Integer Programming – Continued

Linear Assignment 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. It needs to assign each job to a plant so as to minimise the total distance between jobs and plants.

In general terms, this can be written as:

$$min \sum_{i,j} c_{ij} x_{ij}$$

Subject to:

$$\sum_i x_{ij} = 1 \quad \forall\, j \qquad \text{LAP(1)}$$
$$\sum_j x_{ij} = 1 \quad \forall\, i \qquad \text{LAP(2)}$$
$$x_{ij} \in \{0,1\} \quad \forall\, i,j$$

Where x_{ij} is 1 if job i is assigned to plant j. The equations enforce the requirement that each plant has one job assigned to it, and vice versa. It is easy to formulate and solve these sorts of problems in python. See LAP.py

It turns out that assignment problems are "totally unimodular". This means that the solutions of the LP relaxation will be integer.

Definition: A unimodular matrix is a square matrix of integers with determinant of $+1$ or -1. Equivalently, it is an integer matrix that is invertible over the integers. A totally unimodular matrix is a matrix for which every square non-singular sub-matrix is unimodular. This implies that any totally unimodular matrix has only 0, $+1$ or -1 entries.

Theorem: Let A be an m by n matrix whose rows can be partitioned into two disjoint sets B and C, with the following properties:

- Every column of A contains at most two non-zero entries;
- Every entry in A is 0, $+1$ or -1;
- If two non-zero entries in a column of A have the same sign, then the row of one is in B, and the other in C;
- If two non-zero entries in a column of A have opposite signs, then the rows of both are in B, or both in C.

Then every minor determinant of A is 0, $+1$ or -1.

□

By inspection we can see that LAP satisfies the conditions of this theorem.

Assignment problems can be solved by specialist techniques that take note of the following observation: the optimal solution of the assignment problem does not change if a constant is added to or subtracted from any row or column of the cost matrix C defined by c_{ij}. In this case, all that happens is that the objective function changes by the constants applied.

We define such row and column modifications to the costs as u_i and v_j and define $c'_{ij} = c_{ij} - u_i - v_j$. We can solve the Linear Assignment Problem if we can find x_{ij}, u_i and v_j, with x_{ij} satisfying LAP(1) and LAP(2), and such that $c'_{ij} = 0$ if $x_{ij} = 1$ and $c'_{ij} \geq 0$ if $x_{ij} = 0$. The optimal objective value will then be: $\sum_i u_i + \sum_j v_j$.

(Alert readers will observe that c'_{ij} correspond to reduced costs, u_i and v_j correspond to dual variables and the conditions given are the complementary slackness conditions).

Consider the cost matrix from the example:

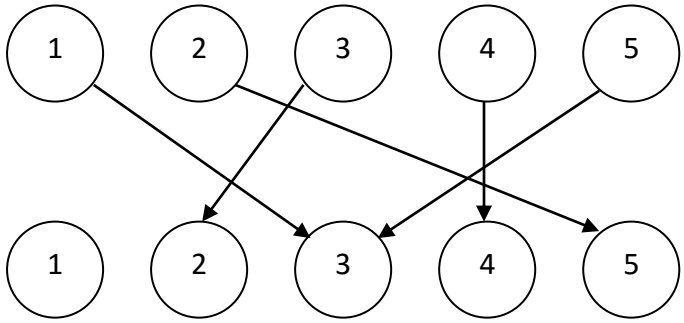
	1	**2**	**3**	**4**	**5**
1	48	56	31	59	34
2	47	54	94	74	36
3	67	12	67	31	43
4	74	31	37	23	55
5	60	41	27	39	44

If we extract the minimum entry from each row, we get the following modified cost matrix.

	1	**2**	**3**	**4**	**5**	u_i
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
v_j	0	0	0	0	0	129

This would not constitute a legal solution because if we use the zero cost assignments then two rows are assigned to column 3, and none are assigned to column 1.

This can be drawn as the following graph:



We need to find the cheapest series of swaps that will move an assignment away from 3 and add an assignment to 1. The series of changes is to move assignment (1,3) to (1,5) which costs 3; and move (2,5) to (2,1) which costs 11. This can be done with the following adjustments to u_i and v_j:
u_1: +3, v_3: -3, u_5: +3, v_1: +11. This gives the new matrix:

	1	**2**	**3**	**4**	**5**	u_i
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
v_j	11	0	-3	0	0	143

You can use dynamic programming to find a chain of swaps like this.

(NB: In class we extracted the minimum from each row and column before doing an update, which resulted in a smaller update).

Linear assignment problems are also known as bipartite matching problems. Non-bipartite matching problems are another interesting case.

See Matching.py (not done in class).

More Puzzles

Consider the problem of filling 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. This is more or less trivial for 2D noughts and crosses but what we present here is easily extendable to 3D.

We number the squares as follows:

0	1	2
3	4	5
6	7	8

Let x_i, be 1 if square i is an X, 0 otherwise. We need to associate an indicator variable y_l with every possible line so that y_l must be 1 if the line is all X's or all O's. In the case of the top line (call this line 0) we can do this by:

$$x_0 + x_1 + x_2 + y_0 \geq 1$$
$$x_0 + x_1 + x_2 - y_0 \leq 2$$

Clearly if the line is all O's then y_0 needs to be 1 to satisfy the first constraint. If the line is all X's then y_0 needs to be 1 to satisfy the second constraint.

More generally for each line l we can define s_{l0}, s_{l1} and s_{l2} to be the indices of the squares in line l. Then for line we have a pair of constraints:

$$x_{s_{l0}} + x_{s_{l1}} + x_{s_{l2}} + y_l \geq 1$$
$$x_{s_{l0}} + x_{s_{l1}} + x_{s_{l2}} - y_l \leq 2$$

We wish to minimise the sum of y_l, and we need to constrain the sum of x_i. This can be done in a straightforward way. See XO.py

Modelling non-linear constraints and objective function

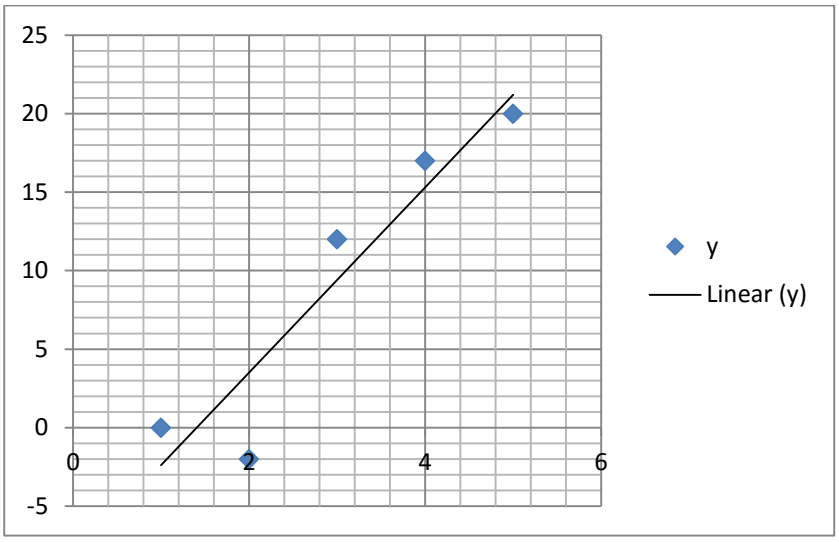
Data Fitting (Not done in class)

Consider the following data.

x	1	2	3	4	5
y	0	-2	12	17	20

We can fit a linear regression to this data in the usual way, to get the least squares fit:

$$y = 5.9x - 8.3$$



Note that this fit is strongly influenced by the second data point, which has the biggest error. The linear regression in effect solves the following problem:

$$min \sum_i e_i^2$$

Subject to:

$$y_i = a_1 x_i + a_0 + e_i \qquad \forall i$$

Here y_i and x_i are data and a_1, a_0 and e_i are variables.

The distortion caused by one data point suggests that it might be more stable to solve:

$$min \sum_i |e_i|$$

Subject to:

$$y_i = a_1 x_i + a_0 + e_i \qquad \forall i$$

But how do we model the non-linear (and indeed non-differentiable at one point) absolute value function using linear variables?

In this case we can take advantage of the minimising nature of the objective function, and write:

$$min \sum_i e_i^+ + e_i^-$$

Subject to:

$$y_i = a_1 x_i + a_0 + e_i^+ - e_i^- \qquad \forall i$$

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

Any error can be represented by a combination of over and under errors. The minimisation requirement means that at least one of these errors will be 0 for each data point.

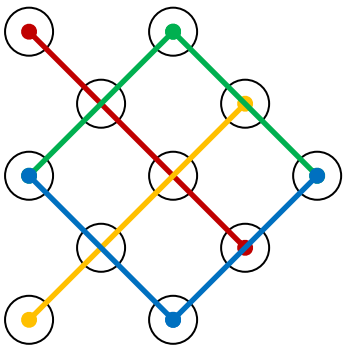
Fit is $y = 5x - 5$

This "linear fit" has the interesting property that it is sure to pass through two of the data points. It is easy to extend to multiple dimensions and it is also easy to add side constraints – a key advantage over least squares estimation.

So can all absolute value functions be treated this way?

Railway Conflict Avoidance

(Not done in class) Consider a rail network, for example the network below:



Assume that the different colours are different lines and, for the purposes of this example, all trains go left to right. For each line we need to determine the departure time. For each

segment of line we need to determine the travel time we will use, subject to a minimum allowed travel time (m_{ln} in the notation below). At each node, we require that any two trains are separated by at least two minutes. We wish to minimise the elapsed time, starting from time zero, for all trains to arrive at their destination.

Consider the following formulation:

t_{l0}	The time the train on line l starts
t_{ln}	The time the train on line l arrives at node n
d_{ln}	Segment duration for the train on line l immediately before node n
D	The overall duration

$$\min D$$

Subject to:

$t_{ln} = t_{ln'} + d_{ln} \;\; \forall\, l, n, n'$ s.t. n' is the node before n on line l
$t_{l0} \geq 0 \;\; \forall\, l$
$d_{ln} \geq m_{ln} \;\; \forall\, l, n$
$D \geq t_{ln} \;\; \forall\, l, n$ s.t. n is the last node on line l
$|t_{l_1 n} - t_{l_2 n}| \geq 2 \;\; \forall\, n, l_1, l_2$ s.t. l_1, l_2 pass through n

This formulation is fine (note in passing that D is the maximum of the t_{ln}), *except* for the last constraint involving absolute value. Assume we do something similar to our previous example and write:

$$t_{l_1 n} - t_{l_2 n} = e^{+}_{l_1 l_2 n} - e^{-}_{l_1 l_2 n}$$
$$e^{+}_{l_1 l_2 n} + e^{-}_{l_1 l_2 n} \geq 2$$

The problem with this is that the e variables can both be increased to satisfy both equations.

To fix this we need to add the pair of equations:

$$e^{+}_{l_1 l_2 n} \leq y_{l_1 l_2 n} M$$
$$e^{-}_{l_1 l_2 n} \leq (1 - y_{l_1 l_2 n}) M$$

Here $y_{l_1 l_2 n}$ is a binary variable and M is a constant chosen large enough to cover any reasonable separation.

Piecewise linear objective function

(Done a different way in class – see Euing.py. Code also uses Gurobi's PWL capability).

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.

Define x as the amount of oil 1 purchased and x_{ij}as the amount of oil i used to produce petrol j $(i, j = 1{,}2)$. Define the following cost function for purchasing oil 1:

$$c(x) = \begin{cases} 2.5x & (0 \leq x \leq 500) \\ 2x + 250 & (500 \leq x \leq 1{,}000) \\ 1.5x + 750 & (1000 \leq x \leq 1{,}500) \end{cases}$$

The formulation is then:

$$max\ 1.2(x_{11} + x_{21}) + 1.4(x_{12} + x_{22}) - c(x)$$

Subject to:

$$\begin{aligned} & x_{11} + x_{12} \leq 500 + x \\ & x_{21} + x_{22} \leq 1{,}000 \\ & x_{11} \geq x_{21} \\ & x_{12} \geq 1.5x_{22} \\ & x_{ij} \geq 0, 0 \leq x \leq 1{,}500 \end{aligned}$$

How do we recast $c(x)$ using integer variables?

$$\begin{aligned} & c(x) = z_1 c(0) + z_2 c(500) + z_3 c(1{,}000) + z_4 c(1{,}500) \\ & x = 0z_1 + 500z_2 + 1{,}000z_3 + 1{,}500z_4 \\ & z_1 \leq y_1, z_2 \leq y_1 + y_2, z_3 \leq y_2 + y_3, z_4 \leq y_3 \\ & z_1 + z_2 + z_3 + z_4 = 1;\ y_1 + y_2 + y_3 = 1 \\ & y_i \in \{0{,}1\};\ z_i \geq 0 \end{aligned}$$

This formulation works because exactly one y_i will be one. This in turn means that only z_i and z_{i+1} can be positive and all other z_i's are zero. This means the appropriate portion of $c(x)$ can be linearly interpolated.

The optimal solution of this problem is to purchase 1,000 litres of oil 1 and use all of oil 1 and oil 2 to make blend 2.

This is an example of what solver packages refer to as Special Ordered Sets.