Topics in Optimization

João Pedro Gonçalves Dionísio

March 7, 2023

Course Structure

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

Github Page

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

- 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

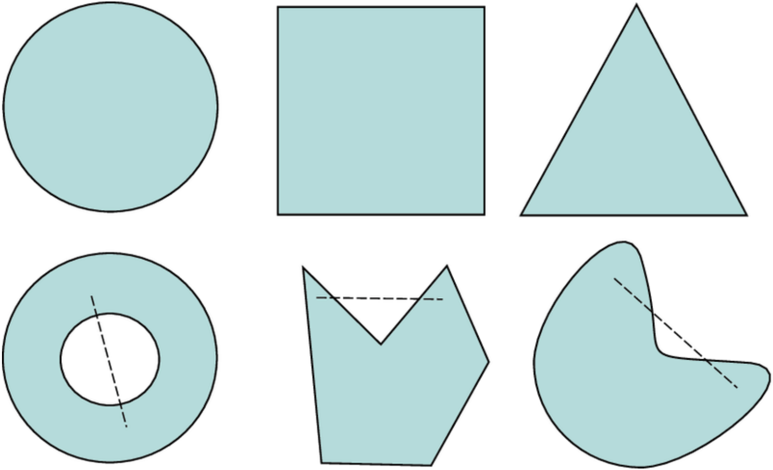


Figure: Examples of convex and non-convex sets

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

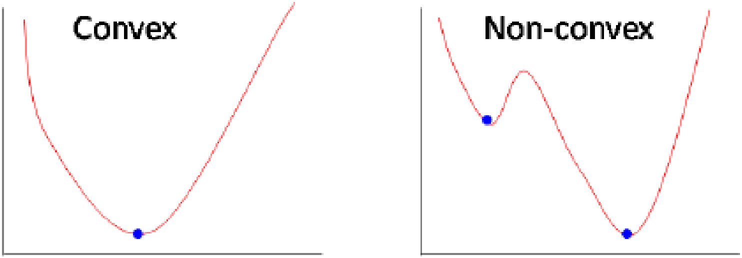


Figure: Examples of convex and non-convex functions

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

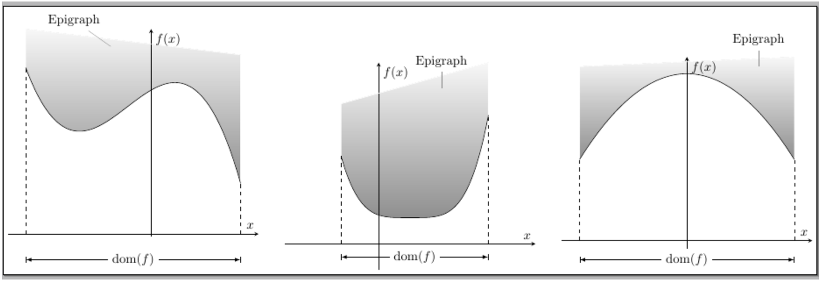


Figure: Examples of epigraph

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

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

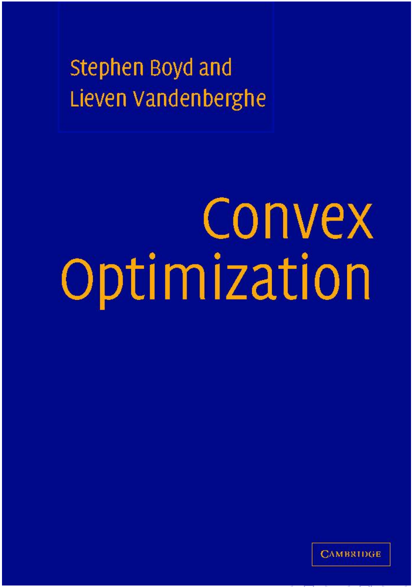
$f, g_i, i = 1 \ldots, m$ convex functions. It implies that $\bigcap_{i \in [m]} \mathrm{epi}(g_i)$ forms a convex set.

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

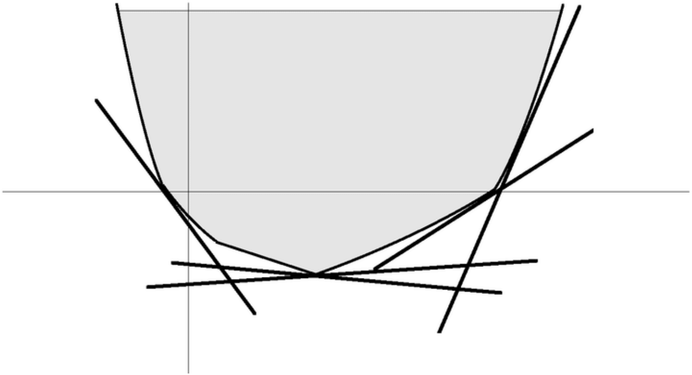


Figure: Visual representation

Duality

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

Duality

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.

Proof sketch

Definition

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

Proof sketch

Definition

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

Lemma

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

Algorithm for solving LPs

With this theorem, we have can create an algorithm to solve LPs.
How? How many vertices do we need to check?

Simplex Motivation

Linear programming is a convex optimization problem. Local optimality $\implies$ Global optimality. We only need to visit vertices that do not worsen the current solution.

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 domain's border.

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

Successive iterations decrease the value of μ.

IPM Motivation

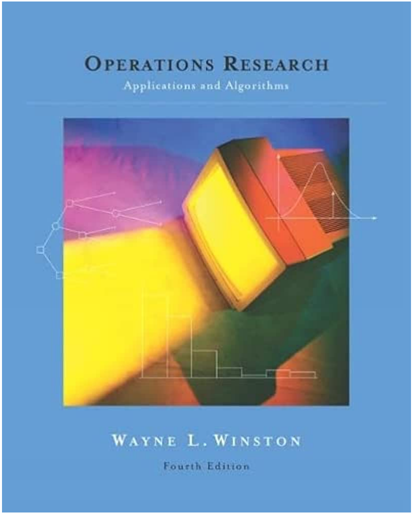
We replace the constraints with what is called a *barrier function*, most commonly a logarithm, to discourage solutions close to the domain's border.

$$\min_{x} \quad c^{\mathsf{T}}x - \mu \sum_{i=1}^{m} log(c_i(x))$$

Successive iterations decrease the value of μ. IPM has a

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

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 number of the power set of size n require us to write $O(2^n)$ numbers. What about $2n$?

What about multiplying two numbers?

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.

Vertex-Cover

Given a graph $G(V, E)$ is there a subset of vertices V' such that $|V'| = k$ and

$$x_u + x_v \geq 1, \forall (uv) \in E$$
$$x_v \in \{0, 1\}$$

where $x_k = \mathbb{1}_{x_k \in V'}$

Knapsack Decision Problem

Can you fit items with value and weight into a bag such that the total weight does not exceed W and the total value is at most k?

$$\sum_{i=1}^{n} w_i x_i \leq W$$
$$x_i \in \{0, 1\}$$

Cutting-Stock Decision Problem

Given

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

Traveling-Salesman Decision Problem

Given a graph $G(V,E)$ with weighted edges, can you find a Hamiltonian tour with total weight less than k?

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

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

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

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

Reductions

Other complexity classes

A problem is **NP**-hard if every problem in **NP** can be reduced to it. If it is also in **NP**, it is called **NP**-complete.

Some classes visualized

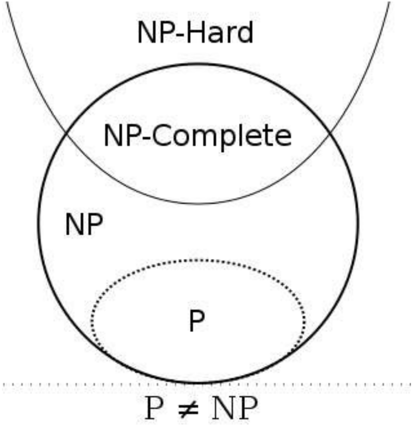


Figure: Some complexity classes

Min Vertex-Cover

Min Vertex-Cover

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

Max Knapsack

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

Cutting-Stock

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

Traveling-Salesman

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{i \neq j, i=1}^{n} x_{ij} = 1, j \in [n] \\
& \sum_{i \neq j, j=1}^{n} x_{ij} = 1, i \in [n] \\
& \sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1, \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2 \\
& x_{ij} \in \{0, 1\}
\end{aligned}
$$

Exercises

Integer Programming

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

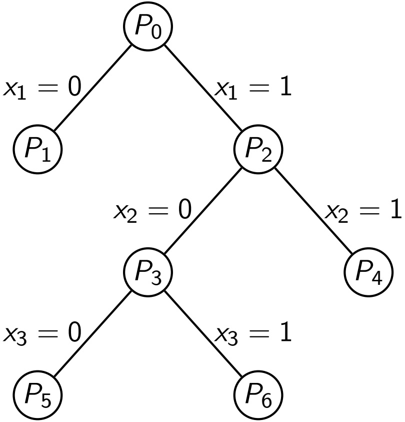
Non-convex

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

Linear Relaxations

Relaxing constraints provides a bound on the optimal solution.
Can we use this to solve IPs?

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

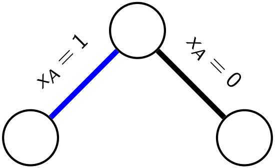
1. Sort items by price density (price/weight)
2. Pick items until capacity is exceeded
3. Remove the excess of the last item

Knapsack instance

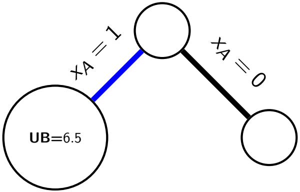
Item	Weight	Value	Value/Weight
A	3	5	1.67
B	2	3	1.5
C	1	1	1

Knapsack capacity: 4

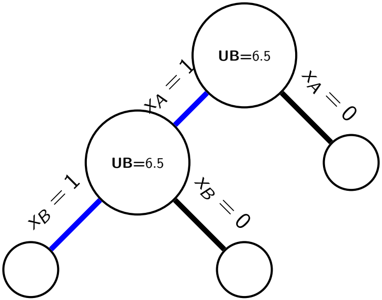
Branch and Bound



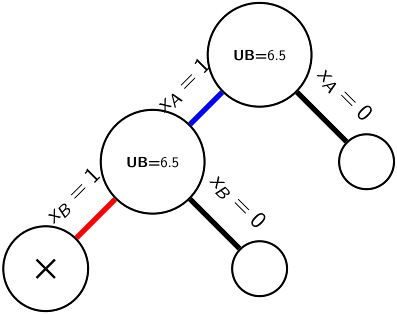
Branch and Bound



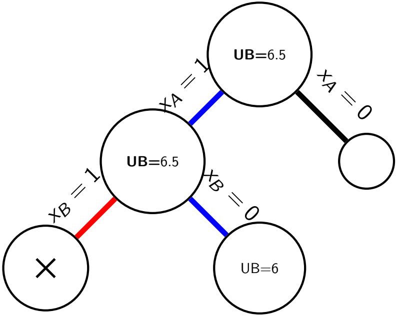
Branch and Bound



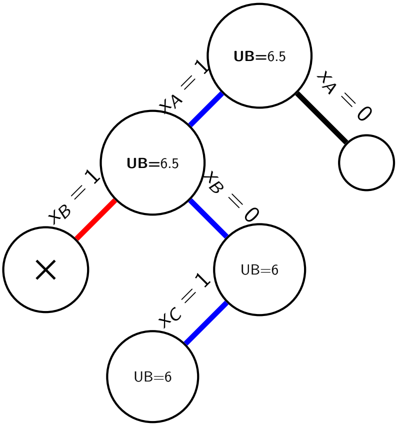
Branch and Bound



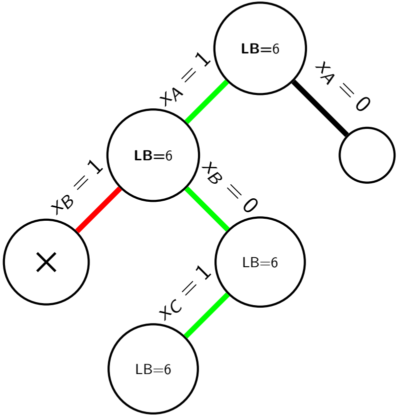
Branch and Bound



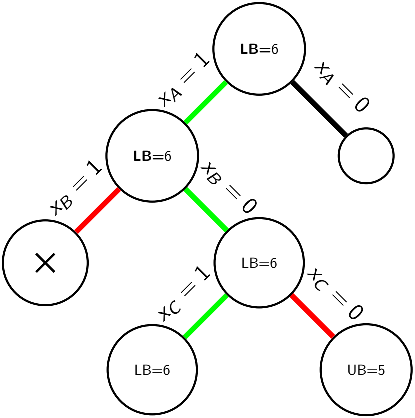
Branch and Bound



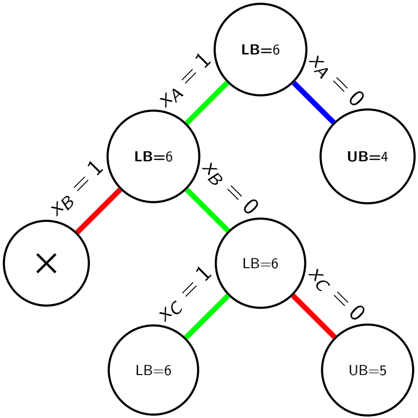
Branch and Bound



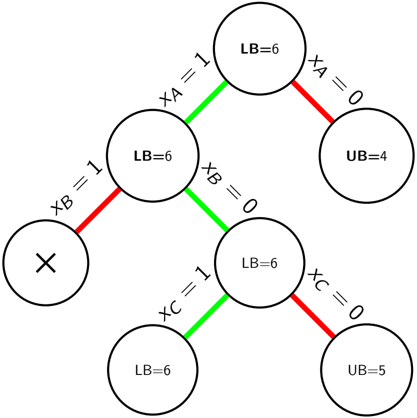
Branch and Bound



Branch and Bound



Branch and Bound



Heuristics

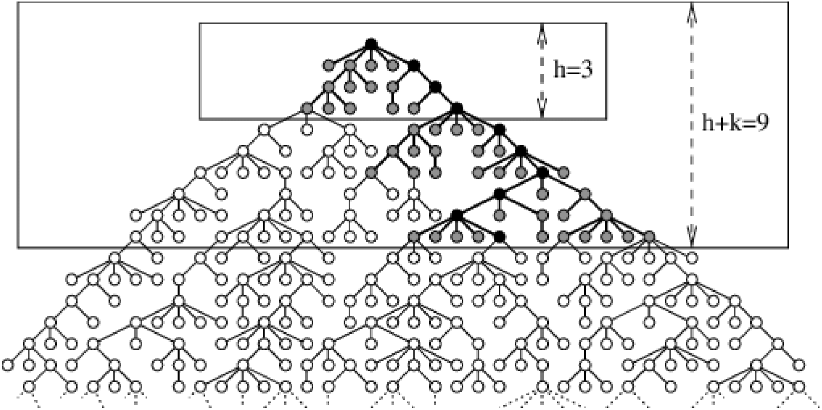


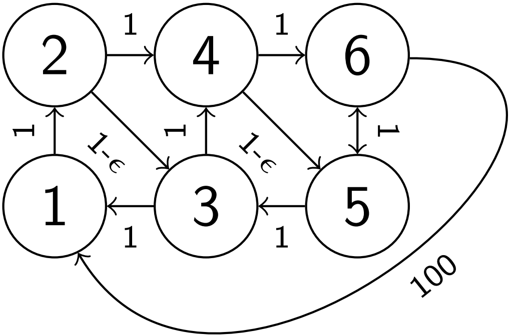
Figure: Branch and Bound trees can get very big. How big?

Heuristics

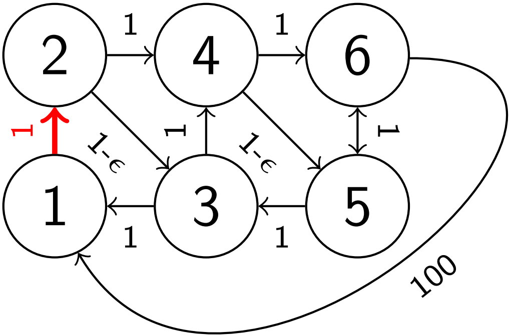
We will focus on heuristics for the TSP.

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \\
& \sum_{i \neq j, i=1}^{n} x_{ij} = 1, j \in [n] \\
& \sum_{i \neq j, j=1}^{n} x_{ij} = 1, i \in [n] \\
& \sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1, \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2 \\
& x_{ij} \in \{0, 1\}
\end{aligned}
$$

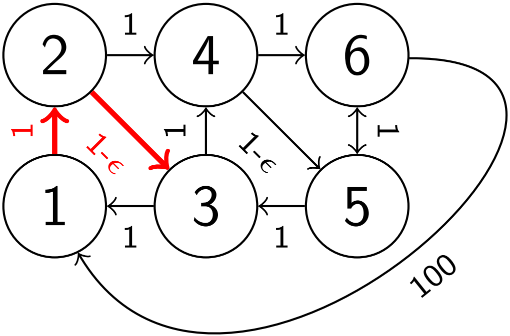
Nearest Neighbor

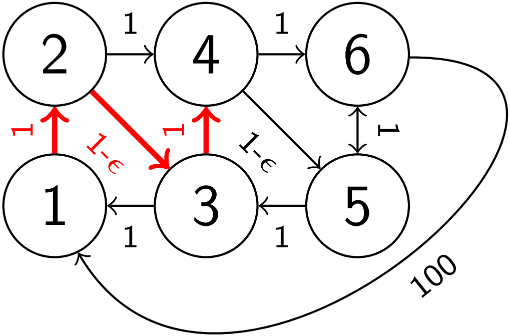


Nearest Neighbor

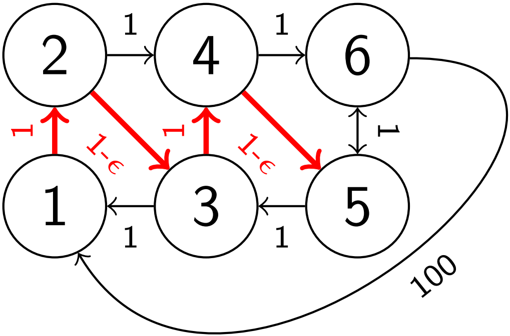


Nearest Neighbor

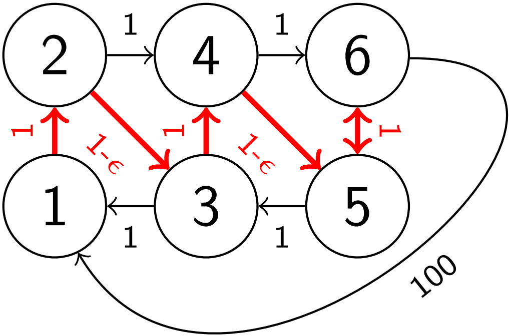



2
4
6
1
3
5
1
1
1
1-ϵ
1
1-ϵ
1
1
1
100

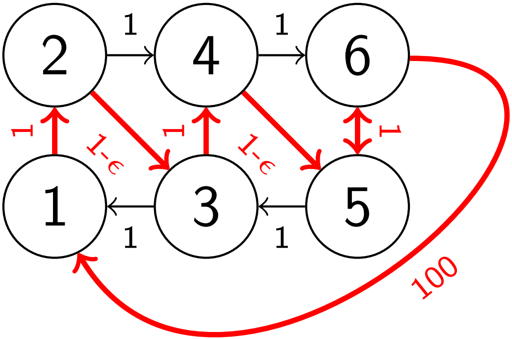
Nearest Neighbor



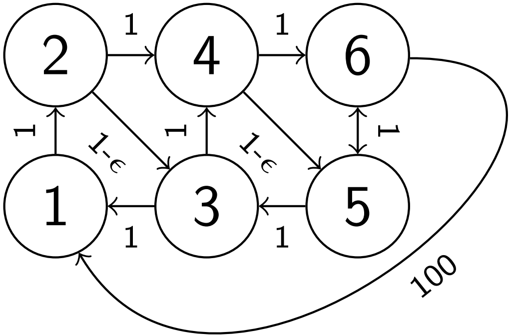
Nearest Neighbor



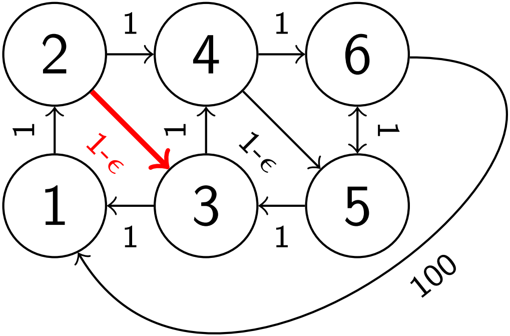
Nearest Neighbor



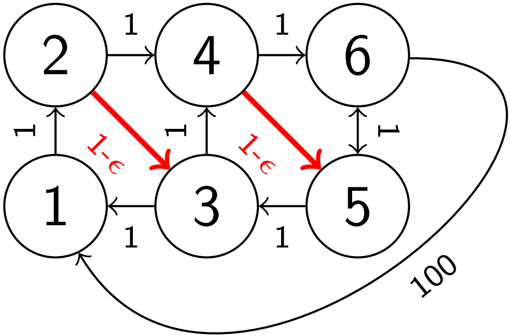
Greedy Algorithm



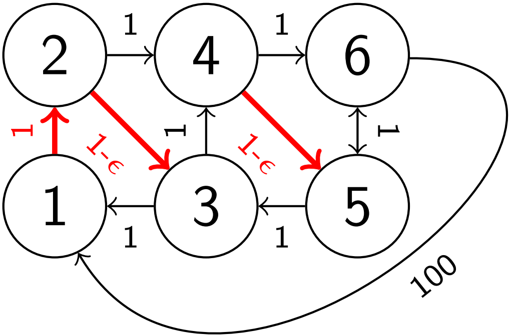
Greedy Algorithm



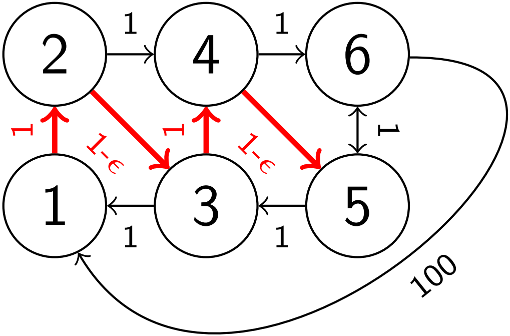
Greedy Algorithm



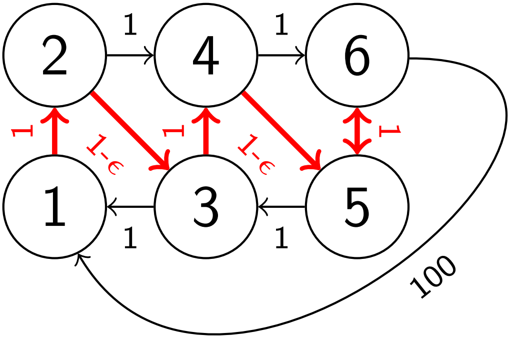
Greedy Algorithm



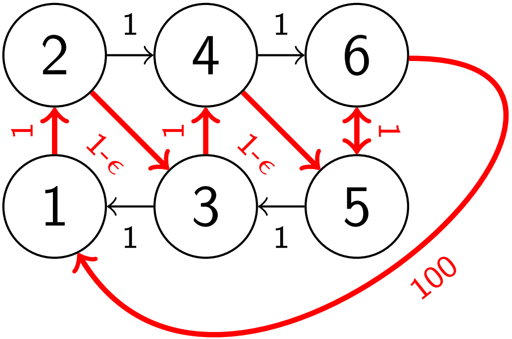
Greedy Algorithm



Greedy Algorithm



Greedy Algorithm



2-OPT algorithm

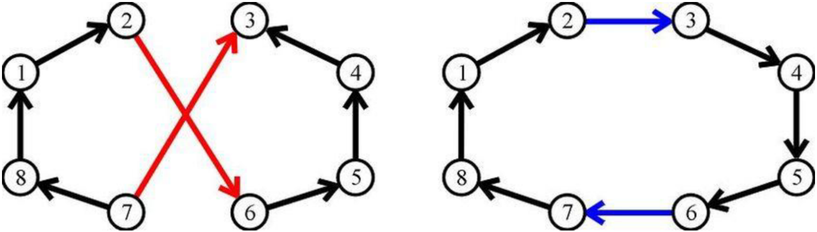


Figure: If two edges cross, we can find a strictly better solution

TSP: 2-opt visualization

Large TSP

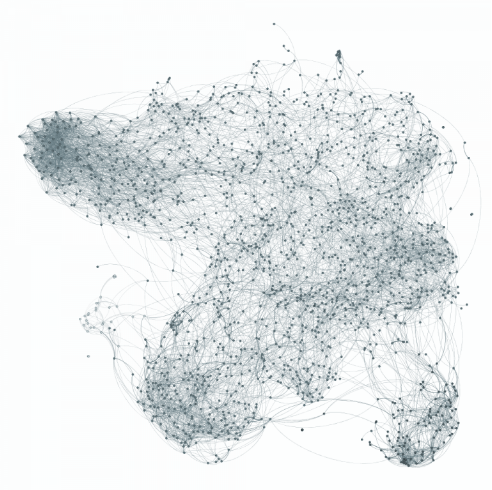


Figure: Heuristics are for large instances

Approximation algorithms

Some heuristics can have theoretical guarantees of their solution quality.

We will study a 2-approximation algorithm for TSP where the cost function satisfies the triangle inequality.

Minimum spanning tree

Definition

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

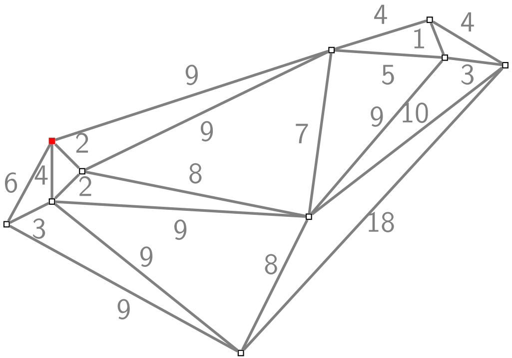


Figure: TSP instance

Minimum spanning tree

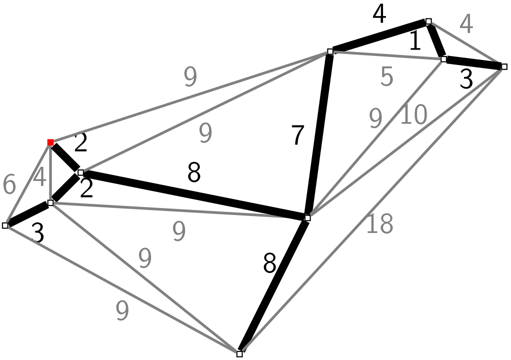


Figure: Minimal spanning tree

Depth-first search

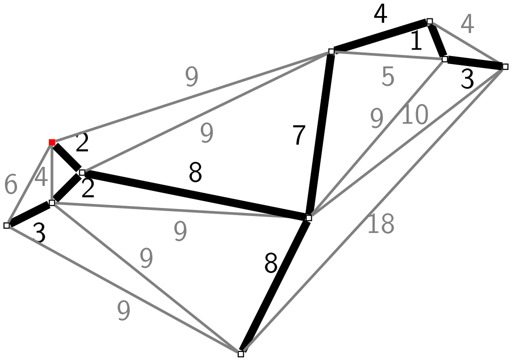


Figure: DFS

Depth-first search

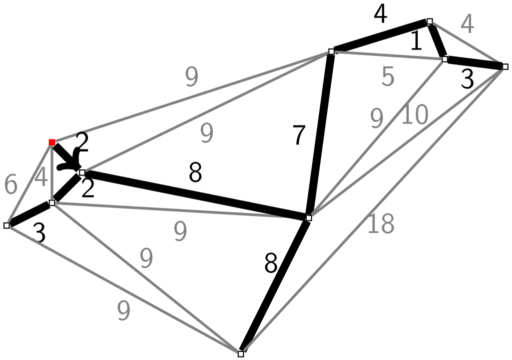


Figure: DFS

Depth-first search

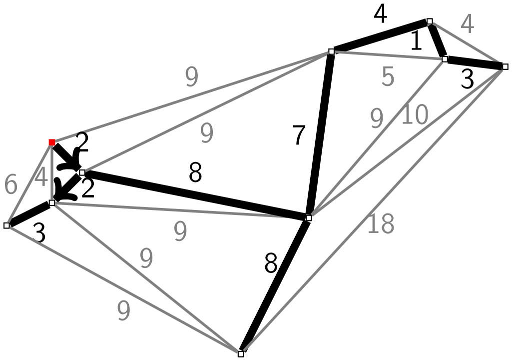


Figure: DFS

Depth-first search

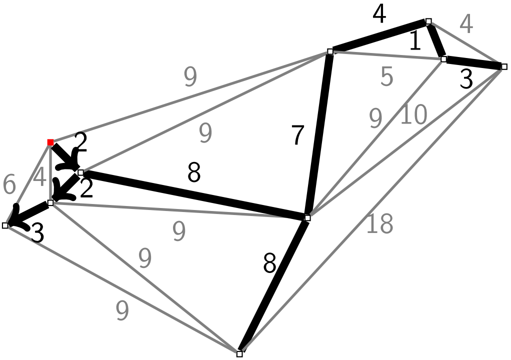


Figure: DFS

Depth-first search

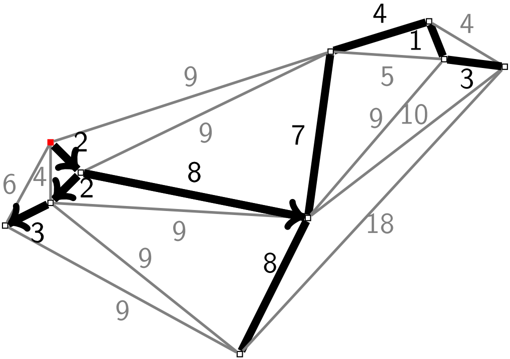


Figure: DFS

Depth-first search

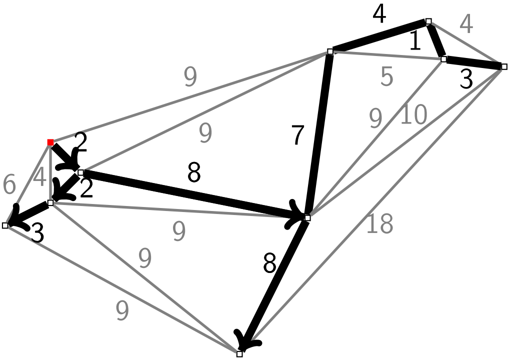


Figure: DFS

Depth-first search

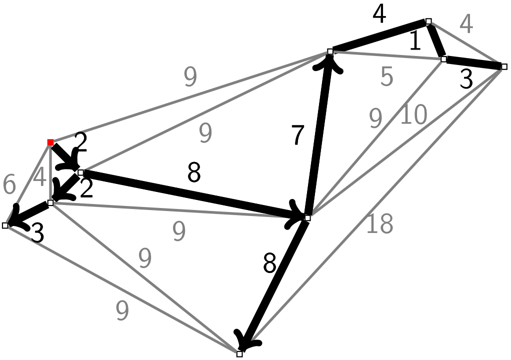


Figure: DFS

Depth-first search

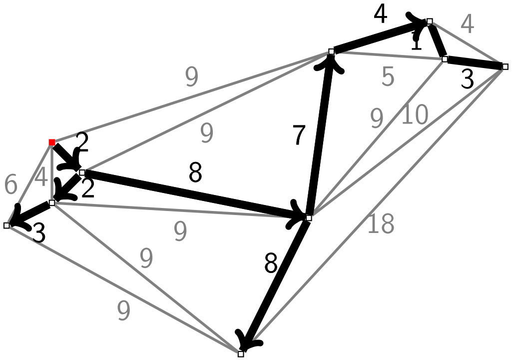


Figure: DFS

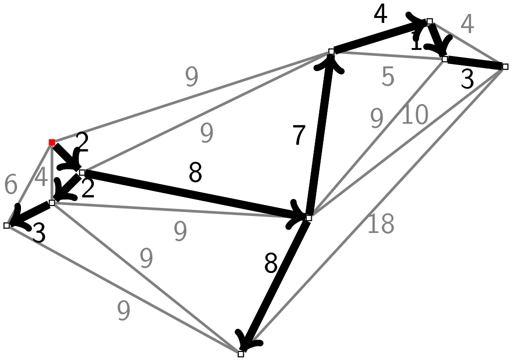


Figure: DFS

Depth-first search

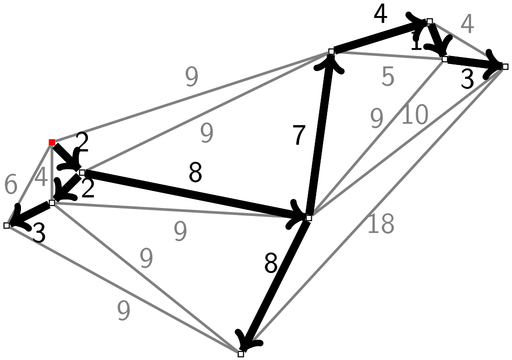


Figure: DFS

2-approx algorithm

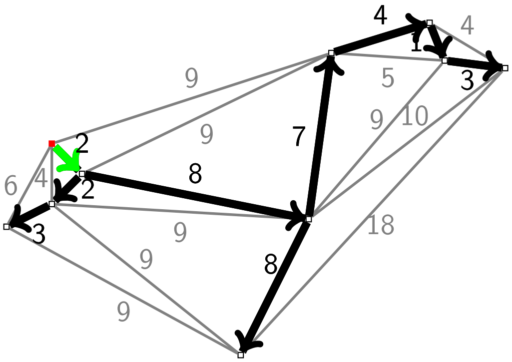


Figure: 2-approximation algorithm

2-approx algorithm

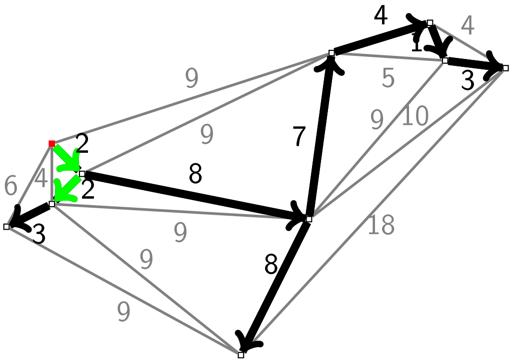


Figure: 2-approximation algorithm

2-approx algorithm

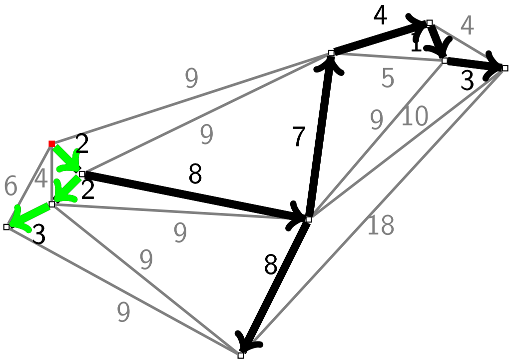


Figure: 2-approximation algorithm

2-approx algorithm

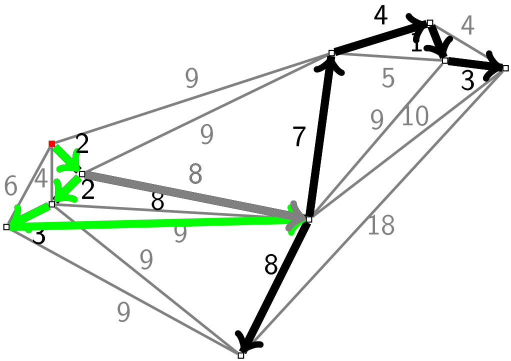


Figure: 2-approximation algorithm

2-approx algorithm

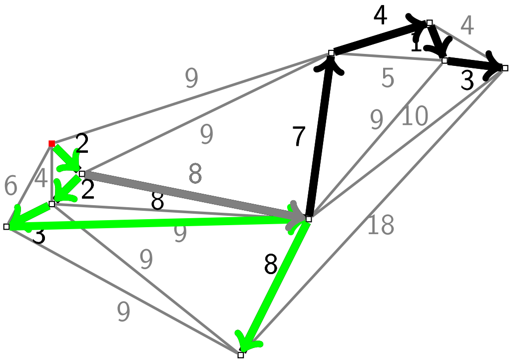


Figure: 2-approximation algorithm

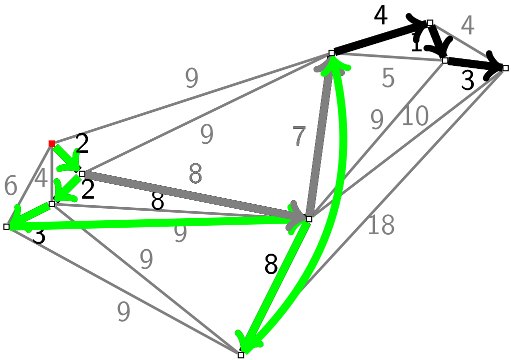


Figure: 2-approximation algorithm

2-approx algorithm

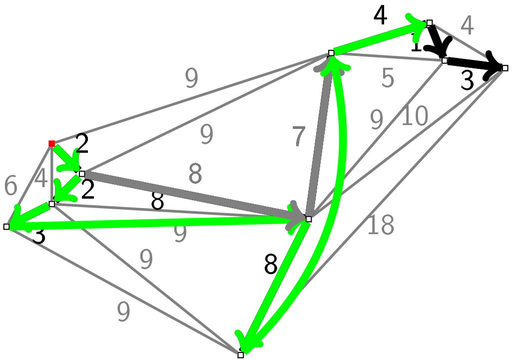


Figure: 2-approximation algorithm

2-approx algorithm

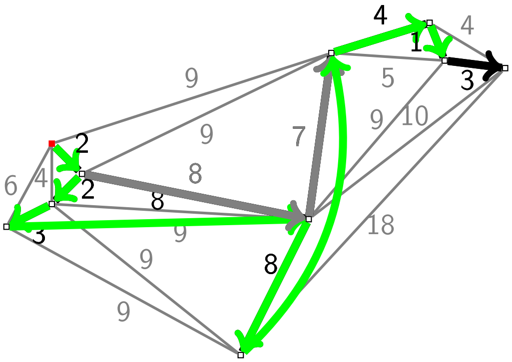


Figure: 2-approximation algorithm

2-approx algorithm

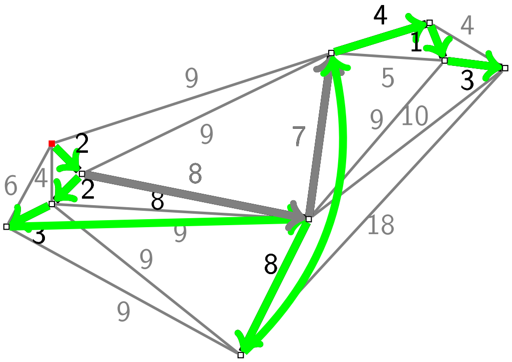


Figure: 2-approximation algorithm

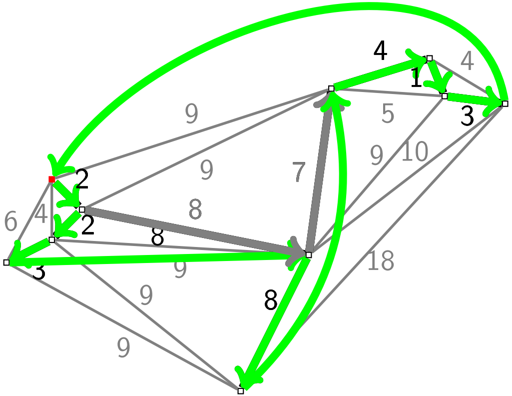


Figure: 2-approximation algorithm

Approximation ratio

The solution given by this heuristics is at most twice as bad as the optimal solution.

SCIP heuristics

Primal Heuristics	ExecTime	SetupTime	Calls	Found	Best
LP solutions	0.00	-	-	3	3
relax solutions	0.00	-	-	0	0
pseudo solutions	0.00	-	-	0	0
strong branching	0.00	-	-	0	0
actconsdiving	0.00	0.00	0	0	0
adaptivediving	1.00	0.00	18	0	0
alns	0.00	0.00	1	0	0
bound	0.00	0.00	0	0	0
clique	0.00	0.00	1	0	0
coefdiving	0.00	0.00	0	0	0
completesol	0.00	0.00	0	0	0
conflictdiving	10.00	0.00	27	0	0
crossover	4.00	0.00	1	0	0
dins	0.00	0.00	0	0	0
distributiondivin	7.00	0.00	26	0	0
dps	0.00	0.00	0	0	0
dualval	0.00	0.00	0	0	0
farkasdiving	5.00	0.00	2	0	0
feaspump	2.00	0.00	1	0	0
fixandinfer	0.00	0.00	0	0	0
fracdiving	17.00	0.00	27	0	0
gins	48.00	0.00	4	4	2
guideddiving	11.00	0.00	27	0	0
indicator	0.00	0.00	0	0	0
intdiving	0.00	0.00	0	0	0
intshifting	1.00	0.00	11	0	0
linesearchdiving	9.00	0.00	27	0	0
localbranching	0.00	0.00	0	0	0
locks	0.00	0.00	1	0	0
lpface	0.00	0.00	0	0	0
mpec	6.00	0.00	2	0	0
multistart	0.00	0.00	0	0	0
mutation	0.00	0.00	0	0	0
nlpdiving	22.00	0.00	1	17	0
objpscostdiving	7.00	0.00	5	0	0
octane	0.00	0.00	0	0	0
ofins	0.00	0.00	0	0	0
oneopt	0.00	0.00	8	0	0
padm	0.00	0.00	0	0	0
proximity	0.00	0.00	0	0	0
pscostdiving	12.00	0.00	27	0	0
randrounding	5.00	0.00	182	0	0
rens	0.00	0.00	0	0	0
reoptsols	0.00	0.00	0	0	0

Figure: Modern solvers employ a lot of heuristics

Meta-Heuristics

Heuristics can often get stuck in local optima.

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

Many accept worsening moves, which tends to work well in avoiding local optima,

Simulated Annealing

Randomly decide to move to a neighboring solution based on its fitness. The probabilities decrease with time.
(Show simulated annealing example)

Tabu Search

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

Genetic Algorithm

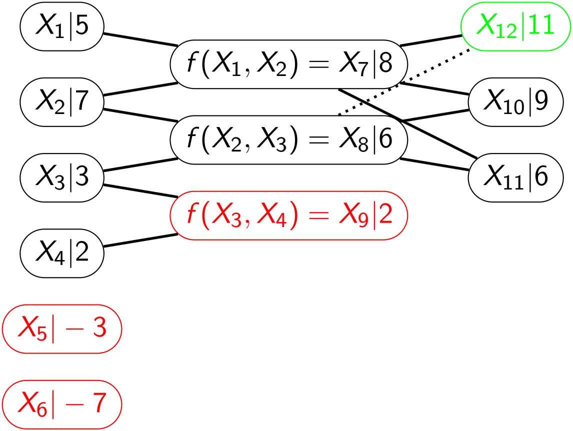


Figure: Illustration of a genetic algorithm

Reformulations

Logical Constraints

With binary variables, we can model some logical constraints.

Logical Constraints

With binary variables, we can model some logical constraints.

$$\neg x \quad \Big|$$

Logical Constraints

With binary variables, we can model some logical constraints.

$$
\begin{array}{c|c}
\neg x & 1 - x \\
x \implies y &
\end{array}
$$

Logical Constraints

With binary variables, we can model some logical constraints.

$$
\begin{array}{c|c}
\neg x & 1 - x \\
x \implies y & x \leq y \\
x \wedge y &
\end{array}
$$

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

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

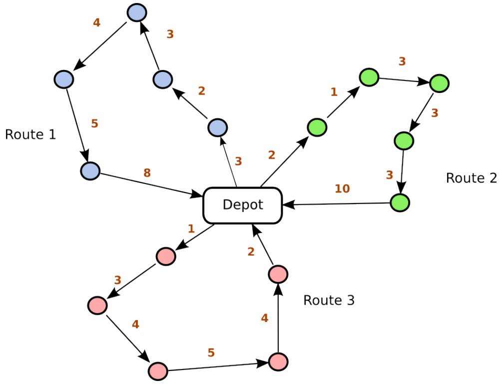


Figure: The vehicle routing problem has a lot of symmetry

Presolving

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

E.g. $x \leq 1.5, y \geq 0.5, x + y \leq 1 \implies x \leq 0.5$

Presolving

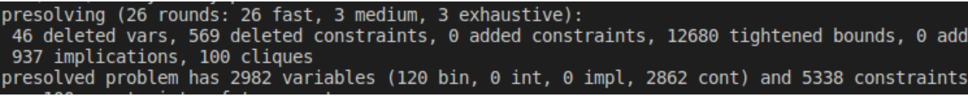


Figure: Example of SCIP presolving

Cutting planes

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

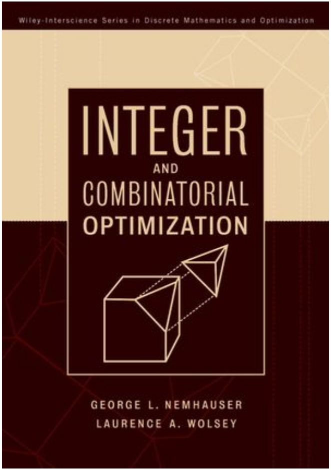
Cutting planes

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

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

Exercises

Bibliography



Suggestions

Light read on integer programming: Link

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

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: dynamically add columns

Master Problem

Subproblem

Reduced Cost

Cutting Stock Example

PySCIPOpt

Algorithm diagram

Branch-and-Price

Column generation embedded in a Branch-and-bound tree. Branch on fractional variables.

Dantzig-Wolfe Decomposition

(Weak) Minkowsi-Weyl Theorem revisited

Theorem

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

(The actual theorem is an equivalence and with possibly unbounded polyhedron)

Dantzig-Wolfe Decomposition (continued)

Benders' Decomposition

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

Benders' Decomposition

Fix y to $\overline{y}$.

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

Algorithm Idea

Benders' Decomposition

The previous problem has the following dual:

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

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 Problem

Inner Problem

Duality revisited

Optimality cuts

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

Feasibility cuts

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

Exercises

Suggestions

Light read on column-generation: Link

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