Topics in Optimization

João Pedro Gonçalves Dionísio

April 4, 2023

Course Structure

- 6 weeks?
- 1h lecture + 30min exercises

Github Page

https://github.com/Joao-Dionisio/Minicurso

- Slides (unfinished)
- Lecture Notes (unfinished)
- Some Code (unfinished)
- Competition Details

Competition

Course Contents

1. Convexity Theory
2. Linear Programming
3. Complexity Theory
4. Integer Programming
5. Decomposition Methods

Convex sets

Definition

A set $X \in \mathbb{R}^n$ is said to be convex if

$$\forall x, y \in X, (1-t)x + ty \in X, t \in [0,1]$$

Convex sets

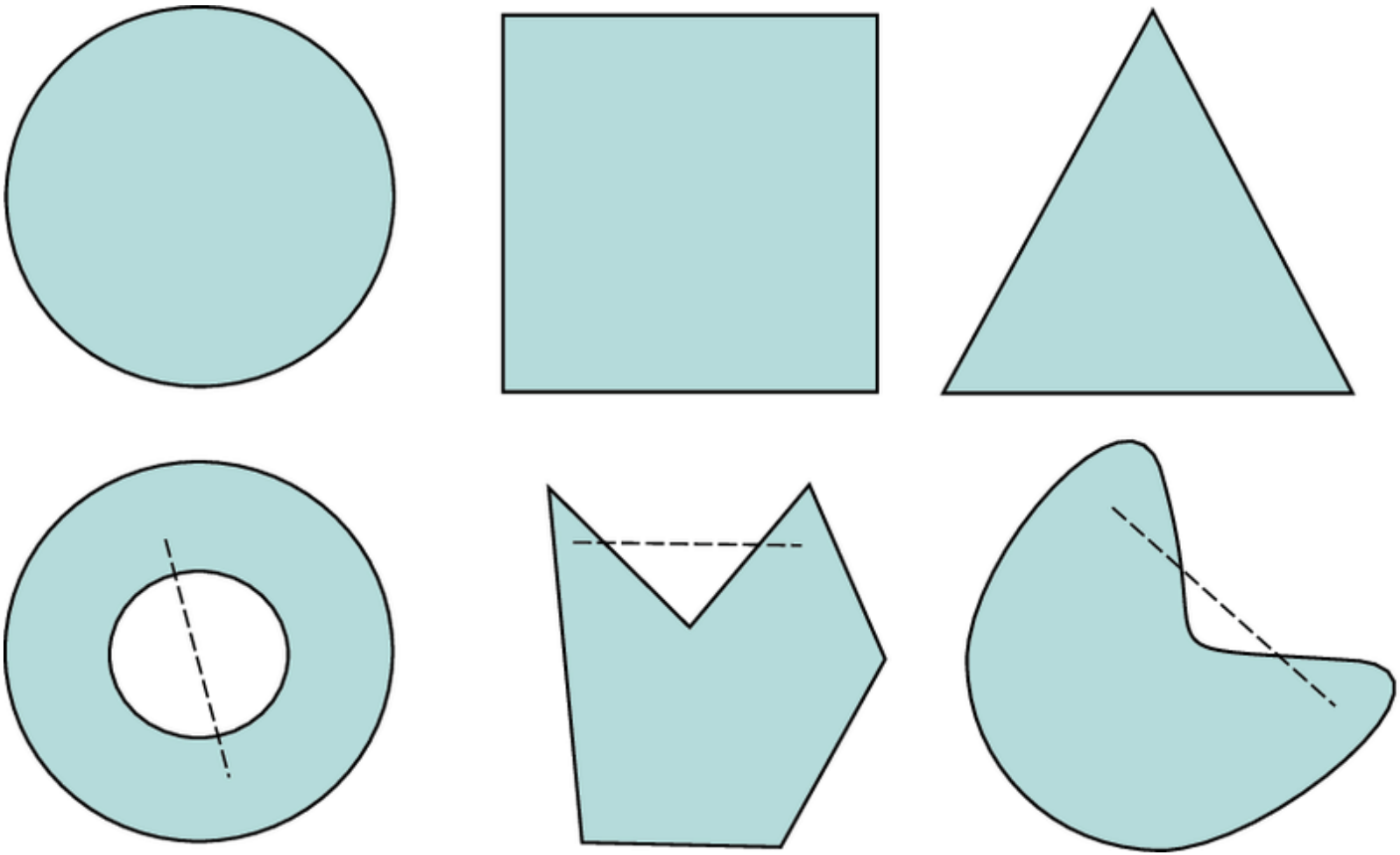


Figure: Examples of convex and non-convex sets

Convex Functions

Definition

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be convex if $\forall x, y \in \mathbb{R}^n, \forall \lambda \in [0, 1]$ we have

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y)$$

Convex Functions

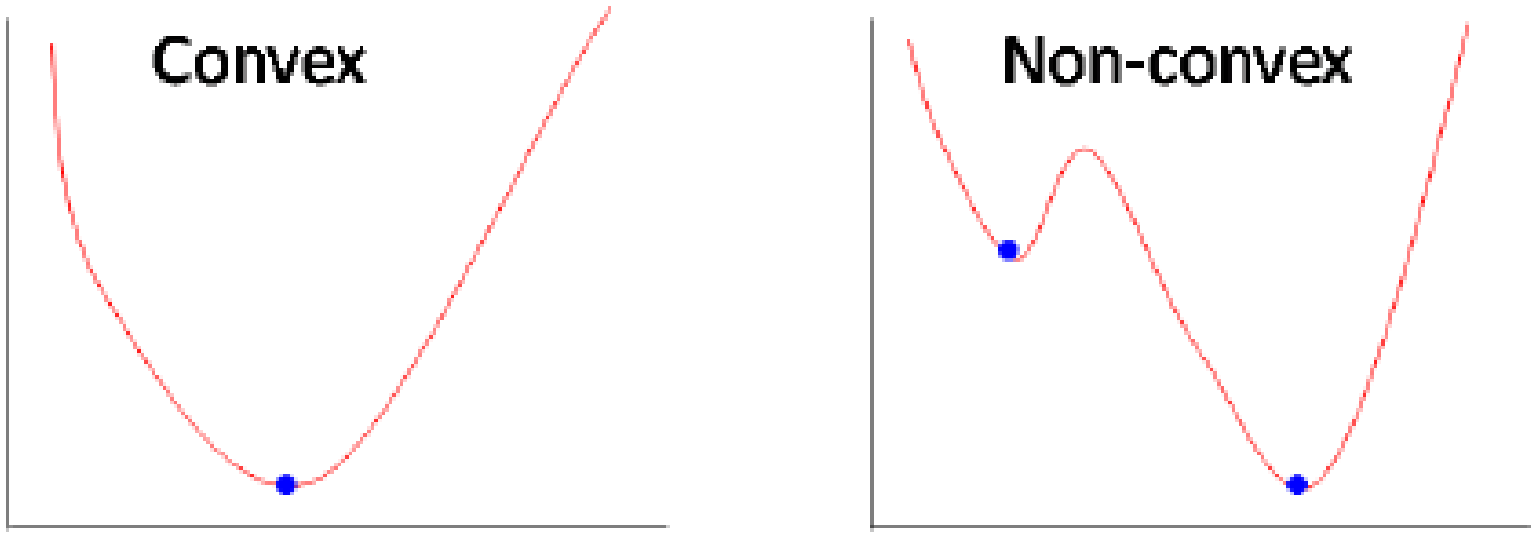


Figure: Examples of convex and non-convex functions

Epigraph

Definition

We call the epigraph of a function $f : X \to \mathbb{R}$, denoted by epi(f), to the following set:

$$epi(f) = \{(x, y) \in X \times \mathbb{R} \mid f(x) \leq y\}$$

Epigraph

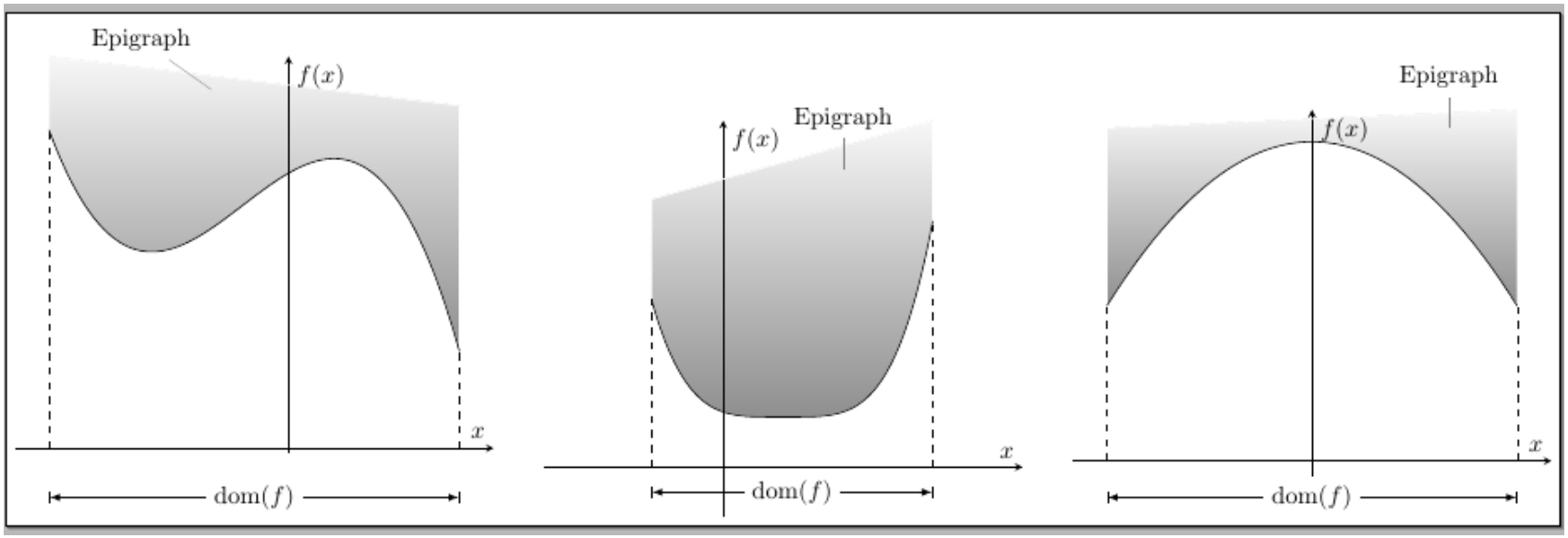


Figure: Examples of epigraph

Convex sets and convex functions

Proposition

The following two statements are equivalent:

1. *f is convex;*
2. *$epi(f)$ is a convex set.*

Convex Optimization

Definition

A convex optimization problem is

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \geq 0, i \in [m] \end{aligned}$$

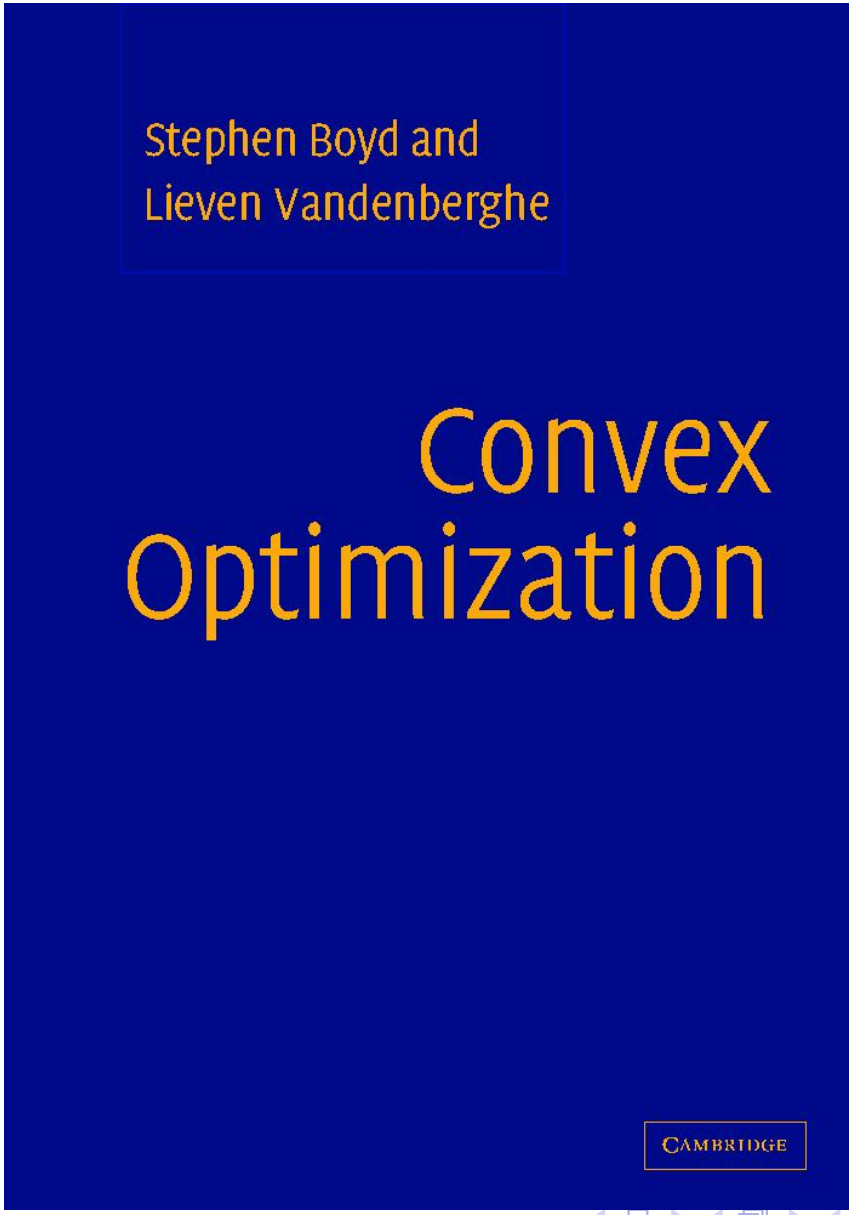
$f, g_i, i = 1 \ldots, m$ convex functions. It implies that $\bigcap_{i \in [m]} \text{epi}(g_i)$ forms a convex set.

Sufficient Optimality Condition

Theorem

Let $\mathcal{P}$ be a convex optimization problem, and without loss of generality, assume it is a minimization problem. Then, if x^* is a local minimizer, then x^* is a global minimizer.

Bibliography



Linear Programming

$$\begin{aligned} \min_{x} \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

The feasible region is a polyhedron.

Small Example

Dunder Mifflin can produce two types of industrial-sized sheets, type A and type B. Type A *can be produced* at a ratio of **200m** per hour, while type B can be produced at a ratio of **140m** per hour. *The profits* from each type of paper are **25** cents per meter and **30** cents per meter, respectively. Taking the market demand into account, next week's production schedule *cannot exceed* **6000m** for paper of type A and **4000m** for paper of type B. If on that week there is a *limit of* **40** production hours, how many meters of each product should be produced to maximize the profit?

Small Example

$$
\begin{aligned}
\max_{A,B} \quad & 25A + 30B \\
\text{s.t.} \quad & A/200 + B/140 \leq 40 \\
& A \leq 6000 \\
& B \leq 4000 \\
& A, B \geq 0
\end{aligned}
$$

Example - Max flow

Suppose you have a network of pipes and receive money based on the amount of a valuable liquid that reaches a single destination, coming from a single source. Assume that the pipes have limited capacity and none of the liquid is gained or lost along the way. This is the max-flow problem, whose linear programming model follows.

Example - Max Flow

Definition

Let $G(V, E)$ be a graph with $s, t \in V$ being defined as the source and target. The **max-flow problem** is the following LP:

$$\begin{aligned} \max \quad & \sum_{v:(s,v)\in E} f_{sv} \\ \text{s.t.} \quad & f_{uv} \leq c_{uv}, \forall (u,v) \in E \\ & \sum_{u} f_{uv} - \sum_{w} f_{vw} = 0, \forall v \in V \setminus \{s,t\} \end{aligned}$$

PySCIPOpt

Supporting hyperplane theorem

Theorem (Supporting Hyperplanes)

Let $C \subset \mathbb{R}^n$ be a convex set, and $x \in \partial C$. Then, there exists a hyperplane H, s.t. $x \in H \cap C$ and C is contained in one of the half-spaces bounded by H.

It justifies the importance of LPs in the more general Convex Optimization.

Supporting hyperplane theorem

Figure: Visual representation

Duality

How can we know if we are close to the optimal solution? Can we derive bounds?

Definition

The dual problem of an LP of the form 16 (which we now call the primal) is:

$$\begin{aligned} \max_{y} \quad & b^\intercal y \\ \text{s.t.} \quad & A^\intercal y \geq c \\ & y \geq 0 \end{aligned}$$

Every primal constraint has an associated dual variable, and vice-versa.

Strong Duality

Theorem (Strong Duality for LPs)

Let P be an LP, D the corresponding dual, and x^*, y^* be the respective optimal solutions. We have that $b^\intercal y^* = c^\intercal x^*$.

Interpretations of the Dual

The maximum amount of money that the decision maker will be willing to spend to buy an additional resource.

Optimal solution at vertex

Theorem

Consider the following LP:

$$\begin{aligned} \min_{x} \quad & c^{\mathsf{T}} x \\ s.t. \quad & Ax \leq b \quad (P) \\ & x \geq 0 \end{aligned}$$

Suppose it has at least one vertex. Then, if an optimal solution exists, there is also an optimal solution at a vertex.

Definition

A point x of a convex set C is an **extreme point** (vertex) if $\nexists y, z \in C, \lambda \in]0,1[\mid x = \lambda y + (1-\lambda)z$.

Definition

A point x of a convex set C is an **extreme point** (vertex) if $\nexists y, z \in C, \lambda \in]0,1[\mid x = \lambda y + (1-\lambda)z$.

Definition

A polyhedron P contains a **line** if $\exists d \in \mathbb{R}^n, x \in P \mid \forall \lambda \in \mathbb{R}, x + \lambda d \in P$.

Definition

A point x of a convex set C is an **extreme point** (vertex) if $\nexists y, z \in C, \lambda \in]0,1[\mid x = \lambda y + (1-\lambda)z$.

Definition

A polyhedron P contains a **line** if $\exists d \in \mathbb{R}^n, x \in P \mid \forall \lambda \in \mathbb{R}, x + \lambda d \in P$.

Lemma

P has a line $\iff$ P does not have an extreme point.

Proof Sketch

P has an extreme point $\implies$ P has no line.

Proof Sketch

P has an extreme point $\implies$ P has no line. Q the set of optimal points has no line ($Q \subseteq P$), so it has an extreme point.

Proof Sketch

P has an extreme point $\implies$ P has no line. Q the set of optimal points has no line ($Q \subseteq P$), so it has an extreme point. Let x^* be an extreme point of Q. We want to show that it is an extreme point of P.

Proof Sketch

P has an extreme point $\implies$ P has no line. Q the set of optimal points has no line ($Q \subseteq P$), so it has an extreme point. Let x^* be an extreme point of Q. We want to show that it is an extreme point of P. We assume it is not, and write it as $x^* = \lambda y + (1 - \lambda)z$.

Proof Sketch

P has an extreme point $\implies$ P has no line. Q the set of optimal points has no line ($Q \subseteq P$), so it has an extreme point. Let x^* be an extreme point of Q. We want to show that it is an extreme point of P. We assume it is not, and write it as $x^* = \lambda y + (1-\lambda)z$. We then multiply both sides by $c^\intercal$ on the left.

Proof Sketch

P has an extreme point $\implies$ P has no line. Q the set of optimal points has no line ($Q \subseteq P$), so it has an extreme point. Let x^* be an extreme point of Q. We want to show that it is an extreme point of P. We assume it is not, and write it as $x^* = \lambda y + (1 - \lambda)z$. We then multiply both sides by $c^\intercal$ on the left. Some reasoning gives us that y and z are also optimal points.

Proof Sketch

P has an extreme point $\implies$ P has no line. Q the set of optimal points has no line ($Q \subseteq P$), so it has an extreme point. Let x^* be an extreme point of Q. We want to show that it is an extreme point of P. We assume it is not, and write it as $x^* = \lambda y + (1 - \lambda)z$. We then multiply both sides by $c^\intercal$ on the left. Some reasoning gives us that y and z are also optimal points. But then x^* is not an extreme point of Q. Absurd.

Proof Sketch

P has an extreme point $\implies$ P has no line. Q the set of optimal points has no line ($Q \subseteq P$), so it has an extreme point. Let x^* be an extreme point of Q. We want to show that it is an extreme point of P. We assume it is not, and write it as $x^* = \lambda y + (1 - \lambda)z$. We then multiply both sides by $c^\intercal$ on the left. Some reasoning gives us that y and z are also optimal points. But then x^* is not an extreme point of Q. Absurd. So x^* is an extreme point of P.

Algorithm for solving LPs

With this theorem, we have can create an algorithm to solve LPs. How?

Algorithm for solving LPs

With this theorem, we have can create an algorithm to solve LPs.
How? Need to check all vertices (an exponential number).

Simplex Motivation

How can this algorithm be improved?

Simplex Motivation

How can this algorithm be improved?

Linear programming is a convex optimization problem.

Simplex Motivation

How can this algorithm be improved?

Linear programming is a convex optimization problem. Local optimality $\implies$ Global optimality.

Simplex Motivation

How can this algorithm be improved?

Linear programming is a convex optimization problem. Local optimality $\implies$ Global optimality. We only need to visit vertices that improve the current solution.

Simplex Motivation

How can this algorithm be improved?

Linear programming is a convex optimization problem. Local optimality $\implies$ Global optimality. We only need to visit vertices that improve the current solution.

João, say stuff about Dantzig, Von Neumann, and WWII.

Simplex visualization

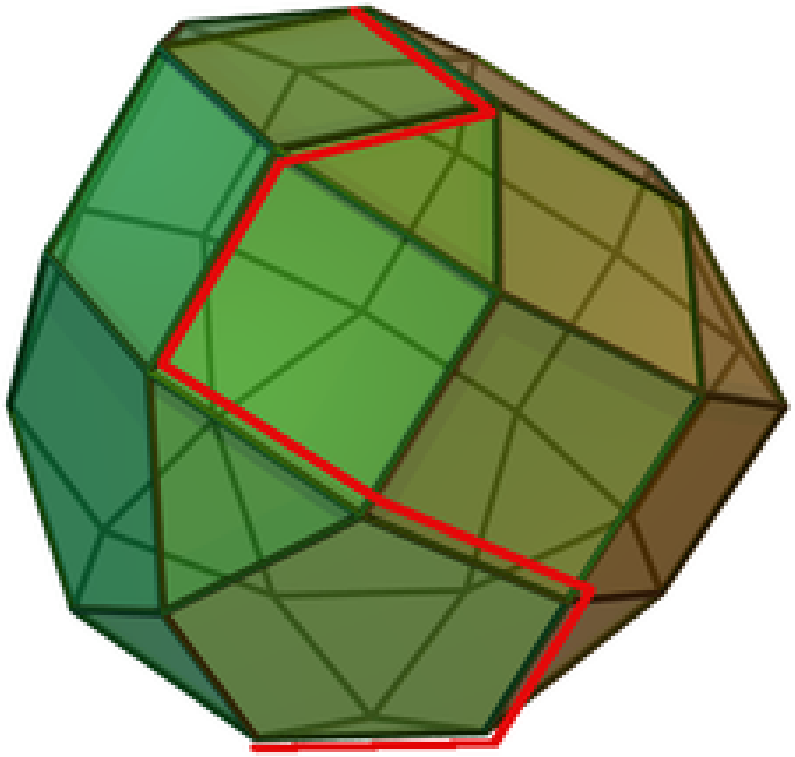


Figure: Example of simplex - only moves forward

Simplex Complexity

For the vast majority of problems, the Simplex Method runs in polynomial time, but so-called pathological examples have been found that ensure that the Simplex Method has to visit an exponential number of vertices.

IPM Motivation

Can be used on general NLPs.

$$\begin{aligned} \min_{x} \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & c_i(x) \geq 0, i = 1, \ldots, m \end{aligned}$$

IPM Motivation

We replace the constraints with what is called a *barrier function*, most commonly a logarithm, to discourage solutions close to the border of the feasible region.

$$\min_x \quad c^\intercal x - \mu \sum_{i=1}^{m} log(c_i(x))$$

Successive iterations decrease the value of μ.

IPM Motivation

We replace the constraints with what is called a *barrier function*, most commonly a logarithm, to discourage solutions close to the border of the feasible region.

$$\min_x \quad c^\intercal x - \mu \sum_{i=1}^{m} log(c_i(x))$$

Successive iterations decrease the value of μ. IPM has a

polynomial running time ($O(n^{3.5} log(1/\varepsilon))$, in fact).

IPM visualization

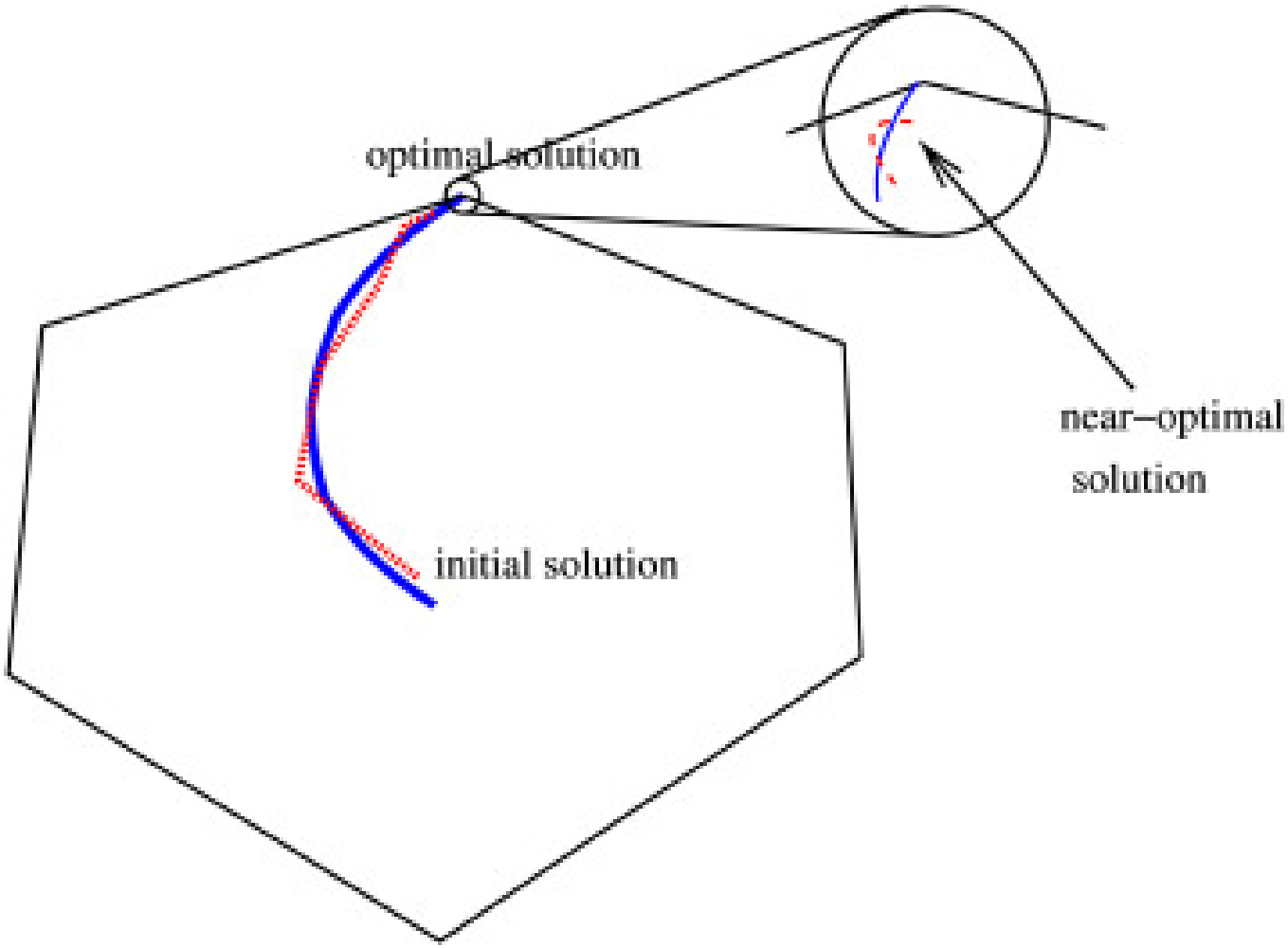
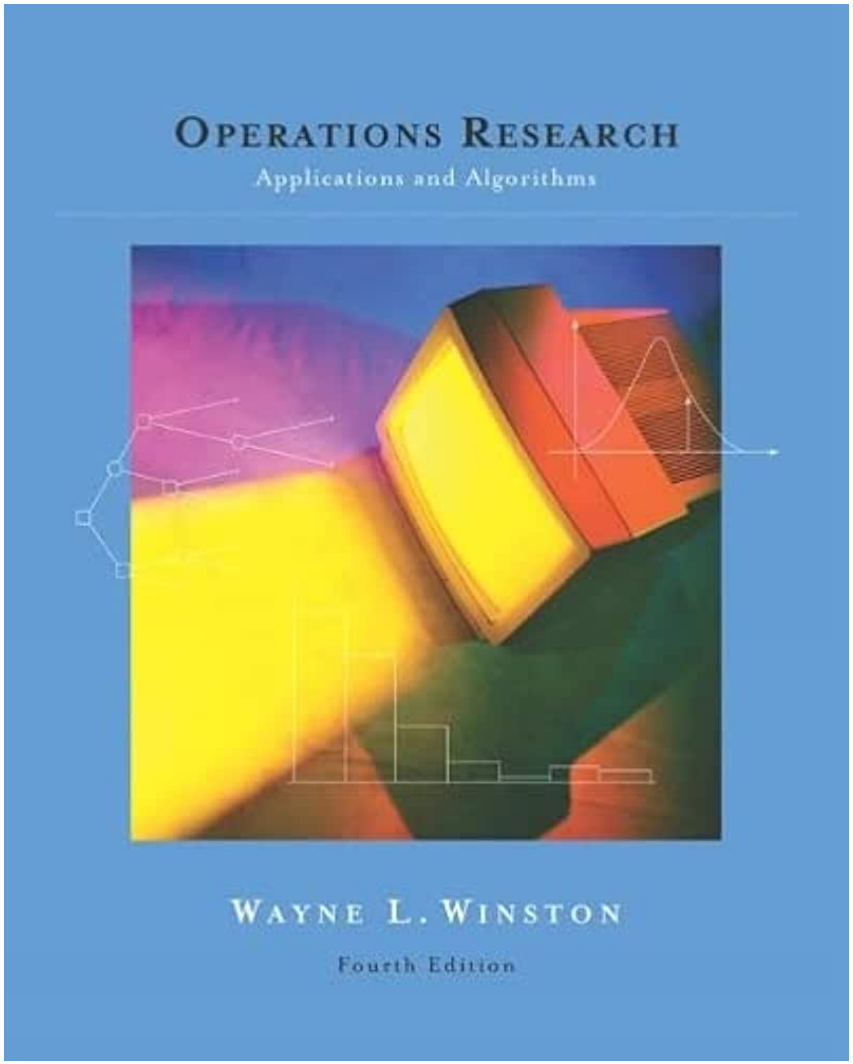


Figure: Example of IPM iterations

Bibliography



Exercises

Big-O

Definition

Given functions $f, g : \mathbb{R}^n \to \mathbb{R}$, we say that $f \in O(g(x))$ if

$$\forall x \geq x_0, |f(x)| \leq Mg(x)$$

Algorithmic complexity

- Writing every number from 1 to n requires us to write $O(n)$ numbers. What about 1 to $2n$?
- Writing every element of the power set of size n require us to write $O(2^n)$ numbers. What about $2n$?

Complexity Classes

Skipping over a lot of details, (time) complexity classes are sets of problems characterized by the difficulty (time) of solving them.
E.g.: **P**, **EXP**, ...

P and NP

P is the set of problems for which an algorithm that runs in polynomial time solves it. **NP** is the set of problems for which an algorithm that runs in polynomial time can verify if a given candidate solution is indeed a solution.

Min Vertex-Cover

Given a graph $G(V, E)$ find the minimum number of vertices that cover all edges

Min Vertex-Cover

Given a graph $G(V, E)$ find the minimum number of vertices that cover all edges

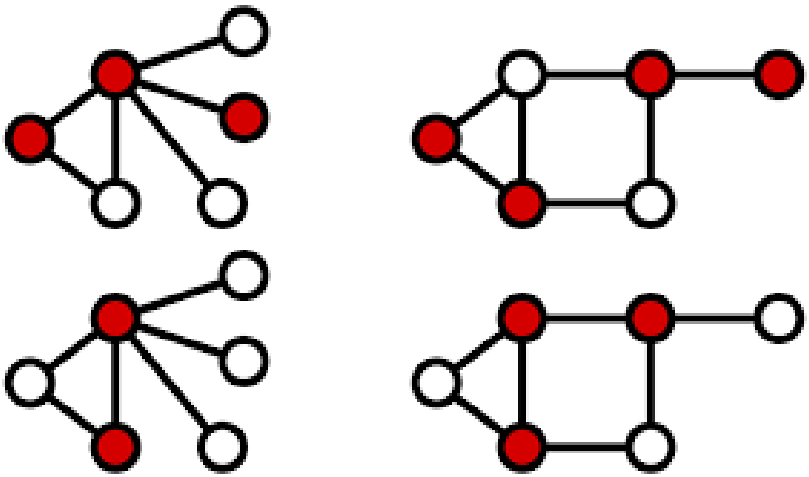


Figure: Example of Vertex Cover solutions

Min Vertex-Cover formulation

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad & x_u + x_v \geq 1, \forall (uv) \in E \\ & x_v \in \{0, 1\} \end{aligned}$$

PySCIPOpt

Max Knapsack

Given items with value and weight, fit them into a bag such that the total weight does not exceed W and the value is maximized.

Max Knapsack

Given items with value and weight, fit them into a bag such that the total weight does not exceed W and the value is maximized.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0, 1\} \end{aligned}$$

Traveling-Salesman

Given a graph $G(V, E)$ with weighted edges, find the least costly Hamiltonian cycle.

Traveling-Salesman

Given a graph $G(V, E)$ with weighted edges, find the least costly Hamiltonian cycle.

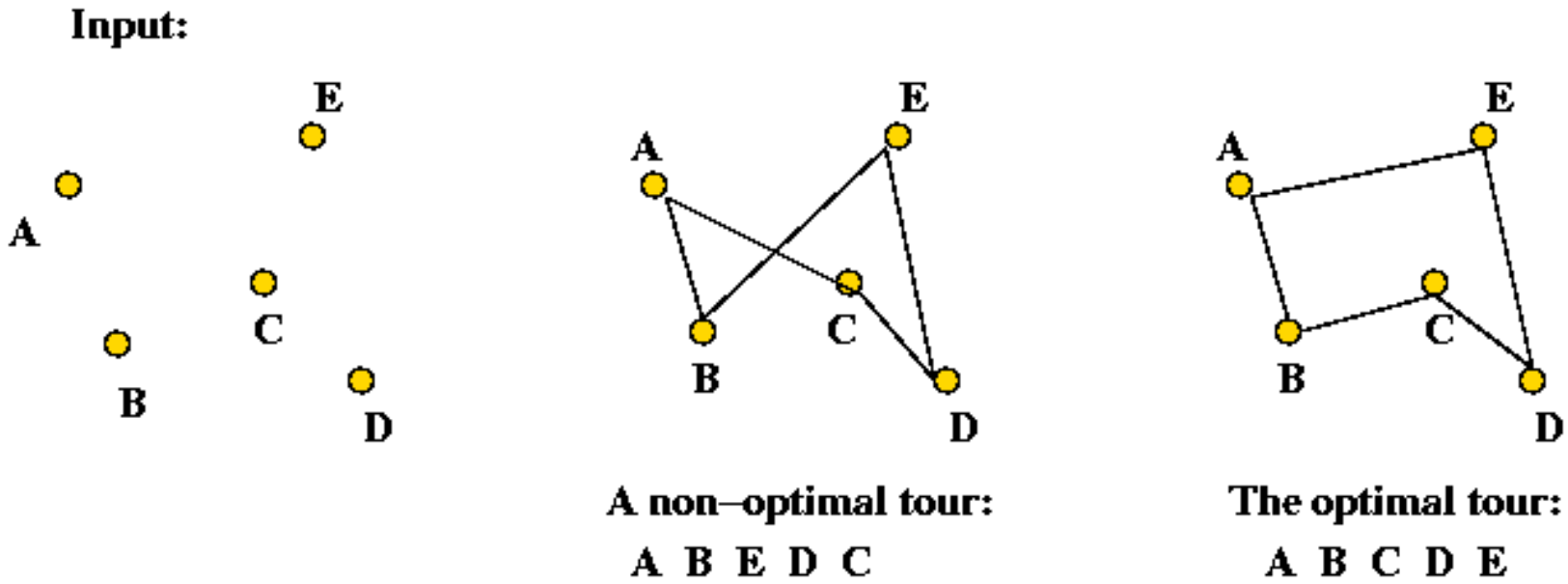


Figure: TSP example

TSP formulation

$$\min \quad \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{i \neq j, i=1}^{n} x_{ij} = 1, j \in [n]$$

$$\sum_{i \neq j, j=1}^{n} x_{ij} = 1, i \in [n]$$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1, \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2$$

$$x_{ij} \in \{0, 1\}$$

Cutting-Stock

Minimize the number of sheets of metal that, when cut into smaller sheets, satisfies a demand.

Cutting-Stock

Minimize the number of sheets of metal that, when cut into smaller sheets, satisfies a demand.

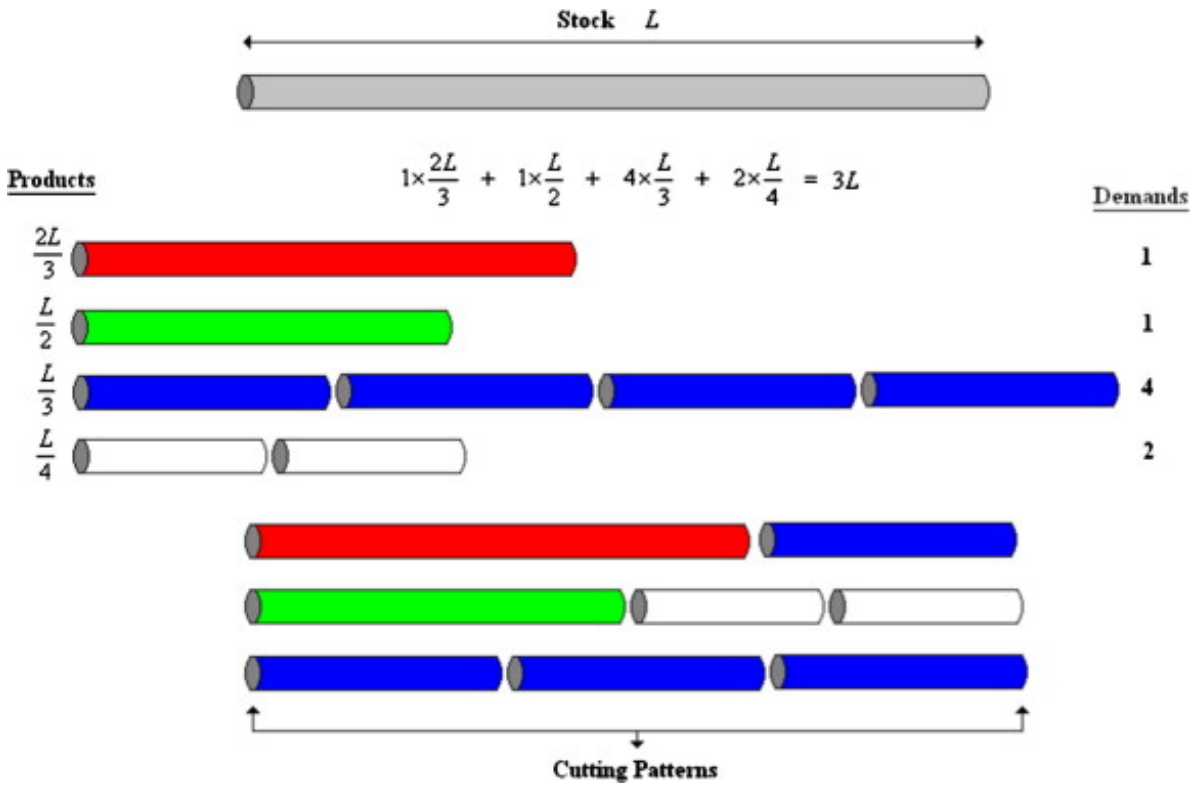


Figure: Cutting stock example

Cutting-Stock Formulation

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{M} y_j & & \\
\text{s.t.} \quad & \sum_{j=1}^{M} x_{ij} = d_i, & & i \in [n] \\
& \sum_{i=1}^{n} l_i x_{ij} \leq L, & & j \in [m] \\
& x_{ij} \leq d_i y_j, & & i \in [n], j \in [m] \\
& x_{ij} \in \mathbb{Z}^+, & & i \in [n], j \in [m] \\
& y_j \in \{0, 1\}, & & j \in [m]
\end{aligned}
$$

Reductions

Knowing the solution to some optimization problems can give us the solution to other optimization problems.

Reductions

Knowing the solution to some optimization problems can give us the solution to other optimization problems.

1. Vehicle routing and TSP

Reductions

Knowing the solution to some optimization problems can give us the solution to other optimization problems.

1. Vehicle routing and TSP
2. Min Vertex Cover and Max Independent set

Reductions

Knowing the solution to some optimization problems can give us the solution to other optimization problems.

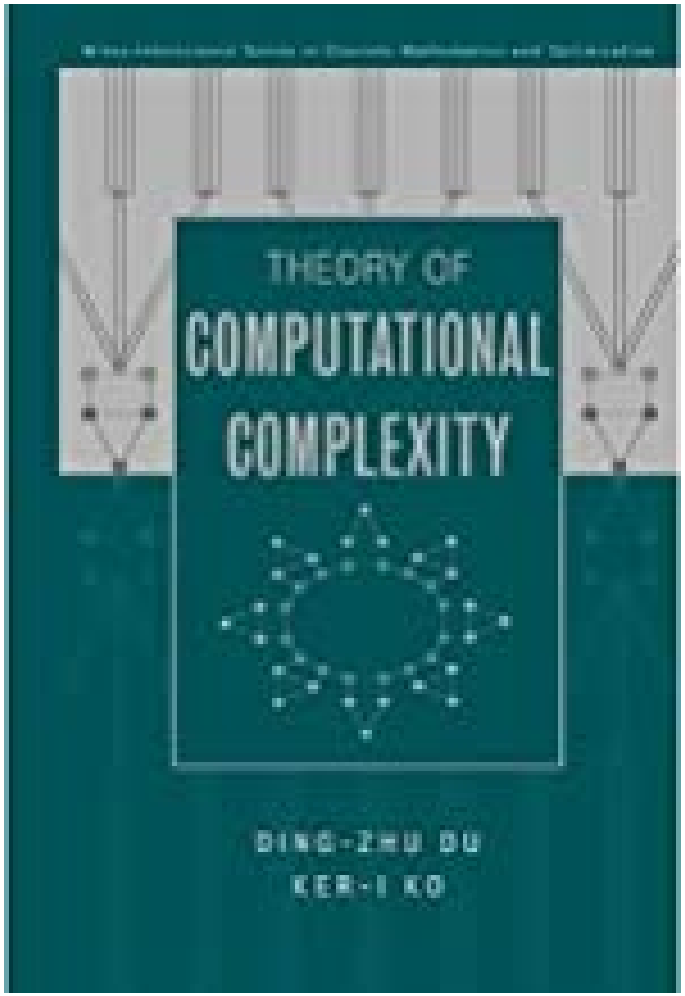
1. Vehicle routing and TSP
2. Min Vertex Cover and Max Independent set
3. Cutting Stock and Bin packing

Reductions

Knowing the solution to some optimization problems can give us the solution to other optimization problems.

1. Vehicle routing and TSP
2. Min Vertex Cover and Max Independent set
3. Cutting Stock and Bin packing
4. Knapsack and Cutting Stock

Bibliography



Exercises

$$
\begin{aligned}
\min_{x,y} \quad & c^{\mathsf{T}}(xy)^{\mathsf{T}} \\
\text{s.t.} \quad & f(x,y) \leq 0 \\
& x \geq 0 \\
& y \in \mathbb{Z}
\end{aligned}
$$

Innumerous applications

- Health Industry (Kidney Exchange, INEM stuff)
- Scheduling (Sports, Classes)
- Game Theory (Chess, Go, Economy)

Non-convex

Integer programming is not convex. Which strategies can we employ to solve it?

Non-convex

Integer programming is not convex. Which strategies can we employ to solve it?

Explicit enumeration?

Non-convex

Integer programming is not convex. Which strategies can we employ to solve it?

Explicit enumeration? Generally out of the question. An exponential number of solutions.

Non-convex

Integer programming is not convex. Which strategies can we employ to solve it?

Explicit enumeration? Generally out of the question. An exponential number of solutions.

Pick integer point closest to the optimum for the LP?

Non-convex

Integer programming is not convex. Which strategies can we employ to solve it?

Explicit enumeration? Generally out of the question. An exponential number of solutions.

Pick integer point closest to the optimum for the LP? They can be arbitrarily far away.

LP vs. IP example

$$\begin{aligned} \max_{x,y} \quad & 10x + y \\ \text{s.t.} \quad & y \leq 5x + 4 \\ & x, y \in \mathbb{N}_0 \end{aligned}$$

LP vs. IP example

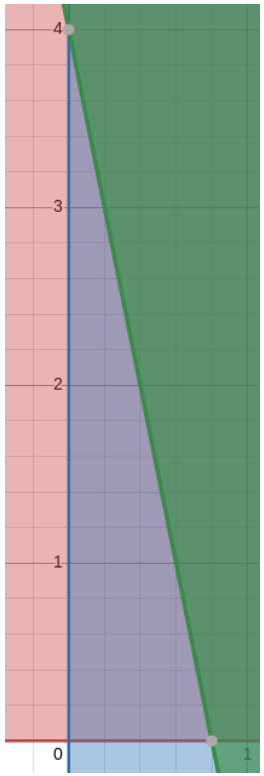


Figure: Feasible region

LP vs. IP example

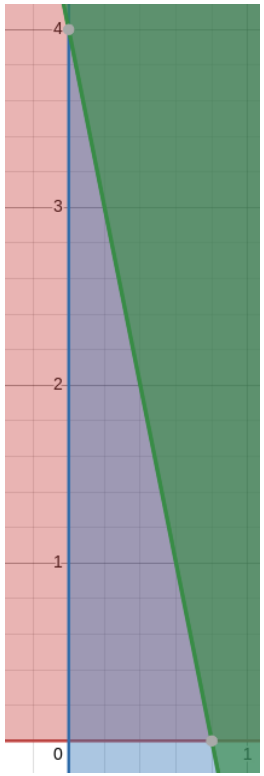


Figure: Feasible region

The optimal solution is $(0, 4)$. With $x, y \geq 0$ instead, optimal solution is $(\frac{4}{5}, 0)$.

Relaxations

$$\begin{aligned} \max_{x} \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & Ax \leq b \\ & Dx \leq e \end{aligned}$$

Relaxations

$$\begin{aligned} \max_{x} \quad & c^{\intercal}x \\ \text{s.t.} \quad & Ax \leq b \\ & Dx \leq e \end{aligned}$$

$$\begin{aligned} \max_{x} \quad & c^{\intercal}x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

Relaxations

$$\begin{aligned} \max_{x} \quad & c^{\intercal}x \\ \text{s.t.} \quad & Ax \leq b \\ & Dx \leq e \end{aligned}$$

$$\begin{aligned} \max_{x} \quad & c^{\intercal}x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

What can we deduce about these two problems?

Relaxations

$$\begin{aligned} \max_{x} \quad & c^\intercal x \\ \text{s.t.} \quad & Ax \leq b \\ & Dx \leq e \end{aligned}$$

$$\begin{aligned} \max_{x} \quad & c^\intercal x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

What can we deduce about these two problems? The optimal value of the second cannot be worse than that of the first. It provides an upper bound.

Relaxations

Relaxing constraints provides a bound on the optimal solution.
Can this help us solve IPs?

Relaxations

Relaxing constraints provides a bound on the optimal solution. Can this help us solve IPs? Yes! We can easily derive bounds.

$$\begin{aligned} \max_{x} \quad & c^\intercal x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z} \end{aligned}$$

$$\begin{aligned} \max_{x} \quad & c^\intercal x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

Branching

We can fix a binary variable x to 0 and 1, and solve the resulting two subproblems. The best optimal value will be the optimal value of the problem. This is called **branching**.

Branching

We can fix a binary variable x to 0 and 1, and solve the resulting two subproblems. The best optimal value will be the optimal value of the problem. This is called **branching**.

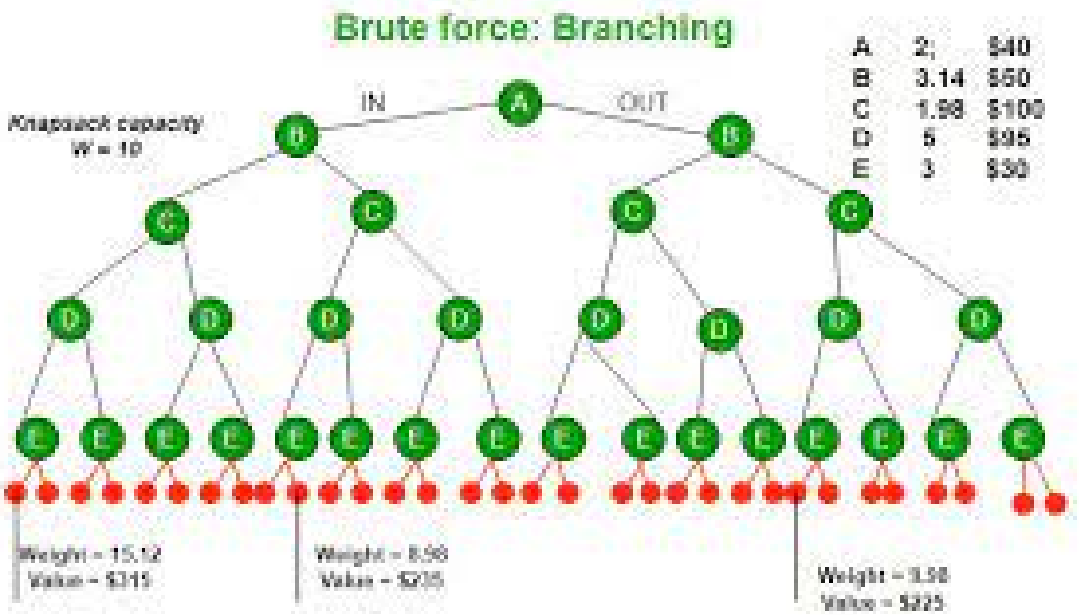


Figure: Branching on all variables (explicit enumeration)

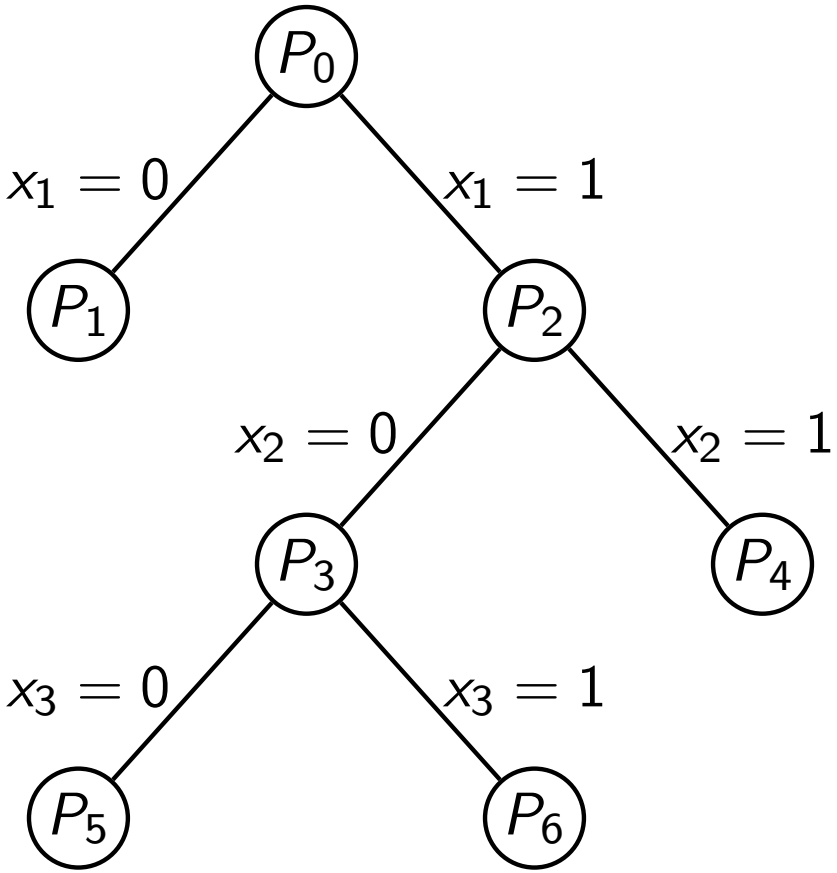
Branch-and-Bound Idea

- Pick a variable and branch on it. Creates problems P_1, P_2
- If the linear relaxation of P_1 is worse than our best solution, ignore it
- If it is better, pick a variable in P_1 and branch on it
- If all variables are integer, save the solution if it's better
- Repeat until all subproblems have been explored

Branch-and-Bound

```
1  L ← {X} // List of unexplored subproblems
2  z̄ = ∞ // Upper bound
3  while L ≠ {} do
4  |   select a subproblem S from L // search strategy
5  |   solve LP relaxation of S, with solution x′ and objective z′
6  |   remove S from L
7  |   if S infeasible or z′ ≥ z̄ then
8  |   |   continue
9  |   if x′ is integer and z′ < z̄ then
10 |   |   z̄ ← z′
11 |   |   x* = x′ // x′ becomes new best solution
12 |   |   continue
13 |   // otherwise S is not pruned, x′ is feasible and fractional
14 |   split S into subproblems S1, S2
15 |   insert S1, S2 into L
16 return x*
```

Branch-and-Bound



Revisiting Knapsack

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0, 1\} \end{aligned}$$

It has an easy LP-relaxation

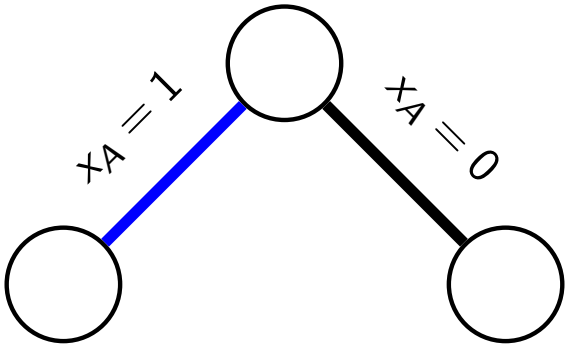
1. Sort items by price density (price/weight)
2. Pick items until capacity is exceeded
3. Remove the excess of the last item

Knapsack instance

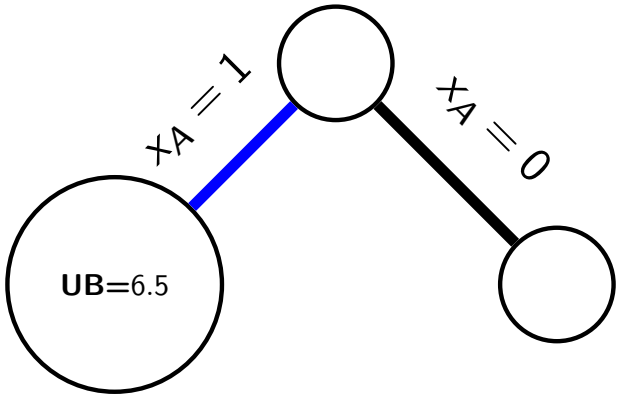
Item	Weight	Value	Value/Weight
A	3	5	1.67
B	2	3	1.5
C	1	1	1

Knapsack capacity: 4

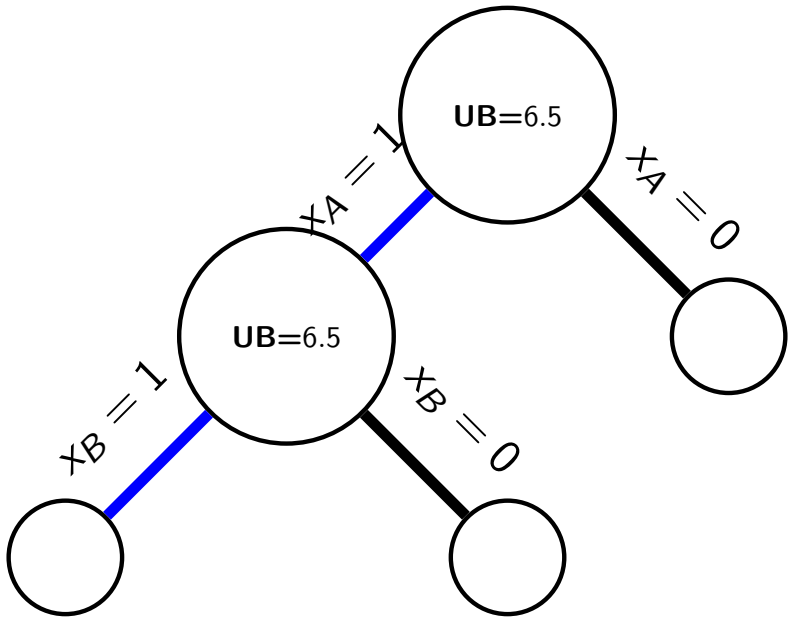
Branch and Bound



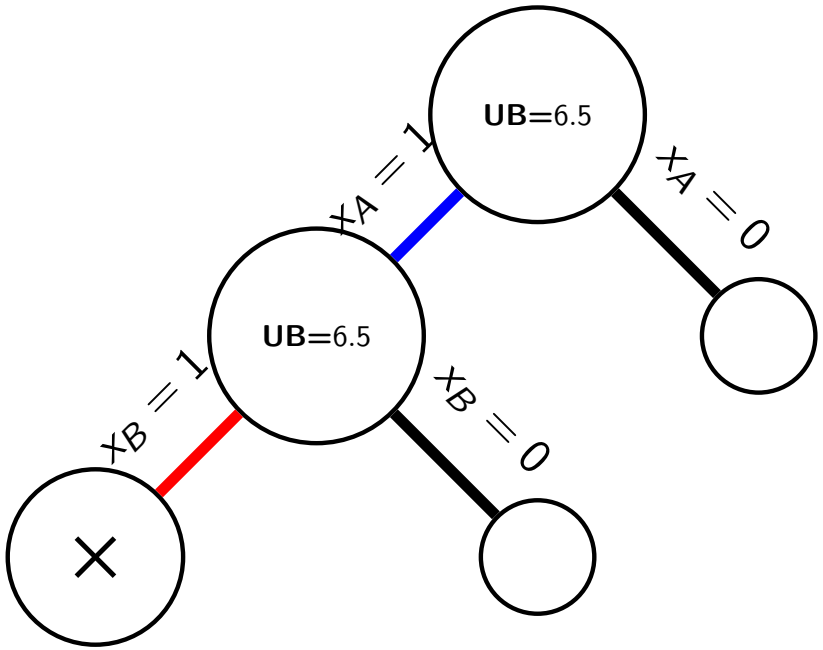
Branch and Bound



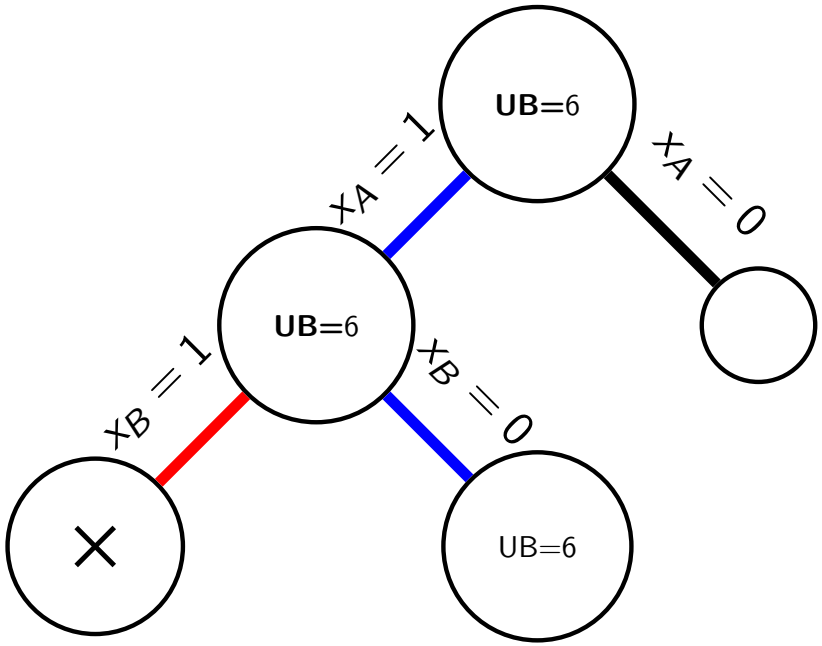
Branch and Bound



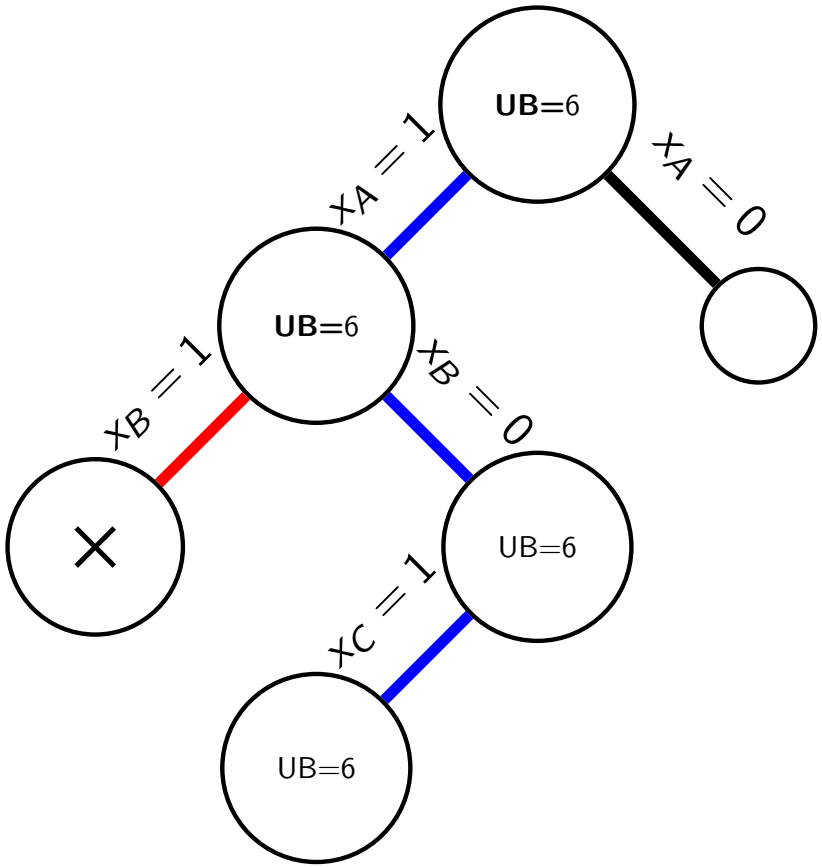
Branch and Bound



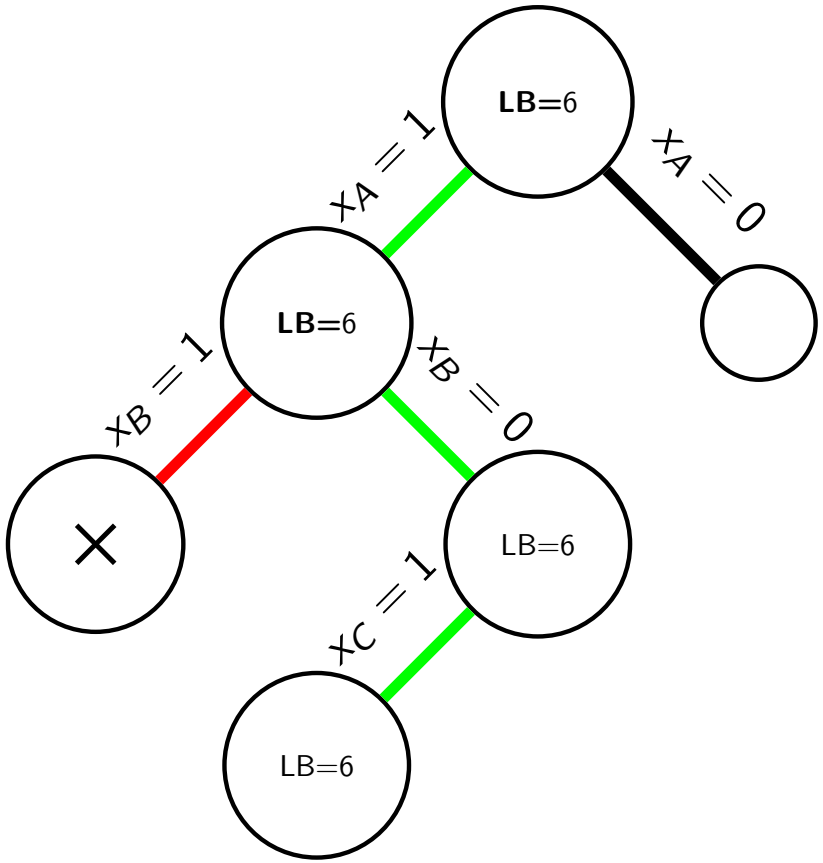
Branch and Bound



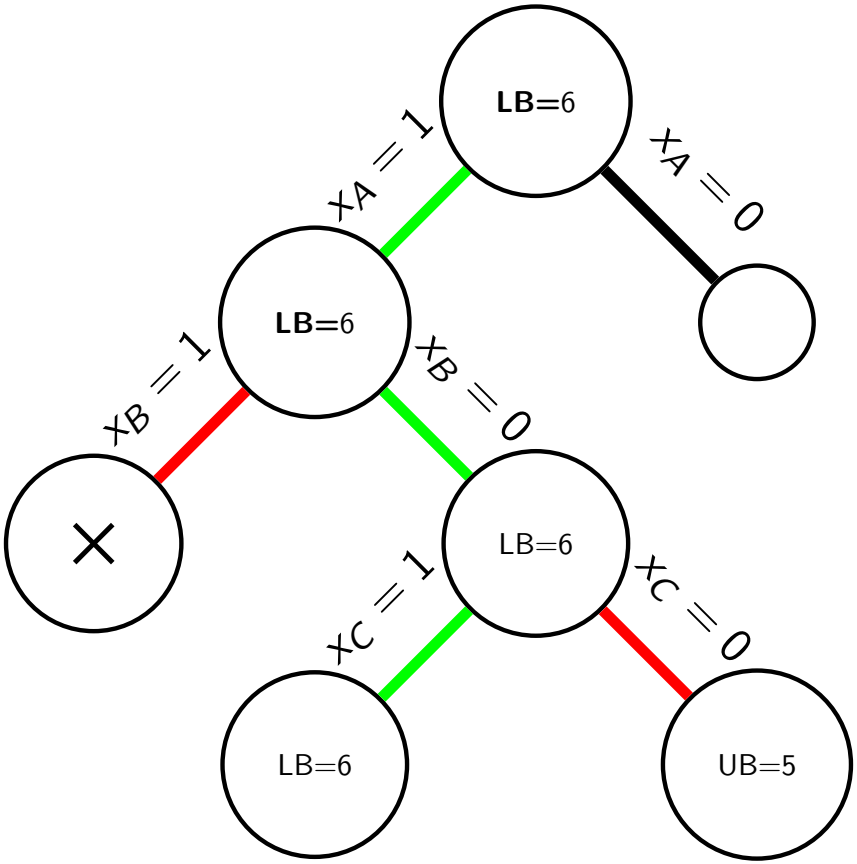
Branch and Bound



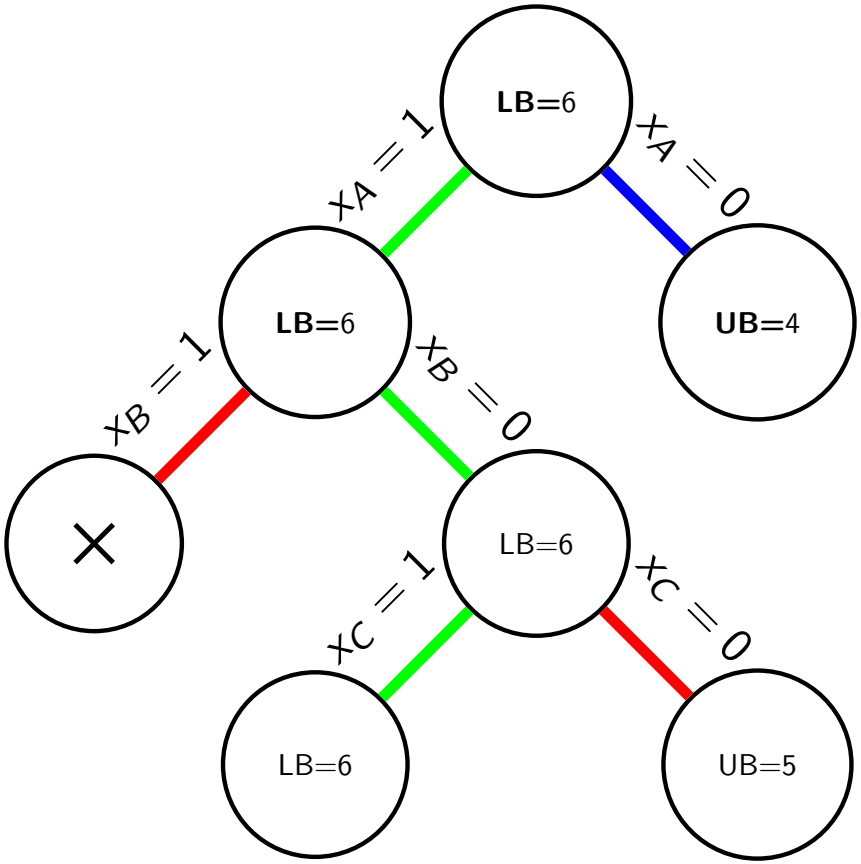
Branch and Bound



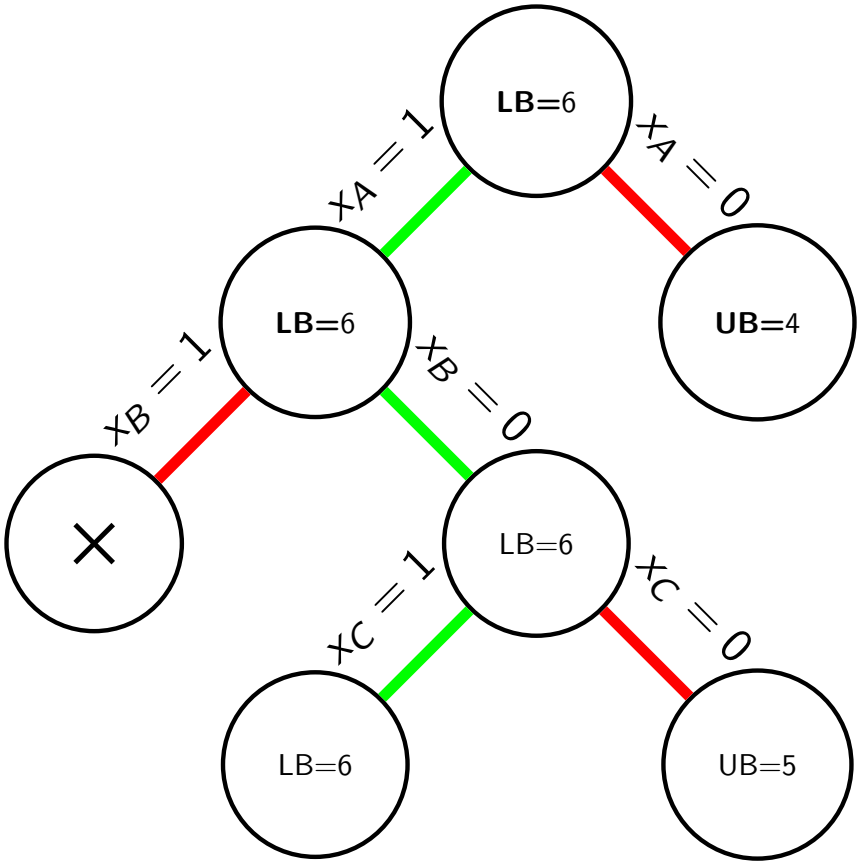
Branch and Bound



Branch and Bound



Branch and Bound



Heuristics

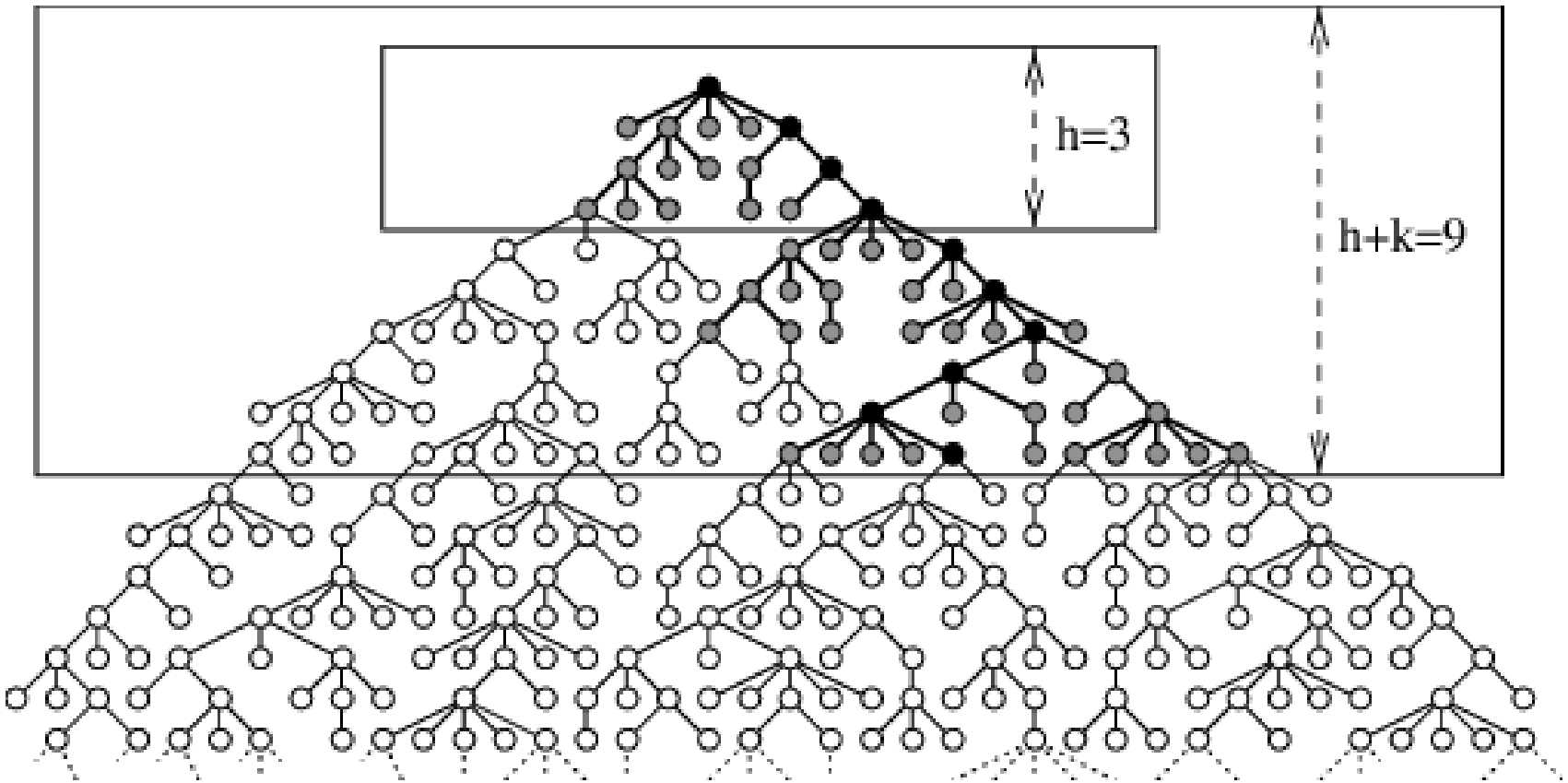


Figure: Branch and Bound trees can get very big. How big?

We will focus on heuristics for the TSP.

$$\min \quad \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$

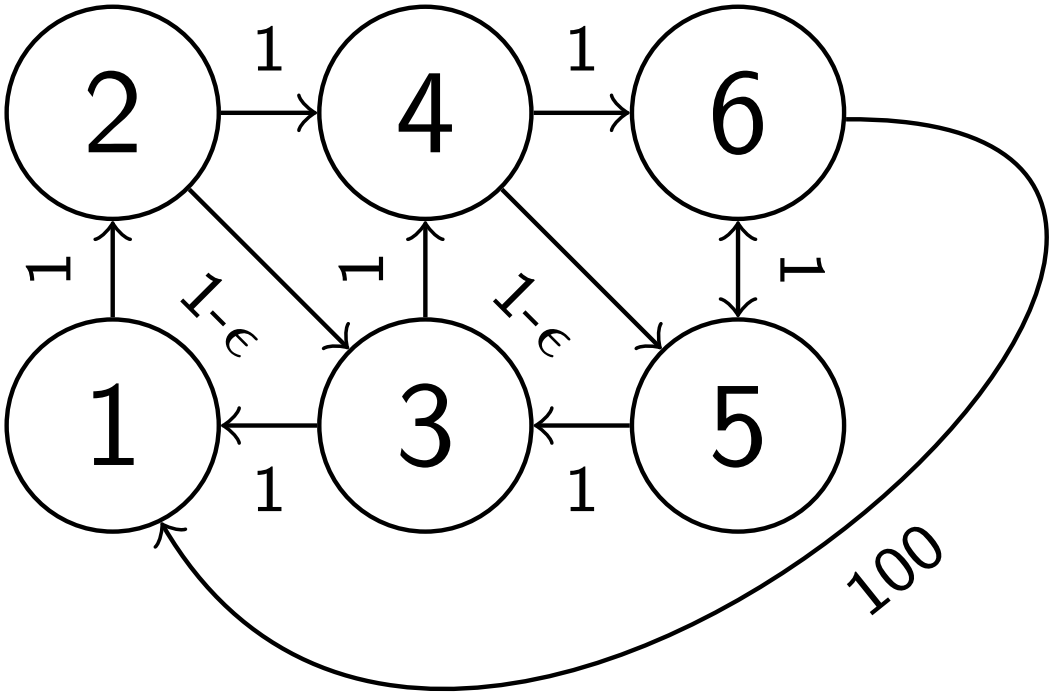
$$\sum_{i \neq j, i=1}^{n} x_{ij} = 1, j \in [n]$$

$$\sum_{i \neq j, j=1}^{n} x_{ij} = 1, i \in [n]$$

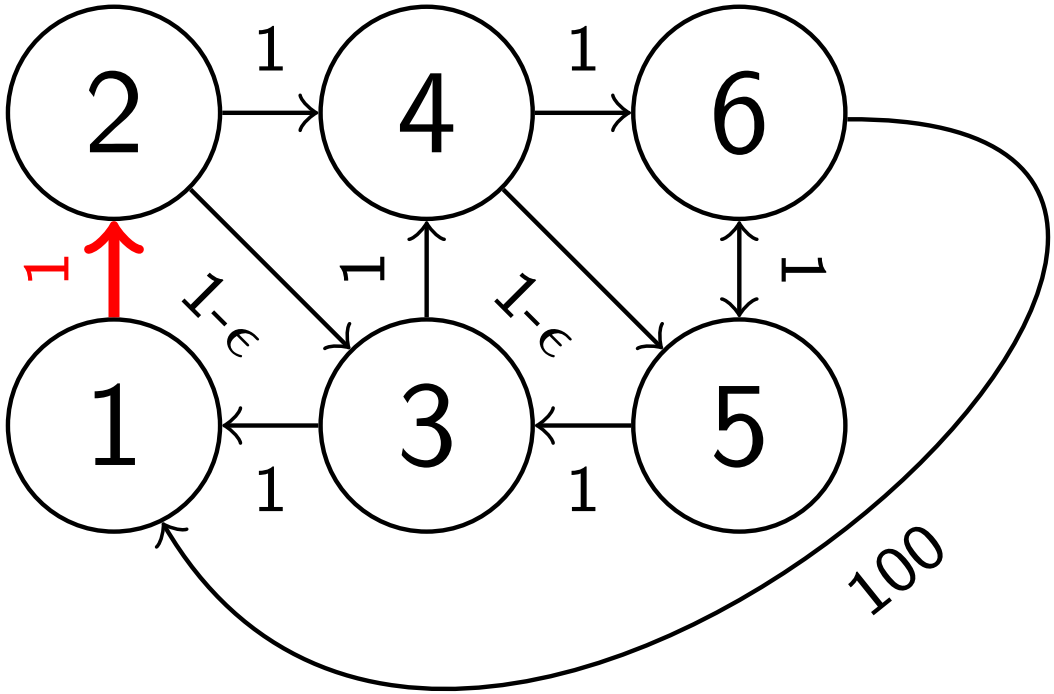
$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1, \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2$$

$$x_{ij} \in \{0, 1\}$$

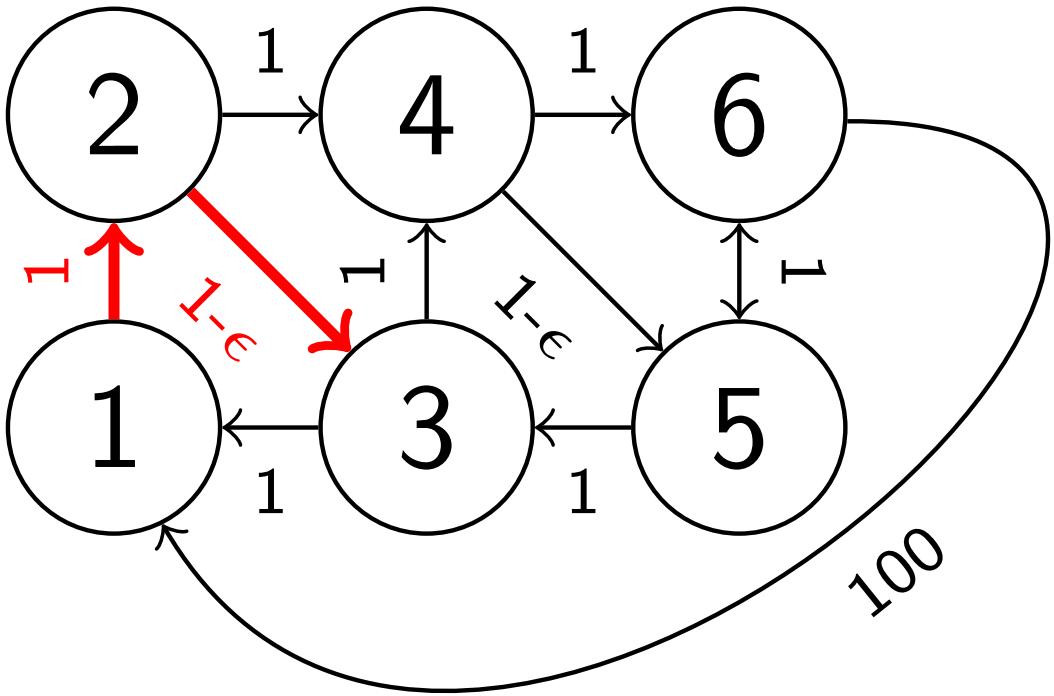
Nearest Neighbor



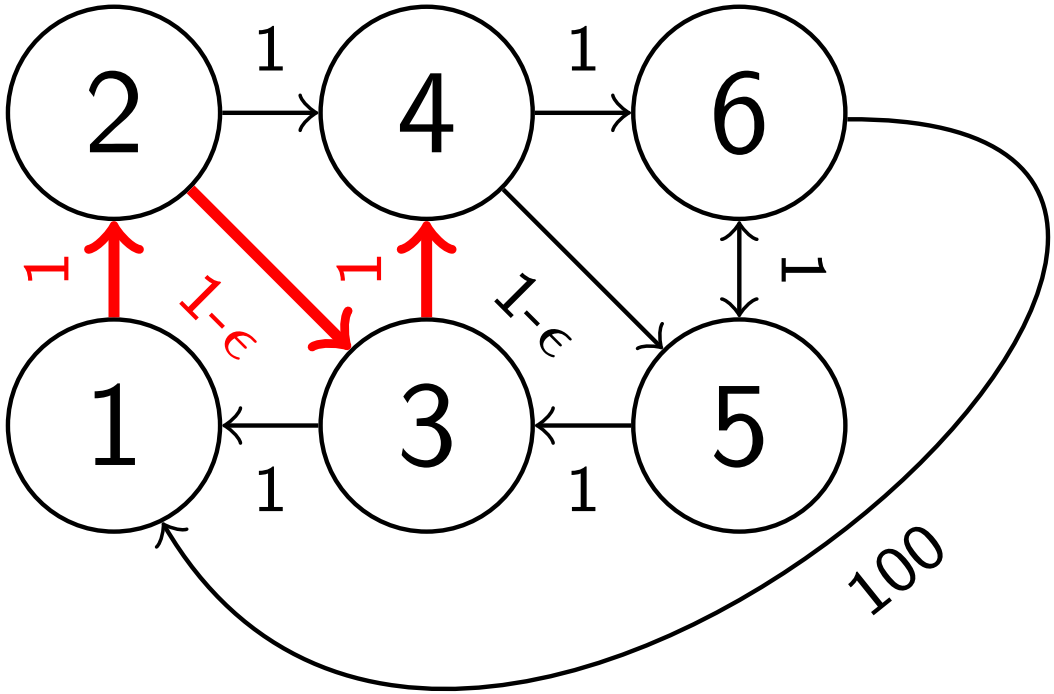
Nearest Neighbor



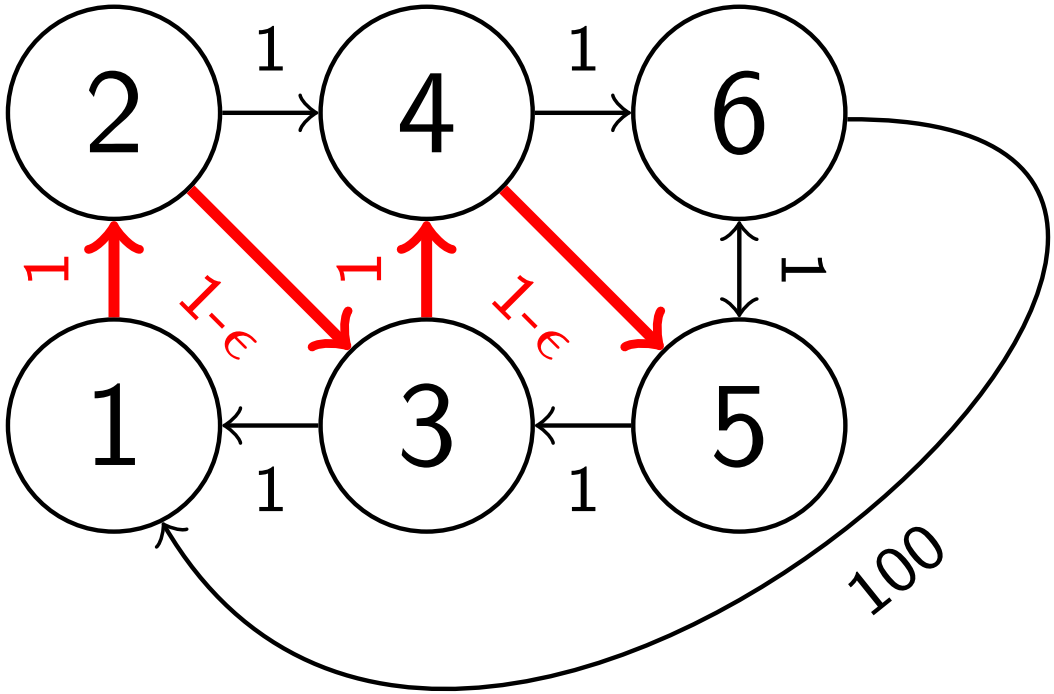
Nearest Neighbor



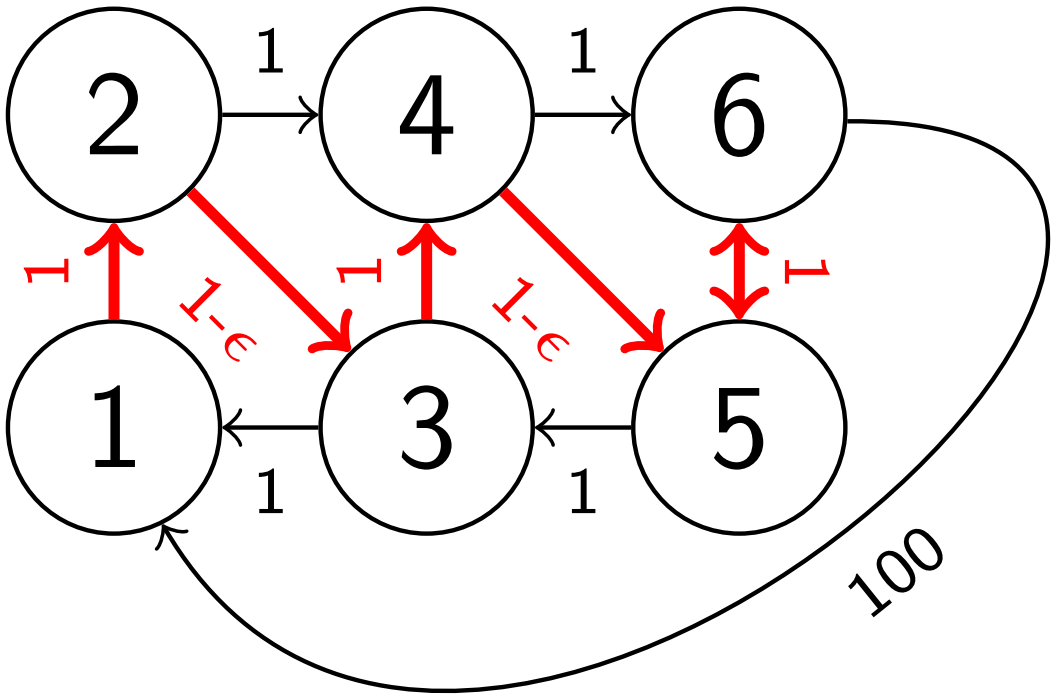
Nearest Neighbor



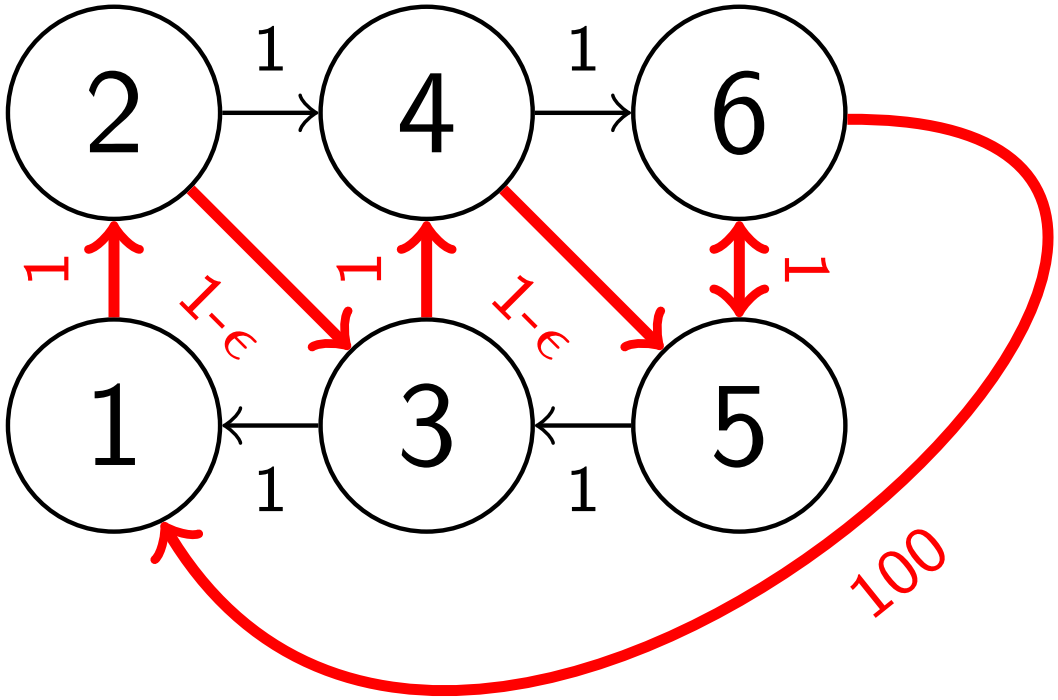
Nearest Neighbor



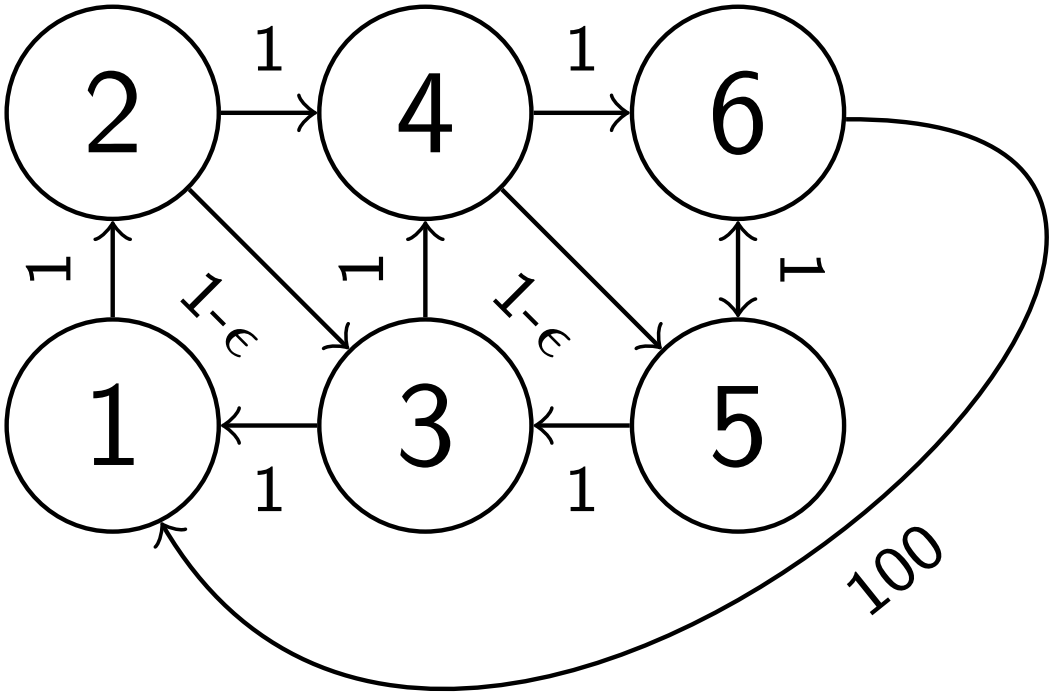
Nearest Neighbor



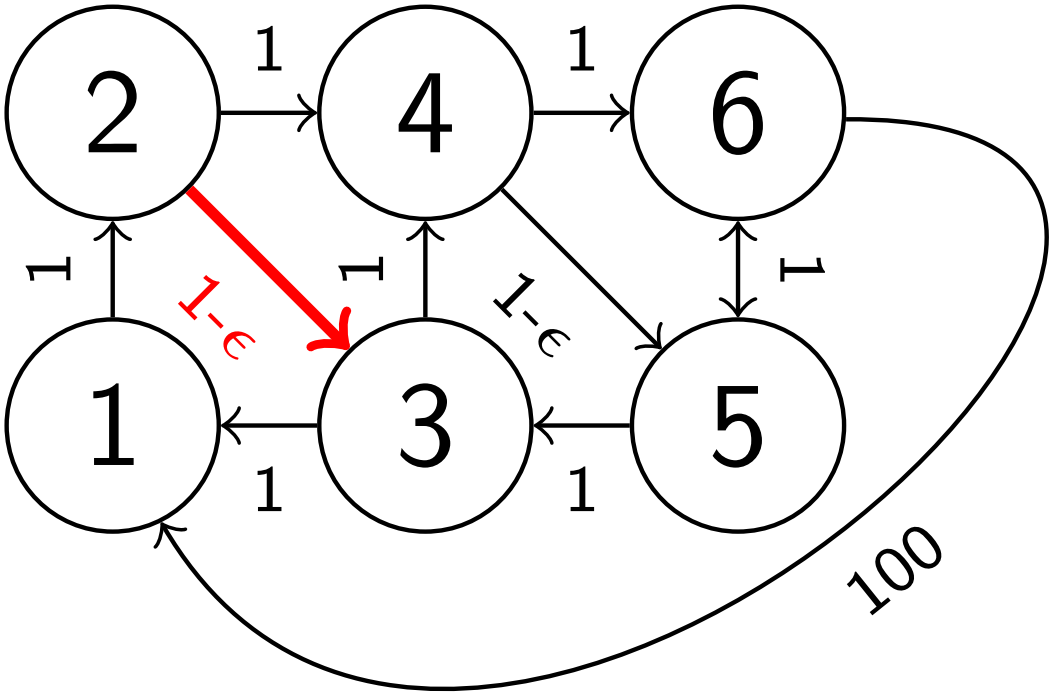
Nearest Neighbor



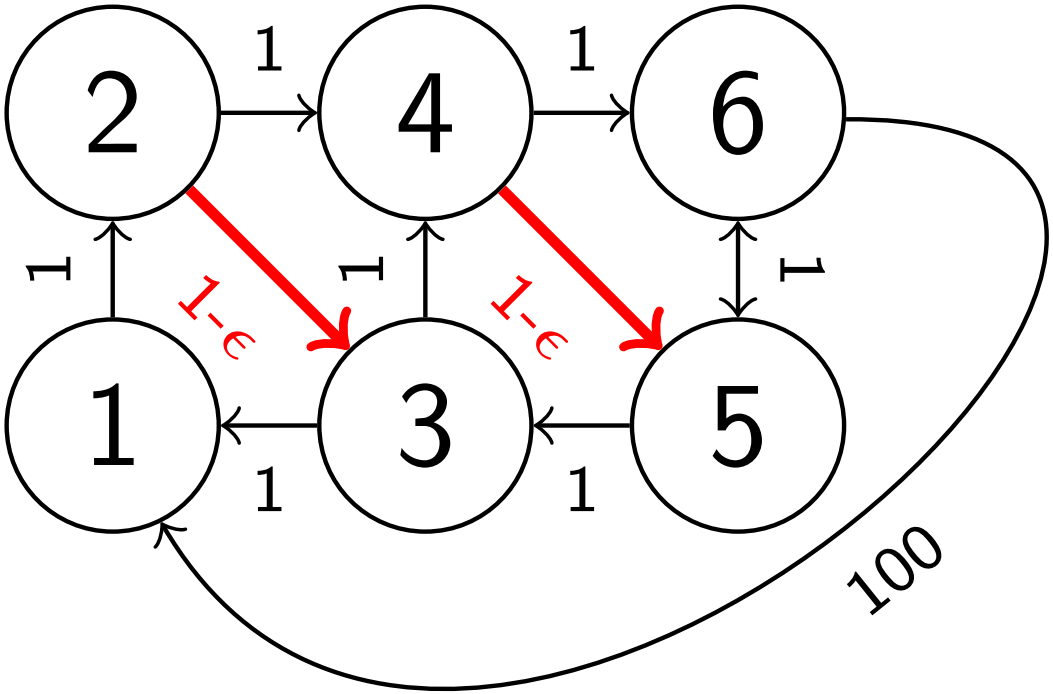
Greedy Algorithm



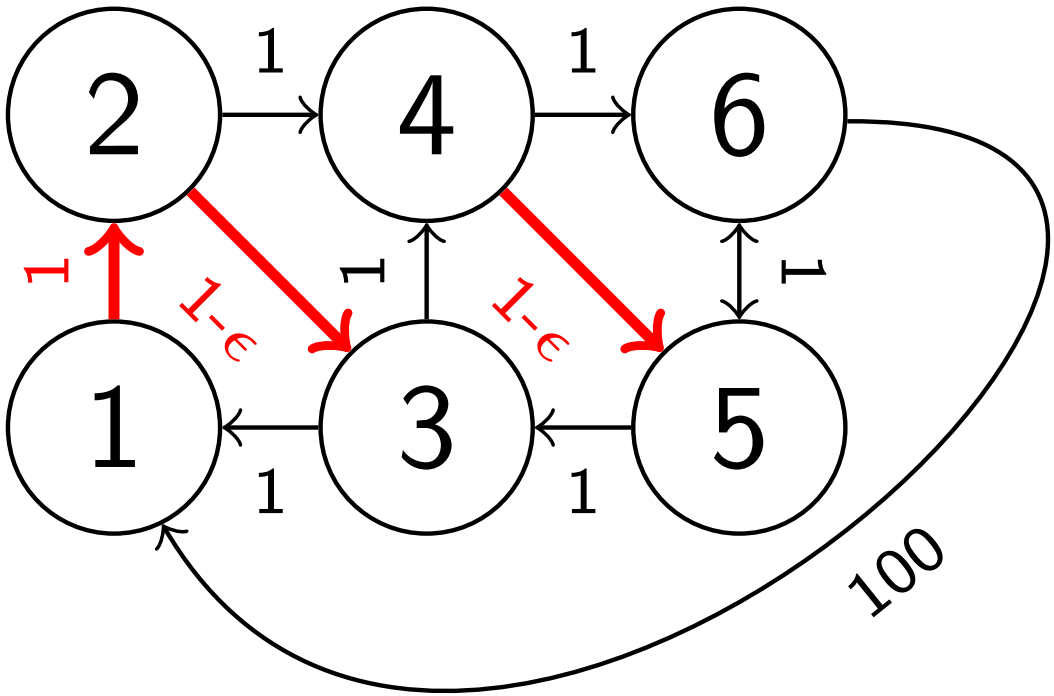
Greedy Algorithm



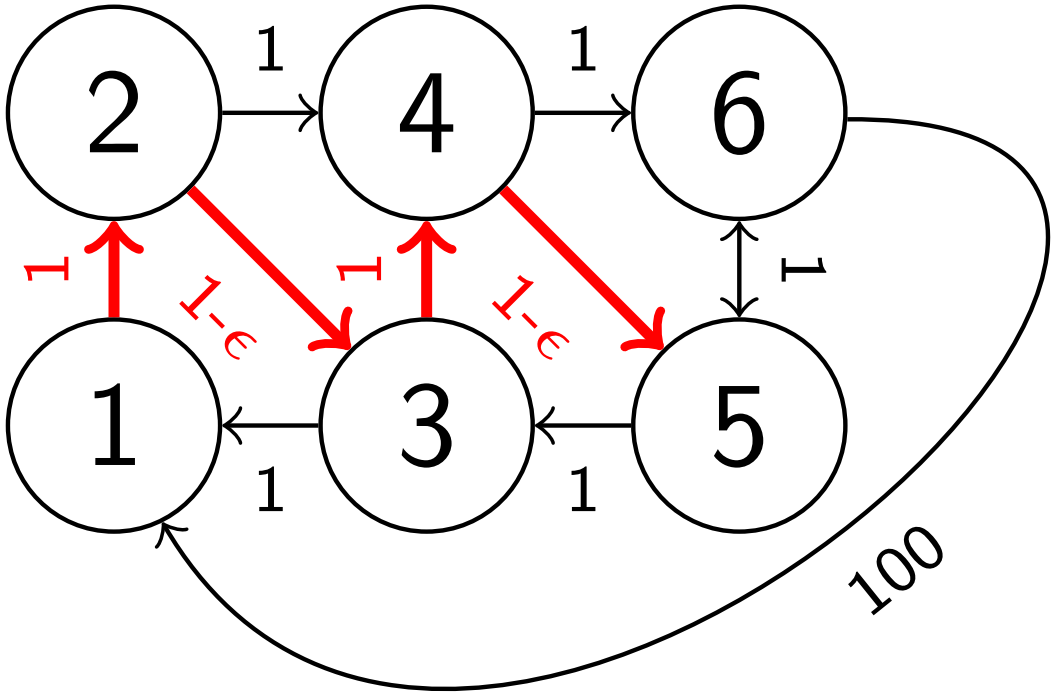
Greedy Algorithm



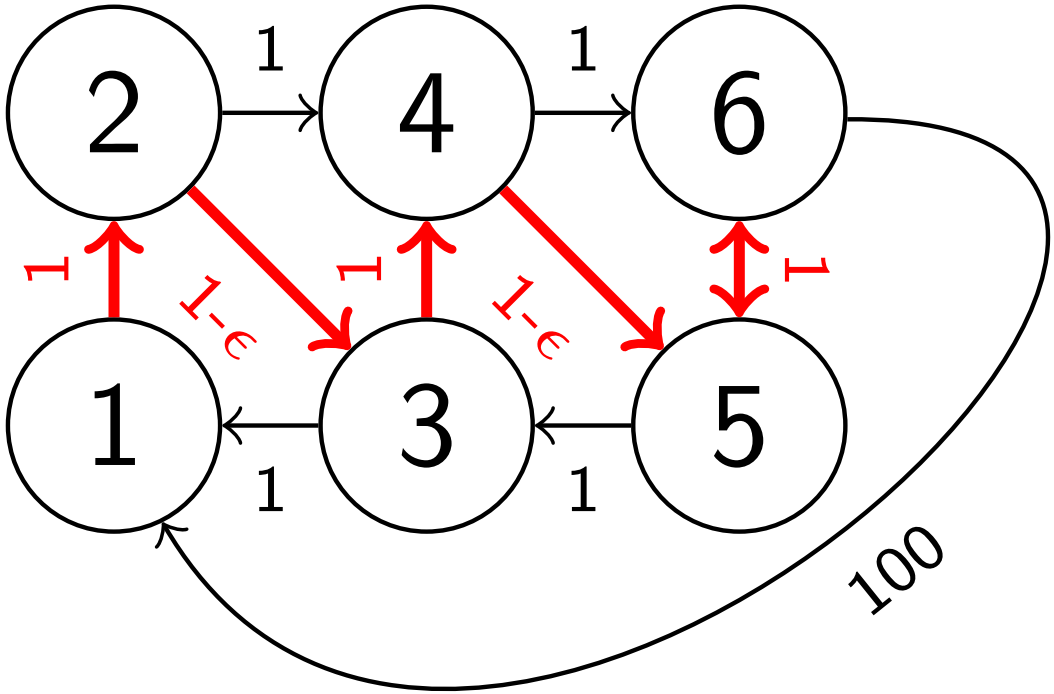
Greedy Algorithm



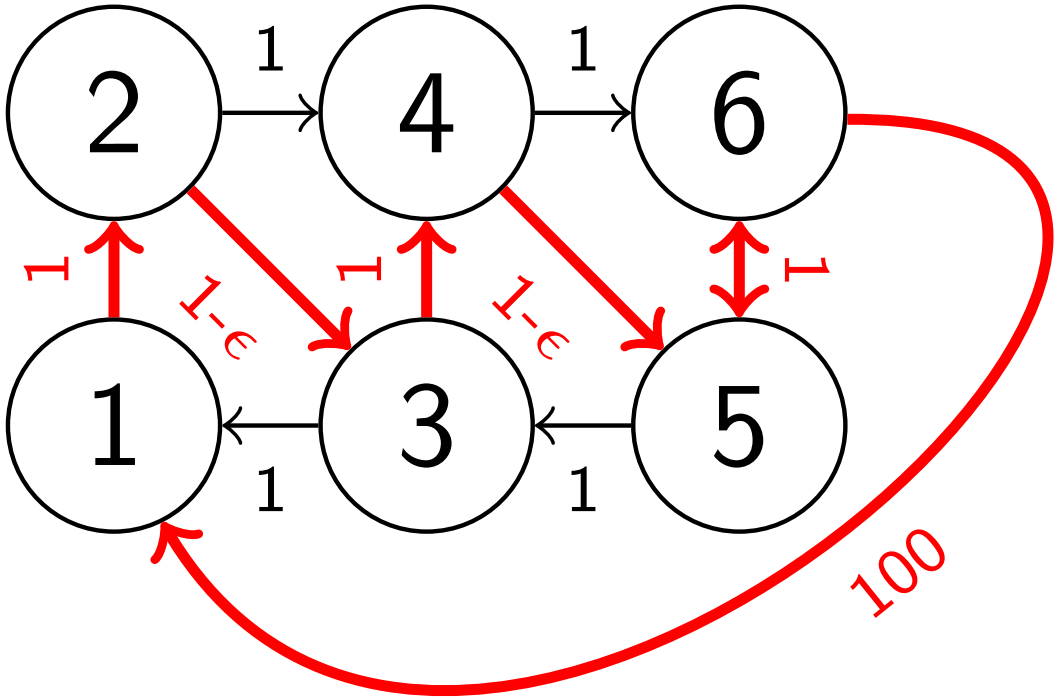
Greedy Algorithm



Greedy Algorithm



Greedy Algorithm



2-OPT algorithm

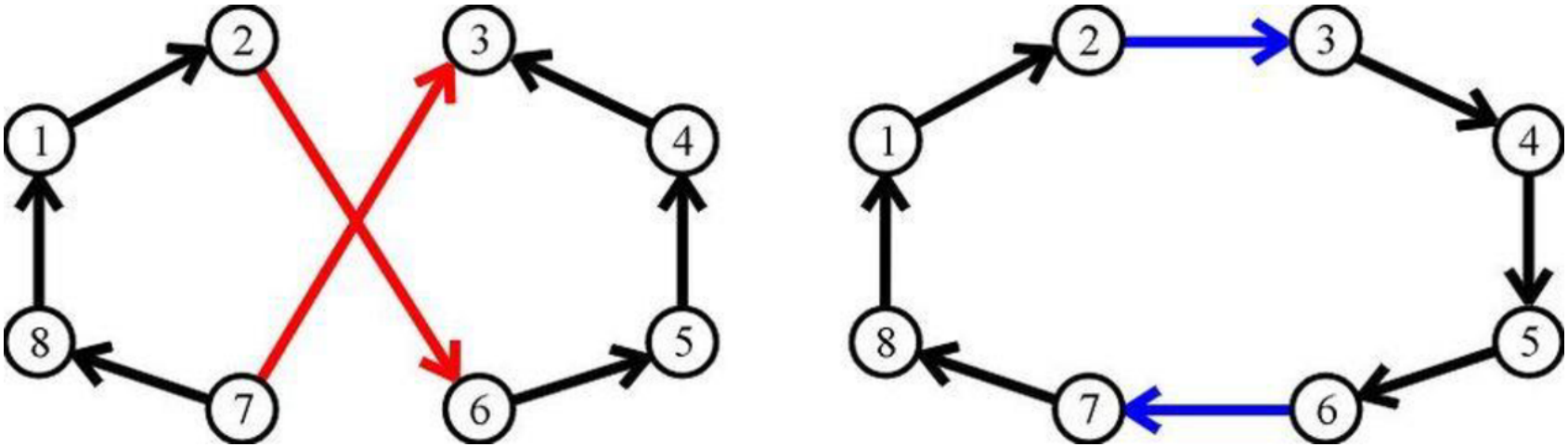


Figure: If two edges cross, we can find a strictly better solution

TSP: 2-opt visualization

Large TSP

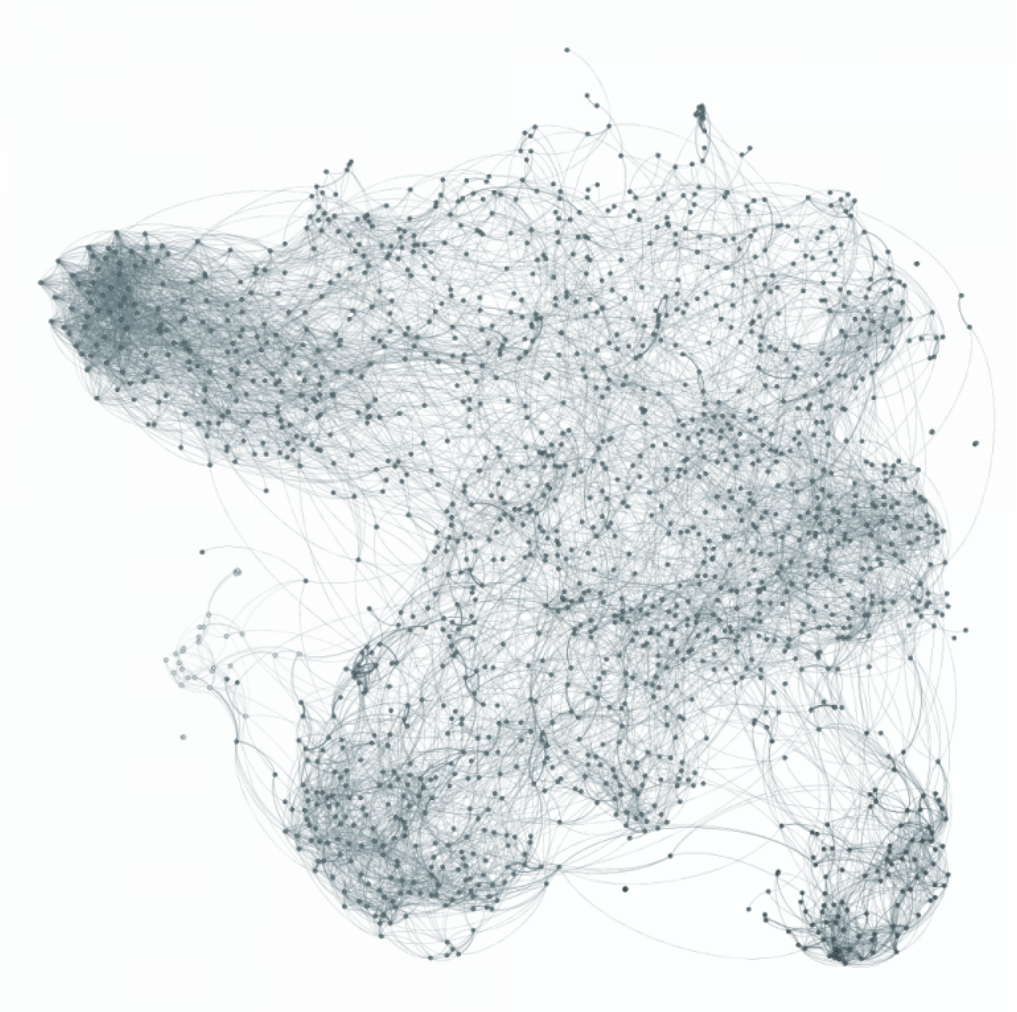


Figure: Heuristics are for large instances

Approximation algorithms

Some heuristics can have theoretical guarantees of their solution quality.

We will study an approximation algorithm for TSP where the cost function satisfies the triangle inequality.

Approximation algorithms

Some heuristics can have theoretical guarantees of their solution quality.

We will study an approximation algorithm for TSP where the cost function satisfies the triangle inequality.

But first, some preliminaries.

Trees

Definition

A **tree** is an undirected graph where any two vertices are connected by exactly one path. Equivalently, it is an undirected graph without cycles.

Definition

A **tree** is an undirected graph where any two vertices are connected by exactly one path. Equivalently, it is an undirected graph without cycles.

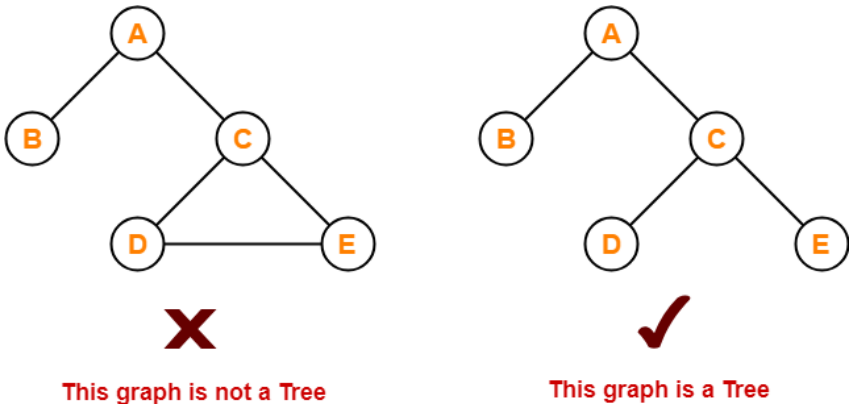


Figure: Example of tree and non-tree

Minimum spanning tree

Definition

A **minimum spanning tree** is a subset of the edges of a graph that connects all vertices, without any cycles, such that total edge weight is minimum.

Minimum spanning tree

Definition

A **minimum spanning tree** is a subset of the edges of a graph that connects all vertices, without any cycles, such that total edge weight is minimum.

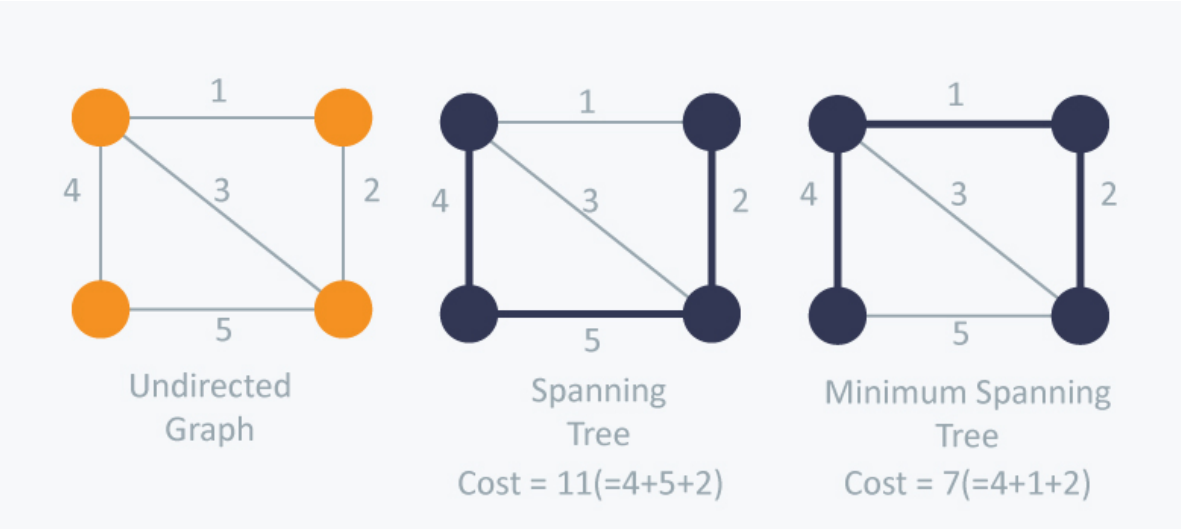


Figure: Spanning Tree vs Minimum spanning tree

Depth-First Search

Depth-First Search is an algorithm for traversing trees.

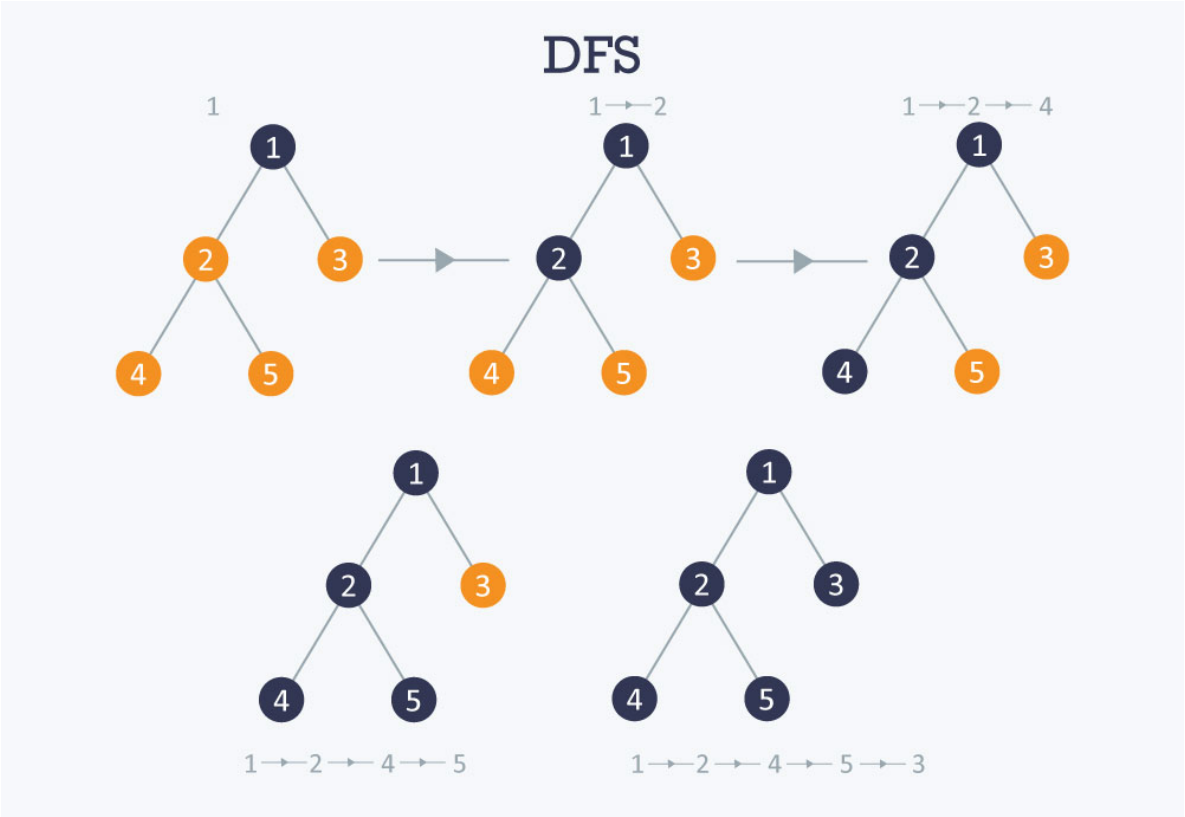


Figure: Example of a Depth-First Search

Minimum spanning tree

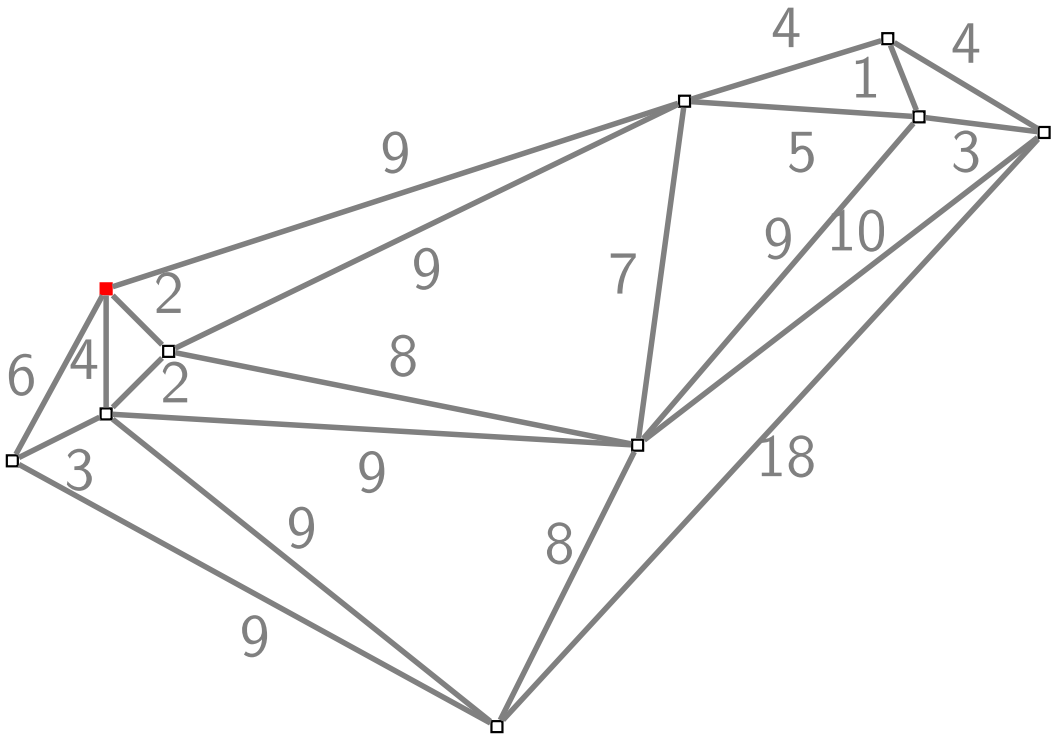


Figure: TSP instance

Minimum spanning tree

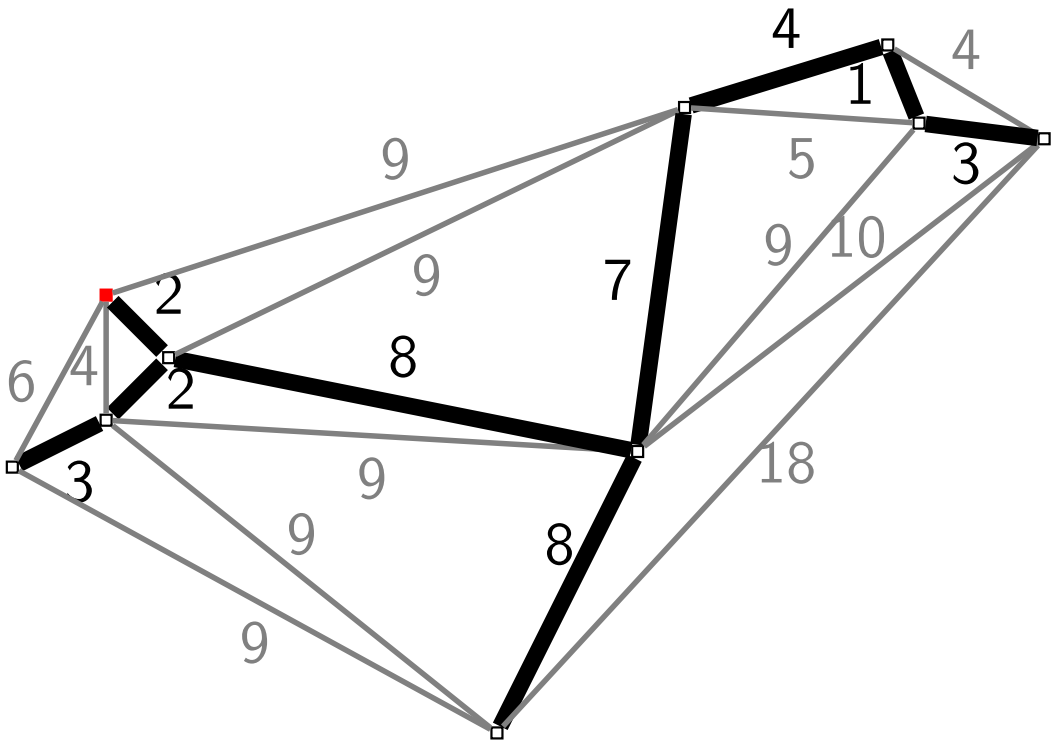


Figure: Minimal spanning tree

Depth-first search

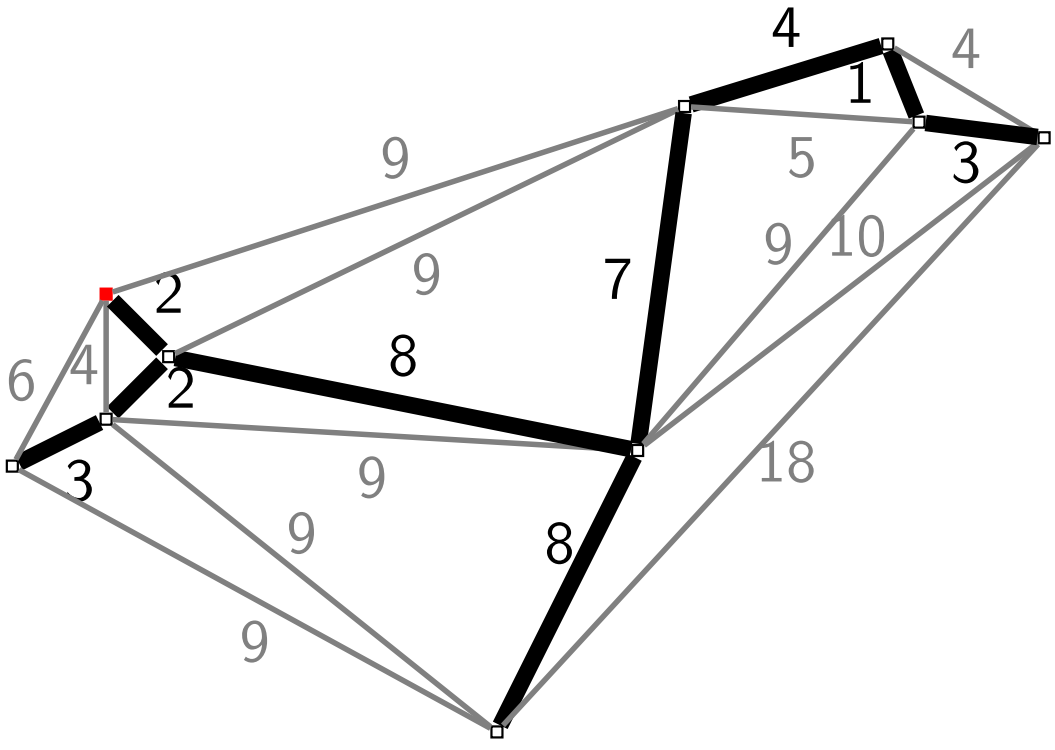


Figure: DFS

Depth-first search

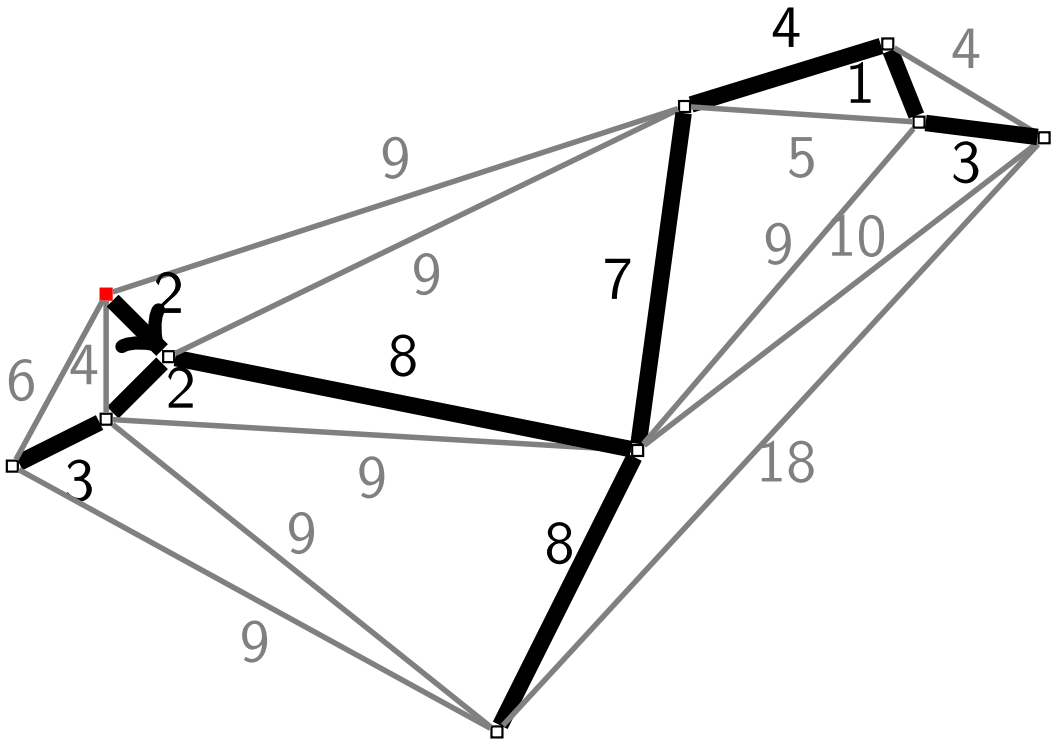


Figure: DFS

Depth-first search

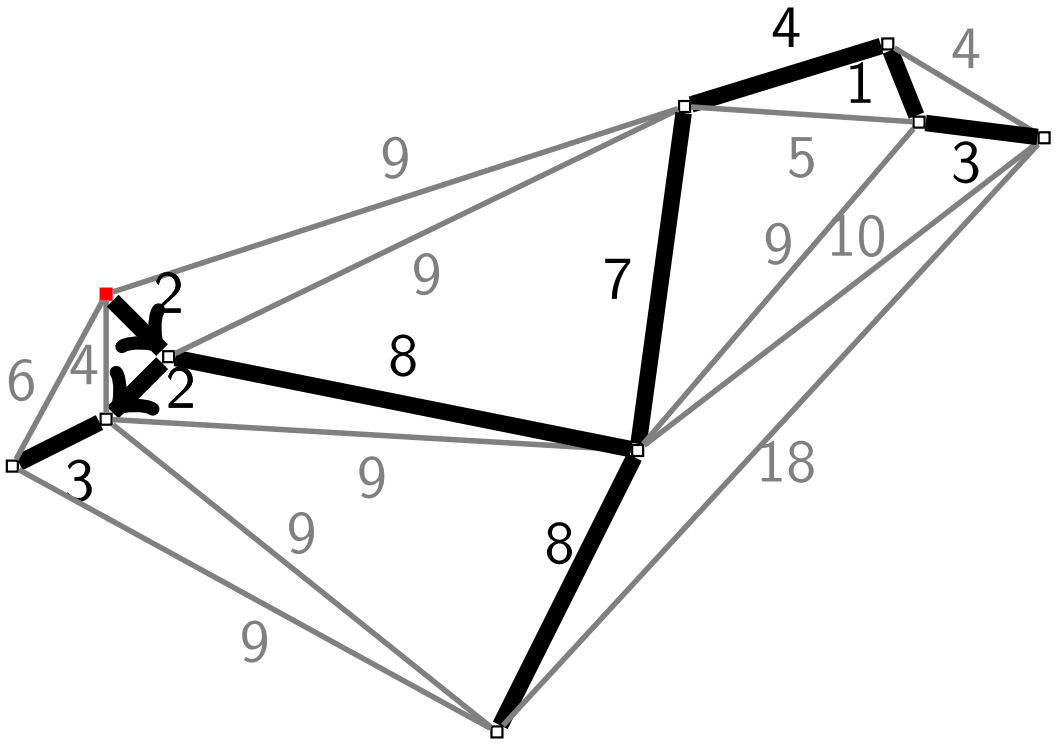


Figure: DFS

Depth-first search

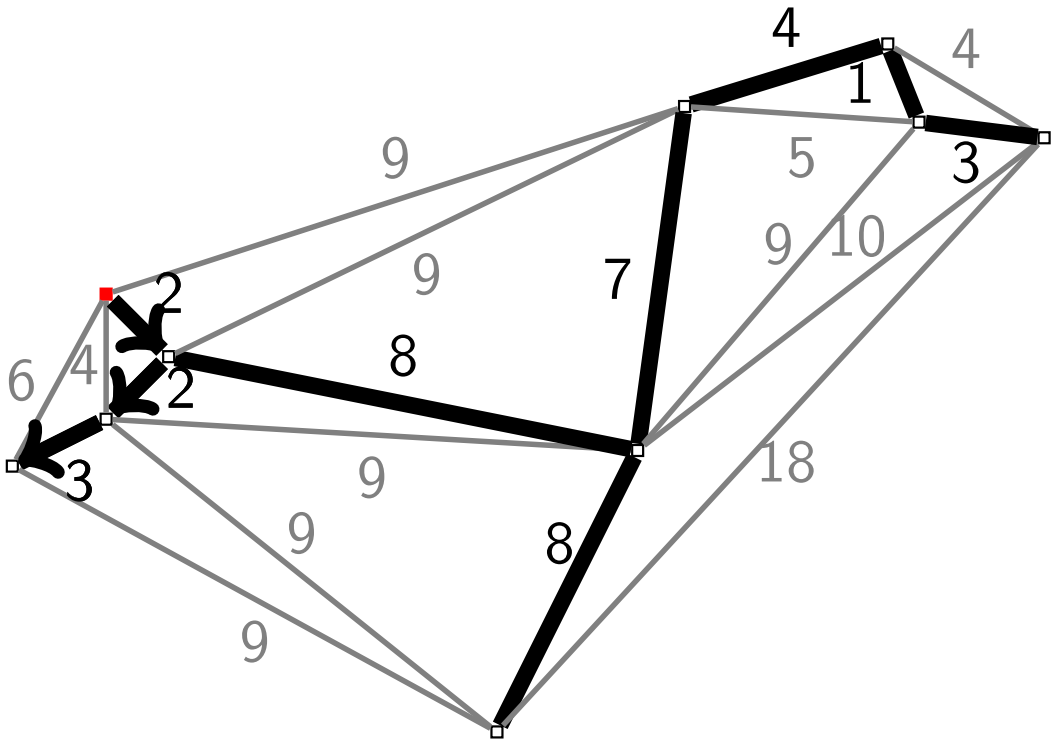


Figure: DFS

Depth-first search

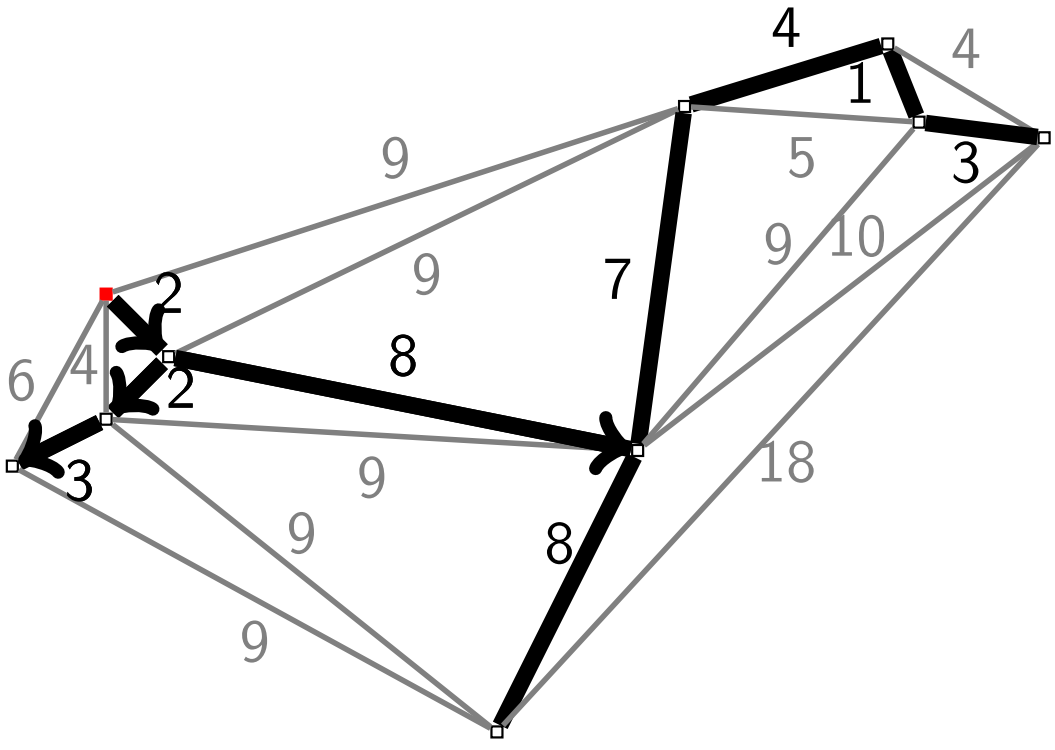


Figure: DFS

Depth-first search

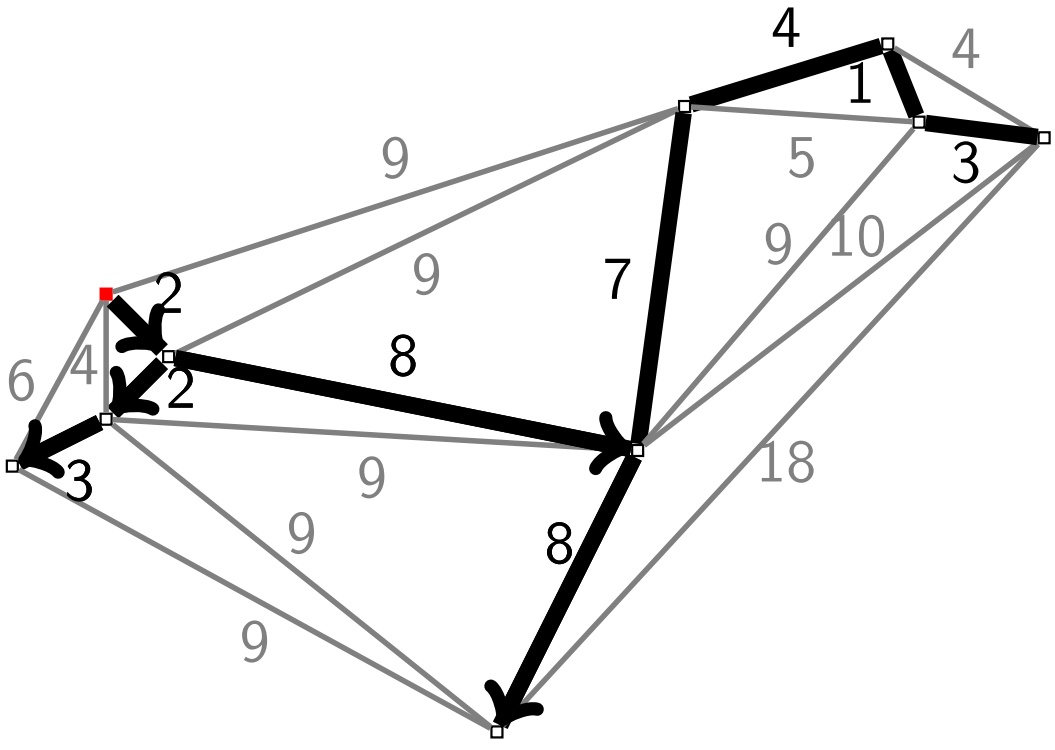


Figure: DFS

Depth-first search

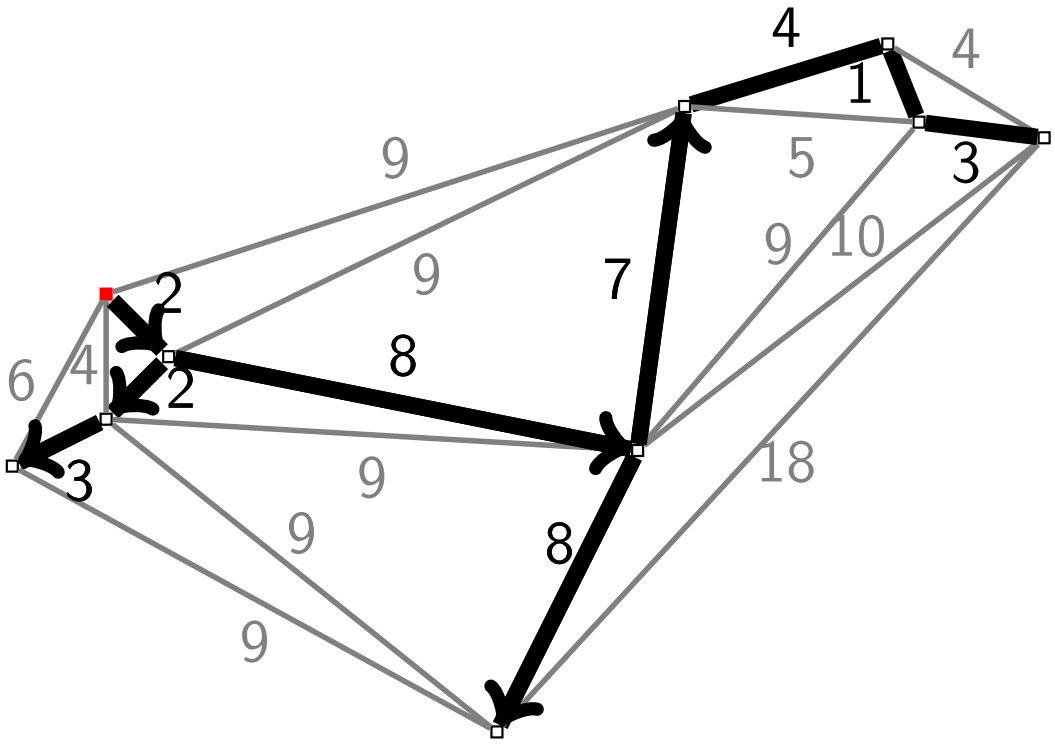


Figure: DFS

Depth-first search

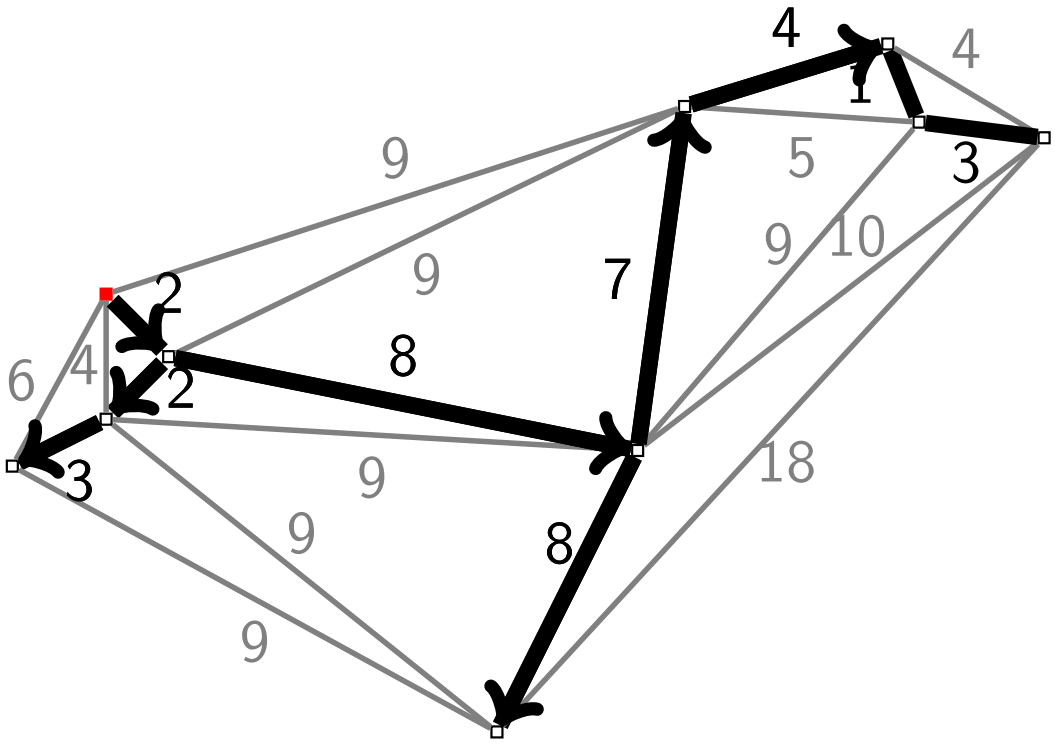


Figure: DFS

Depth-first search

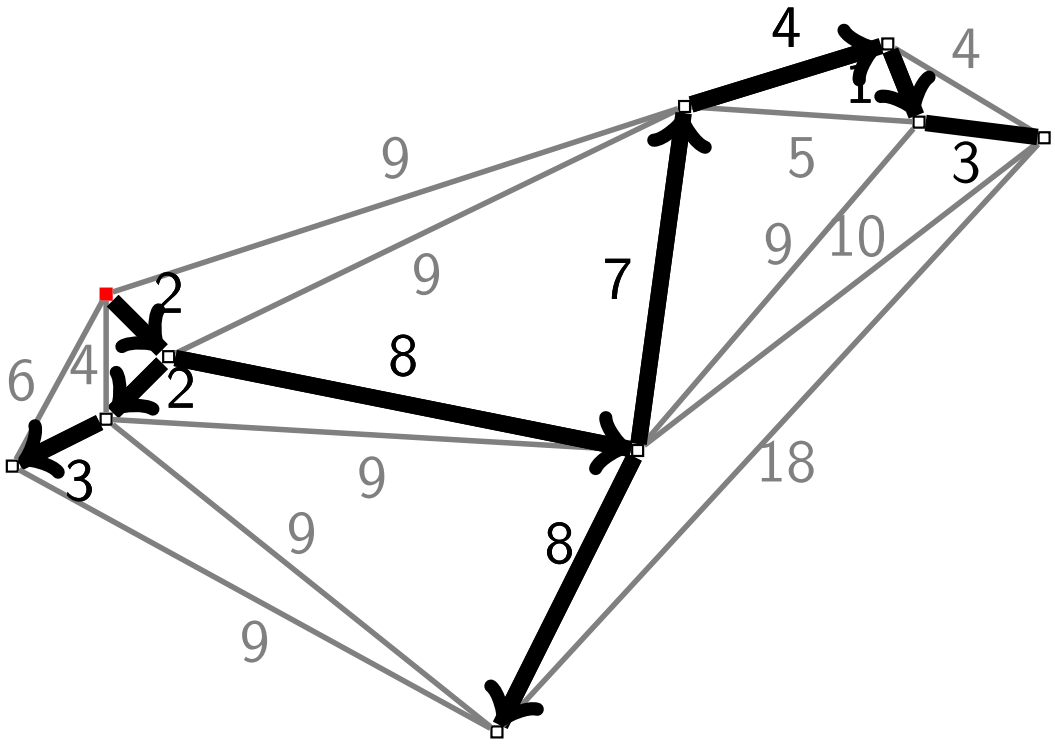


Figure: DFS

Depth-first search

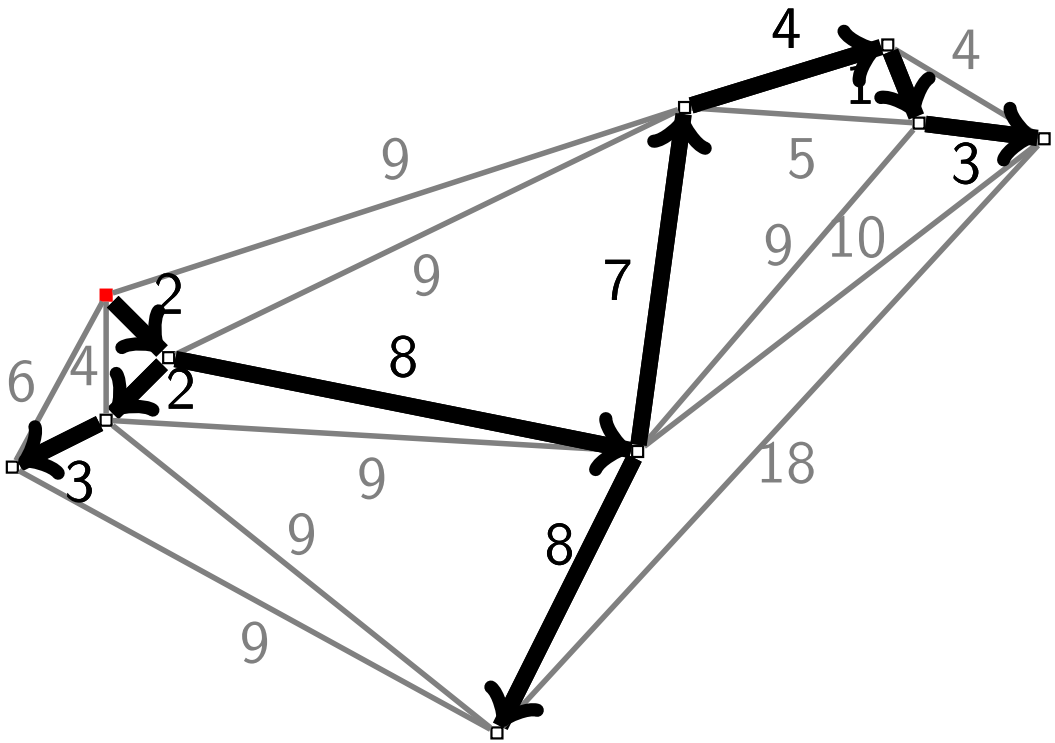


Figure: DFS

Approx algorithm

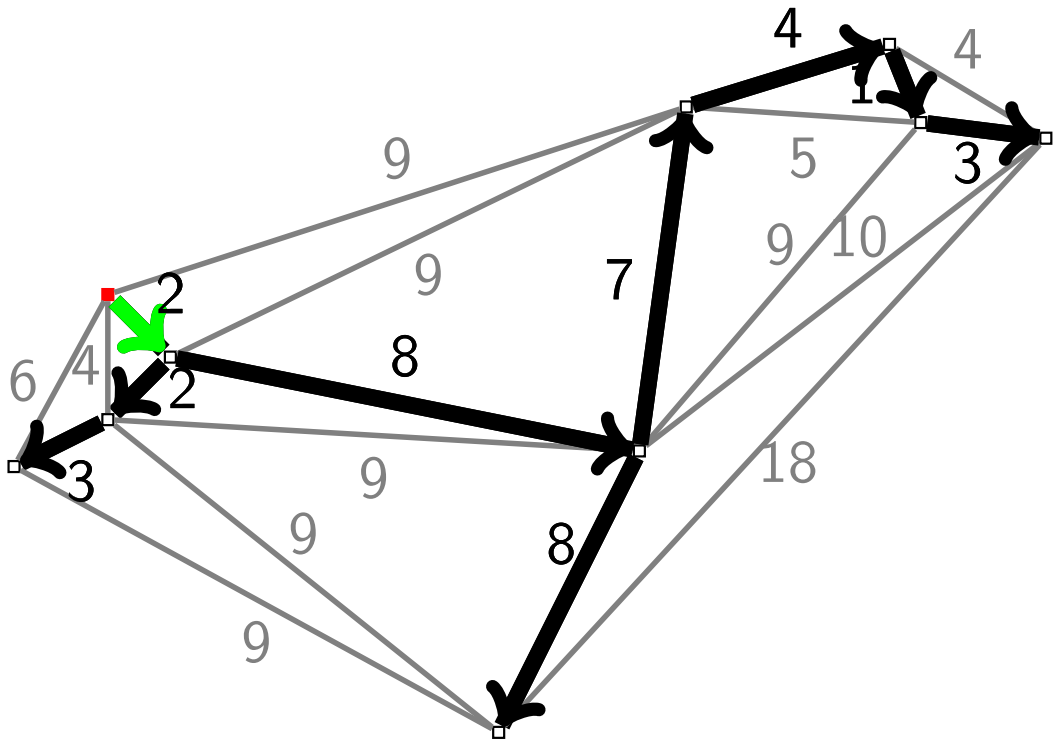


Figure: Approximation algorithm

Approx algorithm

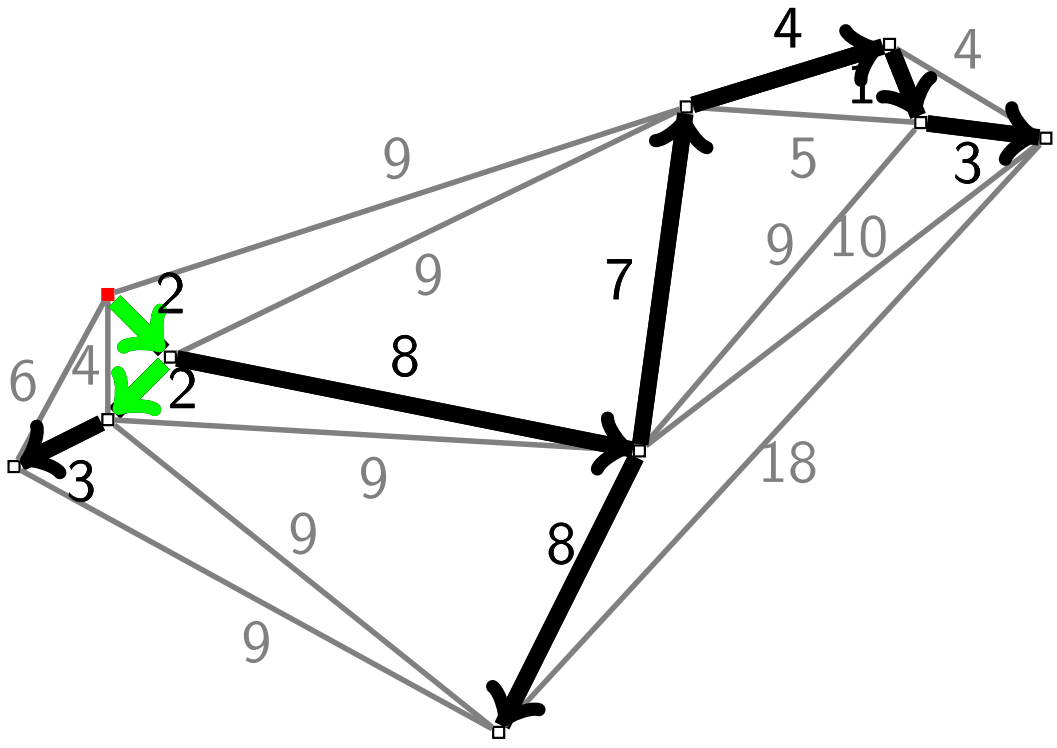


Figure: Approximation algorithm

Approx algorithm

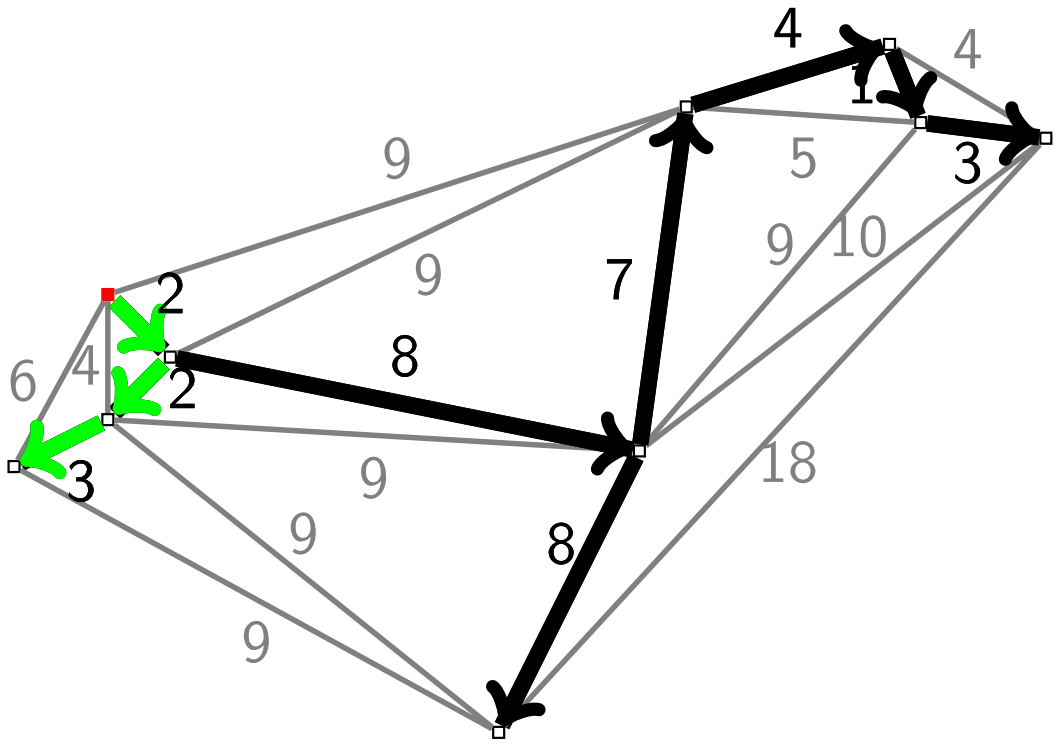


Figure: Approximation algorithm

Approx algorithm

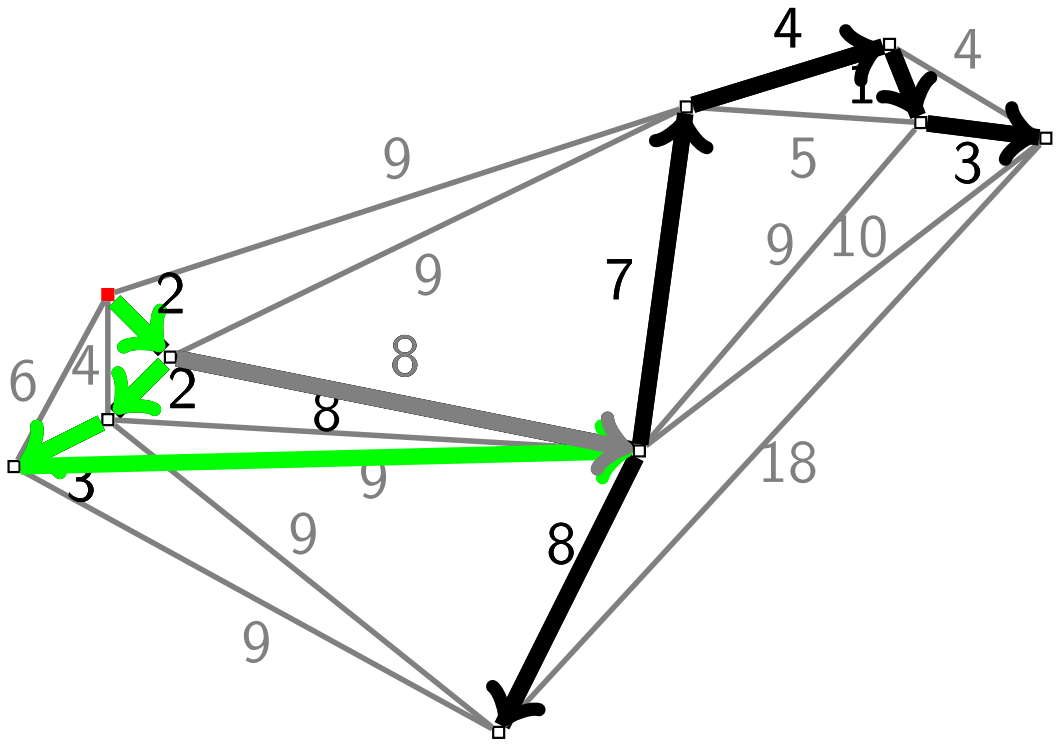


Figure: Approximation algorithm

Approx algorithm

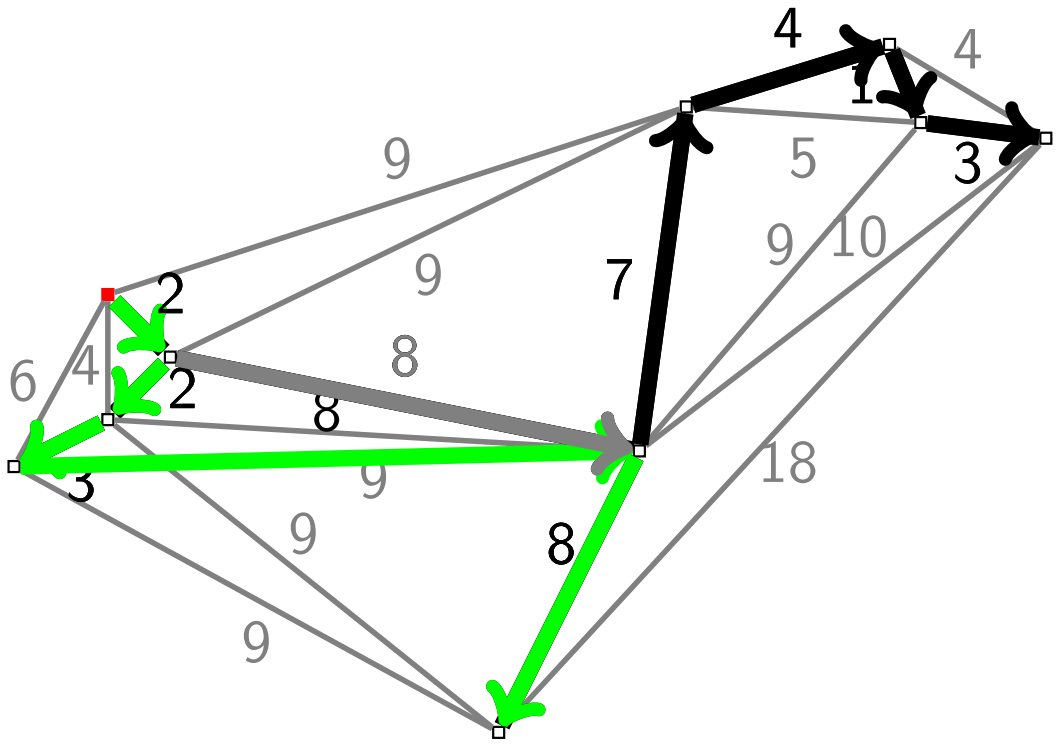


Figure: Approximation algorithm

Approx algorithm

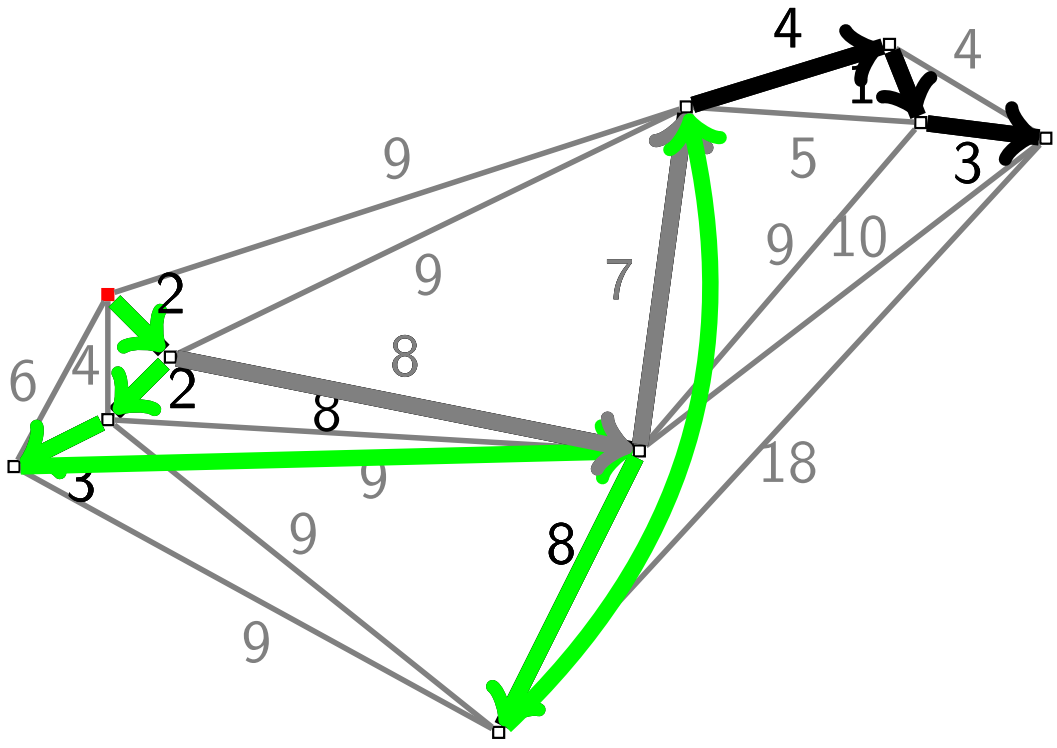


Figure: Approximation algorithm

Approx algorithm

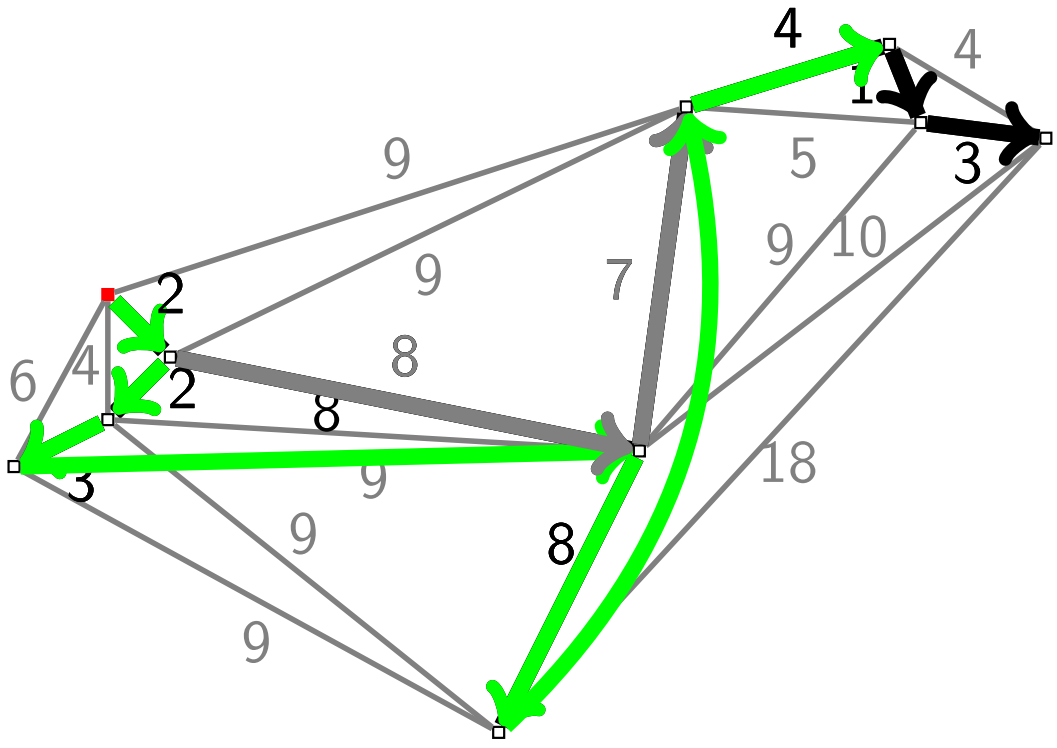


Figure: Approximation algorithm

Approx algorithm

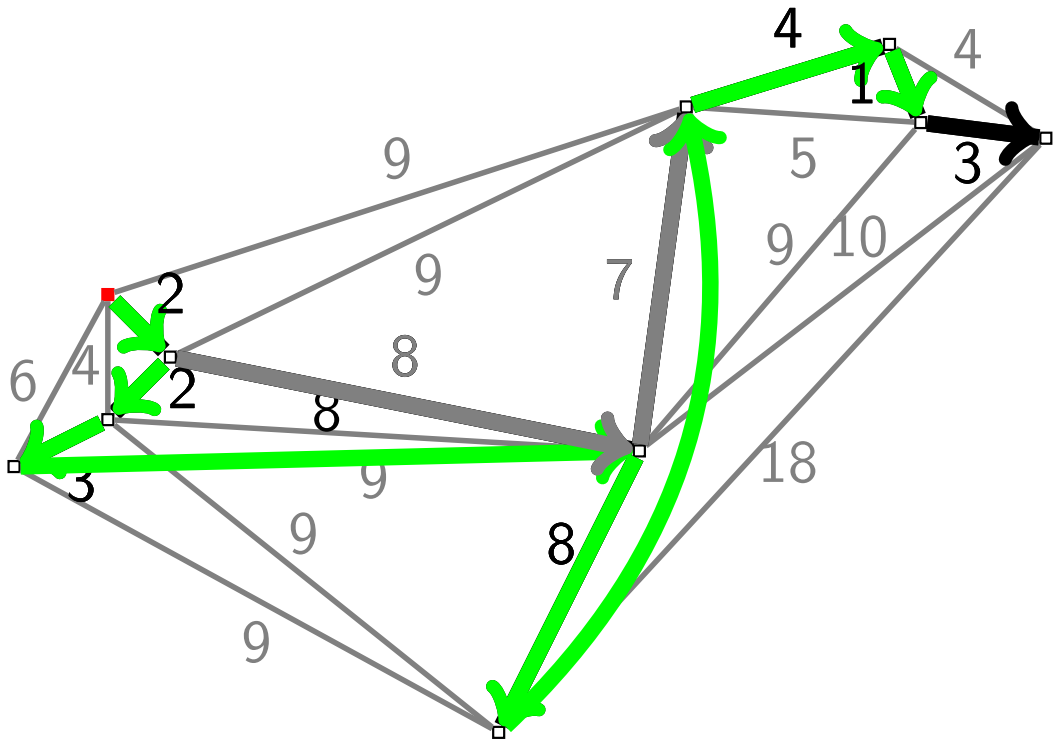


Figure: Approximation algorithm

Approx algorithm

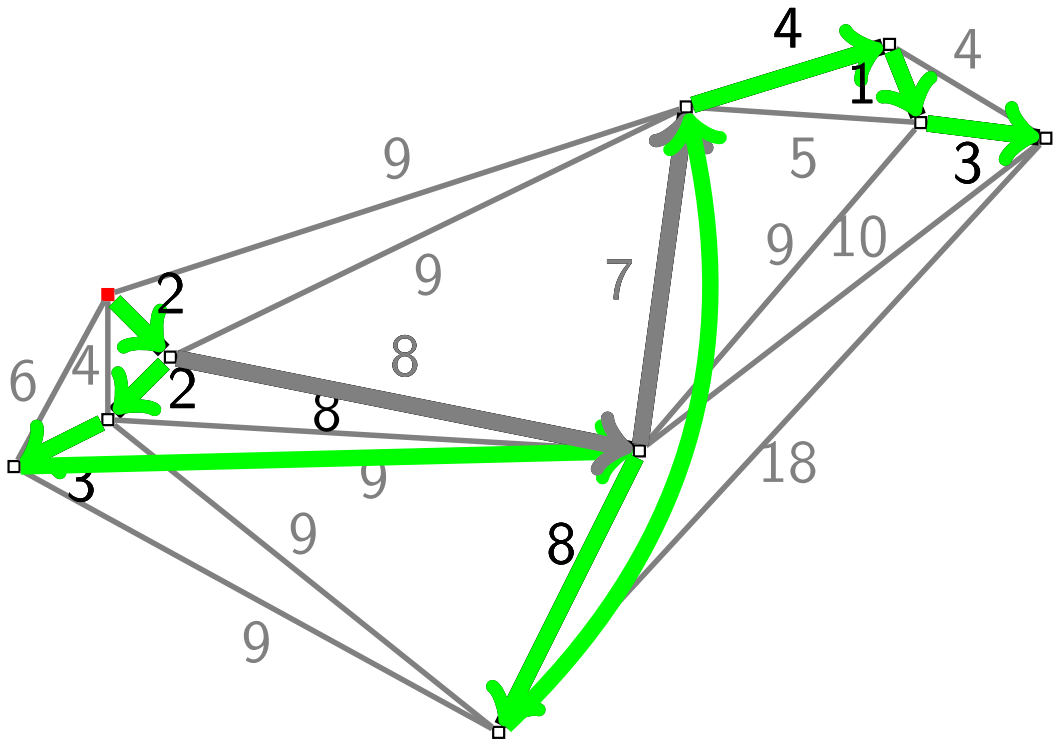


Figure: Approximation algorithm

Approx algorithm

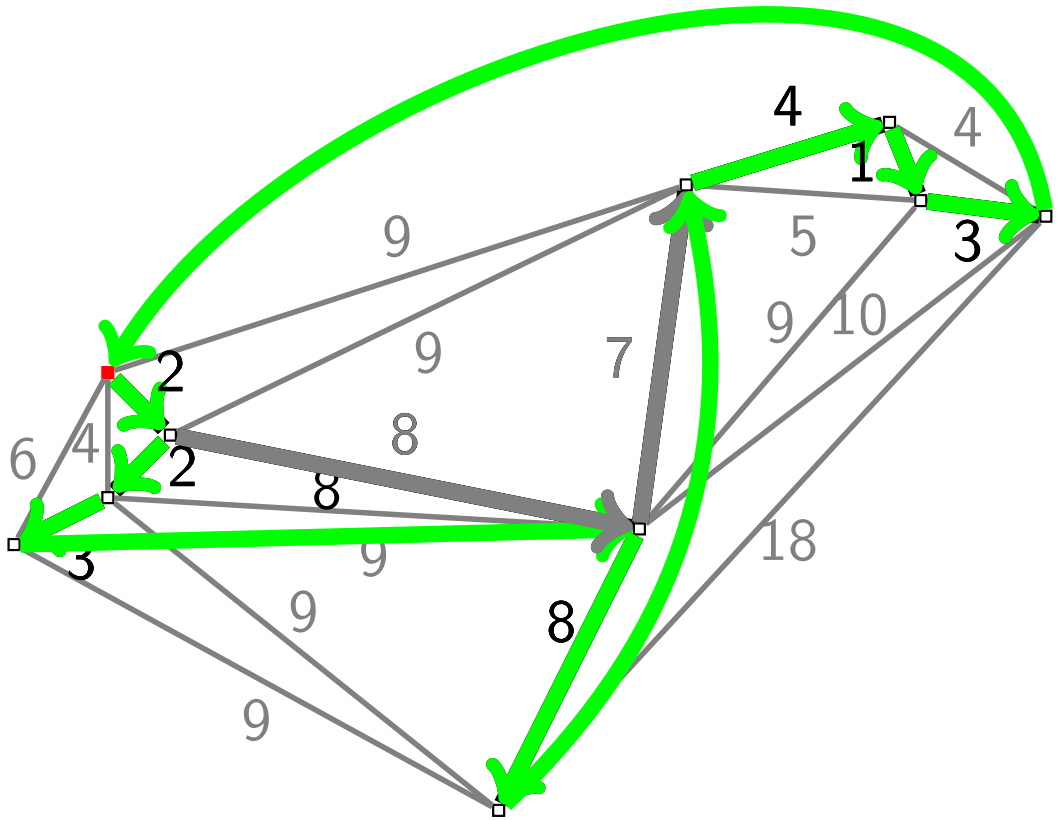


Figure: Approximation algorithm

Approximation ratio

The solution given by this heuristic is at most twice as bad as the optimal solution. Why?

Approximation ratio

The solution given by this heuristic is at most twice as bad as the optimal solution. Why?

Assumption of the triangle inequality. The solution is less than twice the MST. MST is a lower bound.

Approximation ratio

The solution given by this heuristic is at most twice as bad as the optimal solution. Why?

Assumption of the triangle inequality. The solution is less than twice the MST. MST is a lower bound. We say it is a 2-approx algorithm.

SCIP heuristics

file	heur_actconsdiving.h LP diving heuristic that chooses fixings w.r.t. the active constraints the variable appear in.
file	heur_adaptivediving.h diving heuristic that selects adaptively between the existing, public dive sets
file	heur_alns.h Adaptive large neighborhood search heuristic that orchestrates popular LNS heuristics.
file	heur_bound.h heuristic which fixes all integer variables to a bound (lower/upper) and solves the remaining LP
file	heur_clique.h LNS heuristic using a clique partition to restrict the search neighborhood.
file	heur_coefdiving.h LP diving heuristic that chooses fixings w.r.t. the matrix coefficients.
file	heur_completesol.h primal heuristic trying to complete given partial solutions
file	heur_conflictdiving.h LP diving heuristic that chooses fixings w.r.t. conflict locks.
file	heur_crossover.h LNS heuristic that tries to combine several feasible solutions.
file	heur_dins.h DINS primal heuristic.
file	heur_distributiondiving.h Diving heuristic that chooses fixings w.r.t. changes in the solution density after Pryor and Chinneck.
file	heur_dps.h dynamic partition search
file	heur_dualval.h primal heuristic that uses dualvalues for successive switching variable values
file	heur_farkasdiving.h LP diving heuristic that tries to construct a Farkas-proof.
file	heur_feaspump.h Objective Feasibility Pump 2.0.
file	heur_fixandinfer.h fix-and-infer primal heuristic
file	heur_fracdiving.h LP diving heuristic that chooses fixings w.r.t. the fractionalities.
file	heur_gins.h LNS heuristic that tries to delimit the search region to a neighborhood in the constraint graph.
file	heur_guideddiving.h LP diving heuristic that chooses fixings in direction of incumbent solutions.
file	heur_indicator.h handle partial solutions for linear problems with indicators and otherwise continuous variables
file	heur_intdiving.h LP diving heuristic that fixes variables with integral LP value.
file	heur_intshifting.h LP rounding heuristic that tries to recover from intermediate infeasibilities, shifts integer variables, and solves a

file	heur_linesearchdiving.h LP diving heuristic that fixes variables with a large difference to their root solution.
file	heur_localbranching.h Local branching heuristic according to Fischetti and Lodi.
file	heur_locks.h locks primal heuristic
file	heur_lpface.h LNS heuristic that tries to compute integral solution on optimal LP face.
file	heur_mpec.h mpec primal heuristic
file	heur_multistart.h multistart heuristic for convex and nonconvex MINLPs
file	heur_mutation.h LNS heuristic that tries to randomly mutate the incumbent solution.
file	heur_nlpdiving.h NLP diving heuristic that chooses fixings w.r.t. the fractionalities.
file	heur_objpscostdiving.h LP diving heuristic that changes variable's objective value instead of bounds, using pseudo cost values as guide.
file	heur_octane.h octane primal heuristic based on Balas, Ceria, Dawande, Margot, and Pataki
file	heur_ofins.h OFINS - Objective Function Induced Neighborhood Search - a primal heuristic for reoptimization.
file	heur_oneopt.h Improvement heuristic that alters single variable values.
file	heur_padm.h PADM primal heuristic based on ideas published in the paper "A Decomposition Heuristic for Mixed-Integer Supply Chain Problems" by Martin Schmidt, Lars Schewe, and Dieter Weninger.
file	heur_proximity.h improvement heuristic which uses an auxiliary objective instead of the original objective function which is itself added as a constraint to a sub-SCIP instance. The heuristic was presented by Matteo Fischetti and Michele Monaci
file	heur_pscostdiving.h LP diving heuristic that chooses fixings w.r.t. the pseudo cost values.
file	heur_randrounding.h randomized LP rounding heuristic which also generates conflicts via an auxiliary probing tree
file	heur_rens.h LNS heuristic that finds the optimal rounding to a given point.
file	heur_reoptsols.h reoptsols primal heuristic
file	heur_repair.h repair primal heuristic
file	heur_rins.h LNS heuristic that combines the incumbent with the LP optimum.
file	heur_rootsoldiving.h LP diving heuristic that changes variables' objective values using root LP solution as guide.

file	heur_rounding.h LP rounding heuristic that tries to recover from intermediate infeasibilities.
file	heur_shiftandpropagate.h preroot heuristic that alternatingly fixes variables and propagates domains
file	heur_shifting.h LP rounding heuristic that tries to recover from intermediate infeasibilities and shifts continuous variables.
file	heur_simplerounding.h Simple and fast LP rounding heuristic.
file	heur_subnlp.h NLP local search primal heuristic using sub-SCIPs.
file	heur_sync.h primal heuristic that adds given solutions
file	heur_trivial.h trivial primal heuristic
file	heur_trivialnegation.h trivialnegation primal heuristic
file	heur_trustregion.h Large neighborhood search heuristic for Benders' decomposition based on trust region methods.
file	heur_trysol.h primal heuristic that tries a given solution
file	heur_twoopt.h Primal heuristic to improve incumbent solution by flipping pairs of variables.
file	heur_undercover.h Undercover primal heuristic for MINLPs.
file	heur_vbounds.h LNS heuristic uses the variable lower and upper bounds to determine the search neighborhood.
file	heur_veclendiving.h LP diving heuristic that rounds variables with long column vectors.
file	heur_zeroobj.h heuristic that tries to solve the problem without objective. In Gurobi, this heuristic is known as "Hail Mary"
file	heur_zirounding.h ZI Round primal heuristic.
file	heur_cycgreedy.h Greedy primal heuristic. States are assigned to clusters iteratively. At each iteration all possible assignments are computed and the one with the best change in objective value is selected.
file	heur_cyckerlin.h Improvement heuristic that trades bin-variables between clusters.
file	heur_fuzzyround.h primal heuristic that constructs a feasible solution from the lp-relaxation. Round only on the state-variables (binvars) and then reconstruct the rest of the variables accordingly.
file	heur_redsize.h primal heuristic that solves the problem with a sparser matrix as a submip
file	heur_listscheduling.c scheduling specific primal heuristic which is based on bidirectional serial generation scheme.

Modern solvers employ a lot of heuristics.

Metaheuristics

Heuristics can often get stuck in local optima. How can we deal with this?

Metaheuristics

Heuristics can often get stuck in local optima. How can we deal with this?

Potentially accept worsening solutions

Metaheuristics

Heuristics can often get stuck in local optima. How can we deal with this?

Potentially accept worsening solutions

Metaheuristics are abstractions that work with sets of solutions. Eg: TSP paths instead of cities.

Main idea

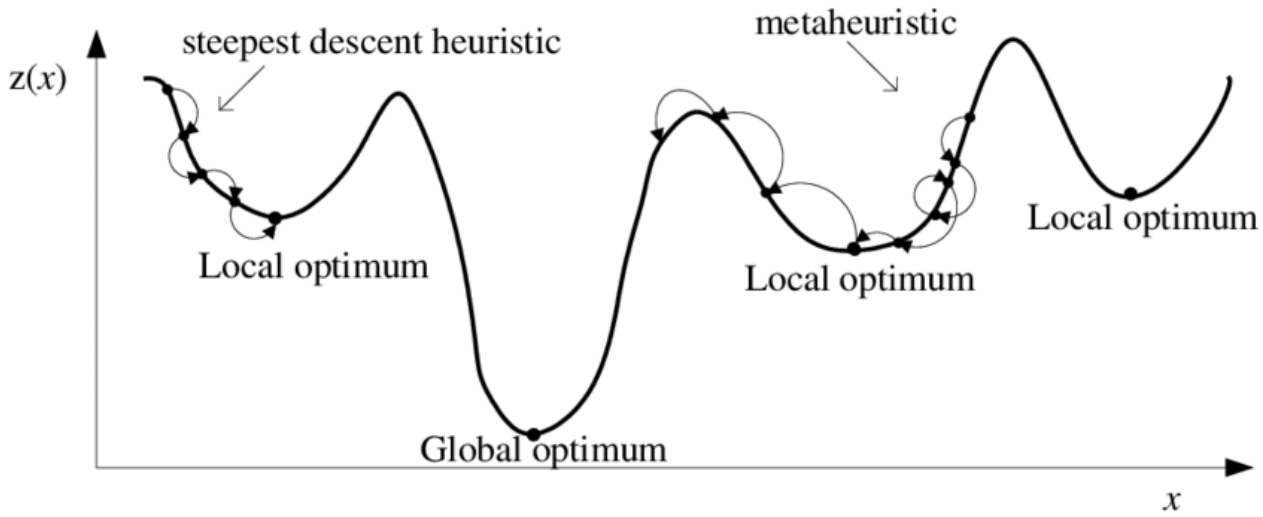


Figure: Escape local optima by accepting worsening moves

Simulated Annealing

Randomly decide to move to a neighboring solution based on its fitness. The probabilities decrease with time.

Traveling Salesman Problem Visualization

Simulated Annealing

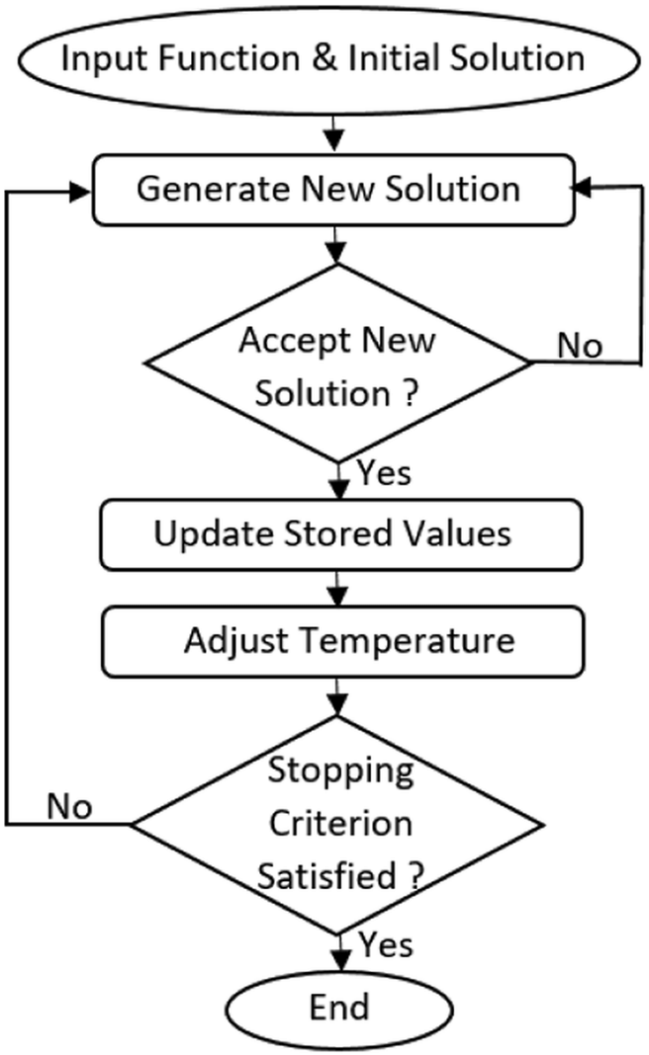


Figure: Simulated Annealing Flowchart

Tabu Search

Initially accept worsening solutions.
Keep a tabu list that forbids recently visited solutions.

Tabu Search

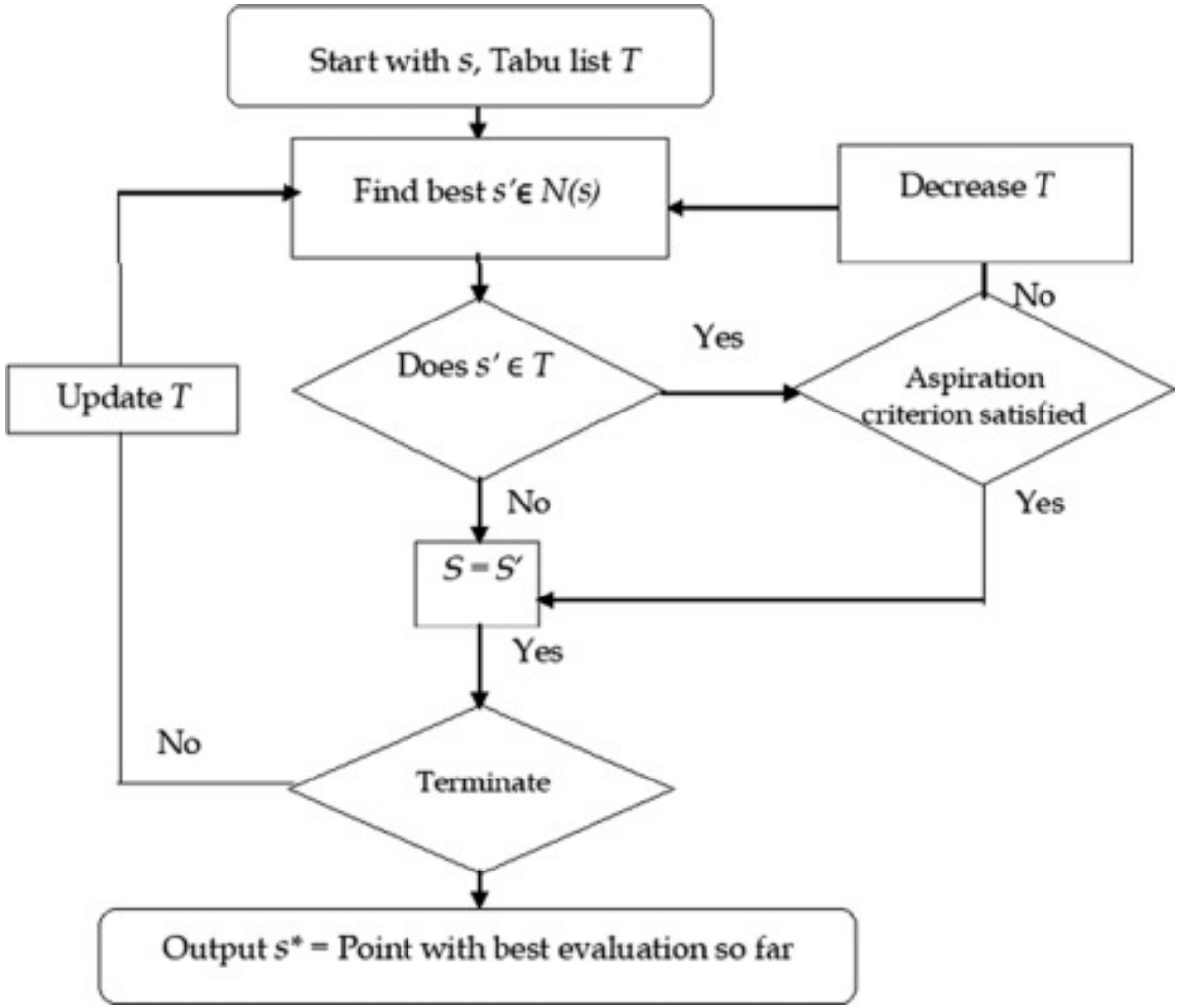


Figure: Tabu Search Flowchart

Genetic Algorithm

Inspired by natural processes, keeps a population of candidate solutions. Iteratively combines them to create new solutions.

Genetic Algorithm

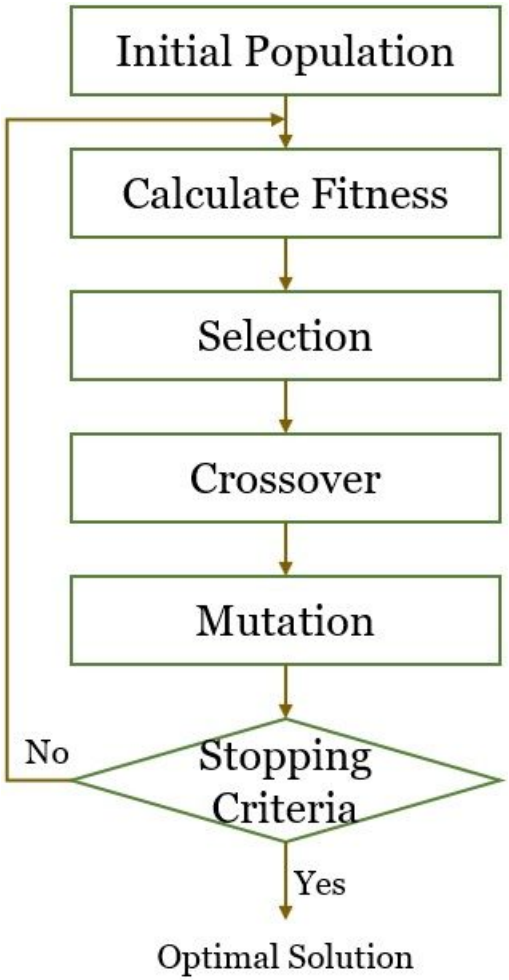


Figure: Genetic Algorithm Flowchart

Meta Heuristics Comparison

Visualization of metaheuristics for the traveling salesman problem

Reformulations

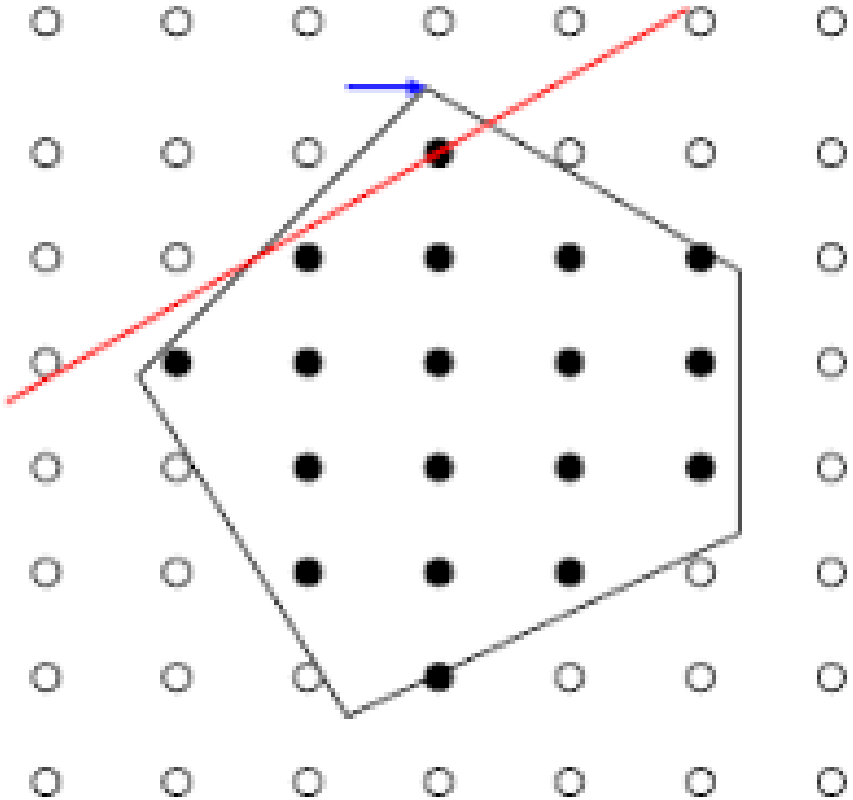


Figure: Integer programs admit infinite formulations

Reformulations

If formulation A is contained in formulation B, then the former is preferred. Why is that?

Reformulations

If formulation A is contained in formulation B, then the former is preferred. Why is that?

The provided bound is better, can prune branch and bound tree earlier.

Reformulations

If formulation A is contained in formulation B, then the former is preferred. Why is that?

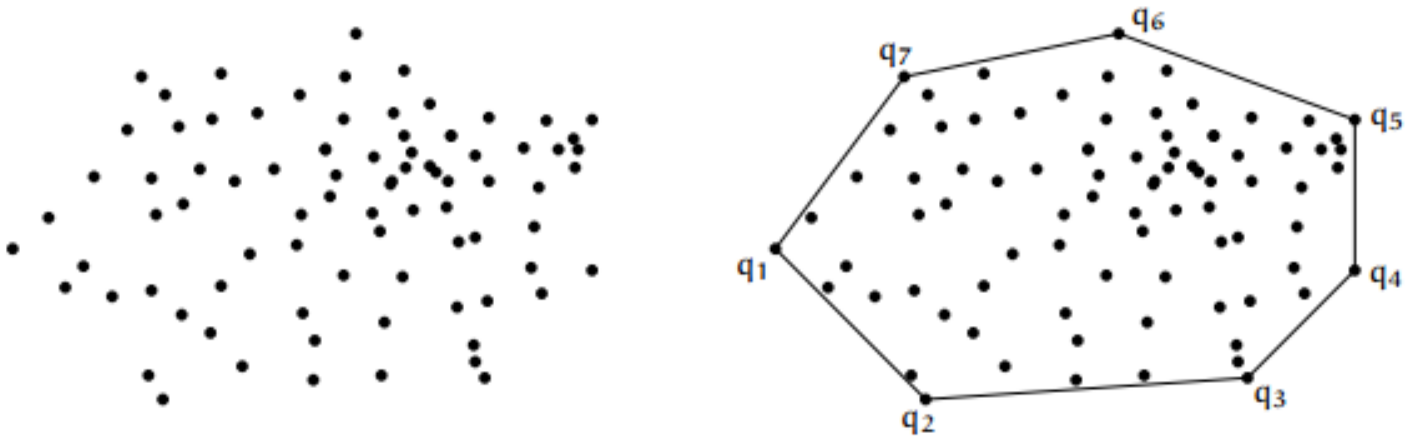
The provided bound is better, can prune branch and bound tree earlier.

Convex Hull

Definition

Let $X \subseteq \mathbb{R}^n$ be a set. The **convex hull** of X, denoted by $conv(X)$, is the intersection of all convex sets containing X. Equivalently, it is the set of all convex combinations of X.

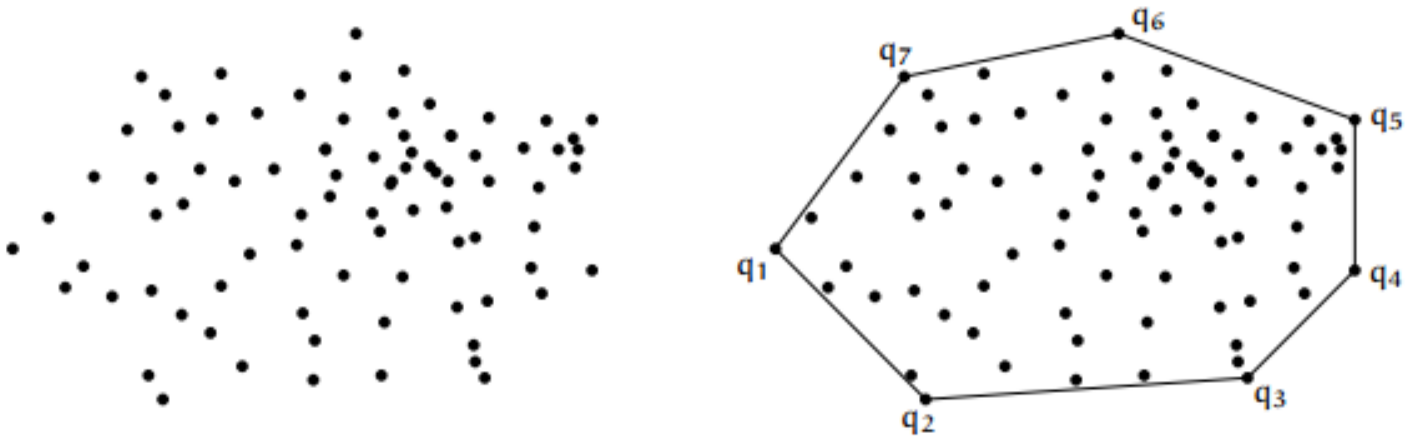
Perfect Formulation



(a) Input. (b) Output.

Figure: Linear relaxation equal to convex hull

Perfect Formulation

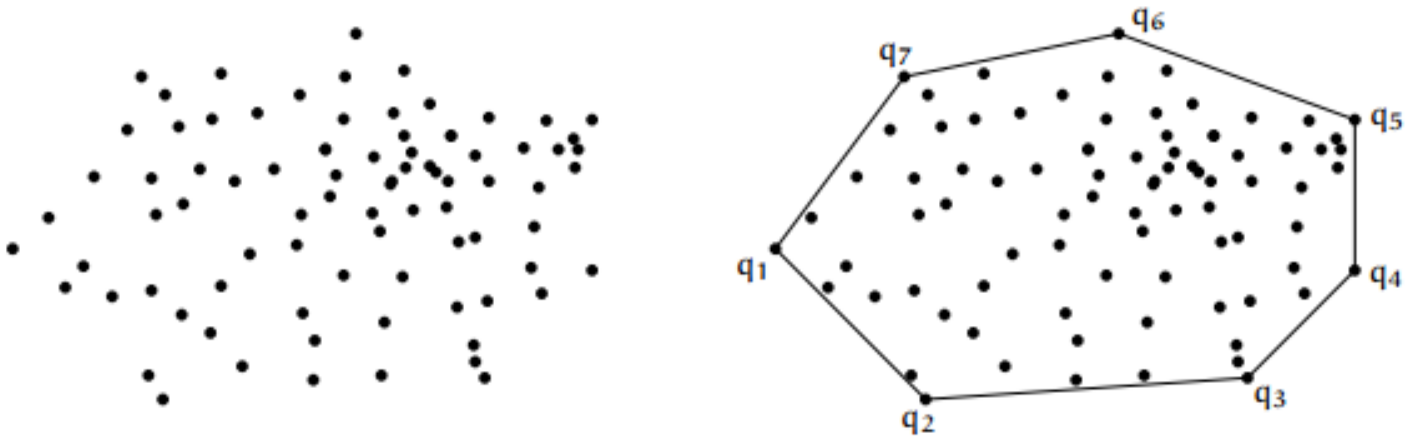


(a) Input. (b) Output.

Figure: Linear relaxation equal to convex hull

Which points do we need to check?

Perfect Formulation



(a) Input. (b) Output.

Figure: Linear relaxation equal to convex hull

Which points do we need to check? Why?

Cutting planes

In theory, every integer program has an equivalent linear programming formulation. Why?

Cutting planes

In theory, every integer program has an equivalent linear programming formulation. Why?

Convex hull, linear relaxation is upper bound, hence optimal solution at vertex.

Throughout the solving process, we keep adding cuts to contract the feasible region.

Trade-off

However, a weaker formulation may sometimes be beneficial.
Why?

Trade-off

However, a weaker formulation may sometimes be beneficial. Why? Tends to require more restrictions, hence the linear relaxation is harder. A trade-off is needed.

Symmetry

The solution space can exhibit a lot of symmetry (equivalent solutions modulo a permutation of the variables, for example). Especially damaging in integer optimization.

Symmetry

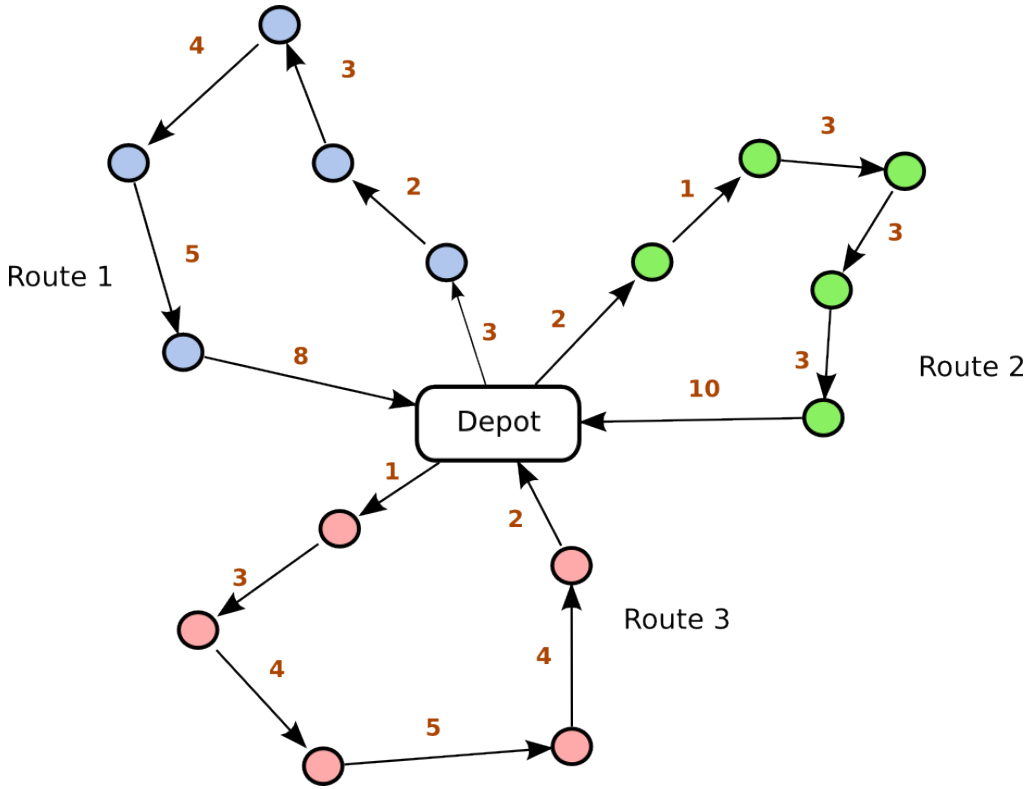


Figure: The vehicle routing problem has a lot of symmetry

Symmetry in chess

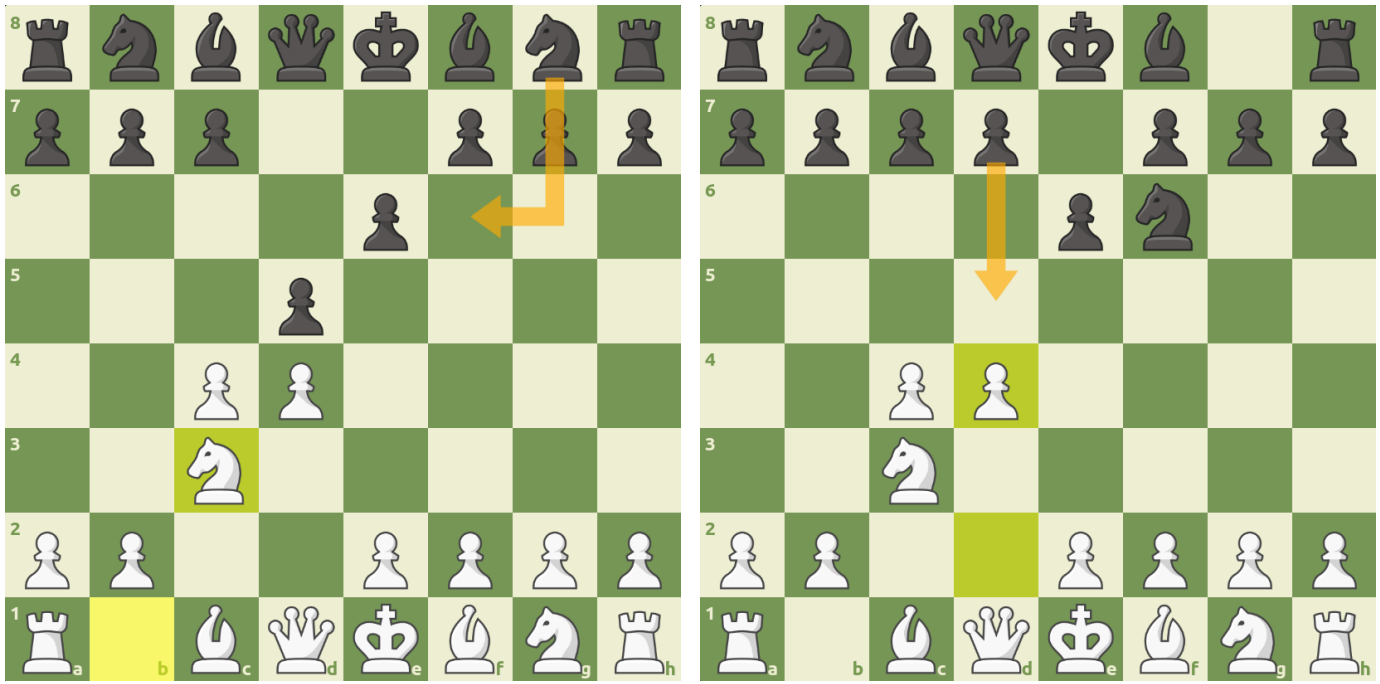


Figure: The same position can be reached from the Queen's Gambit and the English

Symmetry is common in chess. Engines need to record previously studied board positions to avoid wasting time with transpositions.

Presolving

Presolving: Analyze the problems and reduce the solution space by identifying symmetry, redundant or implicit variables, etc.

For example:

- $x \leq 1.5, y \geq 0.5, x + y \leq 1 \implies x \leq 0.5$
- $x \leq 1.5, x \in \mathbb{Z} \implies x \leq 1$

Presolving

```
presolving (26 rounds: 26 fast, 3 medium, 3 exhaustive):
 46 deleted vars, 569 deleted constraints, 0 added constraints, 12680 tightened bounds, 0 add
 937 implications, 100 cliques
presolved problem has 2982 variables (120 bin, 0 int, 0 impl, 2862 cont) and 5338 constraints
```

Figure: Example of SCIP presolving

Logical Constraints

With binary variables, we can model some logical constraints.

Logical Constraints

With binary variables, we can model some logical constraints.

$\neg x$ |

Logical Constraints

With binary variables, we can model some logical constraints.

$\neg x$	$1 - x$
$x \implies y$	

Logical Constraints

With binary variables, we can model some logical constraints.

$\neg x$	$1 - x$
$x \implies y$	$x \leq y$
$x \wedge y$	

Logical Constraints

With binary variables, we can model some logical constraints.

$\neg x$	$1 - x$
$x \implies y$	$x \leq y$
$x \wedge y$	$x + y = 2$
$x \vee y$	$x + y \geq 1$
$x \dot{\vee} y$	$x + y = 1$
$\exists x$	

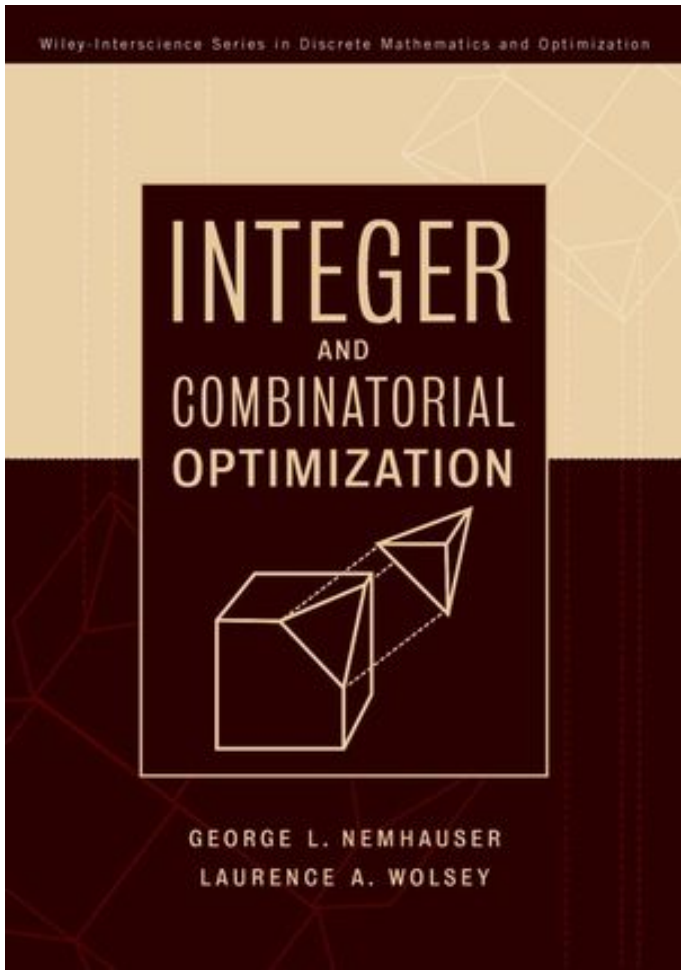
Logical Constraints

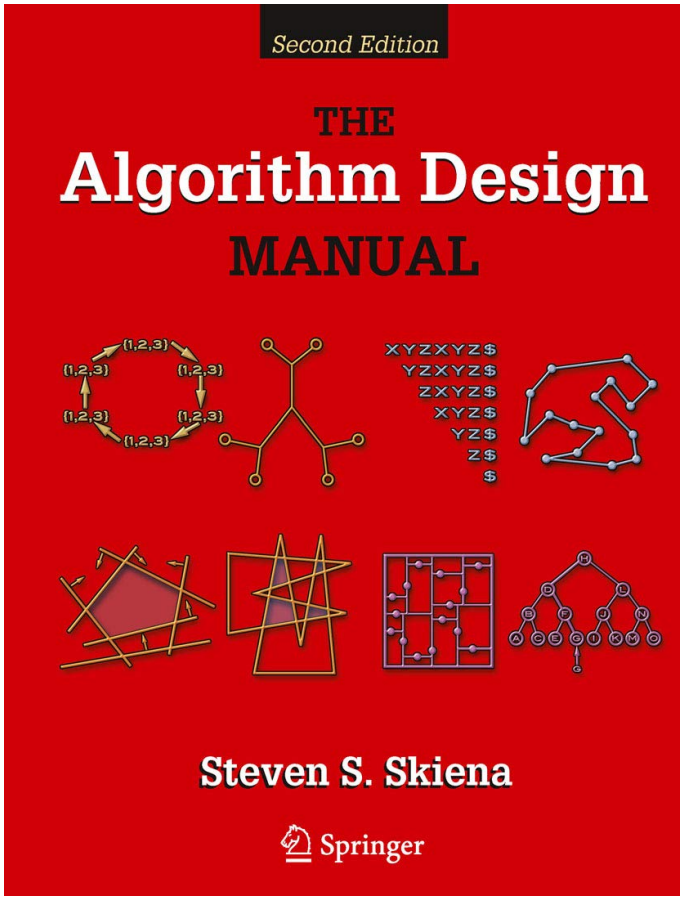
With binary variables, we can model some logical constraints.

$\neg x$	$1 - x$
$x \implies y$	$x \leq y$
$x \wedge y$	$x + y = 2$
$x \vee y$	$x + y \geq 1$
$x \dot{\vee} y$	$x + y = 1$
$\exists x$	$\sum_{i=1}^{n} x_i \geq 1$
$\exists! x$	$\sum_{i=1}^{n} x_i = 1$

Table: Formulating logical expressions with integer programming

Bibliography





Suggestions

Light read on integer programming: Link

Discrete Optimization with Professor Pascal Van Hentenryck, on Coursera: Link

Exercises

Revisiting Cutting-Stock

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{M} y_j & & \\
\text{s.t.} \quad & \sum_{j=1}^{M} x_{ij} = d_i, & & i \in [n] \\
& \sum_{i=1}^{n} l_i x_{ij} \leq L, & & j \in [m] \\
& x_{ij} \leq d_i y_j, & & i \in [n], j \in [m] \\
& x_{ij} \in \mathbb{Z}^+, & & i \in [n], j \in [m] \\
& y_j \in \{0, 1\}, & & j \in [m]
\end{aligned}
$$

Revisiting Cutting-Stock

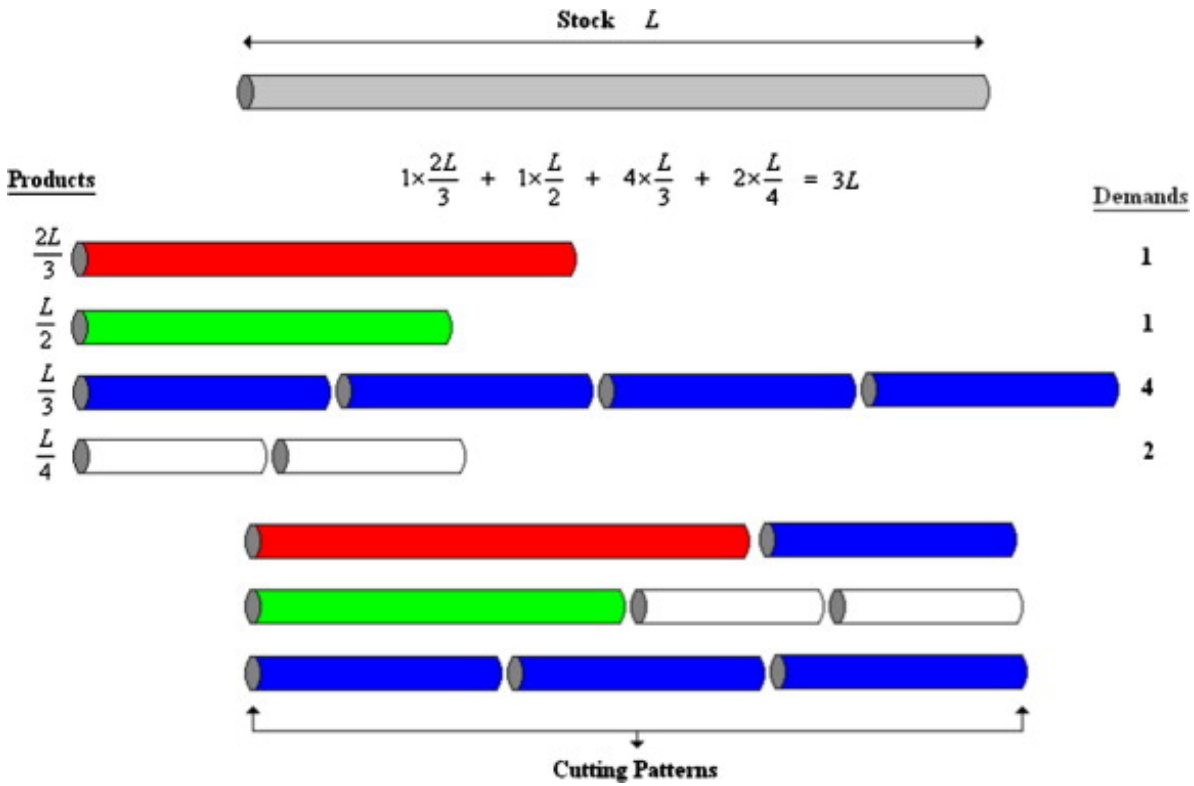


Figure: Cutting stock example

How can we cut the sheet?

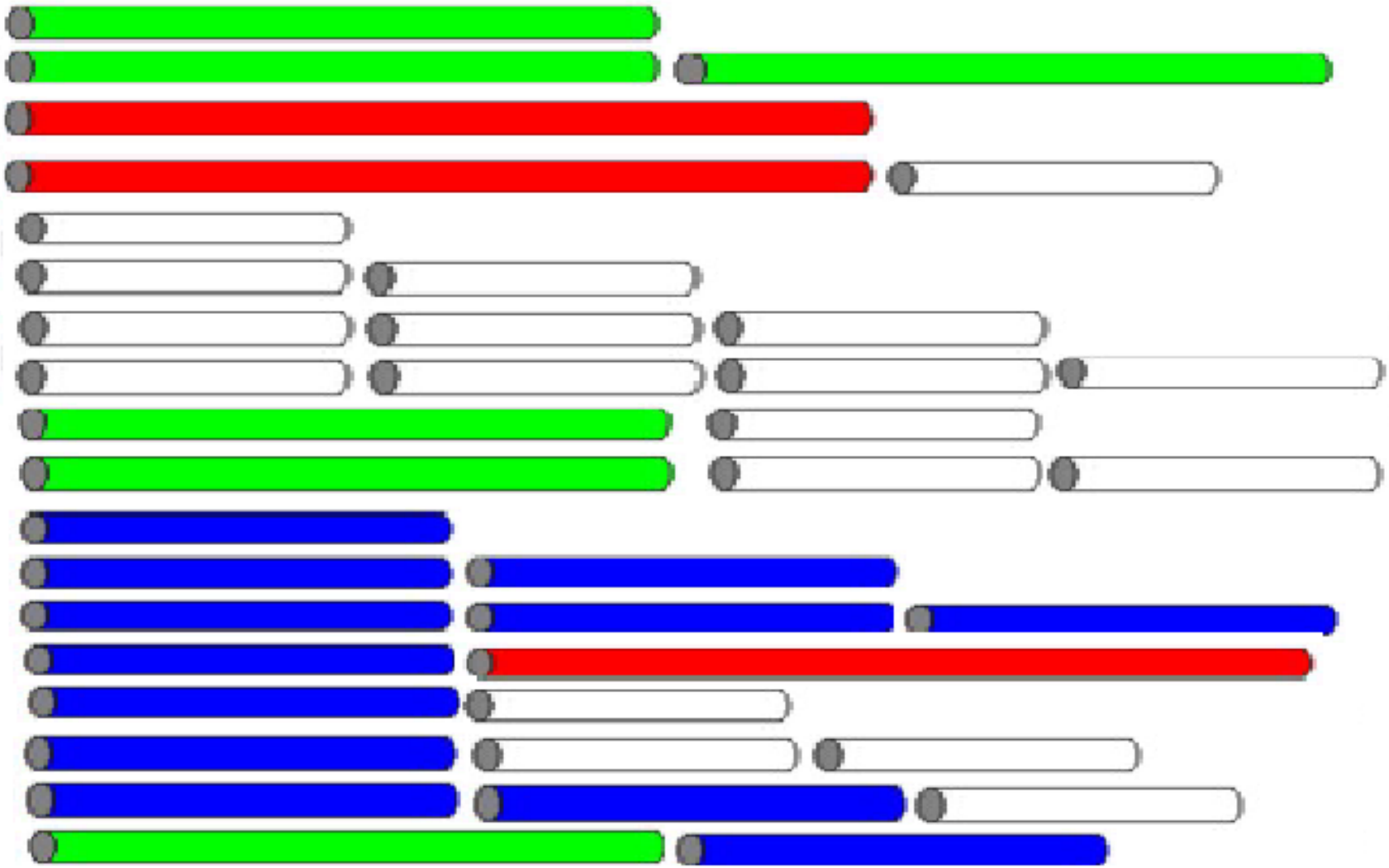


Figure: All possibilities of cutting the sheet into the orders (with Paint!)

Equivalent formulation

Idea: Of all possible combinations, pick the best ones.
Let X be the set of all feasible patterns, and $p_{i,x}$ the number of orders of type i in pattern x.

Equivalent formulation

Idea: Of all possible combinations, pick the best ones.
Let X be the set of all feasible patterns, and $p_{i,x}$ the number of orders of type i in pattern x.

$$\begin{aligned} \min_{x} \quad & \sum_{x \in X} x \\ \text{s.t.} \quad & \sum_{x \in X} x p_{i,x} = d_i \\ & x \in \mathbb{Z}_0^+ \end{aligned}$$

Equivalent formulation

Idea: Of all possible combinations, pick the best ones.
Let X be the set of all feasible patterns, and $p_{i,x}$ the number of orders of type i in pattern x.

$$\begin{aligned} \min_{x} \quad & \sum_{x \in X} x \\ \text{s.t.} \quad & \sum_{x \in X} x p_{i,x} = d_i \\ & x \in \mathbb{Z}_0^+ \end{aligned}$$

Problem: This has an exponential number of possibilities. The problem is even harder!

Restricted Master Problem

Notice that the majority of patterns are not used (not a coincidence...)

Restricted Master Problem

Notice that the majority of patterns are not used (not a coincidence...)

Main idea: Start with a small subset of patterns and solve the easy problem. Add new patterns that will improve the solution.

Some patterns can be worse than others

Pricing Problem

Which new pattern should we add? We want patterns that:

- Prioritize orders that are harming the solution
- Do not exceed the length of the metal sheet

Pricing Problem

Which new pattern should we add? We want patterns that:

- Prioritize orders that are harming the solution
- Do not exceed the length of the metal sheet

This is precisely a knapsack problem!

Pricing Problem

Which new pattern should we add? We want patterns that:

- Prioritize orders that are harming the solution
- Do not exceed the length of the metal sheet

This is precisely a knapsack problem!

- Priority orders $\rightarrow$ Valuable items

Pricing Problem

Which new pattern should we add? We want patterns that:

- Prioritize orders that are harming the solution
- Do not exceed the length of the metal sheet

This is precisely a knapsack problem!

- Priority orders $\rightarrow$ Valuable items
- Order length $\rightarrow$ Item weight

Pricing Problem

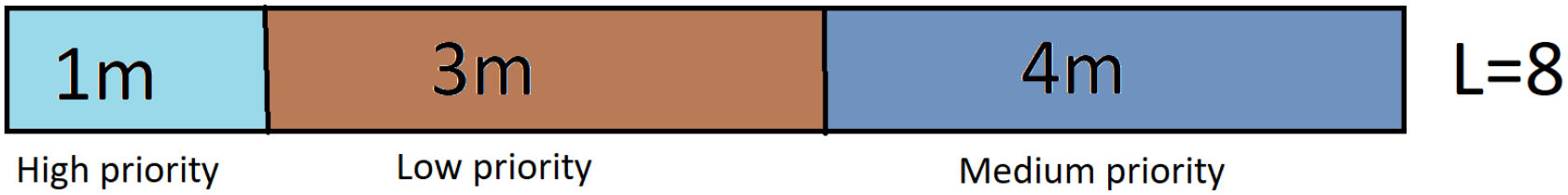
Which new pattern should we add? We want patterns that:

- Prioritize orders that are harming the solution
- Do not exceed the length of the metal sheet

This is precisely a knapsack problem!

- Priority orders $\rightarrow$ Valuable items
- Order length $\rightarrow$ Item weight
- Sheet length $\rightarrow$ Knapsack capacity

Cutting-Stock and Knapsack



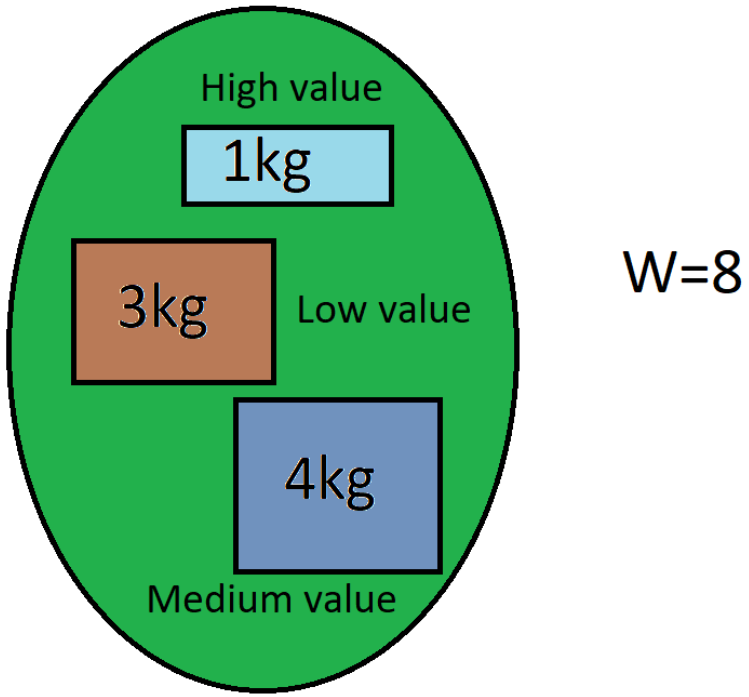


Figure: Cutting stock pattern and knapsack instance

Priority orders?

How should we decide the prices for the items?

Priority orders?

How should we decide the prices for the items?

The dual values of the constraints can give us the answer!

Duality revisited

$$\begin{aligned} \max_{y} \quad & b^\intercal y \\ \text{s.t.} \quad & A^\intercal y \geq c \\ & y \geq 0 \end{aligned}$$

Recall that the dual variable associated with a constraint gives us the infinitesimal increase of the objective per infinitesimal increase of the right-hand side of the constraint.

E.g.: constraint $f(x) \leq b$ having a dual of -3 means that (more or less), $f(x) \leq b + 1$ would decrease the objective by 3.

Duality in Cutting-Stock

In the context of this problem, the dual associated with the demand of item i tells us the cost that a slightly increased demand would have on the number of used sheets.

When do we stop?

We know (very informally) what the benefit of using the generated pattern would be ($c^\intercal x$). We know what the cost of using it would be as well. By using a new pattern, we are using a new roll, thus worsening the solution by 1.

We stop the process when the benefit of the best new pattern doesn't outweigh the cost of using it. So, when $1 - c^\intercal x \geq 0$.

The idea

1. Initialize problem with a small subset of patterns

The idea

1. Initialize problem with a small subset of patterns
2. Optimize the RMP to get the best pattern

The idea

1. Initialize problem with a small subset of patterns
2. Optimize the RMP to get the best pattern
3. Solve the knapsack subproblem

The idea

1. Initialize problem with a small subset of patterns
2. Optimize the RMP to get the best pattern
3. Solve the knapsack subproblem
4. If the best pattern cannot improve RMP, the optimal solution found

The idea

1. Initialize problem with a small subset of patterns
2. Optimize the RMP to get the best pattern
3. Solve the knapsack subproblem
4. If the best pattern cannot improve RMP, the optimal solution found
5. Else, add the new pattern to RMP and go to step 2

The idea

1. Initialize problem with a small subset of patterns
2. Optimize the RMP to get the best pattern
3. Solve the knapsack subproblem
4. If the best pattern cannot improve RMP, the optimal solution found
5. Else, add the new pattern to RMP and go to step 2

I lied a little bit here, but I will explain things later.

```
9 orders | Time original: 0.35 | Time col. generation: 0.04 | Time difference: 0.31
18 orders | Time original: 1.90 | Time col. generation: 0.02 | Time difference: 1.88
19 orders | Time original: 0.77 | Time col. generation: 0.02 | Time difference: 0.75
22 orders | Time original: 4.59 | Time col. generation: 0.01 | Time difference: 4.58
11 orders | Time original: 0.58 | Time col. generation: 0.01 | Time difference: 0.57
11 orders | Time original: 0.52 | Time col. generation: 0.04 | Time difference: 0.48
20 orders | Time original: 7.31 | Time col. generation: 0.03 | Time difference: 7.28
22 orders | Time original: 7.56 | Time col. generation: 0.02 | Time difference: 7.55
14 orders | Time original: 0.40 | Time col. generation: 0.05 | Time difference: 0.36
20 orders | Time original: 1.07 | Time col. generation: 0.05 | Time difference: 1.02
```

Figure: Example of speed of reformulation in some instances

Column Generation

Big linear programs tend to only use a small subset of columns in the optimal solution.

Column Generation

Big linear programs tend to only use a small subset of columns in the optimal solution.

Column generation idea:

- Start with a small subset of columns. Optimize the resulting problem.

Column Generation

Big linear programs tend to only use a small subset of columns in the optimal solution.

Column generation idea:

- Start with a small subset of columns. Optimize the resulting problem.
- Use dual values of the optimal solution to generate new columns

Column Generation

Big linear programs tend to only use a small subset of columns in the optimal solution.

Column generation idea:

- Start with a small subset of columns. Optimize the resulting problem.
- Use dual values of the optimal solution to generate new columns
- If no columns can improve the solution, it is optimal

Column generation

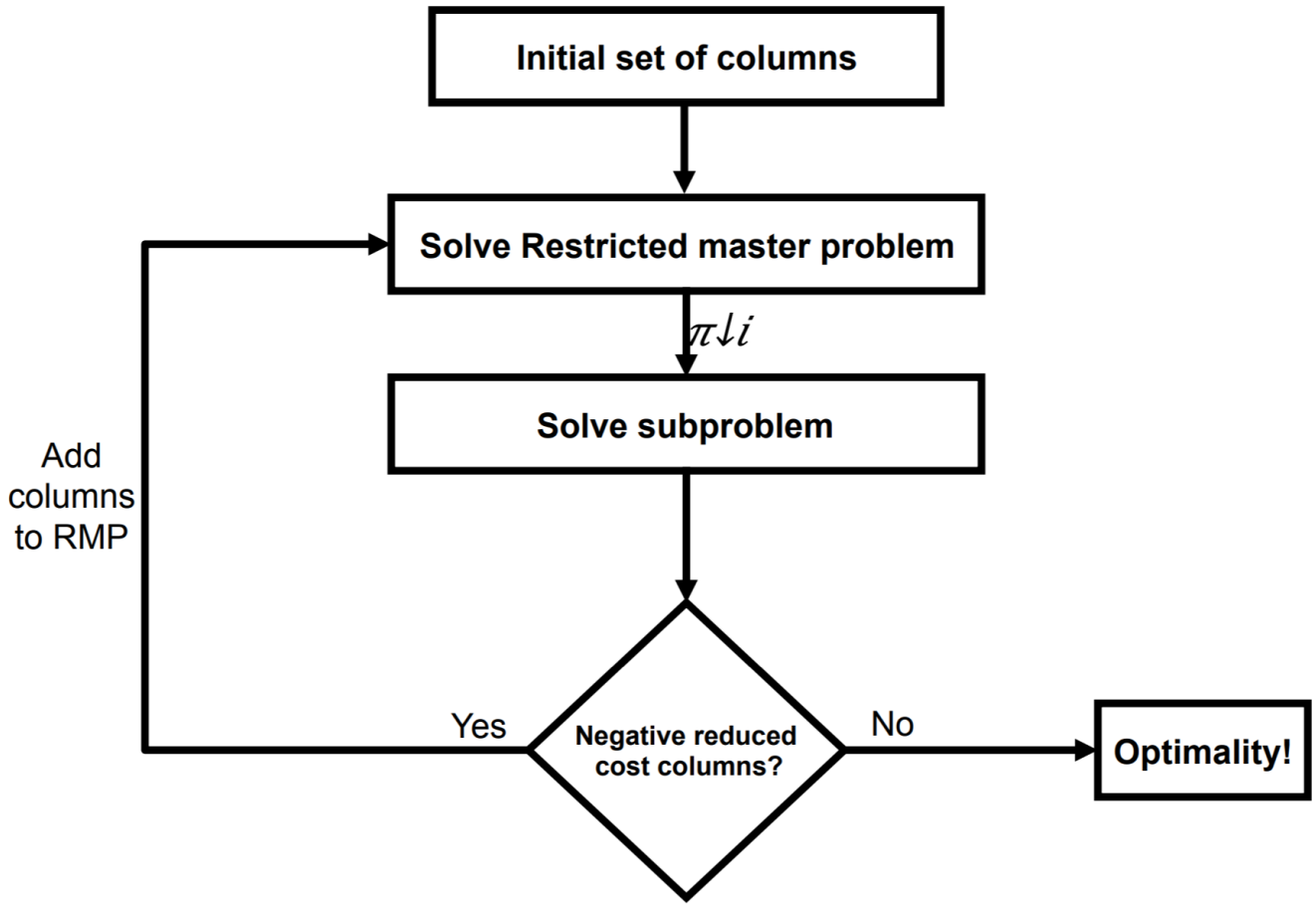


Figure: Column generation Flowchart

Master Problem

With the available columns, pick the ones that optimize the objective.

$$\begin{aligned} \min \quad & c^\intercal x \\ \text{s.t.} \quad & \text{Column satisfies original constraints} \end{aligned}$$

Subproblem

Out of all available columns, pick one that improves the solution of the restricted master problem. How?

Reduced Cost

Get the change in the objective by increasing a variable by a small amount, i.e., the first derivative from a certain point on the polyhedron that constrains the problem.

Reduced Cost

Get the change in the objective by increasing a variable by a small amount, i.e., the first derivative from a certain point on the polyhedron that constrains the problem. The **reduced cost** can be computed in the following manner:

$$c - A^{\intercal} y$$

Subproblem

$$\begin{aligned} \min \quad & c - A^{\mathsf{T}} y \\ \text{s.t.} \quad & \text{Column satisfies original constraints} \end{aligned}$$

Column generation Tips

- The pricing problem should be easy. (Knapsack with dynamic programming!)
- Solve the pricing problem heuristically. How many times does it need to be solved optimally?
- Add multiple columns per iteration.
- Remove columns from the RMP that have been inactive for many iterations.

Branch-and-Price

Column generation is only applicable to linear problems[1] (duality).
We then need to branch on fractional variables/columns/patterns.
We do this by embedding column generation in a Branch-and-Bound tree.

One of the reasons it works so well in the cutting-stock problem is that the linear relaxation of the knapsack is so easy.

[1] This was the lie I talked about earlier.

Branch-and-Price diagram

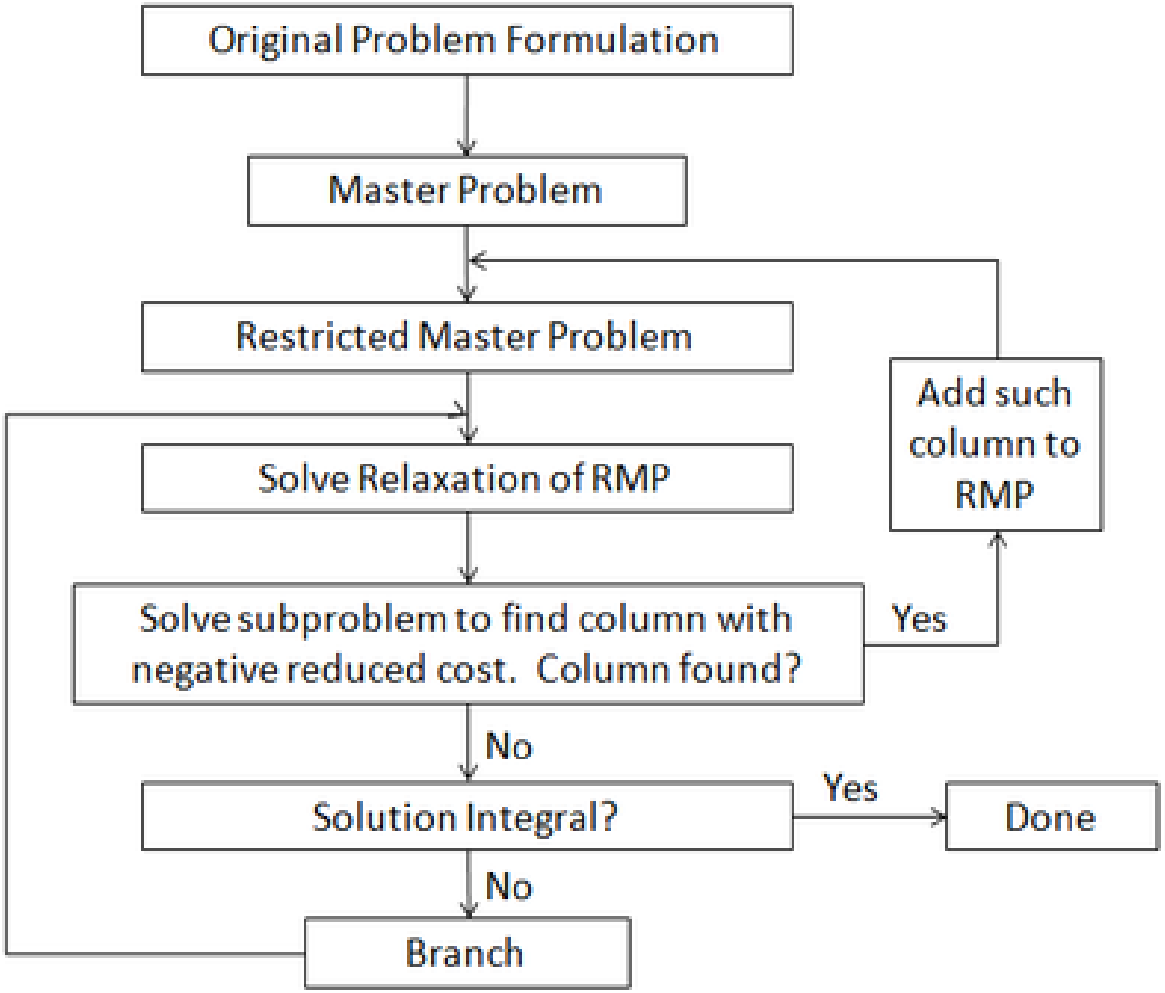
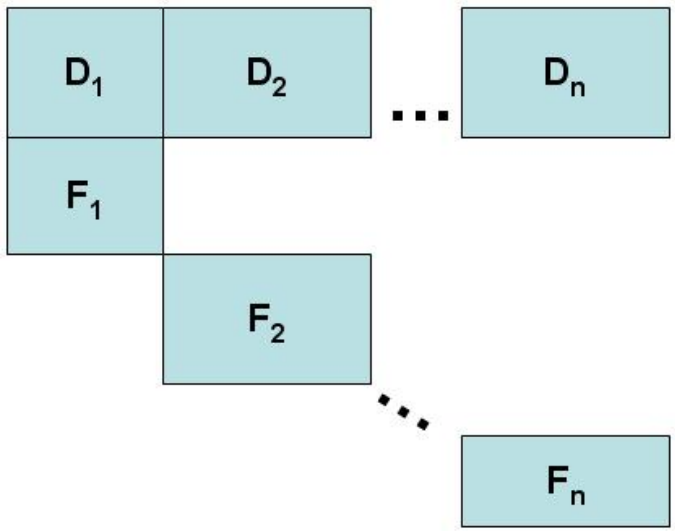


Figure: Branch-and-price diagram

Problem structure

Sometimes difficult problems have a structure that can be explored.

For example, there may be a set of variables that are difficulting the problem:



Problem structure

Other times it's a set of variables that are leading to a difficult problem.

$$
\begin{array}{lll}
\min & a^\intercal x + b^\intercal y & + c^\intercal z \\
\text{s.t.} & Ax & \leq B \\
& Cx + Dy & \leq E \\
& Fx + Gy & + Hz \leq I \\
& x, y, z \geq 0 &
\end{array}
$$

Dantzig-Wolfe Decomposition

When can we use column generation? Is there a systematic way to do it?

Dantzig-Wolfe Decomposition

When can we use column generation? Is there a systematic way to do it?

Yes, with a reformulation of the problem called Dantzig-Wolfe decomposition. First, some preliminaries.

Minkowsi-Weyl Theorem

Theorem

Let $P \subseteq \mathbb{R}^n$ be a bounded convex polyhedron. Then there is a finite set Q such that $P = conv(Q)$.

Minkowsi-Weyl Theorem

Theorem

Let $P \subseteq \mathbb{R}^n$ be a bounded convex polyhedron. Then there is a finite set Q such that $P = conv(Q)$.

In other words

$$P = \{x \in R^n \mid x = \sum_{q \in Q} \lambda_q x_q, \sum_q \lambda_q = 1, \lambda \geq 0\}$$

So, a bounded convex polyhedron can either be represented by the intersection of half-spaces (constraints), or by a convex combination of its extreme points. These representations are equivalent.

Note

Both the theorem and the following methods work for unbounded polyhedra, when including conic combinations of extreme rays. However, this would introduce unnecessary complexity, so I will present things like this. Nevertheless, I leave here the definition of extreme rays.

Definition

A ray (semirreta) r of a polyhedron P is said to be an **extreme ray** if

$$\nexists r_1, r_2 \in P, \lambda \in]0,1[|\ r = \lambda r_1 + (1-\lambda) r_2$$

Where r_1, r_2 are rays.

Extreme ray example

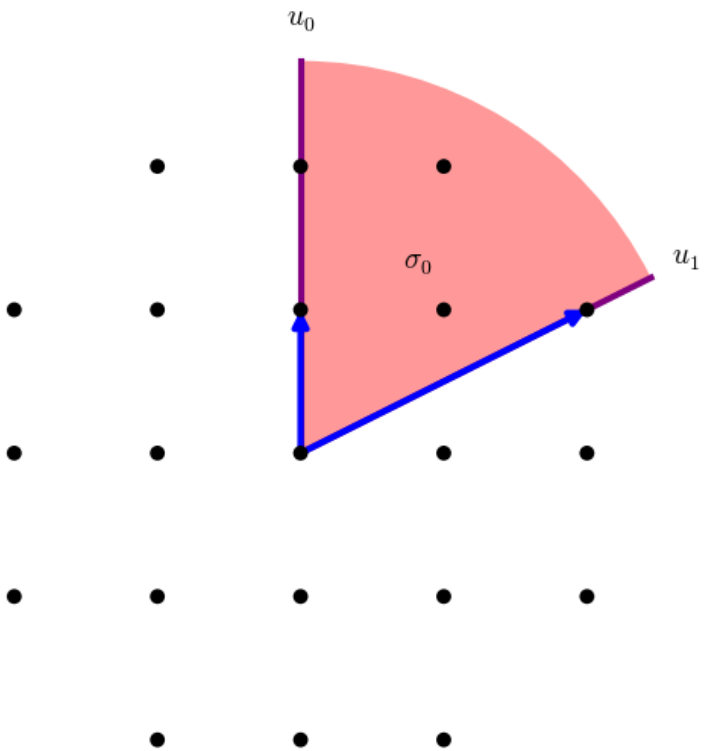


Figure: Example of extreme rays

Dantzig-Wolfe structure

Imagine an LP with a set of easy and a set of complicating constraints:

$$\begin{aligned} \min \quad & c^\intercal x \\ \text{s.t.} \quad & Ax \geq b \\ & Dx \geq d \\ & x \geq 0 \end{aligned}$$

For example, a problem with linking constraints fits this description.

Dantzig-Wolfe reformulation

Dantzig-Wolfe reformulation is the reformulation of the easy constraints of the LP above to the extreme points representation. We do this for the set $X = \{x \geq 0 \mid Dx \geq d\}$.

Dantzig-Wolfe reformulation

Dantzig-Wolfe reformulation is the reformulation of the easy constraints of the LP above to the extreme points representation. We do this for the set $X = \{x \geq 0 \mid Dx \geq d\}$.

With extreme points $Q = \{x_1, \ldots, x_{|Q|}\}$

$$x = \sum_{q \in Q} \lambda_q x_q$$

$$\sum_{q \in Q} \lambda_q = 1$$

$$\lambda_q \geq 0$$

Dantzig-Wolfe reformulation

Dantzig-Wolfe reformulation is the reformulation of the easy constraints of the LP above to the extreme points representation. We do this for the set $X = \{x \geq 0 \mid Dx \geq d\}$.

With extreme points $Q = \{x_1, \ldots, x_{|Q|}\}$

$$x = \sum_{q \in Q} \lambda_q x_q$$

$$\sum_{q \in Q} \lambda_q = 1$$

$$\lambda_q \geq 0$$

Remember, this is characterizing every point in X.

Dantzig-Wolfe master problem

Replacing x in the original LP with this representation we get

$$\begin{aligned}
\min \quad & c^\intercal \left(\sum_{q \in Q} \lambda_q x_q \right) \\
\text{s.t.} \quad & A \left(\sum_{q \in Q} \lambda_q x_q \right) \geq b \\
& \sum_{q \in Q} \lambda_q = 1 \\
& \lambda_q \geq 0
\end{aligned}$$

Dantzig-Wolfe master problem

Replacing x in the original LP with this representation we get

$$\begin{aligned}
\min \quad & c^\intercal \left(\sum_{q \in Q} \lambda_q x_q \right) \\
\text{s.t.} \quad & A \left(\sum_{q \in Q} \lambda_q x_q \right) \geq b \\
& \sum_{q \in Q} \lambda_q = 1 \\
& \lambda_q \geq 0
\end{aligned}$$

This is equivalent to the original problem. Why?

Dantzig-Wolfe master problem

Rearranging the terms and defining $c_q := c^\intercal x_q$, $a_q := Ax_q$ we get the Dantzig-Wolfe master problem:

$$
\begin{aligned}
\min \quad & \sum_{q \in Q} c_q \lambda_q \\
\text{s.t.} \quad & \sum_{q \in Q} a_q \lambda_q \geq b \\
& \sum_{q \in Q} \lambda_q = 1 \\
& \lambda_q \geq 0
\end{aligned}
$$

Restricted master problem

This problem is gigantic (there is a very large number of extreme points). So we need to apply column generation. So, we start with a small subset of extreme points.

$$
\begin{aligned}
\min \quad & \sum_{q \in Q'} c_q \lambda_q \\
\text{s.t.} \quad & \sum_{q \in Q'} a_q \lambda_q \geq b \qquad (\pi) \\
& \sum_{q \in Q'} \lambda_q = 1 \qquad (\pi_0) \\
& \lambda_q \geq 0, \\
& \text{with } |Q'| << |Q|.
\end{aligned}
$$

DW-decomposition pricing problem

How do we add new columns? Using the dual variables π from the RMP, we arrive at

$$\min_{x_q \in Q} \left(c^\intercal - \pi^\intercal A\right) x_q$$

DW-decomposition pricing problem

How do we add new columns? Using the dual variables π from the RMP, we arrive at

$$\min_{x_q \in Q} \left(c^\intercal - \pi^\intercal A\right) x_q$$

Which is equivalent to:

$$\begin{aligned} \min \quad & \left(c^\intercal - \pi^\intercal A\right) x \\ \text{s.t.} \quad & Dx \geq d \\ & x \geq 0 \end{aligned}$$

This is an easy problem!

Cases for pricing problem

- Optimal solution < 0. We identified an extreme point that improves the solution. Add column $(Ax_{q^*}\ 1)^\intercal$ to master problem.
- Optimal solution ≥ 0. We found the optimal solution.

Why does this work?

The restricted master problem is very easy (very few columns).
Adding a new column is also easy. Why?

Why does this work?

The restricted master problem is very easy (very few columns). Adding a new column is also easy. Why?

Because we generate columns by solving the pricing problem, which corresponds to the easy constraints!

Recovering solution to original problem

Solving the DW-reformulation, we get the solution in terms of the extreme points. Projecting back to the original variables is easy:

$$x = \sum_{q \in Q} \lambda_q x_q + \sum_{r in R} \lambda_r x_r$$

Exercises

Suggestions

Light read on column-generation: Link

A more in-depth explanation of column-generation/Dantzig-Wolfe: Link

Benders' Decomposition

What if instead of easy subproblems linked by a few constraints, we have a difficult problem that becomes easy if we fix a few variables?

Benders' Decomposition

What if instead of easy subproblems linked by a few constraints, we have a difficult problem that becomes easy if we fix a few variables? For example, a MIP with integer variables fixed, or a problem with linking variables.

Benders' Decomposition Idea

Divide the problem into two subproblems. The first problem (the outer problem), keeps picking the values to fix the difficult variable.

Benders' Decomposition Idea

Divide the problem into two subproblems. The first problem (the outer problem), keeps picking the values to fix the difficult variable. The second problem (the inner problem), uses duality to tell the outer problem which values for y will work best.

Benders' Decomposition Idea

Divide the problem into two subproblems. The first problem (the outer problem), keeps picking the values to fix the difficult variable. The second problem (the inner problem), uses duality to tell the outer problem which values for y will work best. They keep interacting until they find an ϵ-optimal solution.

Algorithm Idea

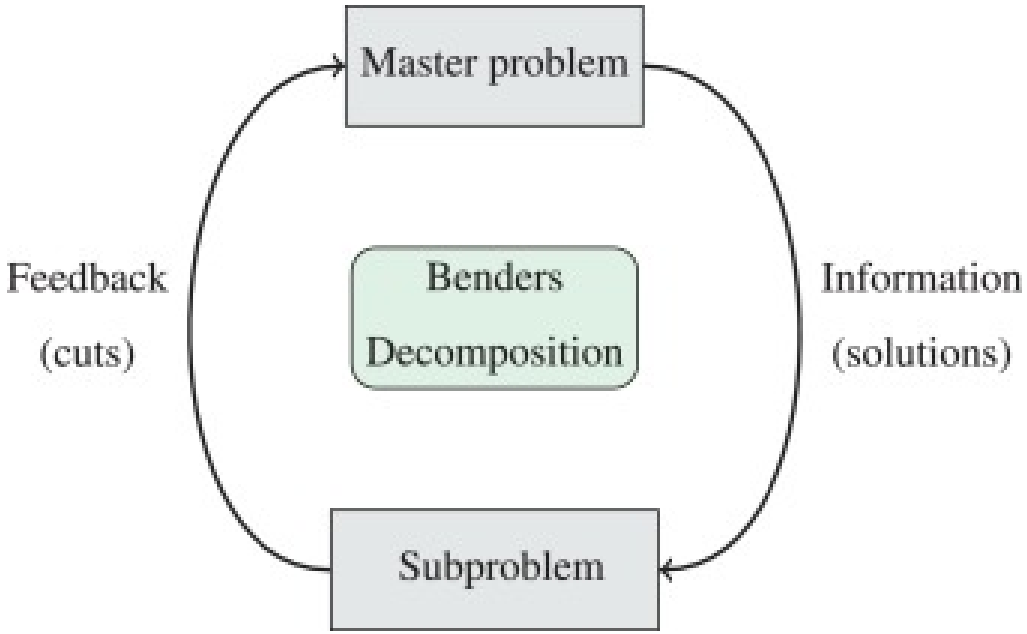


Figure: Benders' Decomposition Idea

Benders' Decomposition

Consider the following problem:

$$\begin{aligned} \min_{x,y} \quad & c^T x + d^T y \\ \text{s.t.} \quad & Ax + By \geq b \\ & y \in Y \\ & x \geq 0 \end{aligned}$$

Here, y will represent the difficult variables.

Benders' Decomposition

Fixing y to $\overline{y}$, we get the following easy problem (LP).

$$\begin{aligned} \min_{x,y} \quad & c^\intercal x + d^\intercal \overline{y} \\ \text{s.t.} \quad & Ax + B\overline{y} \geq b \\ & x \geq 0 \end{aligned}$$

Benders' Decomposition

The original problem is equivalent to optimizing over the y variables first, and then over the x variables.

$$\min_{y \in Y} [d^T y + \min_{x \geq 0} \{c^T x \mid Ax + B\overline{y} \geq b\}]$$

The problem with a fixed y has the following dual:

$$\begin{aligned} \min_{u} \quad & (b - B\overline{y})^{\mathsf{T}} u + d^{\mathsf{T}} \overline{y} \\ \text{s.t.} \quad & A^{\mathsf{T}} u \leq c \\ & u \geq 0 \end{aligned}$$

Benders' Decomposition

The problem with a fixed y has the following dual:

$$\begin{aligned} \min_{u} \quad & (b - B\overline{y})^{\mathsf{T}} u + d^{\mathsf{T}} \overline{y} \\ \text{s.t.} \quad & A^{\mathsf{T}} u \leq c \\ & u \geq 0 \end{aligned}$$

Recall that in LP the dual and the primal have the same optimal solution.

Benders' Decomposition

The original problem is equivalent to:

$$\min_{y \in Y} [d^\intercal y + \max_{u \geq 0} \{(b - By)^\intercal u \mid A^\intercal u \leq c\}]$$

Outer and inner problem

Outer problem: Pick y's from Y, initially with no regard for feasibility or optimality.

Inner Problem: With a fixed y use duality theory to add constraints to the outer problem, saying which y's have been infeasible, which have led to bad solutions, and which have led to good solutions.

Outer and inner problem

Outer problem: Pick y's from Y, initially with no regard for feasibility or optimality.

Inner Problem: With a fixed y use duality theory to add constraints to the outer problem, saying which y's have been infeasible, which have led to bad solutions, and which have led to good solutions.

How can the inner problem provide that information?

Duality revisited

Proposition

Let P be an LP and D the corresponding dual. We have the following:

1. *D is infeasible $\implies$ P is unbounded*
2. *D is unbounded $\implies$ P is infeasible*
3. *D has an optimal solution y^* $\implies$ P has an optimal solution x^* and $b^\intercal y \leq c^\intercal x^*$*

Proof

3 is simply the weak duality theorem, already proven. As the dual of the dual is the primal problem, then we have that 1 $\iff$ 2.

Optimality cuts

If the inner problem is feasible, we can derive bounds for the outer problem. We know that the optimal solution is at least as good as this one.

$$z \geq (b - By)^{\mathsf{T}} \overline{u} + d^{\mathsf{T}} y$$

Feasibility cuts

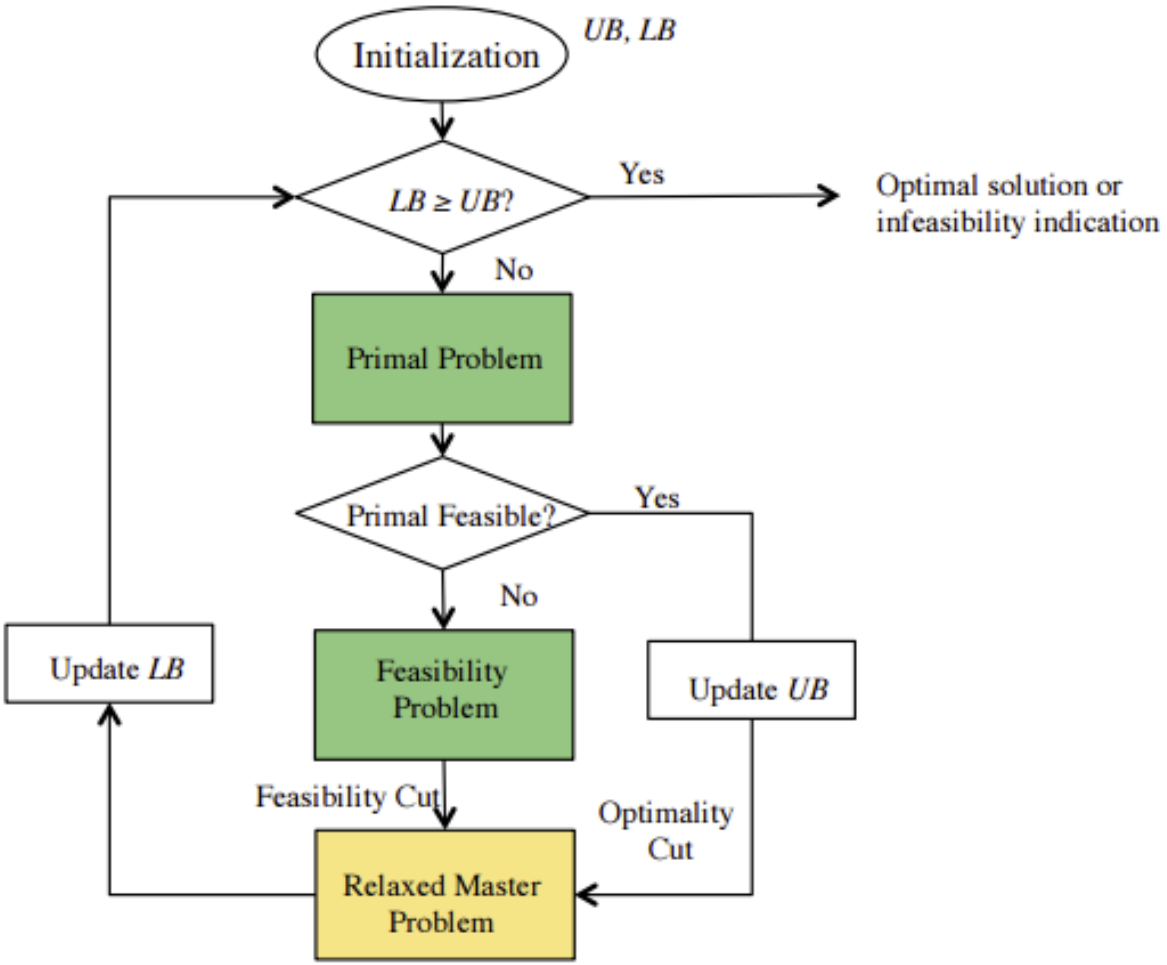
If the inner problem is unbounded, we know the fixed y cannot be chosen. Furthermore, we can also exclude other y's as well.

$$(b - By)^{\intercal}\overline{u} \leq 0$$

When does it stop?

The inner problem gives us lower bounds LB. The outer problem gives us upper bounds UB. Stop when their difference is smaller than a predetermined tolerance, $UB - LB \leq \varepsilon$.

Benders' Decomposition Diagram



Example

t

Exercises

Let us take a quick detour to talk about the standard method for solving the knapsack problem.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0, 1\} \end{aligned}$$

Pick which items to take in order to maximize total value while not exceeding the knapsack's capacity.

Dynamic Programming

- Look at the problem recursively.
- Use a table to store previously found solutions.
- See whether using a new item improves the result.

Dynamic Programming in Knapsack

The table B will store the best solution using items from 1 to n which uses at most 1 to W total weight. The entry $B_{i,j}$ refers to the best solution using items 1 to i, and at most weight j.

Dynamic Programming in Knapsack

The table B will store the best solution using items from 1 to n which uses at most 1 to W total weight. The entry $B_{i,j}$ refers to the best solution using items 1 to i, and at most weight j.

For example, entry $B_{3,10}$ tells us the best solution which uses items $1, 2, 3$ whose weight does not exceed 10.

Where can we find the optimal solution?

Dynamic Programming in Knapsack

The table B will store the best solution using items from 1 to n which uses at most 1 to W total weight. The entry $B_{i,j}$ refers to the best solution using items 1 to i, and at most weight j.

For example, entry $B_{3,10}$ tells us the best solution which uses items $1, 2, 3$ whose weight does not exceed 10.

Where can we find the optimal solution?The entry at $B_{n,W}$ is the optimal solution since we can use every item and the knapsack capacity.

Calculating entries

How can we calculate an entry in the table?
Notation: w_i, p_i the weight and price of item i, and W the knapsack capacity.

Calculating entries

How can we calculate an entry in the table?
Notation: w_i, p_i the weight and price of item i, and W the knapsack capacity.

Entries $B_{0,j} = 0$, since we can pick no items. The same is true for entries $B_{i,0}$, as we can use no weight.
If we can use an additional item, either it belongs to the solution or it doesn't. The optimal solution will be the maximum of these two possibilities.

Calculating entries

How can we calculate an entry in the table?
Notation: w_i, p_i the weight and price of item i, and W the knapsack capacity.

Entries $B_{0,j} = 0$, since we can pick no items. The same is true for entries $B_{i,0}$, as we can use no weight.
If we can use an additional item, either it belongs to the solution or it doesn't. The optimal solution will be the maximum of these two possibilities.

$$B_{i,j} = \max(B_{i-1,j}, V_i + B_{i-1,j-w_i})$$

(Terrible) Visualization

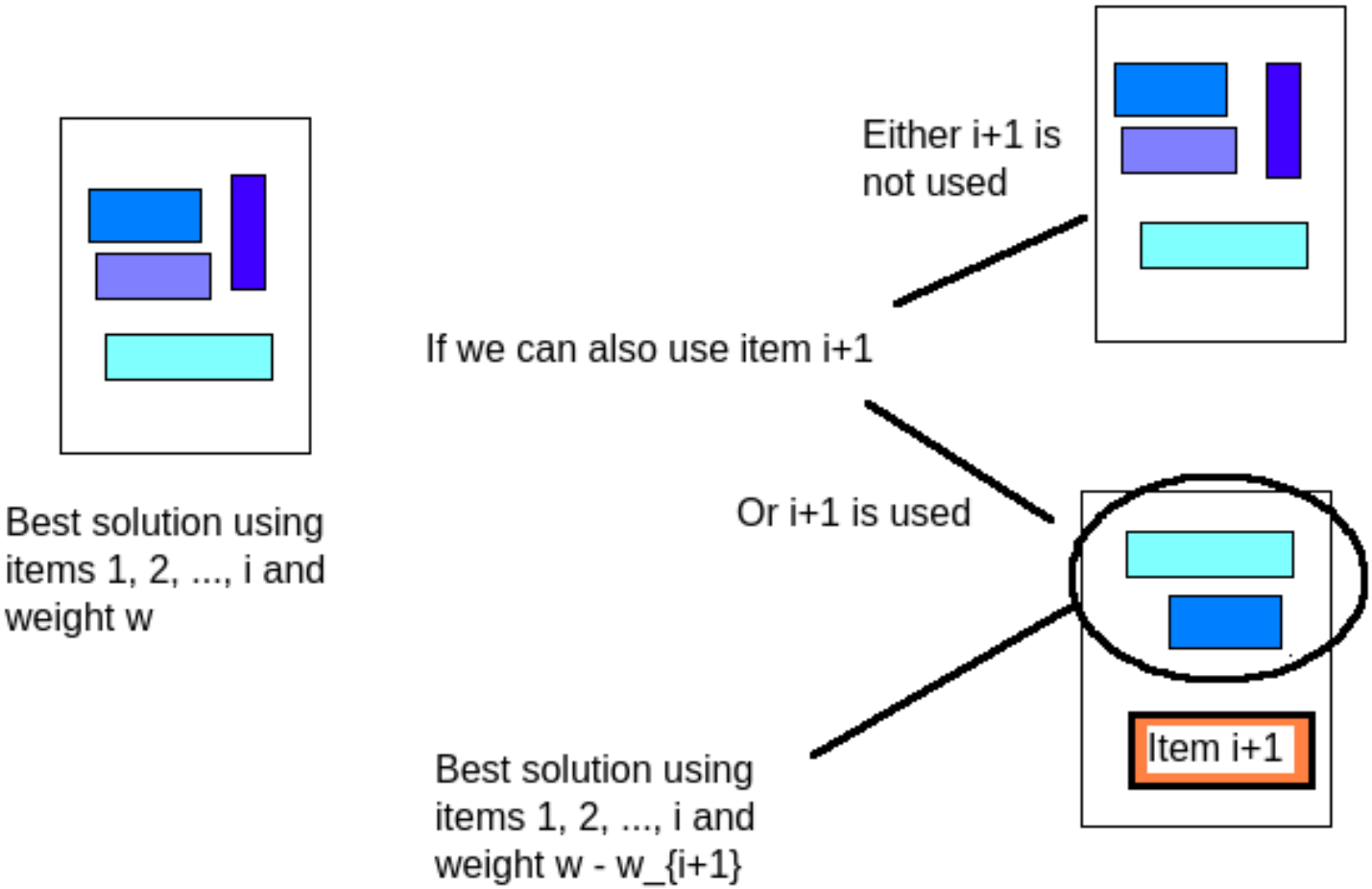


Figure: Example of dynamic programming decision

Example

Item	Weight	Value
1	1	10
2	2	15
3	3	40

Knapsack capacity: 6

Example

If no items can be used, the best objective is 0. If no weight can be used, the best objective is also 0.

Item\Weight	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0						
2	0						
3	0						

Example

If we can use item 1, then the best objective is 0 until we can use w_1 weight, and p_1 from then on.

Item\Weight	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	10	10	10	10	10	10
2	0						
3	0						

Example

Now we can use items 1 and 2.

Item\Weight	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	10	10	10	10	10	10
2	0	10	15	25	25	25	25
3	0						

To get entry $B_{2,3}$, we have to calculate if using item 2 improves the previous best solution of 10, at $B_{1,3}$. If we pick 2, then we must use $w_2 = 2$ weight. The best solution with weight 3 (the current available weight) minus 2 (item 2 weight) is at $B_{1,3-2} = 10$. $10 + v_2 = 10 + 15 > 10$. So using item 2 is better.

Example

Item\Weight	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	10	10	10	10	10	10
2	0	10	15	25	25	25	25
3	0	10	15	40	50	55	65

How can we deduce the optimal solution?

Start at the bottom right cell ($B_{n,W}$)

- If current cell $B_{i,j} = B_{i-1,j}$, we didn't take item i. Go to $B_{i-1,j}$.
- If $B_{i,j} \neq B_{i-1,j}$ we took item i. Go to $B_{i-1,j-w_i}$.
- Repeat until we reach $B_{0,0}$.