Formulating ILP Problems: The TSP

Deepak Ajwani

School of Computer Science

Outline

- Integer Linear Program

  - Modelling problems as ILP
    - Logical Constraints
    - N-colouring Problem
    - Sudoku
    - TSP

  - Why is ILP hard?

Recap

- We've been looking at *linear programming* problems.

Minimise $c_1x_1 + c_2x_2 + \ldots + c_nx_n$

subject to constraints:

$$\begin{array}{cccc}
a_{11}x_1+ & a_{12}x_2+ & \ldots+ & a_{1n}x_n = b_1 \\
a_{21}x_1+ & a_{22}x_2+ & \ldots+ & a_{2n}x_n = b_2 \\
\vdots + & \vdots + & \ldots+ & \vdots = \vdots \\
a_{m1}x_1+ & a_{m2}x_2+ & \ldots+ & a_{mn}x_n = b_m
\end{array}$$

$x_j \geq 0$ *for all* j

- The *Simplex method* finds an optimal solution (if one exists) and is efficient in practice.

Sometimes we want integer solutions!

- For many problems, non-integer solutions don’t make sense.
- Consider the following linear programming problem:

  Suppose a company makes tables and chairs made from either pine or oak. An oak table requires 17 units of oak wood and an oak chair requires 5 units of oak to make. A pine table and chair require respectively 30 and 1 unit of pine. The company has available 150 units of oak and 210 units of pine for this week. If the profit on every table sold is Euro 40 and that of every chair is Euro 15, how many tables and chairs of each type should the company manufacture.

Sometimes we want integer solutions!

- What can we do if Simplex tells us to make 1.5 chairs?! The company only makes whole chairs!
- If we make the extra requirement on an LP that the solution must be integer-only then we speak of an **integer linear programming (ILP)** problem.

Example 1: Knapsack problem

A pure ILP problem with a single linear constraint is called a *knapsack problem.*

$$Maximise \quad c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

Subject to constraints

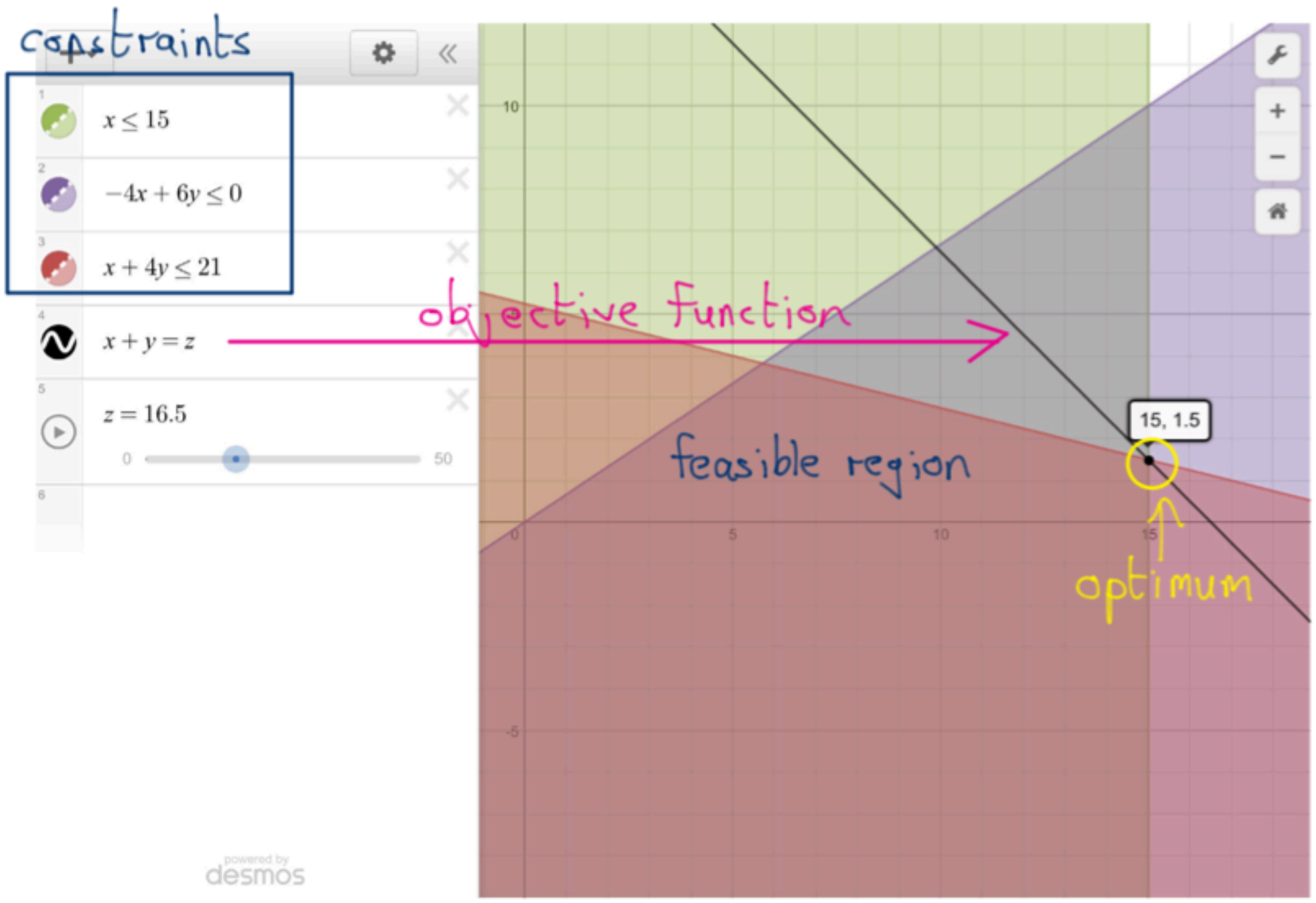
$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$$

$x_i \geq 0$ and x_i is an integer for $i = 1,...,n$

(Imagine a hiker trying to fill her knapsack to a maximum value. Each item has some given value and weight. Constraint comes from overall weight limitation.)

Optimal solutions might be non-integer

- In general, optimal solutions to an LP might **not** be integers (i.e., whole numbers), even when c_j, a_{ij}, b_i **are.**



A classification of ILPs

- *Binary integer programs:* decision variables can take only values 0,1.
    - The case of binary decision variables is quite common. Common for modelling 'yes/no' decisions.
- *Pure integer programs:* all decision variables are required to be integer-only.
- *Mixed integer programs:* contain a mixture of integer valued and non-integer valued decision variables.

Modelling logical constraints

- In the case of binary variables, we may want to model constraints such as:
  - *"If no depot is sited here, then it will not be possible to supply any of the customers from the depot"*
  - *"If this station is closed, then both branch lines terminating at the station must also be closed."*
  - *"No more than five of the ingredients in this class may be included in the blend at any one time."*
- Not so easy to do via ordinary LP. In ILP we can do this via *logical constraints.*

Modelling logical constraints

- Let X_1 and X_2 be 2 propositions (i.e., statements with yes/no answers) and let x_1, x_2 be corresponding binary decision variables*. (i.e.,* x_i equals 1 if X_i is answered 'yes', equals 0 otherwise)
  - 'X_1 or X_2*'* is equivalent to $x_1 + x_2 \geq 1$
  - 'X_1 and X_2*'* is equivalent to $x_1 = 1,\ x_2 = 1$
  - 'not X_1*'* is equivalent to ?
  - 'If X_1 then X_2*'* is equivalent to ?
  - 'X_1 if and only if X_2' is equivalent to ?

Modelling logical constraints

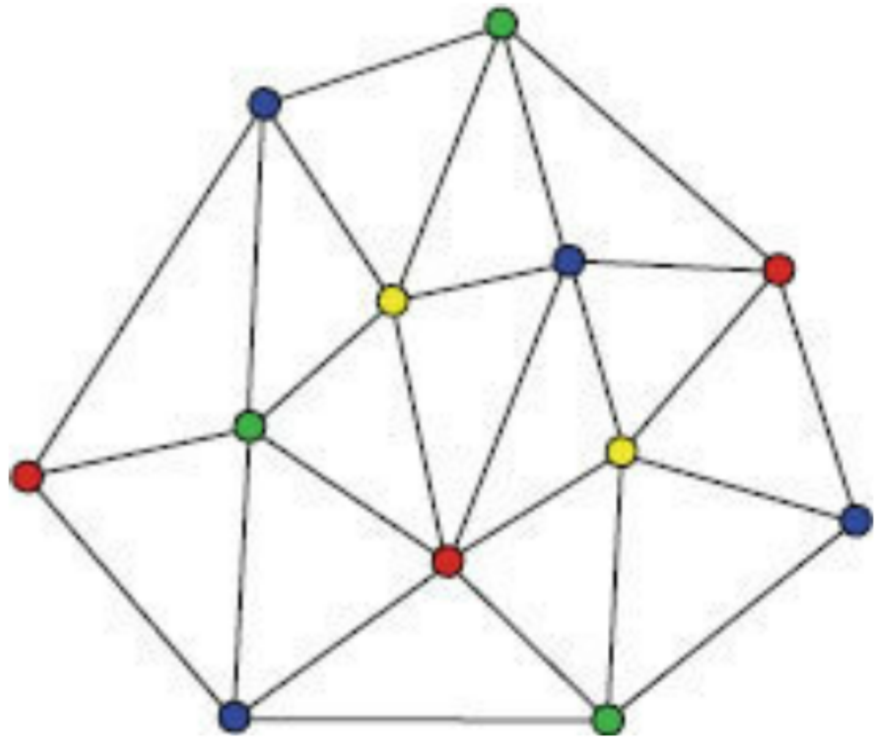
- Let X_1 and X_2 be 2 propositions (i.e., statements with yes/no answers) and let x_1, x_2 be corresponding binary decision variables. *(i.e., x_i* equals 1 if X_i is answered 'yes', equals 0 otherwise)
  - 'X_1 or X_2' is equivalent to $x_1 + x_2 \geq 1$
  - 'X_1 and X_2' is equivalent to $x_1 = 1,\ x_2 = 1$
  - 'not X_1' is equivalent to $x_1 = 0$ (or $1 - x_1 = 1$)
  - 'If X_1 then X_2' is equivalent to $x_1 \leq x_2$, (or $x_1 - x_2 \leq 0$)
  - 'X_1 if and only if X_2' is equivalent to $x_1 = x_2$ or ($x_1 - x_2 = 0$)

Example 2: Colouring problems

The *n*-colouring problem

Given a graph, can we colour the nodes using *n* colours in such a way that no 2 adjacent nodes have the same colour?

Example of a valid 4-colouring:



Is there a valid 3-colouring for this graph?

The 3-colouring problem as an ILP

Assume 3 colours: red, green, blue

What are the decision variables? And how many are there if the graph has *m* nodes?

What are the constraints?

The 3-colouring problem as an ILP

<u>Decision variables</u>

For each node *i* there are 3 variables $x_{i,red}, x_{i,blue}$ and $x_{i,green}$. with $x_{i,red} = 1$ if node *i* is coloured red, $x_{i,red} = 0$ if node i is not coloured red (and similarly for blue, green.). There will be 3*m* decision variables.

The 3-colouring problem as an ILP

Decision variables

For each node *i* there are 3 variables $x_{i,red}$, $x_{i,blue}$ and $x_{i,green}$.
with $x_{i,red} = 1$ if node *i* is coloured red, $x_{i,red} = 0$ if node i is not coloured red (and similarly for blue, green.). There will be 3*m* decision variables.

Constraints

For each i : $x_{i,red} + x_{i,blue} + x_{i,green}$? 1

The 3-colouring problem as an ILP

Decision variables

For each node *i* there are 3 variables $x_{i,red}, x_{i,blue}$ and $x_{i,green}$. with $x_{i,red} = 1$ if node *i* is coloured red, $x_{i,red} = 0$ if node i is not coloured red (and similarly for blue, green.). There will be 3*m* decision variables.

Constraints

For each i : $x_{i,red} + x_{i,blue} + x_{i,green} = 1$

The 3-colouring problem as an ILP

Decision variables

For each node *i* there are 3 variables $x_{i,red}$, $x_{i,blue}$ and $x_{i,green}$.

with $x_{i,red} = 1$ if node *i* is coloured red, $x_{i,red} = 0$ if node i is not coloured red (and similarly for blue, green.). There will be 3*m* decision variables.

Constraints

For each i : $x_{i,red} + x_{i,blue} + x_{i,green} = 1$

For each edge (i, j) in the graph:

$$x_{i,red} + x_{j,red} \ ? \ 1$$
$$x_{i,blue} + x_{j,blue} \ ? \ 1$$
$$x_{i,green} + x_{j,green} \ ? \ 1$$

The 3-colouring problem as an ILP

Decision variables

For each node *i* there are 3 variables $x_{i,red}, x_{i,blue}$ and $x_{i,green}$. with $x_{i,red} = 1$ if node *i* is coloured red, $x_{i,red} = 0$ if node i is not coloured red (and similarly for blue, green.). There will be 3*m* decision variables.

Constraints

For each i : $x_{i,red} + x_{i,blue} + x_{i,green} = 1$

For each edge (i, j) in the graph:

$$x_{i,red} + x_{j,red} \leq 1$$
$$x_{i,blue} + x_{j,blue} \leq 1$$
$$x_{i,green} + x_{j,green} \leq 1$$

The 3-colouring problem as an ILP

Decision variables

For each node *i* there are 3 variables $x_{i,red}, x_{i,blue}$ and $x_{i,green}$. with $x_{i,red} = 1$ if node *i* is coloured red, $x_{i,red} = 0$ if node i is not coloured red (and similarly for blue, green.). There will be 3*m* decision variables.

Constraints

For each i : $x_{i,red} + x_{i,blue} + x_{i,green} = 1$

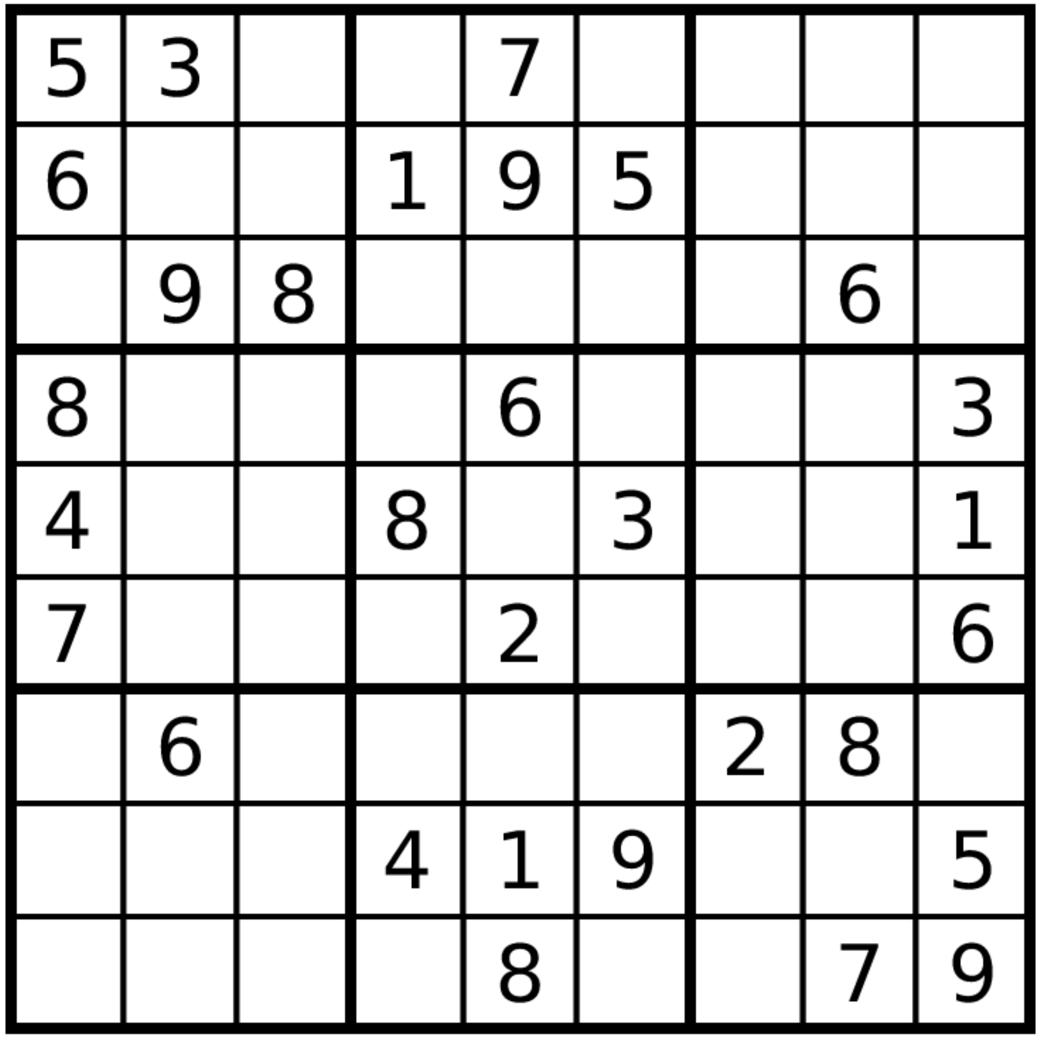
For each edge (i, j) in the graph:

$$\begin{gathered} x_{i,red} + x_{j,red} \leq 1 \\ x_{i,blue} + x_{j,blue} \leq 1 \\ x_{i,green} + x_{j,green} \leq 1 \end{gathered}$$

$x_{i,red}, x_{i,blue}$ *and* $x_{i,green}$ are binary for all i

Example 3: Sudoku

https://jump.dev/JuMP.jl/stable/tutorials/Mixed-integer%20linear%20programs/sudoku/



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

Sudoku

https://jump.dev/JuMP.jl/stable/tutorials/Mixed-integer%20linear%20programs/sudoku/

Sudoku is a popular number puzzle. The goal is to place the digits 1,...,9 on a nine-by-nine grid, with some of the digits already filled in. Your solution must satisfy the following rules:

- The numbers 1 to 9 must appear in each 3x3 square
- The numbers 1 to 9 must appear in each row
- The numbers 1 to 9 must appear in each column

Solving a Sudoku isn't an optimisation problem with an objective; its actually a *feasibility* problem: we wish to find a feasible solution that satisfies these rules. You can think of it as an optimisation problem with an objective of 0.

Sudoku

https://jump.dev/JuMP.jl/stable/tutorials/Mixed-integer%20linear%20programs/sudoku/

Binary variable x_{ijk} represents that the cell corresponding to row i and column j contains the value k

```
@variable(sudoku, x[i = 1:9, j = 1:9, k = 1:9], Bin)
```

Now we can begin to add our constraints. We will actually start with something obvious to us as humans: that there can be only one number per cell

```
for i in 1:9  ## For each row
    for j in 1:9  ## and each column
        # Sum across all the possible digits. One and only one of the digits
        # can be in this cell, so the sum must be equal to one.
        @constraint(sudoku, sum(x[i, j, k] for k in 1:9) == 1)
    end
end
```

Sudoku

https://jump.dev/JuMP.jl/stable/tutorials/Mixed-integer%20linear%20programs/sudoku/

Next, we add the constraint that each value can appear exactly once in each row and each column

```
for ind in 1:9  ## Each row, OR each column
    for k in 1:9  ## Each digit
        # Sum across columns (j) - row constraint
        @constraint(sudoku, sum(x[ind, j, k] for j in 1:9) == 1)
        # Sum across rows (i) - column constraint
        @constraint(sudoku, sum(x[i, ind, k] for i in 1:9) == 1)
    end
end
```

Sudoku

https://jump.dev/JuMP.jl/stable/tutorials/Mixed-integer%20linear%20programs/sudoku/

Finally, we enforce the constraint that each digit appears once in each of the nine 3x3 grids. We index over the top-left corners of each 3x3 square with for loops, then sum over the squares.

```
for i in 1:3:7
    for j in 1:3:7
        for k in 1:9
            # i is the top left row, j is the top left column.
            # We'll sum from i to i+2, e.g. i=4, r=4, 5, 6.
            @constraint(
                sudoku,
                sum(x[r, c, k] for r in i:(i+2), c in j:(j+2)) == 1
            )
        end
    end
end
```

Sudoku

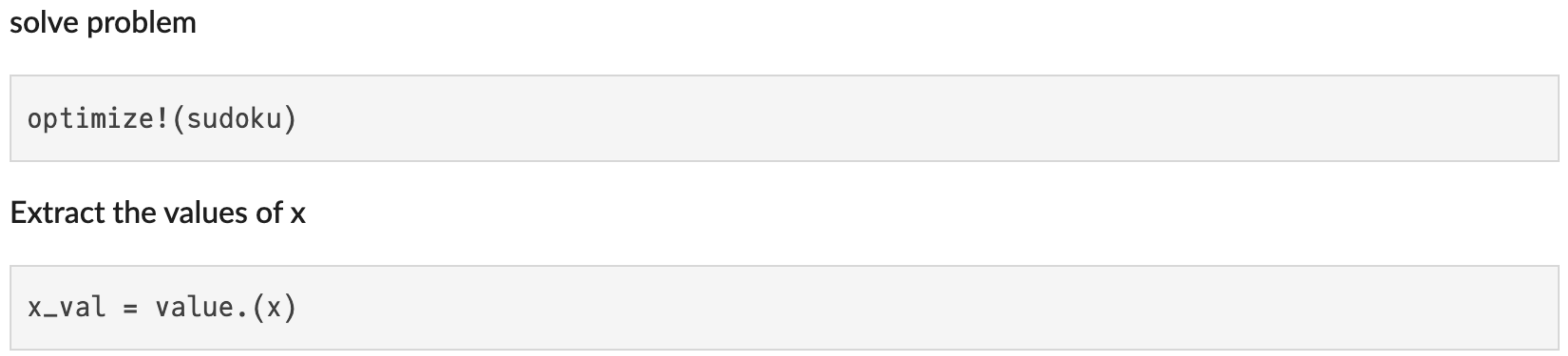
https://jump.dev/JuMP.jl/stable/tutorials/Mixed-integer%20linear%20programs/sudoku/

The final step is to add the initial solution as a set of constraints. We will put a 0 if there is no digit in that location.

```
init_sol = [
    5 3 0 0 7 0 0 0 0
    6 0 0 1 9 5 0 0 0
    0 9 8 0 0 0 0 6 0
    8 0 0 0 6 0 0 0 3
    4 0 0 8 0 3 0 0 1
    7 0 0 0 2 0 0 0 6
    0 6 0 0 0 0 2 8 0
    0 0 0 4 1 9 0 0 5
    0 0 0 0 8 0 0 7 9
]
for i in 1:9
    for j in 1:9
        # If the space isn't empty
        if init_sol[i, j] != 0
            # Then the corresponding variable for that digit and location must
            # be 1.
            fix(x[i, j, init_sol[i, j]], 1; force = true)
        end
    end
end
```

Sudoku

https://jump.dev/JuMP.jl/stable/tutorials/Mixed-integer%20linear%20programs/sudoku/



solve problem

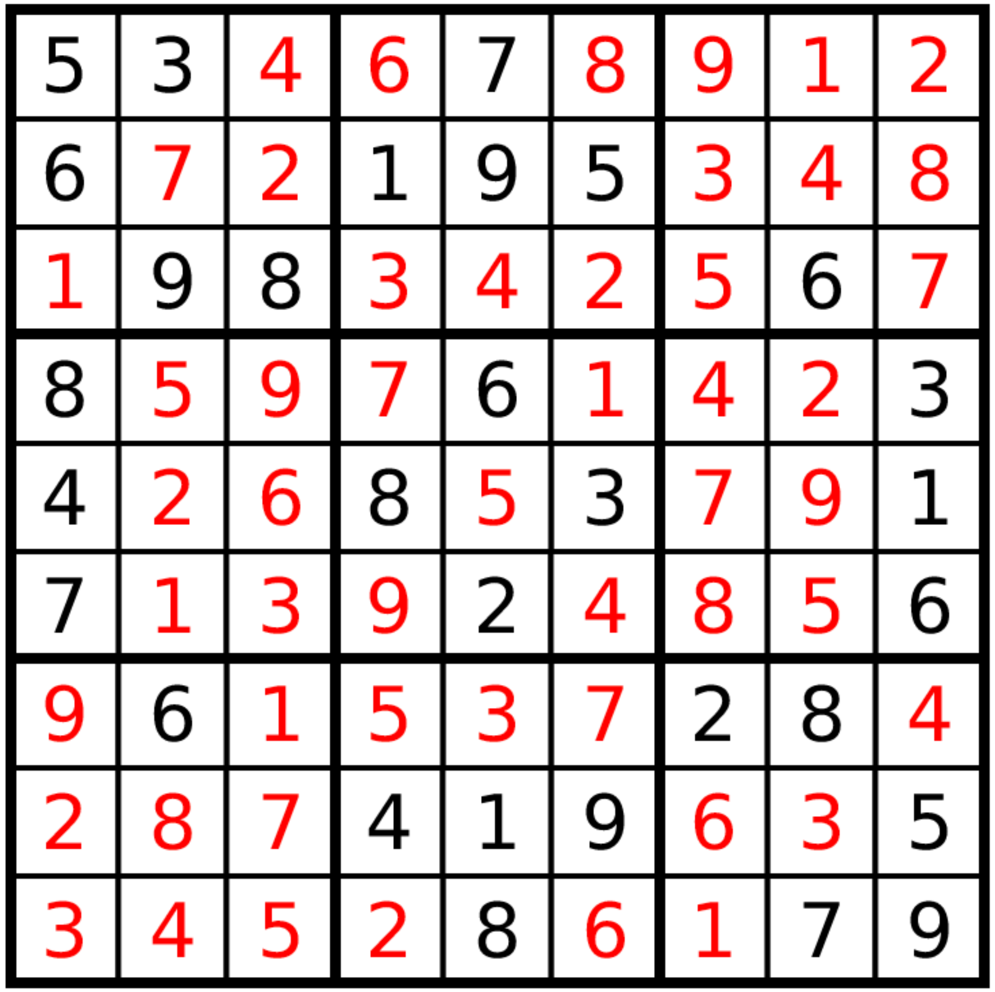
```
optimize!(sudoku)
```

Extract the values of x

```
x_val = value.(x)
```

Sudoku

https://jump.dev/JuMP.jl/stable/tutorials/Mixed-integer%20linear%20programs/sudoku/



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

Any Real-world Example?

- Any mixed integer linear programming problem you encountered in your internship? Or in day-to-day life?
- Actual project done for HSE by a consulting company:
  - Decide the allocation of Covid vaccines to different vaccination centres
    - Objective: Minimise wastage
    - Constraints: Limited vaccination supply
    - Constraints: Vaccination usage depends on the number of booths open, number of staff available at the centre, demand at the centre

Vaccination Distribution Problem

Some concrete numbers:

6 vaccination centres:

A fixed amount of vaccines to be distributed every week

Each of the 6 centres has different operational capacities and different number of booths

Each booth requires 1 vaccinator and 1 observer

For every 10 booth, 1 supervisor

For every 5 booths, 1 pharmacist to mix the solution

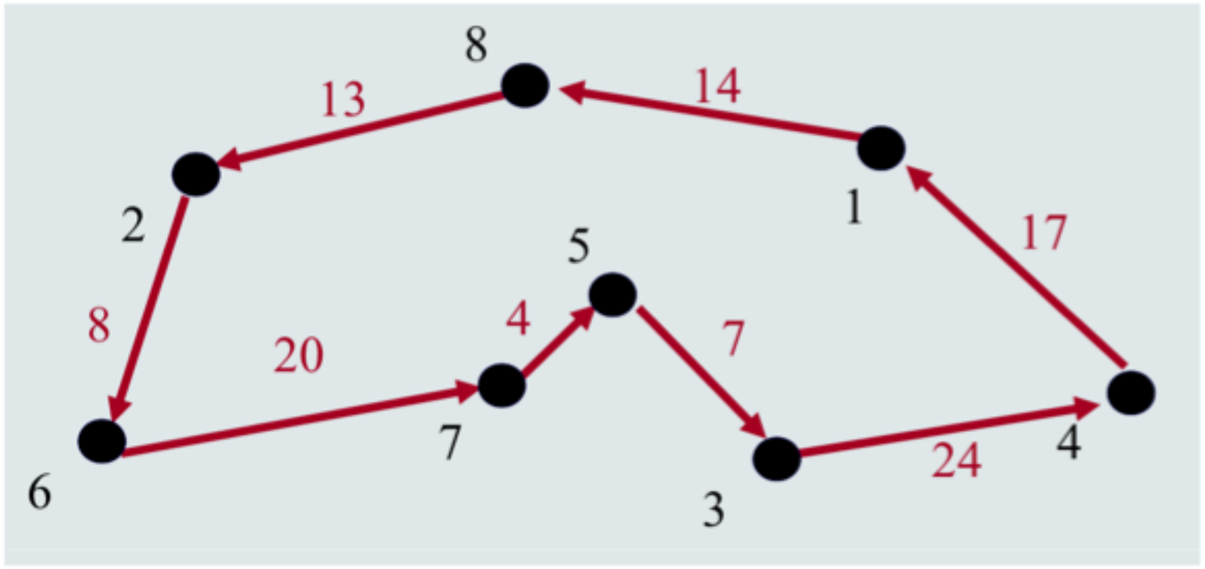
For every 3 booths, 1 nurse for emergency first-aid

Each centre could be open at most 9 hours a day

Different vaccination rates of different centres

A More Complex Example: TSP as ILP

Recall the *Travelling Salesperson Problem*: A salesperson wishes to find a route which visits each of *n* cities once and only once, at minimal distance.



Let’s formulate the TSP as an integer linear program.

A More Complex Example: TSP as ILP

<u>Decision variables</u>

For each pair of cities i and j, we need to decide whether to include the leg $i \to j$ in our tour, so introduce one decision variable x_{ij} for each pair of cities. (note there is a total of n^2 such variables.)

Each possible tour determines the value of x_{ij} as follows:

$$x_{ij} = \begin{cases} 1 & \text{if the leg } i \to j \text{ is included in the tour} \\ 0 & \text{otherwise} \end{cases}$$

(So x_{ij} is a binary integer)

A More Complex Example: TSP as ILP

<u>Objective function</u>

We are aiming to minimise the total distance travelled. Each edge $i \to j$ potentially contributes the distance between i and j to this total. So if we let c_{ij} denote this distance, we arrive at the following objective function (to be minimised):

$$\begin{array}{lcccccccc}
 & c_{11}x_{11} & + & c_{12}x_{12} & + & \dots & + & c_{1n}x_{1n} & + \\
+ & c_{21}x_{21} & + & c_{22}x_{22} & + & \dots & + & c_{2n}x_{2n} & + \\
 & \vdots & & \vdots & & & & \vdots & \\
+ & c_{n1}x_{n1} & + & c_{n2}x_{n2} & + & \dots & + & c_{nn}x_{nn} &
\end{array}$$

A More Complex Example: TSP as ILP

<u>Constraints</u>
Each city should be visited once and only once.

$$x_{1j} + x_{2j} + \ldots + x_{nj} = 1 \quad \textit{for each } j = 1,\ldots,n$$
$$x_{i1} + x_{i2} + \ldots + x_{in} = 1 \quad \textit{for each } i = 1,\ldots,n$$

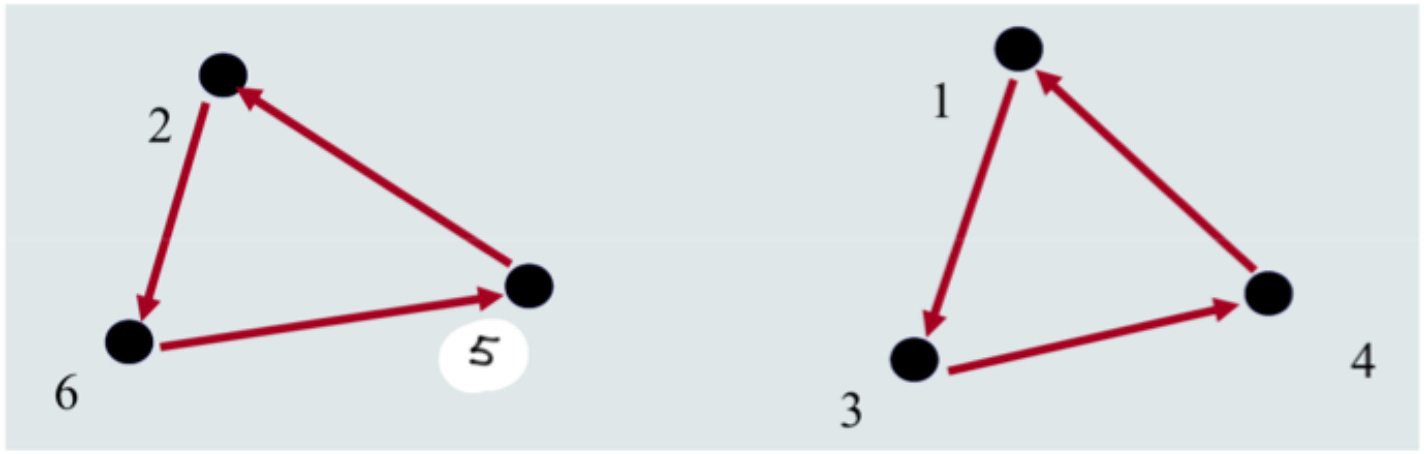
x_{ij} is binary for all i, j

$$\boldsymbol{x}_{ij}$$
$$\begin{pmatrix} & 1 & & & & \\ & & & & & 1 \\ & & & 1 & & \\ 1 & & & & & \\ & & & & 1 & \\ & & 1 & & & \end{pmatrix}$$

Are we done?

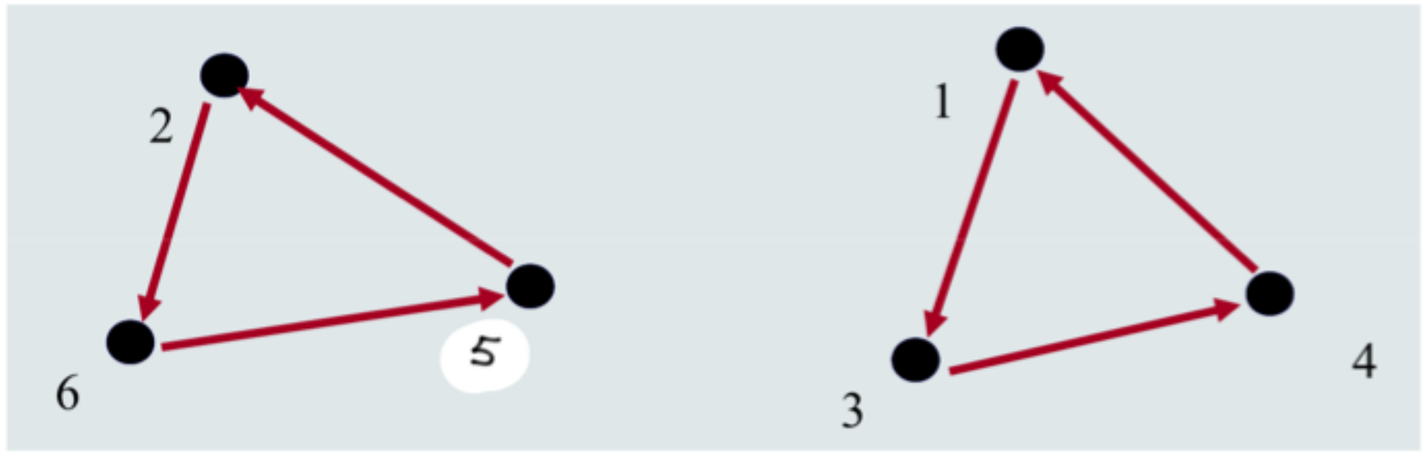
A More Complex Example: TSP as ILP

The following situation is consistent with these constraints (assuming *n* = 6)



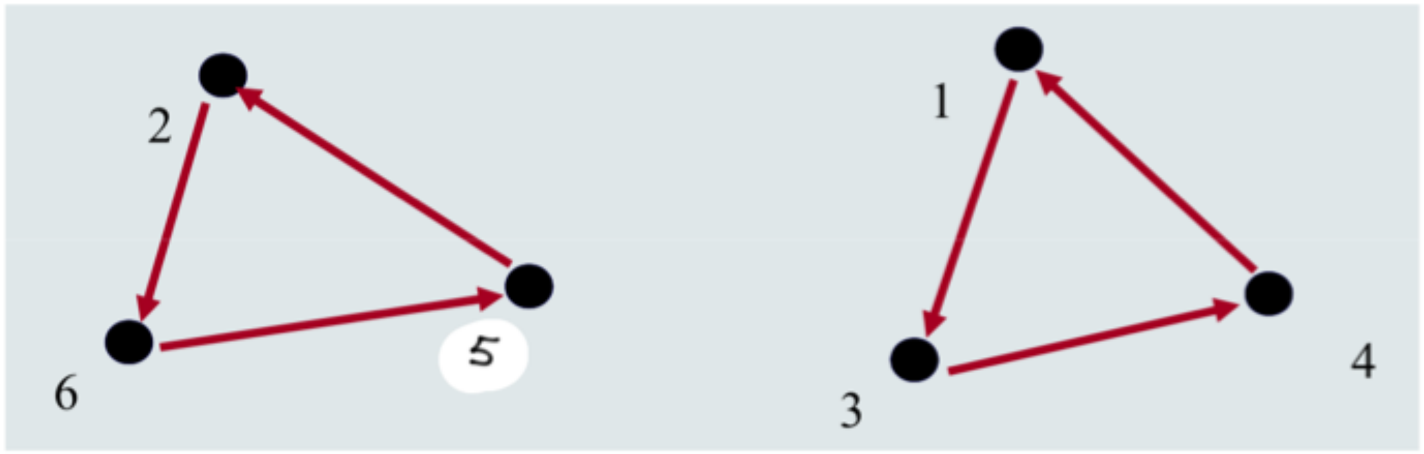
But this is not a tour!! We need more constraints to rule these out!

A More Complex Example: TSP as ILP



Want to ensure "*For any set of nodes S, there exists at least one edge from nodes in S to nodes outside S*"

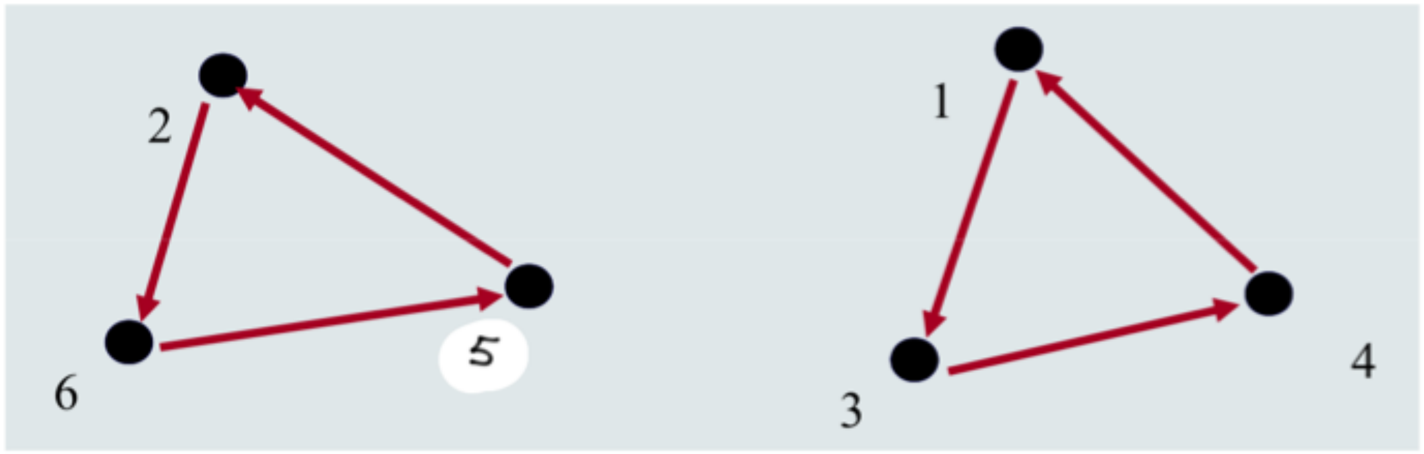
A More Complex Example: TSP as ILP



E.g., taking the case S = {2,5,6} above, we need to add the constraint:

$$x_{26} + x_{62} + x_{25} + x_{52} + x_{56} + x_{65} \leq 2$$

A More Complex Example: TSP as ILP



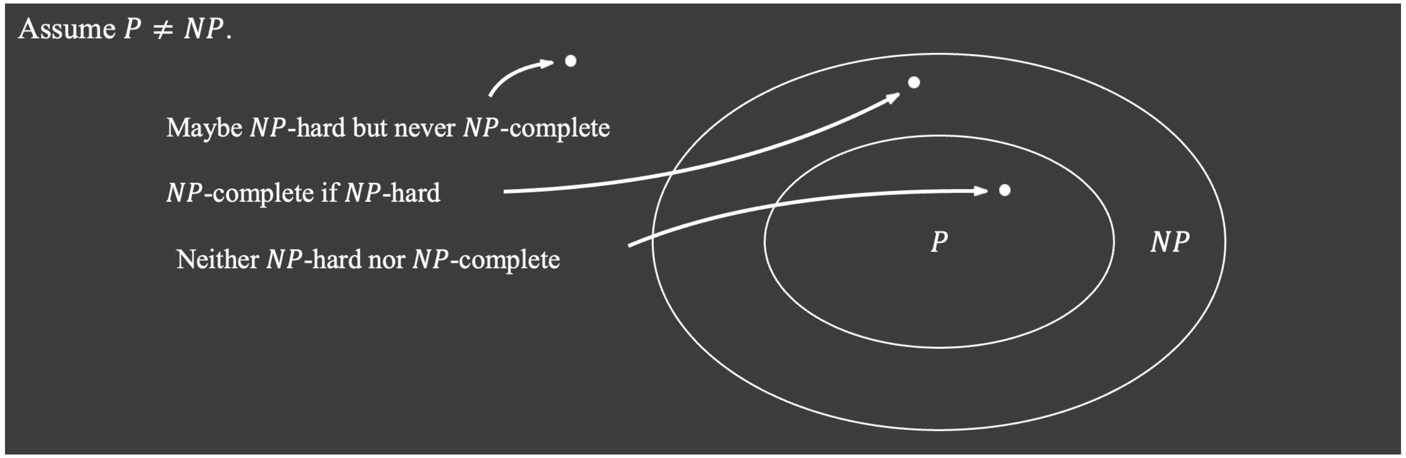
In general, we need to add a constraint:

$$\sum_{i,j \in M, i \neq j} x_{ij} \leq |M| - 1$$

for each $M \subseteq N$ such that $|M| \geq 2$ (where N is the set of all cities).

Complexity of ILP

- Integer linear programming is NP-complete
- NP: Set of decision problems that can be verified in polynomial time, given a proof string of polynomial length
- NP-hard: Set of decision problems that if solved in polynomial time will imply that all problems in class NP can be solved in polynomial time
- NP-complete: NP and NP-hard



Assume $P \neq NP$.
Maybe NP-hard but never NP-complete
NP-complete if NP-hard
Neither NP-hard nor NP-complete
P
NP

Complexity of ILP

- Integer linear programming is NP-complete
- Proving it is in NP is a bit arduous as we need to show that the number of bits needed for the entire computation is limited
- Prove NP-hard using reduction from 3-SAT to binary integer programs
    - Keep a variable for each 3-SAT variable and its negation
    - Sum of variable and its negation equals 1
    - Sum of variables in a clause have to be greater than equal to 1

Complexity of ILP

$F = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$

Variables: Binary variables $x_{i,true}$ and $x_{i,false}$ for all $i \in \{1,2,3,4\}$

Constraints:

$$x_{i,true} + x_{i,false} = 1 \text{ for all } i \in \{1,2,3,4\}$$

$$x_{i,true} + x_{2,true} + x_{3,true} \geq 1$$

…

ILP is harder than LP

- It is inherently more difficult to solve an ILP problem than the corresponding LP problem.
- The size of problem that can be solved successfully by ILP algorithms is an order of magnitude smaller that the size of LP problems that can be easily solved.

How to solve ILP problems?

- Most obvious thing to do is to try ignoring the integer constraints and run the simplex method on the resulting linear program.
- **If** the algorithm returns an integer-valued solution then clearly this must also be the optimal solution to the original problem, and nothing more needs to be done.
- But if the algorithm returns some non-integer values for the decision variables, something more needs to be done…

Can’t we just round off?

- To solve an ILP, we could try solving the corresponding LP and round the solutions to the closest integer. E.g., if $x_1 = 2.75$ then set x_1 to be 3.

- This is plausible if the solution is expected to contain large integers and therefore be insensitive to rounding.

- Otherwise, rounding could lead to problems, since it doesn’t usually give a feasible solution!

The rounding-off problem

Consider the following example ILP:

maximise $5x_1 + 8x_2$

Subject to constraints:

$$x_1 + x_2 \leq 6$$
$$5x_1 + 9x_2 \leq 45$$
$$x_1, x_2 \geq 0$$
x_1, x_2 are integers

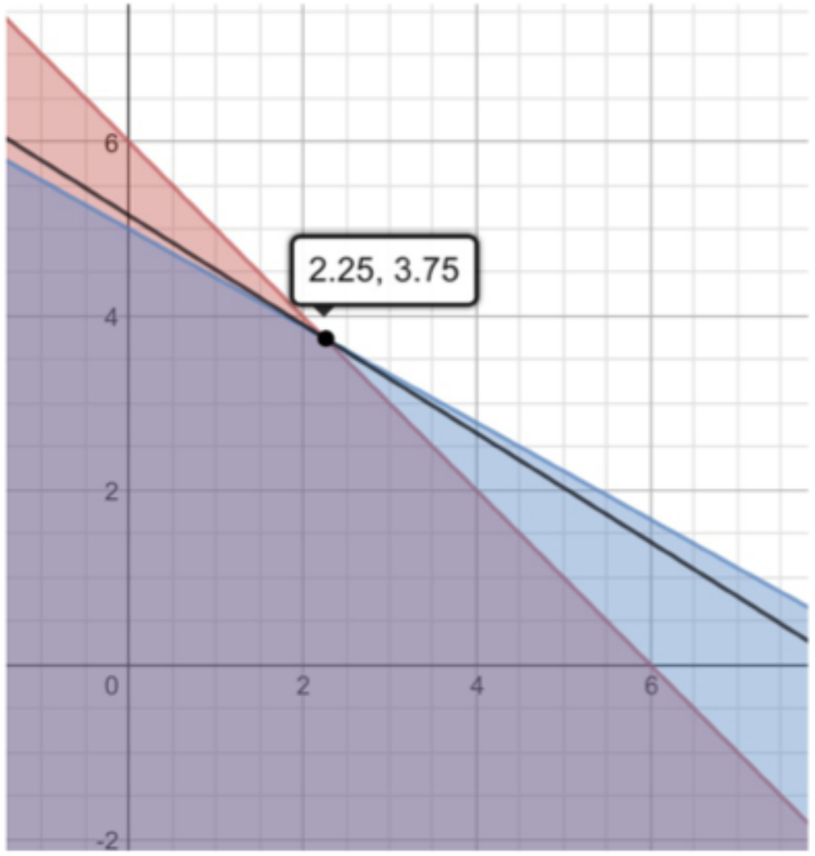
The rounding-off problem

<u>Continuous optimum:</u>
$x_1 = 2.25$, $x_2 = 3.75$
optimal value 41.25

<u>Round off:</u>
$x_1 = 2$, $x_2 = 4$ **INFEASIBLE**

<u>Nearest feasible point</u>
$x_1 = 2$, $x_2 = 3$
Objective function value: 34

<u>Integer optimum</u>
$x_1 = 0$, $x_2 = 5$
Objective function value: 40



Summary

- We started looking at *integer linear programming,* i.e., LP problems in which the decision variables are only allowed to take integer values.
- We’ve seen that the Simplex algorithm can’t be applied directly
    - Simply rounding solutions to nearest integer won’t necessarily work.
- We need a better approach, especially since ILP covers many problems of practical importance.