AI Driven Decision Making

Integer Programming

Based on notes from S. Prestwich and D. Rey
http://publish.ucc.ie/researchprofiles/D005/sprestwich
https://www.skema.edu/en/faculty-and-research/professors/david-rey

Today’s Outline

- Week 2’s lab
- Integer Programming
- 3-Queens Problem
- Warehouse Location Problem
- Logical Constraints
- Travelling Salesman Problem

Integer Programming

- In LP we allow variables to take fractional values, even if this doesn't make sense for our application
    - we assume that rounding the values to the nearest integer will still give a feasible and almost-optimal solution.
    - Last week, we produced 81.82 objects

- In general, we can't do this, because we might get a very sub-optimal solution or even an infeasible one.

- For many problems fractional values are useless, but instead of LP we can use *Integer Programming* (IP, sometimes written ILP where *L* stands for *Linear*).

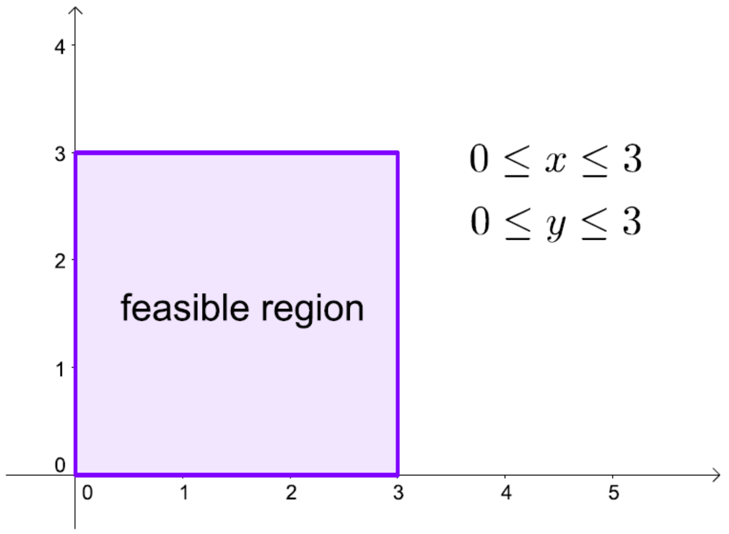
- An IP is simply an LP with extra constraints stating that all the variables must take discrete values (usually integers).

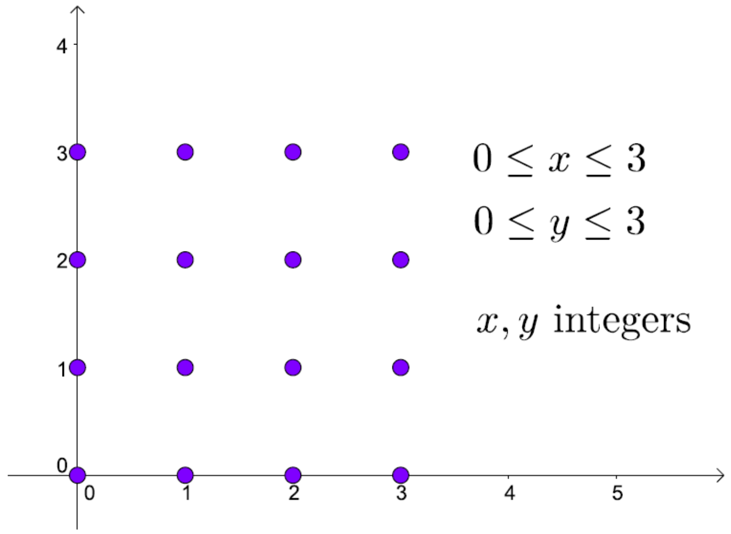
- A problem containing both integer and continuous variables is a *Mixed Integer Program* (MIP or sometimes MILP).

Integer vs Linear Optimization

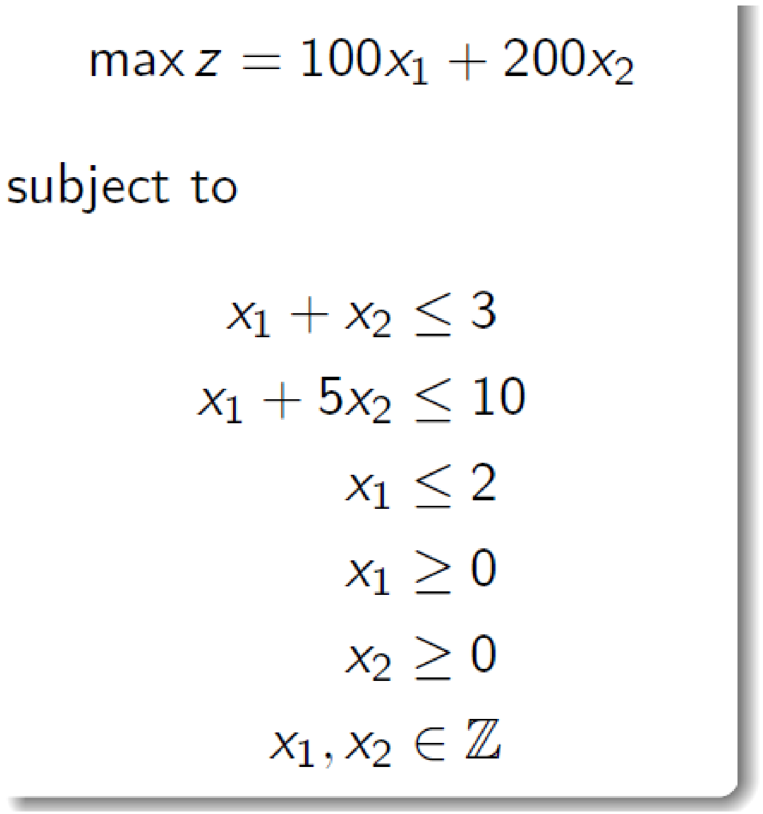
- Although in principle adding constraints makes the problem “smaller” in some sense, IP and MIP are much *harder* to solve than LP.

- There are polynomial-time algorithms for LP but not for IP/MIP because they are NP-hard.





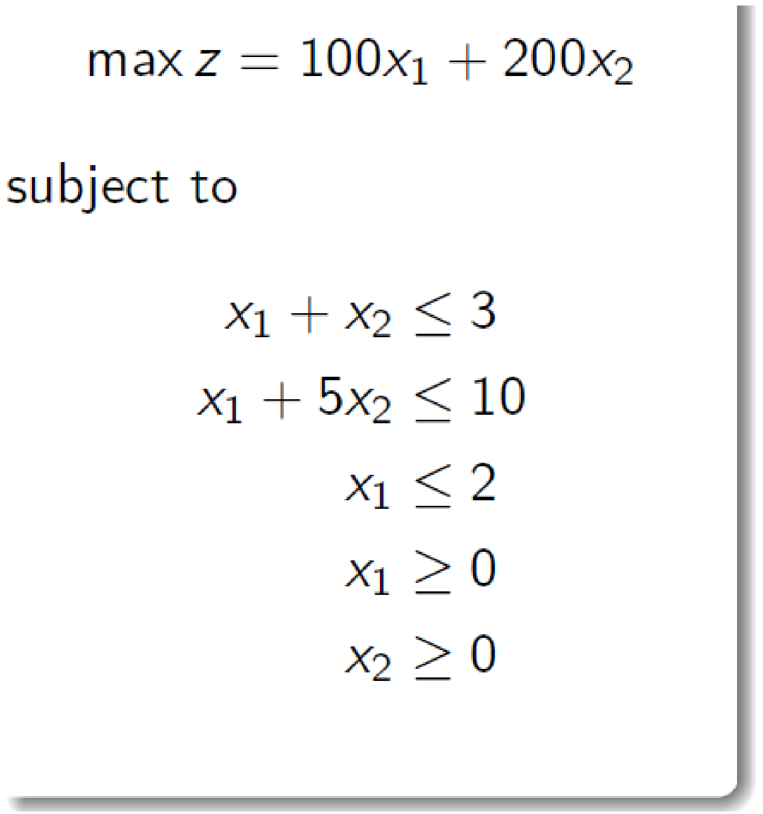
Only the blue dots are feasible solutions of the IP whereas the entire interior of the polyhedron is feasible for the LP



$$\max z = 100x_1 + 200x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 3 \\ x_1 + 5x_2 &\leq 10 \\ x_1 &\leq 2 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \\ x_1, x_2 &\in \mathbb{Z} \end{aligned}$$

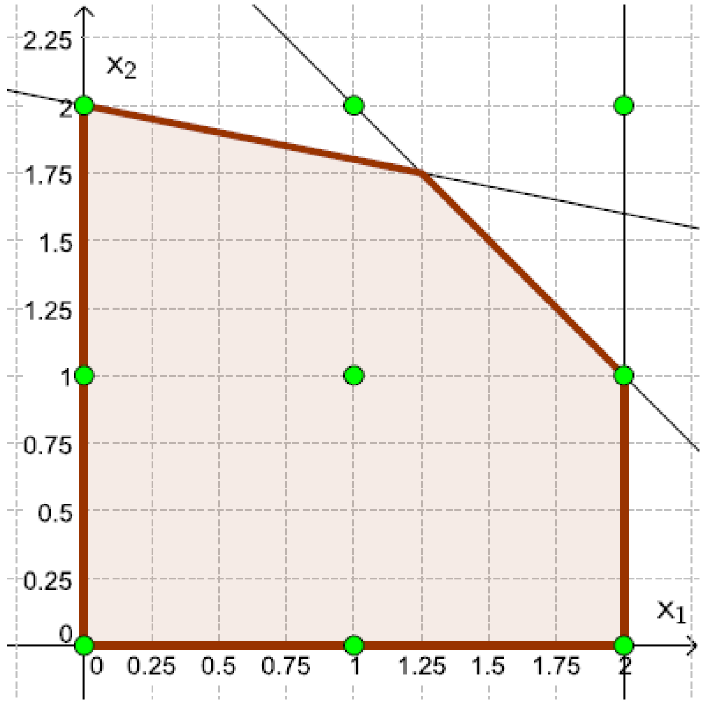


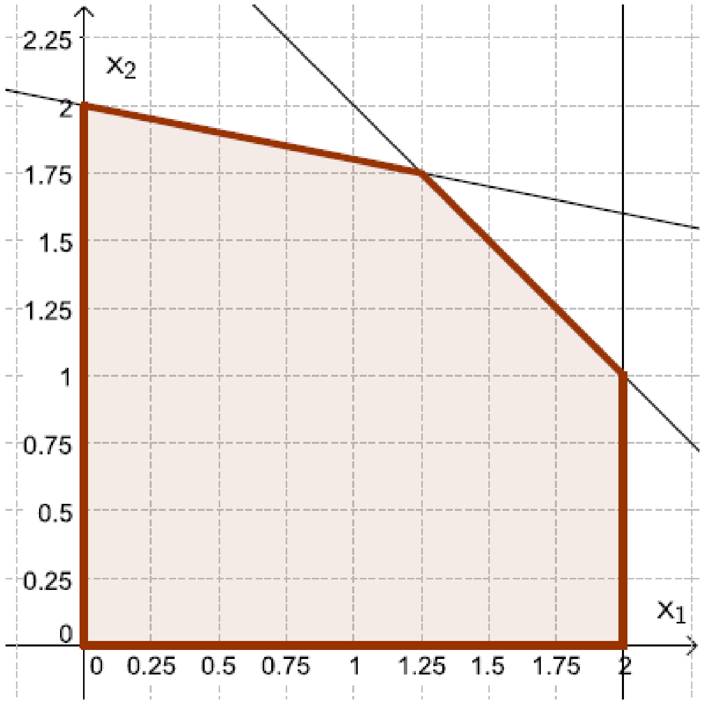
$$\max z = 100x_1 + 200x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 3 \\ x_1 + 5x_2 &\leq 10 \\ x_1 &\leq 2 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Notes: $\mathbb{Z}$ is the set of whole numbers and $x \in \mathbb{Z}$ reads "x is an integer"





Only the green dots are feasible solutions of the IP whereas the entire interior of the polyhedron is feasible for the LP

- **More constraints => smaller feasible region => less degrees of freedom**

- Given a set of linear constraints, adding integer restrictions reduces the feasible region

- For a **maximization problem**: the optimal objective value of the IP is **less** than or equal to that of the LP

- For a **minimization problem**: the optimal objective value of the IP is **greater** than or equal to that of the LP

- There are also problems in which the constraints and/or objective function are nonlinear (for example $x^2 + 3yz^2$), which are called MINLP (Mixed-Integer Non-Linear Program).

- Allowing non-linearity certainly makes it easier to model some problems (that are naturally non-linear), but these problems are even harder to solve than MIP — though there are good algorithms for specific cases such as *Quadratic Programming* (QP) in which the constraints are linear, but the objective function is quadratic.

- We’ll just talk about IP or MIP.

Example: 3-Queens Problem

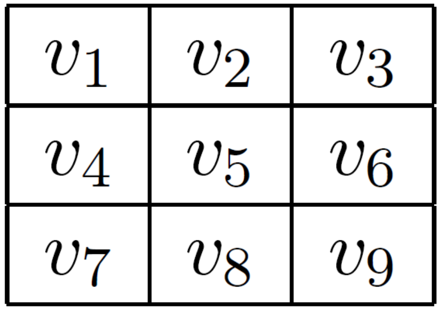
- This is a completely useless problem with no solution, but it’s a nice simple IP example

- Suppose we have a 3 × 3 chess board and 3 queens: can we place the queens on the board so that no 2 attack each other?
    - a queen attacks every square on the same row, column or diagonal

Example: 3-Queens Problem

- Here's one approach. For each of the 9 squares

$i = 1, \dots, 9$ on the board let's have a binary variable v_i which is 0 if there's no queen on it, and 1 if there is a queen on it. Let's arrange them like this:



v_1	v_2	v_3
v_4	v_5	v_6
v_7	v_8	v_9

There's no objective function: we just want to know if there are solutions or not.

Example: 3-Queens Problem

- Constraints:
    - Exactly 3 squares have a queen:

$$\sum_{i=1}^{9} v_i = 3$$

that is:

$$v_1 + v_2 + v_3 + v_4 + v_5 + v_6 + v_7 + v_8 + v_9 = 3$$

Example: 3-Queens Problem

- Constraints:
  - Each row has at most 1 queen:

$$v_1 + v_2 + v_3 \leq 1$$
$$v_4 + v_5 + v_6 \leq 1$$
$$v_7 + v_8 + v_9 \leq 1$$

v_1	v_2	v_3
v_4	v_5	v_6
v_7	v_8	v_9

  - Similarly, each column has at most 1 queen:

$$v_1 + v_4 + v_7 \leq 1$$
$$v_2 + v_5 + v_8 \leq 1$$
$$v_3 + v_6 + v_9 \leq 1$$

Example: 3-Queens Problem

- Constraints:
    - And each diagonal has at most 1 queen:

v_1	v_2	v_3
v_4	v_5	v_6
v_7	v_8	v_9

$$\begin{aligned} v_2 + v_4 &\leq 1 \\ v_3 + v_5 + v_7 &\leq 1 \\ v_6 + v_8 &\leq 1 \\ v_2 + v_6 &\leq 1 \\ v_1 + v_5 + v_9 &\leq 1 \\ v_4 + v_8 &\leq 1 \end{aligned}$$

Example: 3-Queens Problem

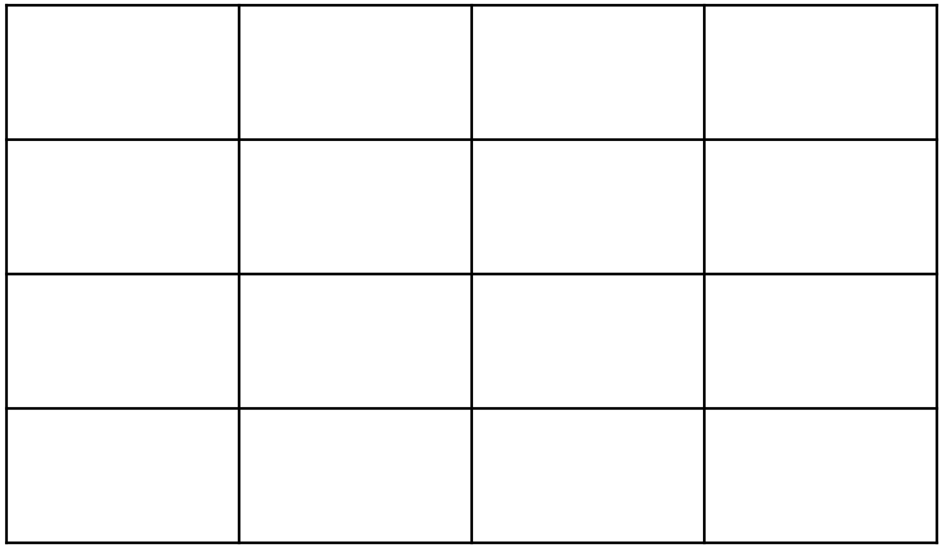
- Rows, columns and diagonals are the only ways that queens can attack each other, so we've completely modelled the problem.

- Actually, after a bit of thought we could use $=$ instead of $\leq$ in the row and column constraints: each queen must be in a different row and there are the same number of queens as rows, so each row contains *exactly* 1 queen, not just *at most*. Similarly for columns. This might make the IP easier to solve. (But the same isn't true of diagonals: there might be a diagonal with no queen on it — see below.)

- There's no solution to this problem, so the software should say something like "no solution" or "infeasible".

Example: 3-Queens Problem

- Is it solvable for a 4×4 board or bigger (and 4 or more queens)?



Example: 3-Queens Problem

- For a 4×4 board or bigger (and 4 or more queens) the problem is solvable, for example:

0	1	0	0
0	0	0	1
1	0	0	0
0	0	1	0

- You can check that no 2 queens attack each other vertically, horizontally or diagonally.

Warehouse location problem

- A company considers opening warehouses at some candidate locations in order to supply its existing stores.
- Each possible warehouse has the same maintenance cost, and a capacity designating the maximum number of stores that it can supply.
- Each store must be supplied by exactly one open warehouse.
- The supply cost to a store depends on the warehouse.
- The objective is to determine which warehouses to open, and which of these warehouses should supply the various stores, such that the sum of the maintenance and supply costs is minimized.

Warehouse location problem

- We can model this problem in a similar way to the LP examples, except that we now have integer variables.

- Define binary variables o_i

$$o_i = \begin{cases} 1; & \textit{if warehouse i is open} \\ 0\ ; & \textit{otherwise} \end{cases}$$

- and s_{ij}

$$s_{ij} = \begin{cases} 1; & \textit{if warehouse i supplies store j} \\ 0\ ; & \textit{otherwise} \end{cases}$$

- This is a typical use of binary variables to represent decisions.

Warehouse location problem

- Now the constraints.
    - We must ensure that each store is supplied by one warehouse:

$$\sum_i s_{ij} = 1$$

    - Each warehouse capacity c_i must be respected:

$$\sum_j s_{ij} \leq c_i$$

    - Of course, if a warehouse is supplied then it must be open:

$$s_{ij} \leq o_i$$

Warehouse location problem

- **Note.** We could collect the last set of constraints into a smaller set using a modelling trick:

$$\sum_{j=1}^{S} s_{ij} \leq S o_i$$

- where S is the number of stores. This does the same as the other constraints, but it needs a bit of thought to see why. We'll return to this.
- The objective is to minimise

$$Z = k \sum_i o_i + \sum_i \sum_j k_i s_{ij}$$

- where k_i is the cost of supplying store j from warehouse i, and k is the warehouse fixed maintenance cost.

Logical constraints

- As we saw in the example, when we're modelling decisions by binary variables there are some modelling tricks that we need to know.

- For example, we might want to ensure that if one decision is made then at least one of two others is; or if several other decisions are made then so is another.

- Before we see more examples, here's a collection of these tricks for some set of binary variables $y_1 \ldots y_n$:

Logical constraints

- The decisions are mutually exclusive:

$$\sum_i y_i \leq 1$$

- At most k of the decisions can be made:

$$\sum_i y_i \leq k$$

- At least k of the decisions can be made:

$$\sum_i y_i \geq k$$

- Exactly k of the decisions can be made:

$$\sum_i y_i = k$$

Logical constraints

- Now a less obvious one. If *any* of the y decisions are made, then so is another decision w:

$$\sum_{i=1}^{n} y_i \leq nw$$

We multiply by n so that the constraint can be satisfied if all the $y_i = 1$.

- If *all* the y decisions are made then so is w:

$$\sum_{i=1}^{n} y_i \leq n - 1 + w$$

- If *at least* k of the y decisions are made, then so is w:

$$\sum_{i=1}^{n} y_i \leq (k-1) + [n - (k-1)]w$$

Logical constraints

- For a MIP, consider the effect of this constraint:

$$x \leq uy$$

where x is real (continuous), y is binary, and u is an upper bound on x.

- If $y = 0$ (decision y is not made) then x must be 0, otherwise can take any value between 0 and u.

- Another way to view this is:

  if $x > 0$ then $y = 1$, so we make decision y if x is positive.

Logical constraints

- We can represent the reverse of a decision y by $1 - y$.

- For example, if we take decision y_1 and we do not take decision y_2 then we take decision w:

$$y_1 + (1 - y_2) \leq 1 + w$$

- which is the same as:

$$y_1 - y_2 - w \leq 0$$

Logical constraints

- These are just a collection of tricks to use when modelling problems.

- If they don't look obvious yet, don't worry: it takes practice.

- What is the difference between $x + y \leq 2w$ and $x + y \leq 1 + w$.

- The first means "if x or y or both is true then so is w" while the second means "if x and y are both true then so is w". (These are special cases of the above rules.)

Example using logical constraints

- Suppose we want to solve the following puzzle:

- If it’s raining then either I take an umbrella or wear a jacket, but not both. If I carry an umbrella then I can’t enter a cafe. I want to enter a cafe. It’s raining. What do I do?

- We’ll use these 0/1 variables:
  - r is 1 it’s raining and 0 otherwise
  - u is 1 if I take an umbrella and 0 otherwise,
  - j is 1 if I wear a jacket and 0 otherwise,
  - c is 1 if I want to enter a cafe and 0 otherwise.

Example using logical constraints

- There's no objective here, just these constraints:
  - If it rains then I carry an umbrella or wear a jacket: $r \leq u + j$.

    If $r = 1$ then the right hand side must be at least 1, which forces at least one of u and j to be 1.

  - I can't carry an umbrella and wear a jacket: $u + j \leq 1$.

    Both u and j can't be 1 because then the left hand side would be 2.

  - If I carry an umbrella then I can't enter a cafe: $u \leq (1 - c)$ or $u + c \leq 1$
  - I want to enter a cafe: $c = 1$
  - It's raining: $r = 1$

Example using logical constraints

- A solver will give the answer $c = r = j = 1$ and $u = 0$:
    - it’s raining, I want to enter a cafe, I wear a jacket, and I don’t carry an umbrella.

Another Example (big-M)

- Suppose that we have a real variable r and a binary variable b. If $b = 1$ then we want $r \leq 100$, otherwise if $b = 0$ then we want $r \geq 100$.

- We can express this by two constraints:

$$r \leq 100 + M(1-b) \qquad r \geq 100 - Mb$$

where M is some large number (certainly greater than 100). We can check that this does what we want it to. If $b = 0$ then these constraints reduce to:

$$r \leq 100 + M \qquad r \geq 100$$

- Because M is large the first constraint has no effect, but the second one does what we wanted.

Another Example (big-M)

- On the other hand, if $b = 1$ then the constraints reduce to

$$r \leq 100 \qquad r \geq 100 - M$$

- The first one does what we wanted, while the second one has no effect because M is large. The purpose of the M was to "turn off" a constraint when b has some value.

- Now suppose we just want to force r to be either ≤ 100 or ≥ 200 but not in between, but there's no variable b in the problem? We can just create one and use the same trick: b isn't use anywhere else, but it must be 0 or 1 so it forces the condition on r.

- These are useful tricks, but unfortunately these "big-M" models of problems have a bad effect on the algorithm usually used to solve IPs (branch-and-bound, more about this later).

- Here’s another modelling technique. Suppose we have a variable x that we want to take discrete values that aren’t consecutive integers, or maybe not even integers, for example x takes values from the set $\{1, 3, 4.5, 16\}$.

- To model this we can introduce new binary variables. In the example there are 4 discrete values for x so we define y_1, y_2, y_3, y_4.

- Then we use constraints

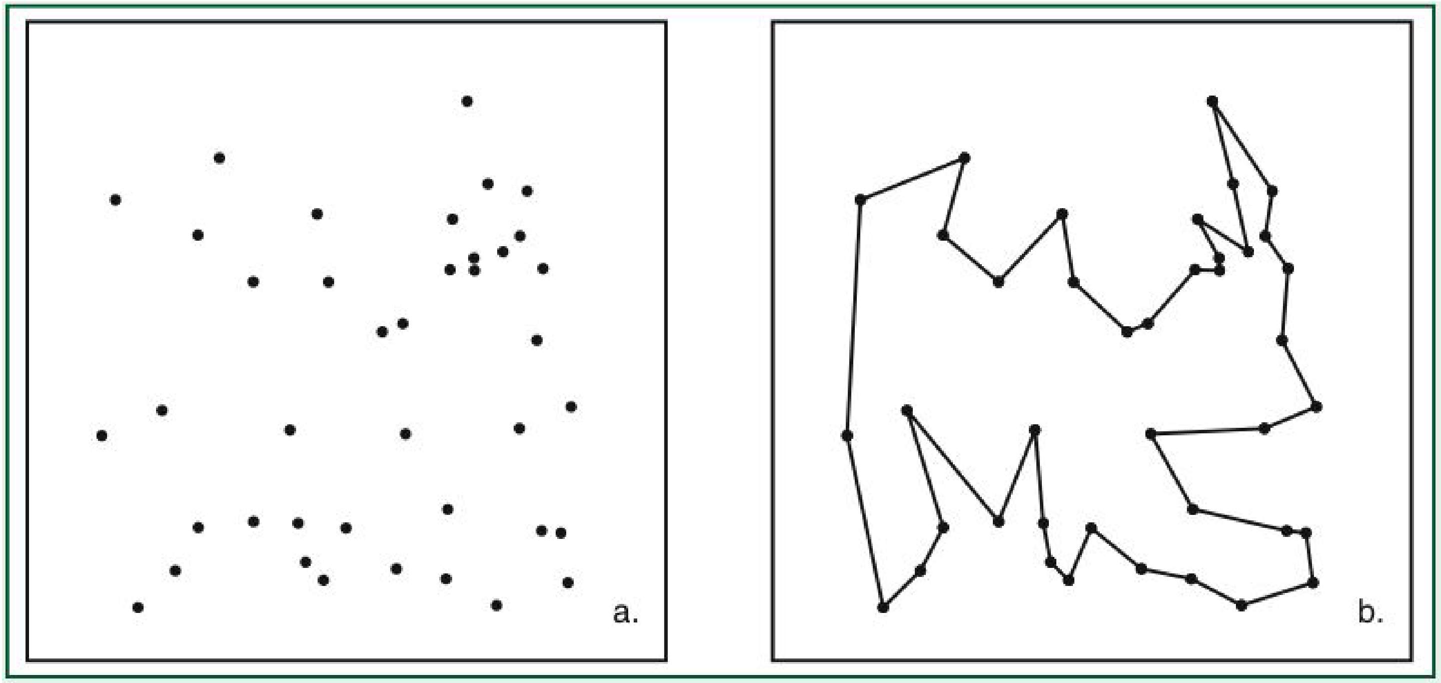
$$x = y_1 + 3y_2 + 4.5y_3 + 16y_4$$
$$y_1 + y_2 + y_3 + y_4 = 1$$

The Travelling Salesman Problem (TSP)

- The TSP is a very well-known problem that has been attacked with all sorts of methods.
- We have a set of cities, or (more generally) nodes in a graph. Between each pair of nodes is an edge with a distance attached to it.
- In some cases, the distance between city A and city B may be different than the distance from city B to city A: this is the *asymmetric* TSP.
- The problem is to plan a tour that visits each city exactly once, finishing up back where we started, while minimising the total distance travelled. (Not all the edges will be used in any tour.)

The Travelling Salesman Problem (TSP)

What is the **shortest route** visiting all N cities exactly once?



There are $(N-1)!$ **different routes**, can we enumerate them?

The Travelling Salesman Problem (TSP)

N	$N! = 1 \times 2 \times \ldots \times N$	Time required for enumeration [1]
5	120	1×10^{-8} seconds
10	$3.6 \times 10^{+6}$	3×10^{-4} seconds
15	$1.3 \times 10^{+12}$	109 seconds
20	$2.4 \times 10^{+18}$	6.4 years
50	$3.0 \times 10^{+64}$	$8 \times 10^{+46}$ years
100	$9.3 \times 10^{+157}$	$2 \times 10^{+140}$ years

Today, the TSP has been solved to (global) optimality on an instance with 24,978 cities in the equivalent of 84.4 years (parallel computing was used to speed-up the computations)!

The Travelling Salesman Problem (TSP)

National College of Ireland

- We represent the distances by a distance matrix, for example:

	1	2	3	4
1	—	27	43	16
2	7	—	16	1
3	20	13	—	35
4	21	16	25	—

- The symmetric TSP is also studied, in which distances are the same each way.
- In real life asymmetry might occur because the roads are of different quality, or the trips might be flights and some flights are cheaper than others.
- (In a symmetric TSP, the matrix is symmetric about the diagonal going from top left to bottom right.)
- When the matrix is asymmetric we have an *asymmetric TSP* (ATSP).

The travelling salesman problem (TSP)

- The TSP is related to the problem of finding a *Hamiltonian cycle* (or *circuit*) in a graph: some nodes may not be connected, or may be connected from A to B but not vice-versa.

- It is also related to the problem of finding a *Hamiltonian path* which visits each node once but doesn’t return to the first one, so the first and last need not be connected. The edges may be 1-way or 2-way.

- The TSP is the problem of finding a Hamiltonian cycle on a complete graph (where every pair of nodes is connected) while minimising the sum of the weights (distances) of the edges used in the cycle.

The travelling salesman problem (TSP)

- The ATSP is easily modelled as an IP. Here's a standard model found in 1954 by Dantzig, Fulkerson and Johnson.

- For each edge between nodes i, j (or trip between cities i, j) define a binary variable x_{ij} . This is 1 if the trip is made, otherwise 0.

- We do not define variables x_{ii} because a trip is never made from a city to itself. If the distance matrix is C_{ij} then we must minimise

$$Z = \sum_i \sum_j C_{ij} x_{ij}$$

- subject to the following constraints:

The travelling salesman problem (TSP)

- Each node has exactly one successor (from every city we must travel to exactly one other):

$$\sum_j x_{ij} = 1$$

for each i.

- Each node has exactly one predecessor (to be where we are now, we must have arrived from exactly one city):

$$\sum_i x_{ij} = 1$$

for each j.

The travelling salesman problem (TSP)

- The constraint that each node (city) must be visited exactly once is not easy to represent directly.
- The standard way is to prevent *subtours*, that is visits to *some* of the nodes that form a cycle. For every possible subtour, represented by a set S of edges, we add a constraint

$$\sum_{(i,j)\in S} x_{ij} \leq |S| - 1$$

- That is, not all the edges in the subtour can be used.

- Actually, we only need to consider subtours of size up to $n/2$ (where n is the number of cities), because if there is a larger subtour then the rest of the nodes must be contained in a smaller one (this might need some thought).

The travelling salesman problem (TSP)

- The drawback with this IP is that there are a lot of subtour elimination constraints: $2^{n-1} - n - 1$ of them. So this is not practical except for small TSPs. At least not with the usual solution methods.

- But there is in fact a way of solving large TSPs with this model: instead of putting all the subtour elimination constraints into a file before solving the IP, they are generated as needed during search.

- This is just one model; there are a few different alternative IPs for the TSP.

- Some of these are easier to solve than others.

An application of the TSP

- The TSP may be used to model other problems that at first seem unrelated.
- For example we might need to drill a number of holes in a sheet of metal as quickly as possible.
- This is a symmetric TSP: we must visit each location on the sheet, stopping at each, and minimise the total distance travelled.
- If you notice that your problem reduces to the TSP then you know how it should be solved, and you can use existing software.

TSP Application: Scheduling aircraft for take-off

- There must be a gap between each take-off, because of turbulence and vortices from the engines.
- The gap depends on the types of plane: a large plane can take off sooner than a small one, because it’s less sensitive to air currents.
- But a large plane creates more disturbance than a small one for the following plane.
- We want a schedule that allows every aircraft to take off, while minimising the total time.
- Such a schedule is a list of all the planes with no repeats, which is just like a list of cities in a tour.

TSP Application: Scheduling aircraft for take-off

- Travelling from the current city A to one of the next ones B, C, D etc. is equivalent to allowing planes B, C, D etc to take off after plane A

- The distance from A to B is equivalent to the time that must be left after A before B can take off.

- This problem has natural asymmetry: a large plane might have to wait for 1 minute after a small one, but a small one might wait 3 minutes after a large one.

- So the “distance” from A to B may be very different from the “distance” from B to A.

TSP Application: Scheduling aircraft for take-off

National College of Ireland

- For example, suppose we have 3 types of plane: Jumbo, Airbus and Cessna in decreasing order of both size and engine power (there is an Airbus bigger than a Jumbo, but let's ignore this):

first→ second↓	Jumbo	Airbus	Cessna
Jumbo	3	2	1
Airbus	4	3	1
Cessna	8	7	2

- Now suppose we have 2 Jumbos, an Airbus and a Cessna to schedule. Let's call these planes J_1, J_2, A,C.

- The "distances" are:

first→ second↓	J_1	J_2	A	C
J_1	—	3	2	1
J_2	3	—	2	1
A	4	4	—	1
C	8	8	7	—

TSP Application: Scheduling aircraft for take-off

- (We didn’t use the Airbus-Airbus or Cessna-Cessna “distances” because there’s only one plane of each type to schedule.)