Day 2 - Mixed-integer programming

Richard Oberdieck

Ørsted A/S

May 26, 2019

Outline

Outline

Mixed-integer programming

Andrew

Schedule selection
I want to assign 70 workers to 70 tasks. How can I do that optimally?

Anne

Network design
I want to design the array cable layout of my offshore wind farm. How can I do that?

James

Facility allocation
I have 9 potential sites for my 3 factories. Where should I build?

Have you had any experience with mixed-integer programming?

Outline

The N-queens problem (`N queens.ipynb`)

The N-queens is the problem of placing N chess queens on an $N \times N$ chessboard so that no two queens attack each other.
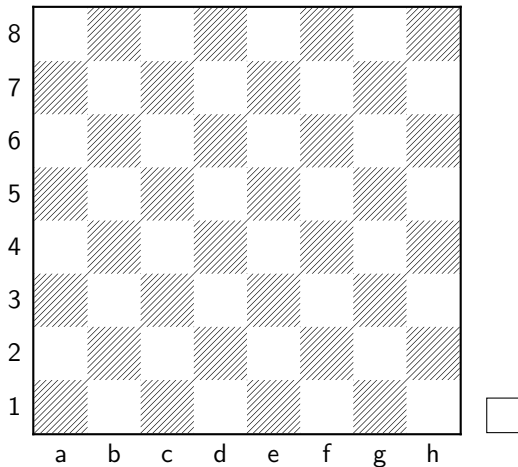
Outline

Now it's your turn: facility location

Engineer

I have 9 potential sites for my 3 factories.
Where should I build?
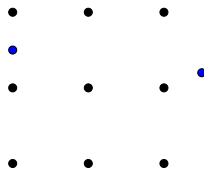
- The possible locations are on a grid with $x, y \in \{0, 1, 2\}$ with
  construction costs as $[3, 2, 3, 1, 3, 3, 4, 3, 2]$ for $(0, 0)$, $(0, 1)$, $(0, 2)$ $\frac{k\$}{km}$
  etc.
- The client sites are at $(0, 1.5)$ and $(2.5, 1.2)$ with 3 and 4 million tons
  demand.
- The mileage cost is $1000\frac{\$}{M \text{ tons}}$.
- The maximum flow avaiable on each connection is 3 million tons.
- We want to make the decision that costs the least amount of money.
- Let's open `Facility location.ipynb` and get to it!

Outline

From linear to mixed-integer programming

- On day 1 we looked at linear programming, which solves a linear objective function in a polytope.
- Now, we add some binary variables $y \in \{0, 1\}^{n_y}$ to the mix:
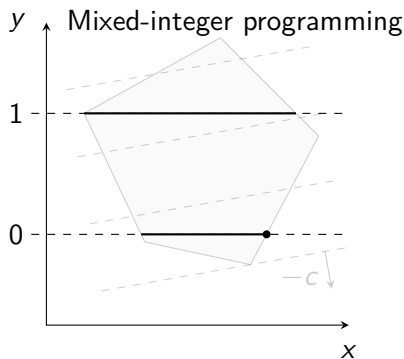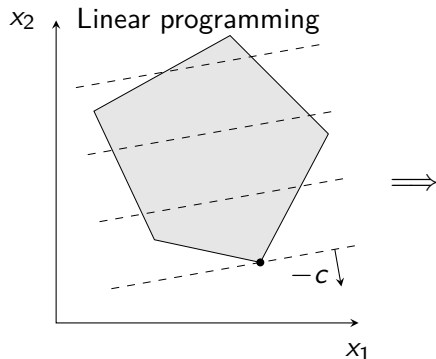
Mixed-integer programming (MIP) problems

$$
\begin{aligned}
z = \quad & \underset{x,y}{\text{minimize}} && c_x^T x + c_y^T y \\
& \text{subject to} && Ax + Ey \leq b \\
& && A_{eq}x + E_{eq}y = b_{eq} \\
& && x \in \mathbb{R}^{n_x}, \ y \in \{0,1\}^{n_y}
\end{aligned}
$$

- Almost always, a binary variable represents a choice. For example this can be a direct choice (build factory or not) or a logical choice (and/or).
- Any bound integer variable can be represented as a set of binary variables. Thus we speak of mixed-integer programming.
- They are the **most** powerful modelling devices at your disposal. We will now learn some of the tricks of mixed-integer programming:

Outline

Mixed-integer modelling tricks

Beyond modelling direct decisions (e.g. opening a valve or not), binary variables are often used for the following three tasks:

Logic: This includes things such as `and`,`or` and `if-then`

Variable modelling: Describing non-standard variables such as semi-continuous variables or fixed costs in the objective function.

Nonlinear modelling: Using binary variables, it is straightforward to model nonlinearities using approximations and reformulations.

Remark

Some of the things shown now are standard parts of modern solvers. However, it is important to know what these commands do under the hood so that you can take the best modelling decisions.

Semi-continuous variable

Suppose you have a variable x which has:

$$x = 0 \text{ or } l \leq x \leq u$$

Let us introduce an indicator variable y:

$$y = \begin{cases} 0 & \text{for } x = 0 \\ 1 & \text{for } l \leq x \leq u \end{cases}$$

$$\Rightarrow ly \leq x \leq uy$$

Note: this type of variable is already part of Xpress (and most other modelling languages with `xp.semicontinuous`. However, if you need y in another part of your model, you still have to model it yourself.

Either-or constraints

Suppose you want one of the two constraints to be used:

$$\sum_{j \in J} a_{1j} x_j \leq b_1$$

$$\sum_{j \in J} a_{1j} x_j \leq b_2$$

Using a binary variable $y \in \{0, 1\}$ the constraints can be rewritten as:

$$\sum_{j \in J} a_{1j} x_j \leq b_1 + M_1 y$$

$$\sum_{j \in J} a_{1j} x_j \leq b_2 + M_2(1 - y)$$

where M_1 and M_2 are *sufficiently large*. Then, if $y = 0$, the second constraint is weakened while $y = 1$ weakens the first constraint.

Conditional constraints (indicator constraints)

If $\sum\limits_{j \in J} a_{1j}x_j \leq b_1$ holds, then $\sum\limits_{j \in J} a_{1j}x_j \leq b_2$ has to to hold:

$$\sum_{j \in J} a_{1j}x_j \leq b_1 \Rightarrow \sum_{j \in J} a_{1j}x_j \leq b_2$$

This is equivalent to saying that

$$\sum_{j \in J} a_{1j}x_j > b_1 \quad \text{or} \quad \sum_{j \in J} a_{1j}x_j \leq b_2$$

has to hold which we just covered. However, you would need to introduce a small tolerance ϵ to convert $>$ into $\geq$.

Special-order sets (SOS)

SOS1

SOS Type 1 is a set of variables where at most one variable may be nonzero. If only binaries are involved, the constraint reduces to:

$$\sum_i y_i \leq 1$$

SOS2

SOS Type 2 is a set of variables where at most two variables may be nonzero. If two variables are nonzero, they must be adjacent in the set.

Note:

- This type of constraint is quite special, and most solvers have a dedicated place for it as they can be treated efficiently.
- Solvers do not restrict the types of variables used (i.e. even continuous variables), however under the hood binary variables are used.

Piecewise linear function
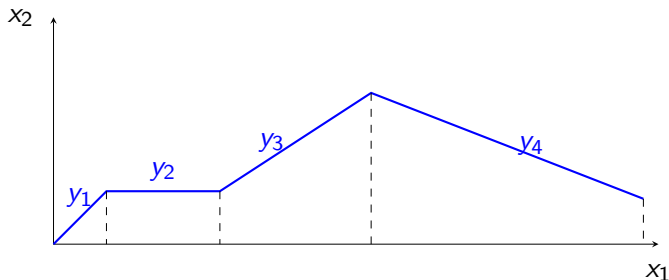
- Unless the piecewise linear function is convex, you need binary variables to encode this.
- Most solvers have ways to do this for you as it is tedious bookkeeping.
- Sadly, Xpress does not provide an implementation for this, so you'll have to do it by hand.

Find *n* best solutions - excluding binary solutions

Sometimes it may be interesting to find the top *n* solutions for a given MIP.

1. Solve the MIP
2. Introduce the following constraint:

$$\sum_{j \in J} y_j - \sum_{t \in T} y_t \le |J| - 1 \tag{1}$$

   where J are the indices where $y_j = 1$, i.e. $J = \{j | y_j = 1\}$ and T are the indices where $y_t = 0$, i.e. $T = \{t | y_t = 0\}$. This will exclude the found solution from the MIP.
3. Solve the problem again and repeat procedure, if necessary.

Product of variables

Suppose you have a term $yf(x)$, where $\delta \in \{0, 1\}$. Then, you can resolve this situation by introducing an auxiliary variable t:

$$t \leq My$$
$$t \geq my$$
$$t \leq f(x) - m(1-y)$$
$$t \geq f(x) - M(1-y)$$

Note that M and m have to be sufficiently large/small, respectively.

Big-M formulation

Definition
Use a *sufficiently large* number M to render certain terms redundant at a certain value of the corresponding binary variable.

- M often is a surrogate for ∞, and therefore has to be *sufficiently large*.
- This means, big-M formulations often suffer from numerical instability and rounding errors, especially when M is hideously large or occurs as a coefficient of a continuous variable.
- When using the big-M formulation, it is **critical** to use the smallest possible M. Sure, 10^{12} will probably work, but it may just make your model numerical nonsense.
- There is no need to use a single M in your model. Rather, for each M you introduce, index it (e.g. M_i) and think of what the lowest value is that gets the job done.

Solving Sudokus using mixed-integer programming

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

- Sudoku is a popular logical puzzle, where you should fill numbers 1-9 such that all rows, columns and 3x3 cells have exactly one of each number.
- You're now going to write a Sudoku solver in `Sudoku.ipynb`!

Outline

Solving MIP problems

$$z = \underset{x}{\text{minimize}} \quad c_x^T x + c_y^T y$$
$$\text{subject to} \quad Ax + Ey \le b$$
$$A_{eq}x + E_{eq}y = b_{eq}$$
$$x \in \mathbb{R}^{n_x}, \ y \in \{0, 1\}^{n_y}$$
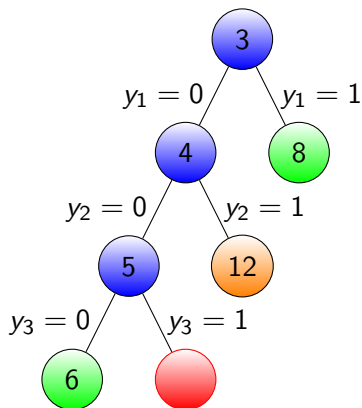
Exhaustive enumeration

Every MIP problem can be solved by exhaustively enumerating all possible combinations of binary variables and solving the resulting 2^{n_y} linear programming problems.

- You should not do this.
- However, it shows how to solve MIP problems: get the binary variables out of the way and solve the LP.
- But, we have to reduce the number of nodes we check, otherwise we cannot solve any realistic problem.

Introducing: branch-and-bound

min. $y_1 - 2y_2$
s.t. $2(y_1 + y_2) \leq 1$
$y_1, y_2 \in \{0, 1\}$

→ solve as continuous problem →

min. $\bar{y}_1 - 2\bar{y}_2$
s.t. $2(\bar{y}_1 + \bar{y}_2) \leq 1$
$\bar{y}_1, \bar{y}_2 \in [0, 1]$

$\bar{y}_1^* = 0, \bar{y}_2^* = 0.5$

$y_1 = 0$

$y_1 = 1$

min. $0 - 2\bar{y}_2$
s.t. $2(0 + \bar{y}_2) \leq 1$
$\bar{y}_2 \in [0, 1]$

$\bar{y}_2^* = 0.5$

min. $1 - 2\bar{y}_2$
s.t. $2(1 + \bar{y}_2) \leq 1$
$\bar{y}_2 \in [0, 1]$

Infeasible

$y_2 = 0$

$y_2 = 1$

min. $0 - 0$
s.t. $2(0 + 0) \leq 1$

$y_1 = 0, y_2 = 0$

min. $0 - 2$
s.t. $2(0 + 1) \leq 1$
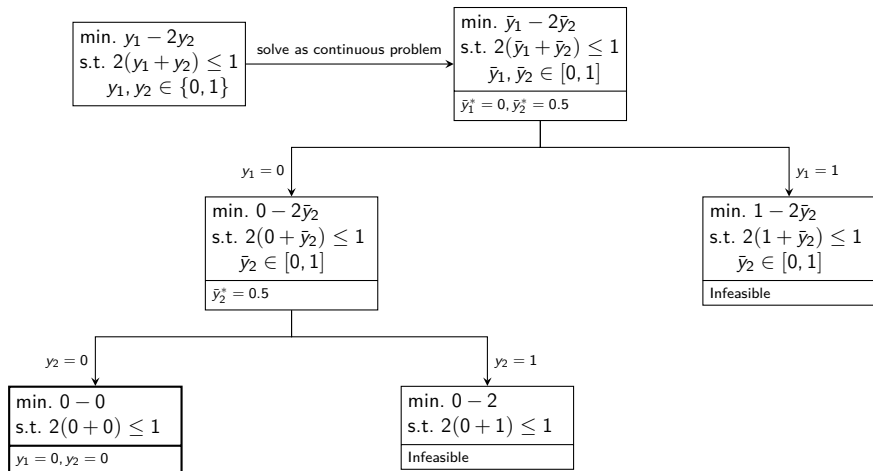
Infeasible

Principles of branch-and-bound

- At each node, **set a binary variable y_i to either 0 or 1** to create two child nodes.
- Every feasible solution to a LP relaxation is a **lower bound**.
- Every feasible integer solution found is an **upper bound**.
- The power of branch-and-bound is to discard nodes and their children, if:
  - ▸ The LP subproblem is infeasible
  - ▸ The solution of the LP subproblem (a lower bound) is greater than the current best upper bound.
  - ▸ The solution of the LP subproblem is an integer solution, which will not improve by going further down the tree.

Extensions to branch-and-bound: branch-and-cut

- Branch-and-bound algorithms are often paired with other methods to increase efficiency.
- One of the most successful ones is **branch-and-cut**, which uses cutting planes (cuts, for short).

Integer solutions

Cutting plane

$-c$
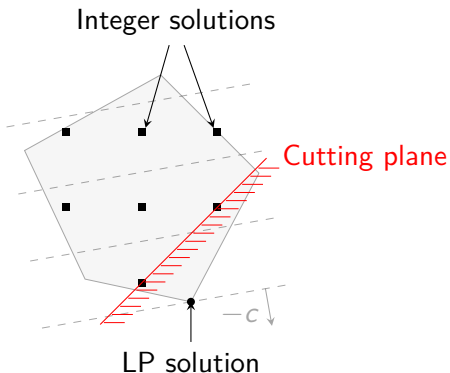
LP solution

Cutting planes (cuts)

"Cuts are constraints added to a model to restrict (cut away) noninteger solutions that would otherwise be solutions of the continuous relaxation. The addition of cuts usually reduces the number of branches needed to solve a MIP." (CPLEX User Manual)

Extensions to branch-and-bound: Benders decomposition

- In case you have many independent "blocks" in your model (e.g. many different scenarios), you can use Benders decomposition.
- Typically you need to code this yourself, although CPLEX has an in-built method (`parameters.benders.strategy`) that works well.

$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & c^T x \\
\text{subject to} \quad & Ax \le b \\
& x \in \mathbb{R}^{n_x} \times \{0, 1\}^{n_y}
\end{aligned}
$$

$$\Downarrow$$

Benders decomposition

The original problem is divided into a master problem and a subproblem using only a subset of the variables based on the block structure. By iteratively solving this master-subproblem set and the generation of appropriate cuts when the master problem becomes infeasible, very large LP/MIP problems can be solved.
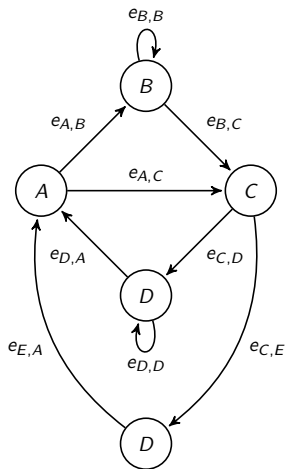
Outline

A case worth mentioning separately: graphs

Graphs

A graph $G(V, E)$ is a mathematical object, where a set of vertices V are linked together by some edges E.

- Graphs are very common in modelling in general, and MIP modelling in particular. Examples include supply chains, electrical circuit design, superstructure design, flow networks etc.

- Modelling with graphs can be very efficient, and many solvers have special heuristics to recognize graphs in the models.

Tips and tricks with graphs

- Two of the most important classifications of graphs are:

 Directed/Undirected: A graph is directed if $A \to B \not\equiv B \to A$, and undirected otherwise.

  Complete: A graph is called complete if any vertex v_i is linked to any other vertex v_j by an edge $e_{i,j}$

- When modelling graphs it is very common to use **semi-continuous variables**, e.g. when using flows.

- Avoid complete graphs when possible, as the number of variables explodes fairly quickly.

- It is common to use additional indices when modelling, e.g. to approximate nonlinearity.
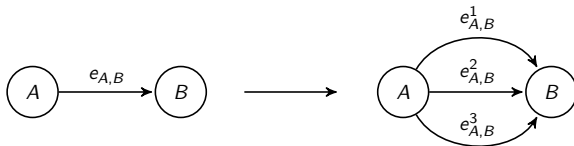
Tree design

Select the edges from this graph such that we have the overall minimum edge weight to connect all nodes with node g (see Tree design.ipynb).
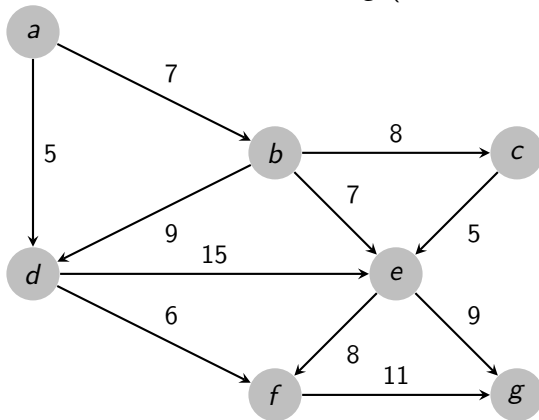
Problem 1 - Nurse scheduling

Problem 2 - Food manufacturing