6. (Mixed) Integer Linear Programming

Roberto Amadini

Department of Computer Science and Engineering, University of Bologna, Italy

Combinatorial Decision Making and Optimization

2^{nd} cycle degree programme in Artificial Intelligence

University of Bologna, Academic Year 2024/25

From Reals to Integers

- We have seen the simplex method to solve LP problems in "*typically polynomial*" time
- We however can always solve LP problems in polynomial time
  - Ellipsoid methods (inefficient, worse than simplex in practice)
  - Interior point methods (e.g., Karmarkar, can outperform simplex)
- If the domain of the variables involved is $\mathbb{R}$ or $\mathbb{Q}$ solving LP problems is not NP-hard
- What if we require domains to be subsets of $\mathbb{Z}$ or $\mathbb{N}$? Would the problem be easier? Or harder? Or the same?
  - Let's go back to the baking example

Baking cakes

```
% We know how to make two sorts of cakes. A banana cake which takes 250g of
% self-raising flour, 2 mashed bananas, 75g sugar and 100g of butter, and a
% chocolate cake which takes 200g of self-raising flour, 75g of cocoa, 150g
% sugar and 150g of butter. We can sell a chocolate cake for $4.50 and a
% banana cake for $4.00. And we have 4kg self-raising flour, 6 bananas,
% 2kg of sugar, 500g of butter and 500g of cocoa. How many of each sort of
% cake should we bake for the fete to maximise the profit

var 0..100: b; % no. of banana cakes
var 0..100: c; % no. of chocolate cakes

% flour
constraint 250*b + 200*c <= 4000;
% bananas
constraint 2*b  <= 6;
% sugar
constraint 75*b + 150*c <= 2000;
% butter
constraint 100*b + 150*c <= 500;
% cocoa
constraint 75*c <= 500;

% maximize our profit
solve maximize 400*b + 450*c;

output ["no. of banana cakes = \(b)\n",
         "no. of chocolate cakes = \(c)\n"];
```

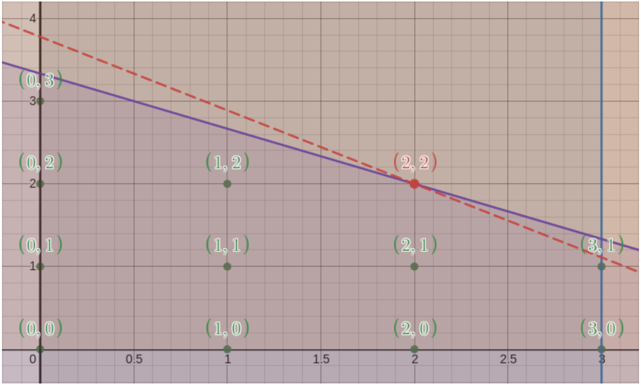
Baking cakes

- b, c $\in$ $0..100$ $= \{0, 1, \ldots, 100\}$ $\subsetneq$ $[0, 100]$ $= \{x \in \mathbb{R} \mid 0 \leq x \leq 100\}$
- $b = 3$, $c = 4/3$ can't be a feasible solution anymore!

Baking cakes

- On the previous episodes, we assumed we could sell slices of cakes
  - E.g., one third of banana cake and two fifths of chocolate cake
- This could be a too strong assumption in a real-world setting
  - Imagine we are selling cars instead of cakes
- The feasible/optimal regions are now sets of integer points
  - "Grids" instead of polyhedra
- What is the optimal solution if we strictly require $b, c \in 0..100$?

Baking cakes



- $b, c \in 0..100 = \{0, 1, \ldots, 100\} \subsetneq [0, 100] = \{x \in \mathbb{R} \mid 0 \leq x \leq 100\}$
- Optimal solution $b = c = 2$

Integer Linear Programming

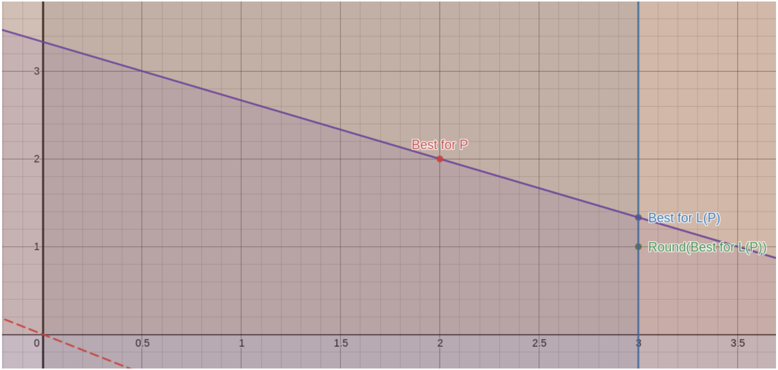
- An **Integer Linear Programming (ILP)** problem has standard form:

$$
\begin{array}{lll}
\max & \sum_{j=1}^{n} c_j x_j & \\
\text{s.t.} & \sum_{j=1}^{n} a_{i,j} x_j = b_i & 1 \leq i \leq m \\
 & x_j \geq 0, \quad x_j \in \mathbb{Z} & 1 \leq j \leq n
\end{array}
$$

- If we require only $k < n$ variables to be integers, then it is a **Mixed-Integer Programming (MIP)** problem
  - ILP a.k.a. Integer Programming (IP)
  - MIP a.k.a. Mixed-Integer Linear Programming (MILP)

- Linear relaxation: MIP problem with no integrality constraints $x_j \in \mathbb{Z}$
  - If $\mathcal{L}(P)$ is the linear relaxation of P, $\mathcal{F}_P \subseteq \mathcal{F}_{\mathcal{L}(P)}$ hence solving $\mathcal{F}_{\mathcal{L}}(P)$ and rounding its optimal solution does not work in general

Linear relaxation



- Optimal solution of linear relaxation: $b = 3, c = 4/3 = 1.\overline{3}$, rounding to nearest integer $b = 3, c = 1$. Obj. value $= 400 \cdot 3 + 450 \cdot 1 = 1650$
- Optimal solution original problem: $b = 2, c = 2$. Obj. value $= 400 \cdot 2 + 450 \cdot 2 = 1700$

Linear relaxation

- Rounding the optimal solution of $\mathcal{L}(P)$ is not sound in general!
- E.g. with $P : \max(x)$ s.t. $x \leq 5/3, x \geq 0, x \in \mathbb{Z}$ we have that $5/3 = 1.\overline{6}$ is optimal for $\mathcal{L}(P)$ but its rounding is $2 \notin \mathcal{F}_P = \{0, 1\}$
  - In this case $\mathcal{O}_P = \{1\} \neq \{\frac{5}{3}\} = \mathcal{O}_{\mathcal{L}(P)}$
- Clearly, $\mathcal{F}_{\mathcal{L}(P)} = \emptyset \implies \mathcal{F}_P = \emptyset$: solving $\mathcal{F}_{\mathcal{L}(P)}$ can help to detect unsatisfiability of P
- If $\mathcal{L}(P)$ unbounded, P can be:
  - bounded, e.g. $P : \max(x_1)$ s.t. $x_1 = \sqrt{2}x_2$ and $x_1, x_2 \in \mathbb{N}$
  - unbounded, e.g. $P : \max(x)$ s.t. $x \in \mathbb{N}$
  - unsatisfiable, e.g. $P : \max(x_1)$ s.t. $0.1 \leq x_2 \leq 0.2$ and $x_1, x_2 \in \mathbb{N}$

Complexity

- Adding integrality has huge impact on the complexity of LP solving
- No known algorithms for solving MIP problems in polynomial time
  - otherwise, $\mathbf{P} = \mathbf{NP}$
- More precisely, MIP problems are NP-complete
  - They are in **NP** (certifying solutions takes polynomial time)
    - *Solvable by NDTM in polynomial time...*
  - They are among the *hardest* problems in **NP** (NP-hard)
- If we could solve a generic MIP problem in polynomial time, we could also solve, e.g., any SAT problem in polynomial time
  - And all the NP problems in polynomial time...

SAT to MIP reduction

- From any generic SAT problem P_{SAT} with clauses $C_1, \ldots, C_m$ and literals $\ell_1, \ldots, \ell_n$ we get an equisatisfiable MIP problem P_{MIP} with n variables x_j (one per literal) and m constraints (one per clause):

$$\begin{array}{lll} \max & 0 & \\ \text{s.t.} & \sum_{\ell_j \in P_i} x_j + \sum_{\ell_j \in N_i} (1 - x_j) \geq 1 & 1 \leq i \leq m \\ & x_j \in \{0,1\} & 1 \leq j \leq n \end{array}$$

  where P_i and N_i are the positive and negative literals of clause C_i

- E.g. $\ell_1 \vee \neg\ell_2 \vee \ell_4 \vee \neg\ell_5 \implies x_1 + (1 - x2) + x_4 + (1 - x_5) \geq 1 \implies x_1 - x_2 + x_4 - x_5 \geq -1 \implies -x_1 + x_2 - x_4 + x_5 \leq 1$

SAT to MIP reduction

- Reducing P_{SAT} to P_{MIP} takes polynomial time
  - $O(mn)$ time
- P_{SAT} feasible $\iff$ P_{MIP} feasible, and any solution of P_{MIP} can be mapped back into a solution of P_{SAT} in polynomial time
  - $O(n)$ time: $x_j = 0 \mapsto \ell_j = \mathit{false}$, $x_j = 1 \mapsto \ell_j = \mathit{true}$
- If P_{MIP} solvable in polynomial time $\implies$ any SAT problem solvable in poly. time $\implies$ any NP-complete problem in poly. time $\implies$ **P=NP**
- Because rounding is in general not applicable, we have to tackle MIP solving with other approaches

Handling NP-hardness

Different ways of handling NP-hard problems:

- Exact algorithms: they guarantee to find an optimal solution although this may take exponential time
  - The focus of this course
- Approximation algorithms: they guarantee in polynomial time a (sub-)optimal solution at most ρ times worse the optimal one
  - $\rho =$ approximation factor
- Heuristic algorithms: no guarantee of optimality nor polynomial runtime, but "in practice" they find good solutions in reasonable time
  - According to empirical evidences

Branch-and-bound

- Branch-and-bound (BB) is based on the divide-et-impera principle: split a “big” problem into sub-problems until a success or a failure
  - Overall solution derived from the solution of sub-problems
- Branch phase = explore the sub-problems
  - sub-trees of the search tree
- Bound phase = compute the bounds of sub-problem optimal solution
  - to possibly prune the search tree if current solution not improvable
- BB is a general paradigm applicable to various NP-hard problems

Branch-and-bound

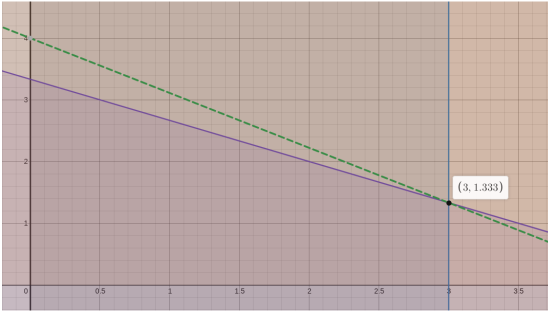
- We can solve a standard MIP problem P via BB. Suppose $P_0 = \mathcal{L}(P)$ has solution $\beta_1, \ldots, \beta_n$ with some $\beta_k \notin \mathbb{Z}$ (*very lucky otherwise!*)
- Pick a x_k s.t. $\beta_k \notin \mathbb{Z}$ and branch: $\begin{cases} P_1 = P_0 \cup \{x_k \leq \lfloor \beta_k \rfloor\} \\ P_2 = P_0 \cup \{x_k \geq \lceil \beta_k \rceil\} \end{cases}$
  - Note $\mathcal{F}_P = \mathcal{F}_{P_1} \cup \mathcal{F}_{P_2}$ and $\mathcal{F}_{P_1} \cap \mathcal{F}_{P_2} = \emptyset$
- We can solve P_1, P_2 to optimality and take the best solution (if any)
  - If integral, optimal for P too! Otherwise, branch again on $P_1, P_2, \ldots$
- So we build a search tree rooted in P_0 with edges $P_i \to P_j$ if child node P_j is a sub-problem of parent node P_i

Branch-and-bound

- If a P_k has integral optimal solution, compare its obj. value z_k with the best obj. value z^* so far (best bound): if $z_k > z^*$, then $z^* \leftarrow z_k$
- Otherwise, we cannot improve z^*
  - In any case, fathom this node: P_k will be a leaf
  - Initially, $z^* \leftarrow -\infty$
- If P_k is unfeasible or has non-integral solution with obj. value $z_k \leq z^*$ also fathom its node (prune sub-tree rooted in P_k)
- At the end, the optimal solution for P is a leaf P_k with obj. value z^*
  - Leaves also called fathomed nodes
  - Current optimal solution also called incumbent solution

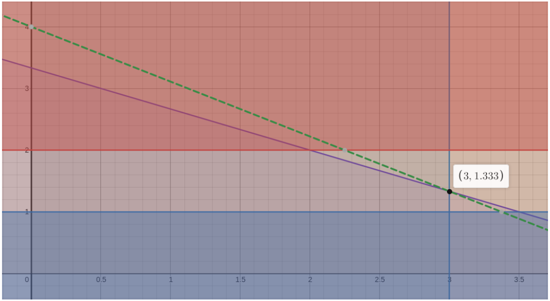
Baking example

Let's how BB works on the baking problem where $B, C \in 0..100$

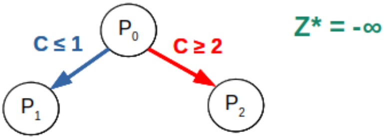


If we solve $P_0 = \mathcal{L}(P)$, optimal solution $B = 3$, $C = 4/3$ not feasible: we need to branch on C: $\begin{cases} P_1 = P_0 \cup \{C \leq \lfloor 4/3 \rfloor = 1\} \\ P_2 = P_0 \cup \{C \geq \lceil 4/3 \rceil = 2\} \end{cases}$

Baking example

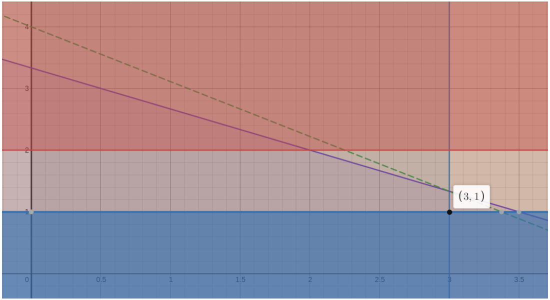


The resulting search tree is:

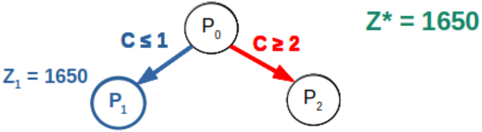


remember objective function is $Z = 400B + 450C$

Baking example

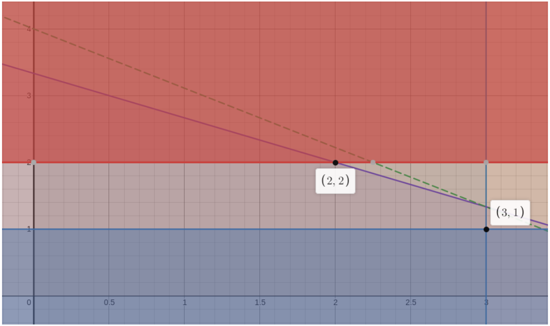


If $C \leq 1$, optimal solution is integral: $B = 3, C = 1$ with value 1650

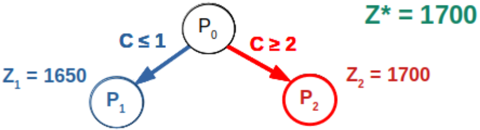


Node P_1 is a leaf: we backtrack and explore P_2

Baking example



If $C \geq 2$, we get a better solution $B = 2$, $C = 2$ with value 1700



search completed: solution of node P_2 is optimal

Branch and Bound

- BB works well typically in combination with other techniques
  - Presolve
  - Cutting planes
  - Heuristics
  - Parallelism
  - ...

Presolve

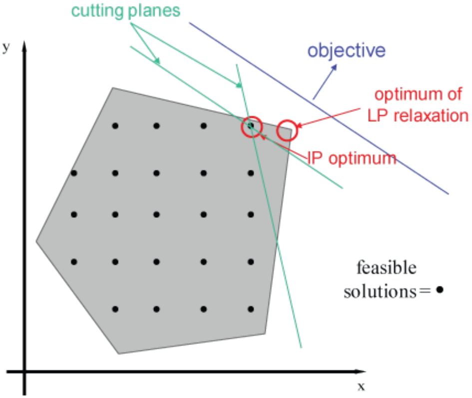
- Presolve means reformulating a problem before its actual solving process to possibly reduce its size
  - Presolve should be computationally efficient
- Bounds tightening, e.g.: $\{x_1 + x_2 \geq 20, x_1 \leq 10\} \models x_2 \geq 10$
  - If $x_2 \in 1..9$, we eagerly detect unsatisfiability
- Problem reduction, e.g.: $\{x_1 + x_2 \leq 0.8\} \models x_1 = x_2 = 0$
  - x_1, x_2 can be removed from problem formulation
- Pre-processing a MIP problem P is important because it can reduce the size of $\mathcal{F}_{\mathcal{L}(P)}$ without altering $\mathcal{F}_P$

Cutting planes

- Cutting planes allowed for significant advancements of MIP solving
- Instead of (*in addition to...*) branching on sub-problems, we repeatedly add new constraints entailed by the original problem P
- The idea is to "cut out" parts of $\mathcal{F}_{\mathcal{L}(P)} - \mathcal{F}_P$ along the solving process to remove non-integral solutions
    - Until we converge to an optimal solution for P
- The existence of a cut separating optimal solution in $\mathcal{F}_{\mathcal{L}(P)} - \mathcal{F}_P$ from $\mathcal{F}(P)$ is always guaranteed
    - But not its uniqueness!

Cutting planes

Formally, a cut for a MIP problem P in standard form is an inequality $px \leq q$ such that $py \leq q$ and $pz > q$ for each $y \in \mathcal{F}_P$ and $z \in \mathcal{O}_{\mathcal{L}(P)}$



Cutting Planes

Image from https://www.gurobi.com/resource/mip-basics/

Gomory's cut

- Proposed by R. Gomory in the 1950s

- Suppose basis $\mathcal{B}^* = \{x_{i_1}, \ldots, x_{i_m}\}$ is optimal for $\mathcal{L}(P)$. The problem in terms of the basic variables (scalar form) is:

$$x_{i_k} = \beta_k + \sum_{j=1}^{n-m} \alpha_{k,j} x_{i_{m+j}} \qquad \text{for } k = 1, \ldots, m$$

  where $x_{i_{m+1}}, x_{i_{m+2}} \ldots, x_{i_n}$ are non-basic variables

- The optimal solution x^* is $x^*_{i_k} = \beta_k$ for $k = 1, \ldots, m$ and $x^*_{i_j} = 0$ for $j = m+1, \ldots, n$. If there is a $\beta_k \notin \mathbb{Z}$, then $x^* \in \mathcal{F}_{\mathcal{L}(P)} - \mathcal{F}_P$ so we generate a cut to separate x^* from $\mathcal{F}_P$

Gomory's cut

- Gomory's cut has the form $-f_k + \sum_{j=1}^{n-m} f_{k,j} x_{i_{m+j}} \geq 0$ where:
  - $f_k = \beta_k - \lfloor \beta_k \rfloor$
  - $f_{k,j} = -\alpha_{k,j} - \lfloor -\alpha_{k,j} \rfloor$
  - $0 < f_k < 1$ and $0 \leq f_{k,j} < 1$ by definition of $\lfloor \cdot \rfloor$ and because $\beta_k \notin \mathbb{Z}$
  - So $-f_k + \cancel{\sum_{j=1}^{n-m} f_{k,j} x^*_{i_{m+j}}} < 0$: the cut is violated by optimal solution x^*
  - We can prove that $-f_k + \sum_{j=1}^{n-m} f_{k,j} x_{i_{m+j}} \geq 0$ holds for each $x \in \mathcal{F}_P$
- Then $\mathcal{L}(P) \leftarrow \mathcal{L}(P) \cup \left\{ y_k = -f_k + \sum_{j=1}^{n-m} f_{k,j} x_{i_{m+j}}, \quad y_k \geq 0 \right\}$
  - A new slack variable y_k added at each round
- $\mathcal{B}' \leftarrow \mathcal{B}^* \cup \{y_k\}$ and solve with dual simplex
  - $\mathcal{B}'$ now unfeasible ($y_k = -f_k < 0$) but dual feasible

Example

- E.g., for baking example we get $\mathcal{B}^* = \{S_1, B, S_3, C, S_5\}$, with optimal solution $B = 3$, $C = 4/3$, $S_1 = \cdots = S_5 = 0$ having value $z^* = 1800$
  - $\alpha_{C,S_2} = 1/3$, $\alpha_{C,S_4} = 1/150$
- $f_C = \frac{4}{3} - \lfloor\frac{4}{3}\rfloor = \frac{4}{3} - 1 = \frac{1}{3}$
- $f_{C,S_2} = -\frac{1}{3} - \lfloor-\frac{1}{3}\rfloor = -\frac{1}{3} + 1 = \frac{2}{3}$
- $f_{C,S_4} = -\frac{1}{150} - \lfloor-\frac{1}{150}\rfloor = \frac{1}{150} + 1 = \frac{149}{150}$
- Gomory's cut is $y_C = -\frac{1}{3} + \frac{2}{3} \cdot S_2 + \frac{149}{150} \cdot S_4 \geq 0$
  - This constraint "cuts-out" current non-integral solution
- Starting basis for dual simplex is $\{S_1, B, S_3, C, S_5, y_C\}$, then y_C out, choose entering variable, reformulate with new basis...

Branch-and-cut

- Gomory's cut considered ineffective at the time because of numerical instability and number of cuts needed for convergence
- In mid 1990s, G. Cornuéjols et al. proved it to be very effective in combination with branch-and-bound: branch-and-cut
- Basically, it runs BB for P and if an optimal solution of $\mathcal{L}(P)$ is not integral it adds cutting planes to tighten $\mathcal{L}(P)$
  - E.g., `https://www.ibm.com/docs/en/icos/22.1.0?topic=c-branch-cut-in-cplex-1`
- Different cutting planes algorithms (or separators) can be used
  - ...and learned: `https://arxiv.org/abs/2311.05650`

Row and Column Generation

- Cutting planes can be seen as "row generation" methods: new constraints are added at each step
- Bender's decomposition is another row generation method dividing a problem P into master problem (min) and sub-problem(s) (max)
  - Idea: iteratively fix the value of some variables and solve the dual of residual sub-problem to get cuts or better objective bounds
  - Logic-Based Bender's Decomposition: sub-problems are generic problems solvable with any solver (e.g., CP or SMT solvers)
- Also column-generation methods exist: start with subset of variables and repeatedly add variables until objective value cannot be improved
  - Assumption: only a small subset of variables is useful
  - E.g., Dantzig–Wolfe decomposition

Heuristics

- Heuristics methods aim to find “good” solutions in “reasonable” time
  - Inherently empirical, weak theoretical guarantees
- Constructive methods: start with empty solution and iteratively extend it to a complete solution
- Local search methods: start with a complete solution and iteratively modify parts to improve it
- Evolutionary methods: explore a population of solutions using mutation, crossover, and selection
- Meta-heuristics methods: higher-level strategies for selecting, combining, tuning, or guiding other heuristics

Primal heuristics

- Modern MIP solvers often use so-called primal heuristics in addition to branching and cutting, e.g.:
- Rounding non-integral solutions and check
  - Easy and fast, typically used at root node
- Diving: iteratively fix fractional variables to some value and solve the remaining sub-problem
  - Helps to quickly find feasible solutions

Primal heuristics

- Neighborhood-based: improve incumbent by solving sub-problems "around it" by fixing some incumbent variables
  - Relaxation-Induced Neighborhood Search (RINS), Large Neighborhood Search (LNS), Local branching, ...
- Node Selection: decide specific nodes to apply heuristics selectively rather than at every node
  - E.g. according to depth, variable fixings, dual gap, ...
- Adaptive dynamic heuristic selection and configuration of heuristics
  - Parameters tuning, portfolios, ML methods...

Warm starts

- MIP solving can be warm-started with initial value(s) to (some of) the variables
- This not necessarily translates to a new incumbent: warm-start solution might be unfeasible or not better than current best bound
  - Or, if partial, it might take too long to compute a complete solution
- A warm start vector might come from the knowledge of a similar problem, or from the expertise of a domain expert
- E.g., warm-start the MIP solver by reusing a route plan or a timetable from a previous day or period

Take-home messages

- Adding integer variables to LP significantly increases its complexity
- MIP solving can be tackled with (a combination of) different approaches
  - Exact
  - Approximate
  - Heuristic
- Rounding non-integral solutions of linear relaxation does not work in general

Take-home messages

- Branch-and-bound: divide-et-impera approach, branches on variables with non-integer value, stop if we cannot improve incumbent solution
- Cutting planes: linear equalities separating non-integral, optimal solutions of linear relaxation from feasible region of original problem
  - Gomory's cut
  - Branch-and-cut
- Heuristics: no strong guarantees on optimality/runtime, but "in practice" they find good solutions in reasonable time
  - Inherently empirical