': (4,5),
     'Bypass': (5,7),
     'Stacker1': (5,6),
     'Stacker2': (5,6),
     'Stacker3': (5,6),
     'Stacker4': (5,6),
     'Reclaimer1': (6,7),
     'Reclaimer2': (6,7),
     'Reclaimer3': (6,7),
     'Load1': (7,8),
     'Load2': (7,8),
     'Back': (8,0)}

T = range(4)

#Data
cap = {'Line1': 100,
       'Line2': 60,
       'Line3': 60,
       'Line4': 100,
       'Unload1': 80,
       'Unload2': 80,
       'Bypass': 20,
       'Stacker1': 40,
       'Stacker2': 40,
       'Stacker3': 40,
       'Stacker4': 40,
       'Reclaimer1': 50,
       'Reclaimer2': 50,
       'Reclaimer3': 50,
       'Load1': 75,
       'Load2': 75}

maintain = {'Line3': 50,
            'Unload2': 15,
            'Bypass': 55,
            'Stacker1': 30,
            'Stacker2': 20,
            'Stacker3': 70,
            'Stacker4': 20,
               'Reclaimer1': 35,
               'Reclaimer2': 35,
               'Load1': 45}

days = [110 for t in T]

#Model
m = Model("Coal Line Maintenance")

#Variables
X = {(a,t): m.addVar() for a in A for t in T}
Y = {(a,t): m.addVar(vtype = GRB.BINARY) for a in A for t in T}

#Objective
m.setObjective(quicksum(X['Back',t] for t in T), GRB.MAXIMIZE)

#Constraints
c1 = [m.addConstr(X[a,t] <= cap[a] * (1 - Y[a,t]))
      for a in cap for t in T]

c2 = [m.addConstr(quicksum(X[a,t] for a in A if A[a][1] == n) ==
                  quicksum(X[a,t] for a in A if A[a][0] == n))
      for n in N for t in T]

c3 = [m.addConstr(quicksum(Y[a,t] * maintain[a] for a in maintain) <=
                  days[t]) for t in T]

c4 = [m.addConstr(quicksum(Y[a,t] for t in T) == 1) for a in maintain]

#Optimise
m.optimize()

#Print results
print("=====Throughput=====")
for a in A:
    print(a, [X[a,t].x for t in T])

print("\n=====Maintenance=====")
for a in maintain:
    print(a, [round(Y[a,t].x) for t in T])

print("\n=====Total Throughput=====")
print(m.objVal, "thousand tonnes")
```

This gave the following result:



```
=====Throughput=====
Line1 [100.0, 100.0, 75.00000000000003, 100.0]
Line2 [20.0, 50.0, 0.0, 20.0]
Line3 [0.0, 0.0, 0.0, 0.0]
Line4 [20.0, 50.0, 0.0, 20.0]
Unload1 [80.0, 80.0, 75.0, 80.0]
Unload2 [40.0, 70.0, 0.0, 40.0]
Bypass [20.0, 0.0, 20.0, 20.0]
Stacker1 [0.0, 40.0, 40.0, 40.0]
Stacker2 [40.0, 40.0, 0.0, 40.0]
Stacker3 [40.0, 30.0, 15.0, 0.0]
Stacker4 [20.0, 40.0, 0.0, 20.0]
Reclaimer1 [0.0, 50.0, 5.0, 50.0]
Reclaimer2 [50.0, 50.0, 0.0, 0.0]
Reclaimer3 [50.0, 50.0, 50.0, 50.0]
Load1 [45.0, 75.0, 0.0, 45.0]
Load2 [75.0, 75.0, 75.0, 75.0]
Back [120.0, 150.0, 75.0, 120.0]

=====Maintenance=====
Line3 [0, 1, 0, 0]
Unload2 [0, 0, 1, 0]
Bypass [0, 1, 0, 0]
Stacker1 [1, 0, 0, 0]
Stacker2 [0, 0, 1, 0]
Stacker3 [0, 0, 0, 1]
Stacker4 [0, 0, 1, 0]
Reclaimer1 [1, 0, 0, 0]
Reclaimer2 [0, 0, 0, 1]
Load1 [0, 0, 1, 0]

=====Total Throughput=====
465.0 thousand tonnes
```

Problem - Part C

They would also like to examine the impact if they could add some additional requirements that would assist the maintenance teams. Specifically, the following would help:

(1) Carrying out maintenance on Stockpile Bypass in the first week.
(2) Carrying out maintenance on Stacker 3 before Stacker 4.
(3) Finishing maintenance on Stacker 2 at least one week before maintenance on Stacker 1 is started.

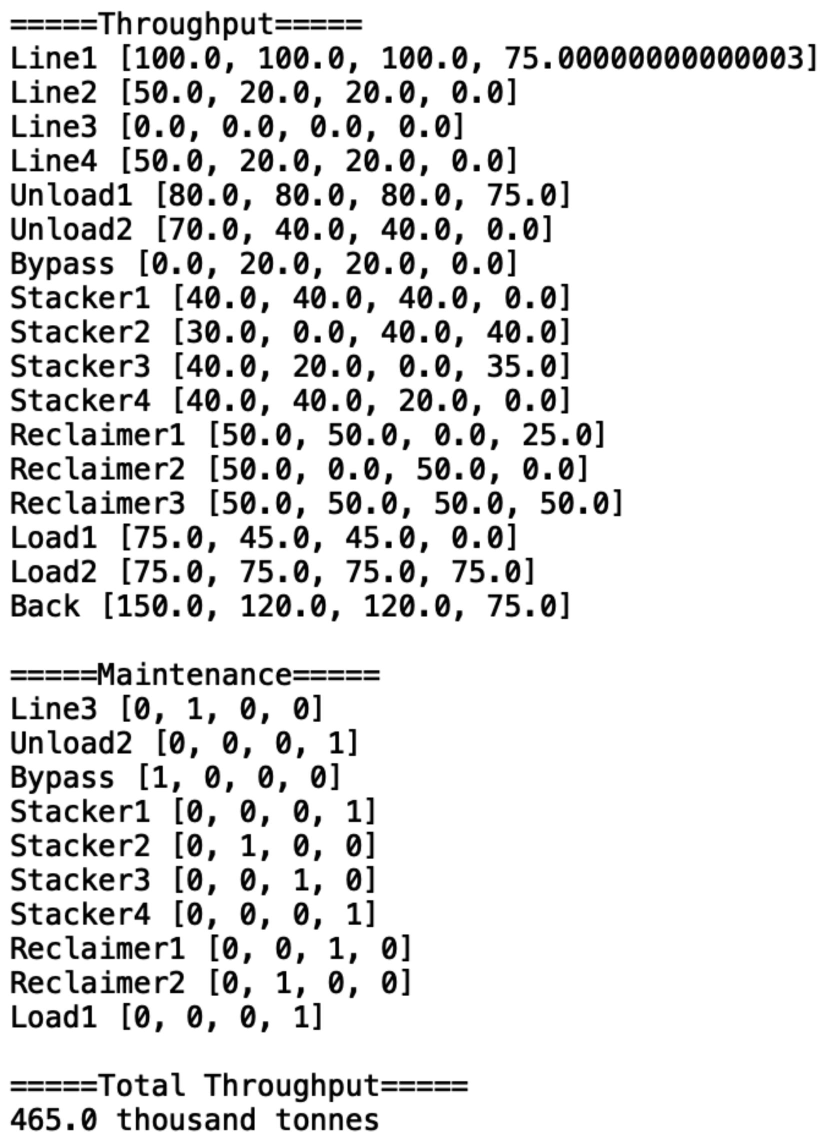
How would these constraints affect the maximum throughput?

Solution - Part C

Code

```
from gurobipy import *

#Sets
N = [0,3,4,5,6,7,8]

A = {'Line1': (0,4),
     'Line2': (0,3),
     'Line3': (0,3),
     'Line4': (3,4),
     'Unload1': (4,5),
     'Unload2': (4,5),
     'Bypass': (5,7),
     'Stacker1': (5,6),
     'Stacker2': (5,6),
     'Stacker3': (5,6),
     'Stacker4': (5,6),
     'Reclaimer1': (6,7),
     'Reclaimer2': (6,7),
     'Reclaimer3': (6,7),
     'Load1': (7,8),
     'Load2': (7,8),
     'Back': (8,0)}

T = range(4)

#Data
cap = {'Line1': 100,
       'Line2': 60,
       'Line3': 60,
       'Line4': 100,
       'Unload1': 80,
       'Unload2': 80,
       'Bypass': 20,
       'Stacker1': 40,
       'Stacker2': 40,
       'Stacker3': 40,
       'Stacker4': 40,
       'Reclaimer1': 50,
       'Reclaimer2': 50,
       'Reclaimer3': 50,
       'Load1': 75,
       'Load2': 75}

maintain = {'Line3': 50,
            'Unload2': 15,
            'Bypass': 55,
            'Stacker1': 30,
            'Stacker2': 20,
            'Stacker3': 70,
            'Stacker4': 20,
              'Reclaimer1': 35,
              'Reclaimer2': 35,
              'Load1': 45}

days = [110 for t in T]

#Model
m = Model("Coal Line Maintenance")

#Variables
X = {(a,t): m.addVar() for a in A for t in T}
Y = {(a,t): m.addVar(vtype = GRB.BINARY) for a in A for t in T}

#Objective
m.setObjective(quicksum(X['Back',t] for t in T), GRB.MAXIMIZE)

#Constraints
c1 = [m.addConstr(X[a,t] <= cap[a] * (1 - Y[a,t]))
      for a in cap for t in T]

c2 = [m.addConstr(quicksum(X[a,t] for a in A if A[a][1] == n) ==
                  quicksum(X[a,t] for a in A if A[a][0] == n))
      for n in N for t in T]

c3 = [m.addConstr(quicksum(Y[a,t] * maintain[a] for a in maintain) <=
                  days[t]) for t in T]

c4 = [m.addConstr(quicksum(Y[a,t] for t in T) == 1) for a in maintain]

c5 = m.addConstr(Y['Bypass',0] == 1)

c6 = [m.addConstr(Y['Stacker4',t] <= quicksum(Y['Stacker3',u] for u in T
                if u < t)) for t in T]

c7 = [m.addConstr(Y['Stacker1',t] <= quicksum(Y['Stacker2',u] for u in T
                  if u+1 < t)) for t in T]

#Optimise
m.optimize()

#Print results
print("=====Throughput=====")
for a in A:
    print(a, [X[a,t].x for t in T])

print("\n=====Maintenance=====")
for a in maintain:
    print(a, [round(Y[a,t].x) for t in T])

print("\n=====Total Throughput=====")
print(m.objVal, "thousand tonnes")
```

This gave the following result:



```
=====Throughput=====
Line1 [100.0, 100.0, 100.0, 75.00000000000003]
Line2 [50.0, 20.0, 20.0, 0.0]
Line3 [0.0, 0.0, 0.0, 0.0]
Line4 [50.0, 20.0, 20.0, 0.0]
Unload1 [80.0, 80.0, 80.0, 75.0]
Unload2 [70.0, 40.0, 40.0, 0.0]
Bypass [0.0, 20.0, 20.0, 0.0]
Stacker1 [40.0, 40.0, 40.0, 0.0]
Stacker2 [30.0, 0.0, 40.0, 40.0]
Stacker3 [40.0, 20.0, 0.0, 35.0]
Stacker4 [40.0, 40.0, 20.0, 0.0]
Reclaimer1 [50.0, 50.0, 0.0, 25.0]
Reclaimer2 [50.0, 0.0, 50.0, 0.0]
Reclaimer3 [50.0, 50.0, 50.0, 50.0]
Load1 [75.0, 45.0, 45.0, 0.0]
Load2 [75.0, 75.0, 75.0, 75.0]
Back [150.0, 120.0, 120.0, 75.0]

=====Maintenance=====
Line3 [0, 1, 0, 0]
Unload2 [0, 0, 0, 1]
Bypass [1, 0, 0, 0]
Stacker1 [0, 0, 0, 1]
Stacker2 [0, 1, 0, 0]
Stacker3 [0, 0, 1, 0]
Stacker4 [0, 0, 0, 1]
Reclaimer1 [0, 0, 1, 0]
Reclaimer2 [0, 1, 0, 0]
Load1 [0, 0, 0, 1]

=====Total Throughput=====
465.0 thousand tonnes
```

As seen, the maximum throughput did not change with these added constraints.