Topics in Optimization

João Pedro Gonçalves Dionísio

March 14, 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

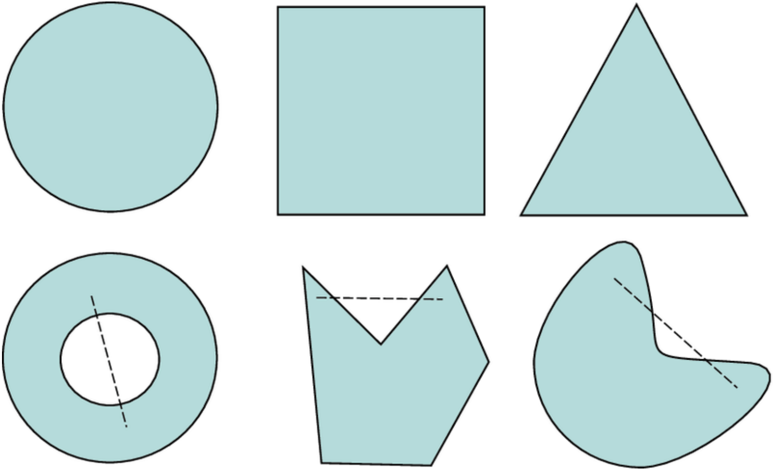


Figure: Examples of convex and non-convex sets

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

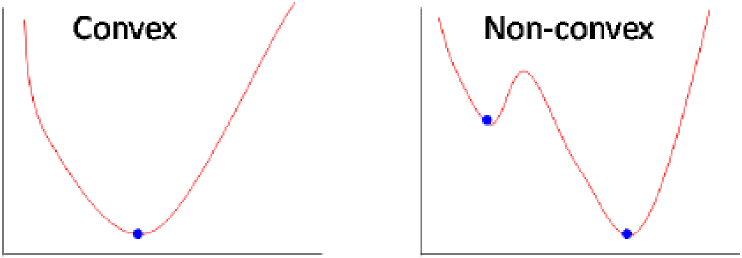


Figure: Examples of convex and non-convex functions

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

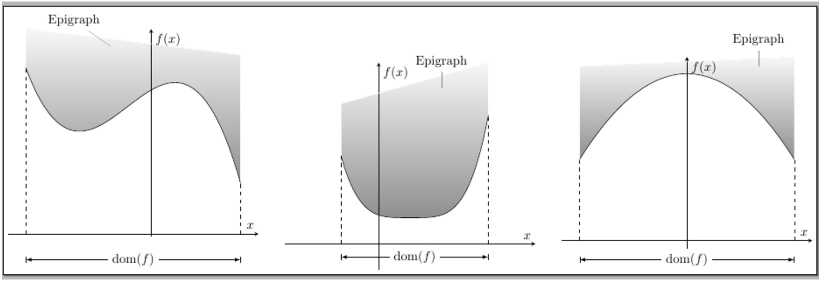


Figure: Examples of epigraph

Proposition

The following two statements are equivalent:

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

Definition

A convex optimization problem is

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

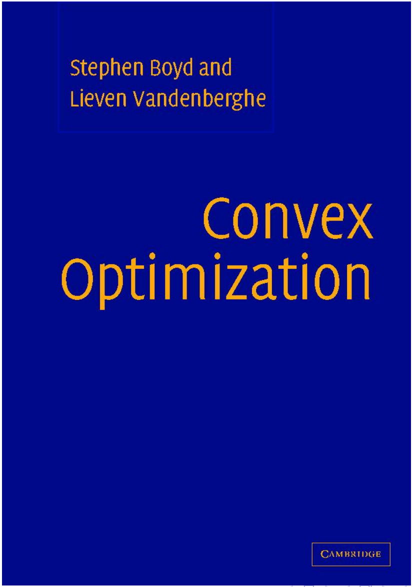
$f, g_i, i = 1 \ldots, m$ convex functions. It implies that $\bigcap_{i \in [m]} \mathrm{epi}(g_i)$ forms a convex set.

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

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

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

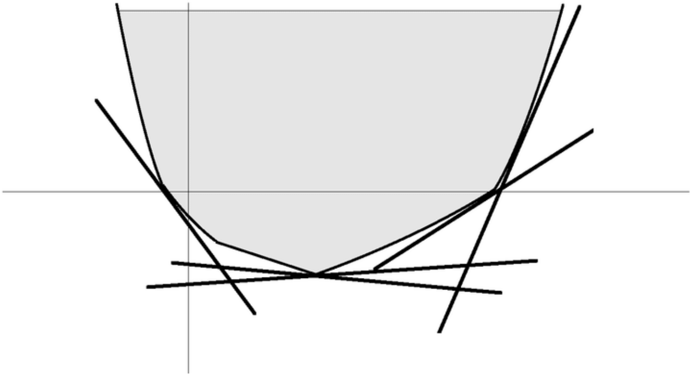


Figure: Visual representation

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:

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

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 $\exists d \in \mathbb{R}^n, x \in P \mid \forall \lambda \in \mathbb{R}, x + \lambda d \in P$.

Preliminaries

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

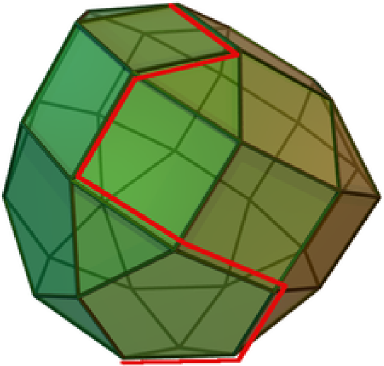


Figure: Example of simplex - only moves forward

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.

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

Successive iterations decrease the value of μ. IPM has a

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

IPM visualization

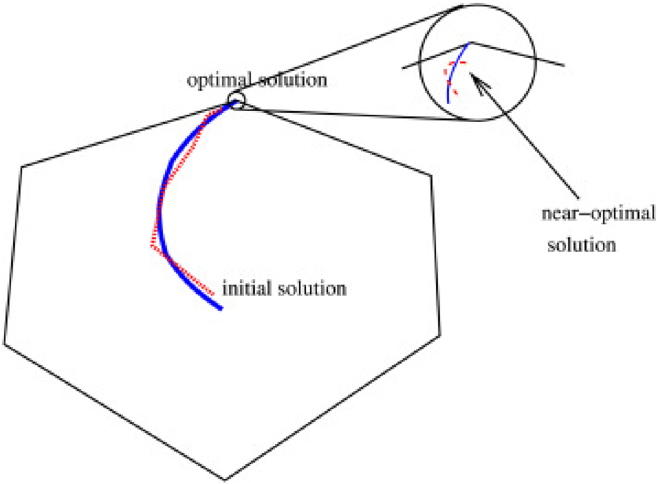
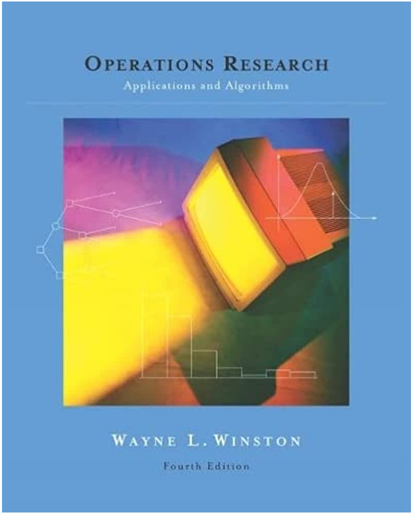


Figure: Example of IPM iterations

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

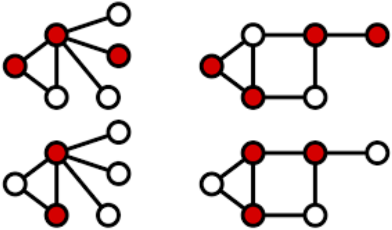


Figure: Example of Vertex Cover solutions

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.

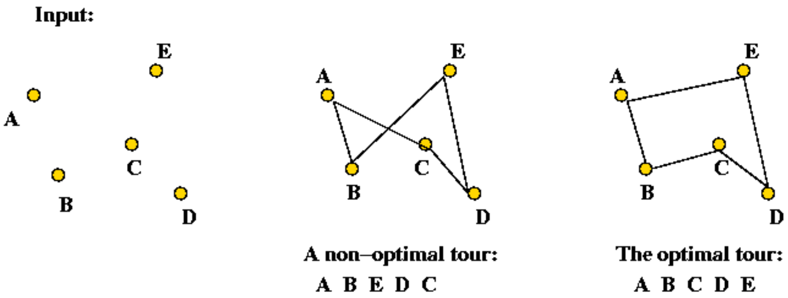


Figure: TSP example

TSP formulation

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

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.

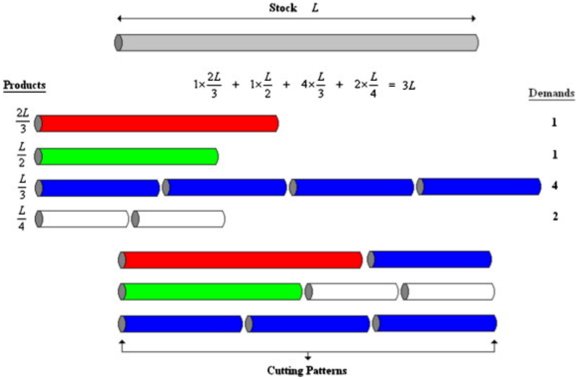


Figure: Cutting stock example

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.

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.

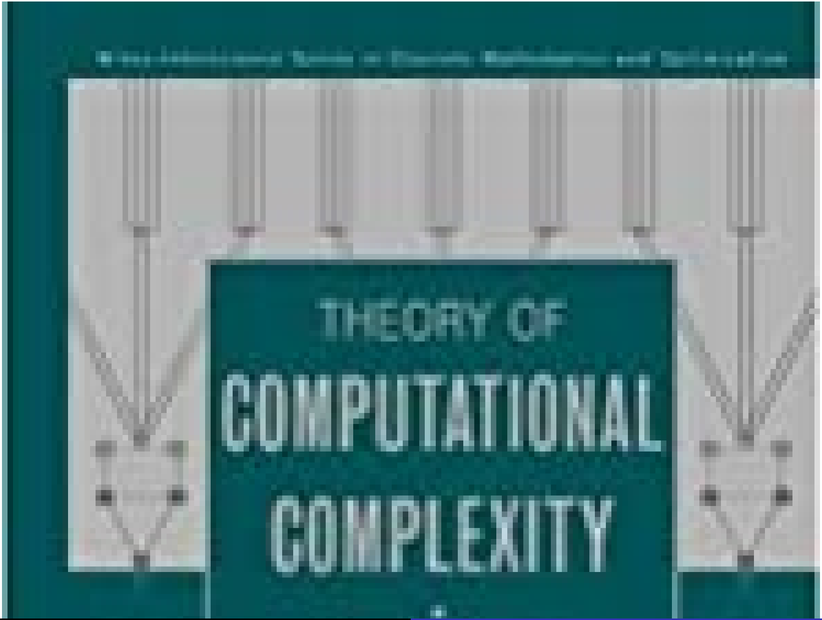
1. Vehicle routing and TSP
2. Min Vertex Cover and Max Independent set
3. Cutting Stock and Bin packing

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

Integer Programming

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

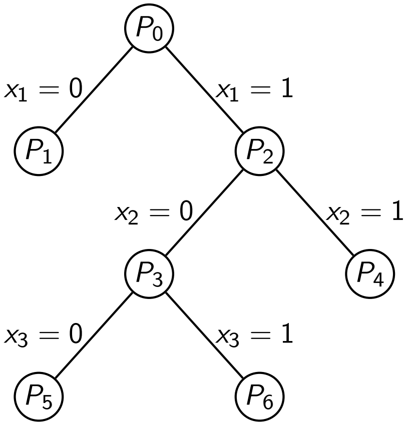
Non-convex

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

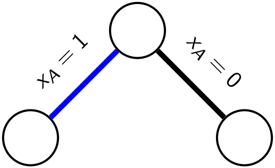
1. Sort items by price density (price/weight)
2. Pick items until capacity is exceeded
3. Remove the excess of the last item

Knapsack instance

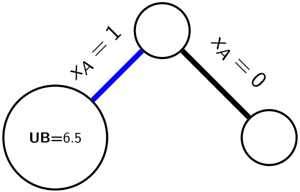
Item	Weight	Value	Value/Weight
A	3	5	1.67
B	2	3	1.5
C	1	1	1

Knapsack capacity: 4

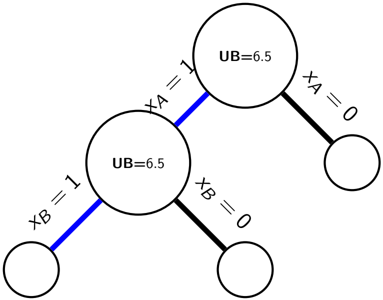
Branch and Bound



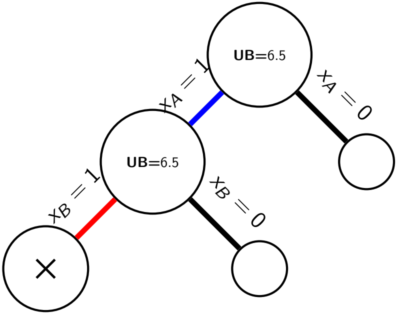
Branch and Bound



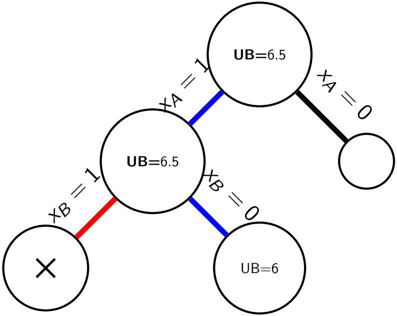
Branch and Bound



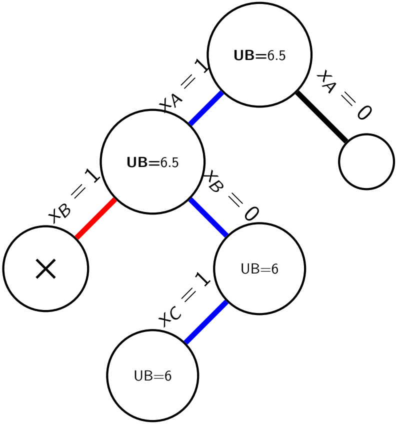
Branch and Bound



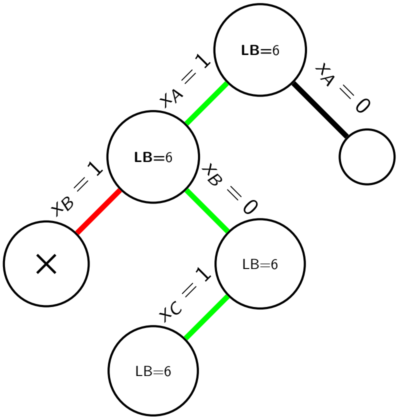
Branch and Bound



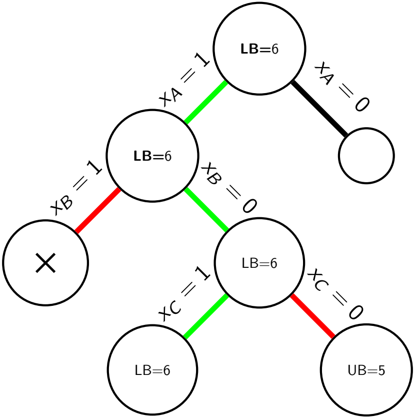
Branch and Bound



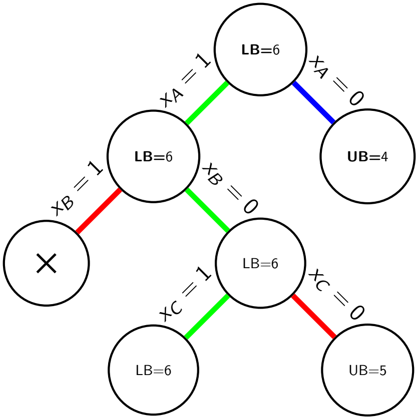
Branch and Bound



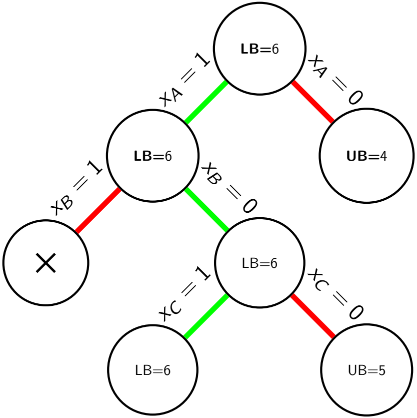
Branch and Bound



Branch and Bound



Branch and Bound



Heuristics

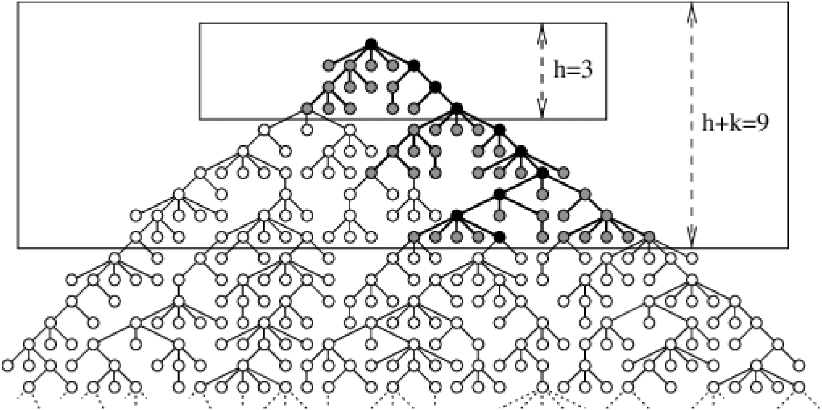


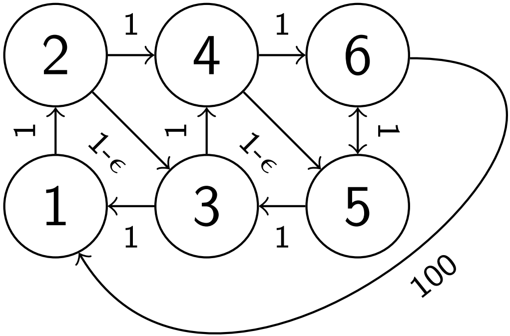
Figure: Branch and Bound trees can get very big. How big?

Heuristics

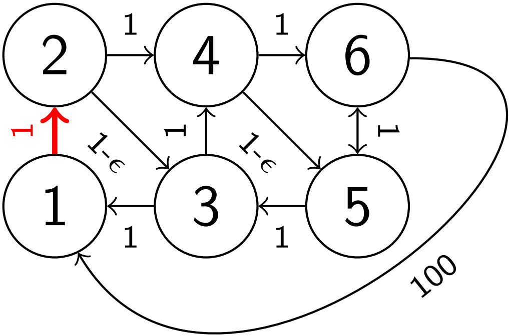
We will focus on heuristics for the TSP.

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

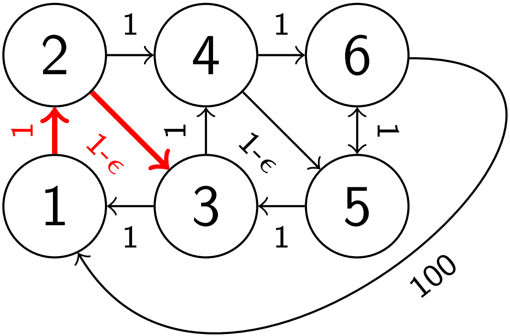
Nearest Neighbor



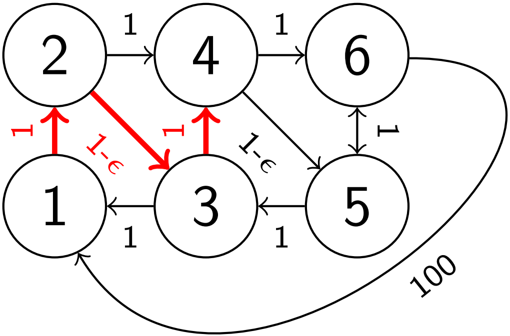
Nearest Neighbor



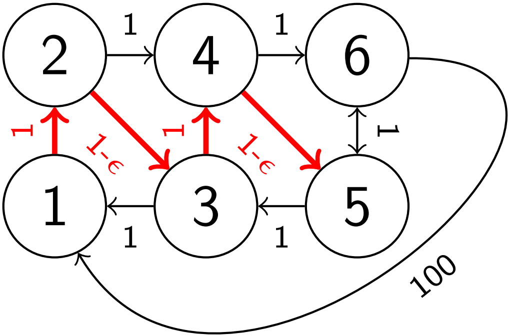
Nearest Neighbor



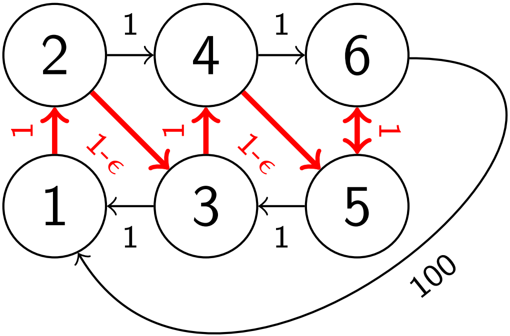
Nearest Neighbor



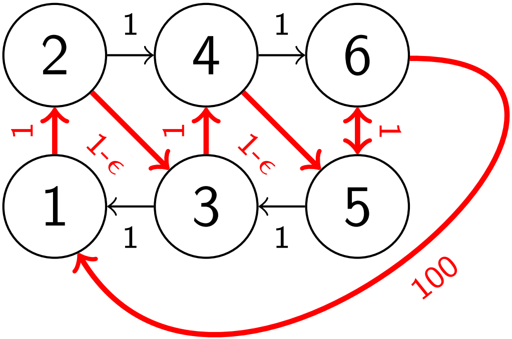
Nearest Neighbor



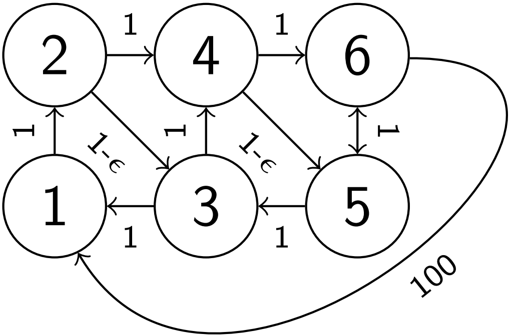
Nearest Neighbor



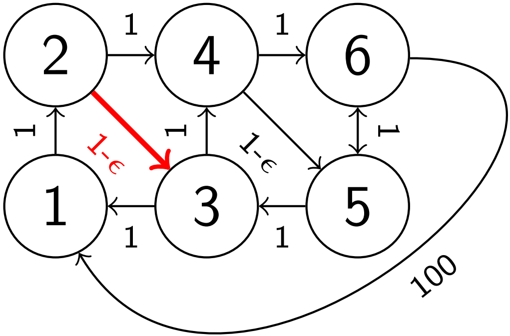
Nearest Neighbor



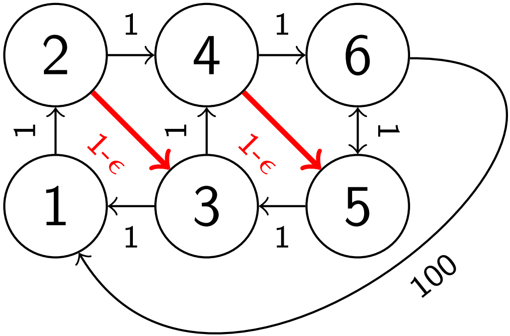
Greedy Algorithm



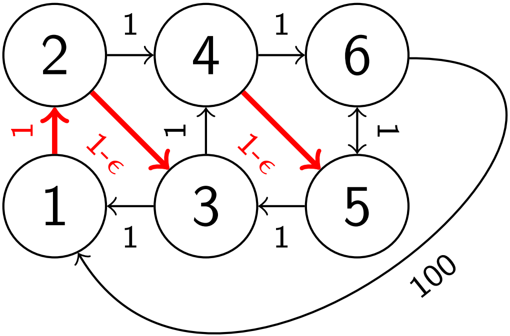
Greedy Algorithm



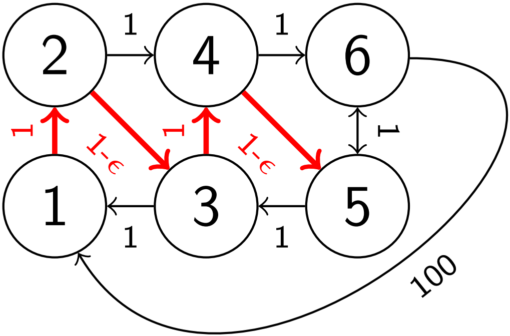
Greedy Algorithm



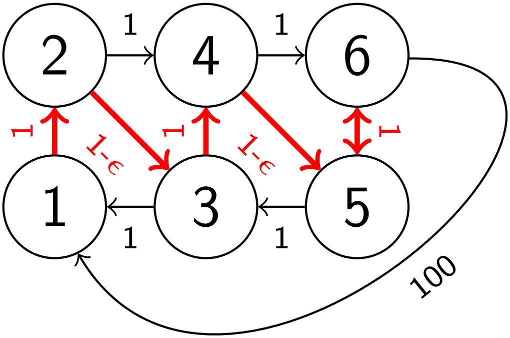
Greedy Algorithm



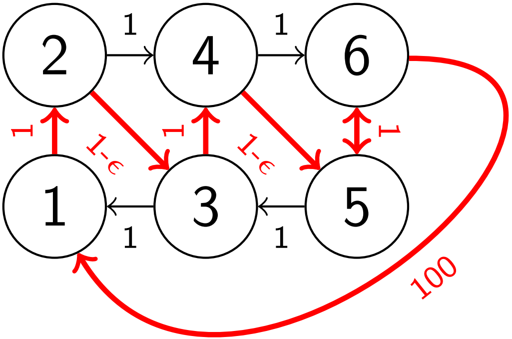
Greedy Algorithm



Greedy Algorithm



Greedy Algorithm



2-OPT algorithm

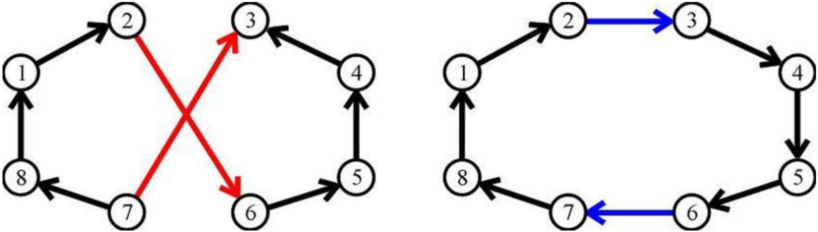


Figure: If two edges cross, we can find a strictly better solution

TSP: 2-opt visualization

Large TSP

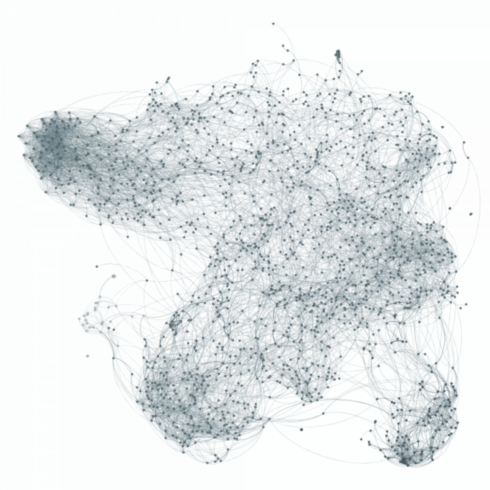


Figure: Heuristics are for large instances

Approximation algorithms

Some heuristics can have theoretical guarantees of their solution quality.

We will study 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

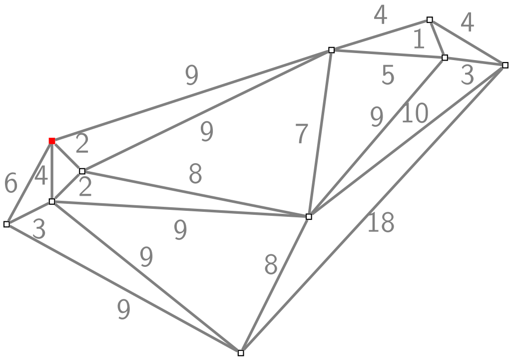


Figure: TSP instance

Minimum spanning tree

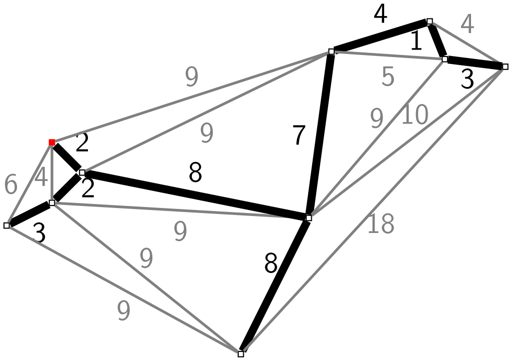


Figure: Minimal spanning tree

Depth-first search

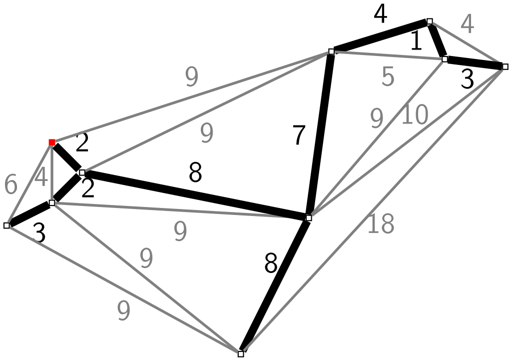


Figure: DFS

Depth-first search

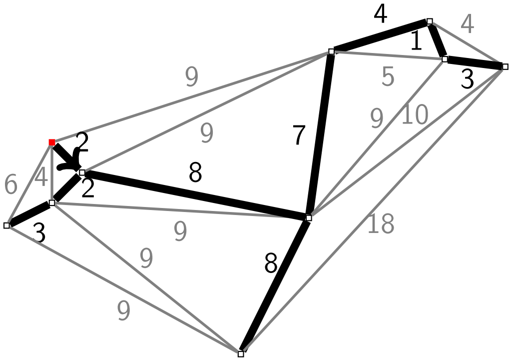


Figure: DFS

Depth-first search

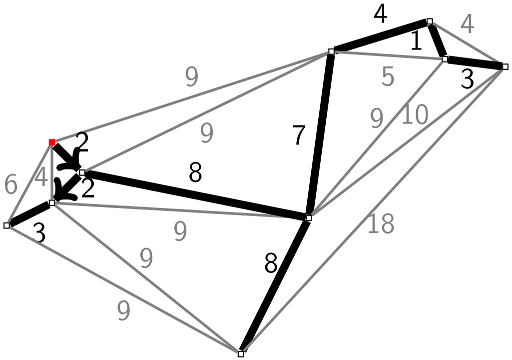


Figure: DFS

Depth-first search

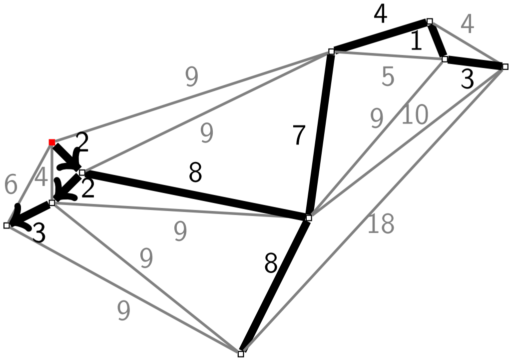


Figure: DFS

Depth-first search

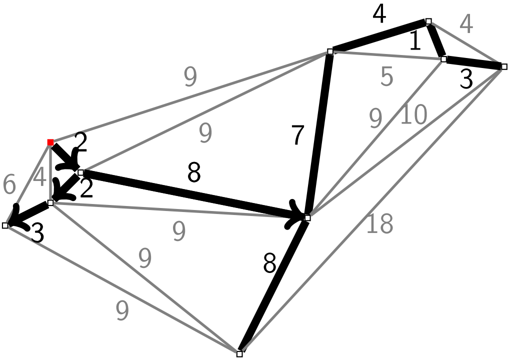


Figure: DFS

Depth-first search

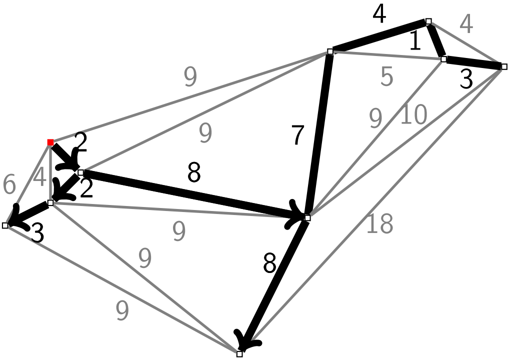


Figure: DFS

Depth-first search

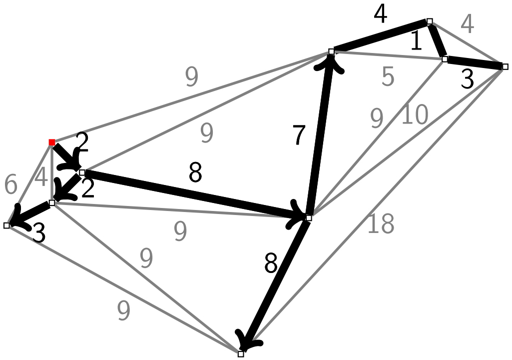


Figure: DFS

Depth-first search

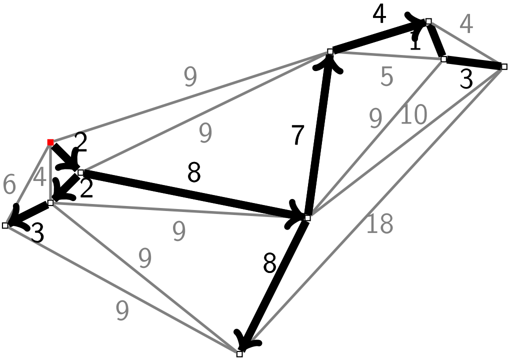


Figure: DFS

Depth-first search

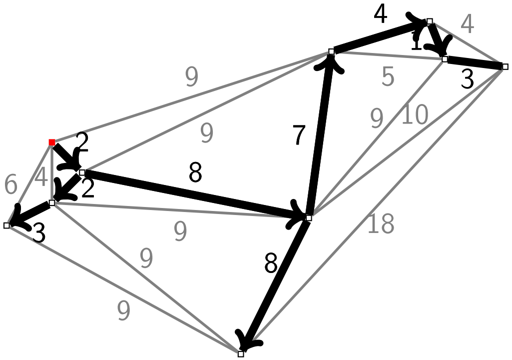


Figure: DFS

Depth-first search

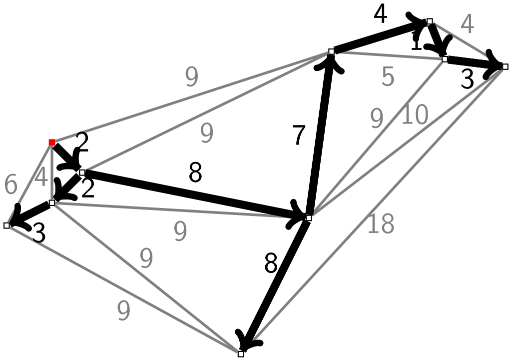


Figure: DFS

2-approx algorithm

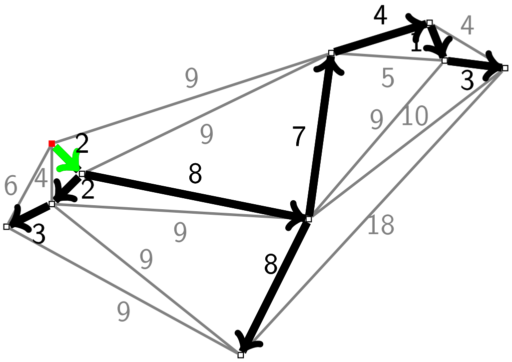


Figure: 2-approximation algorithm

2-approx algorithm

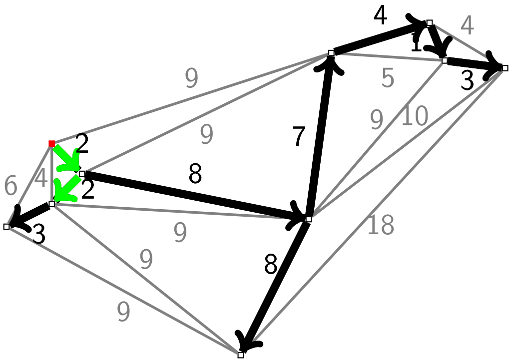


Figure: 2-approximation algorithm

2-approx algorithm

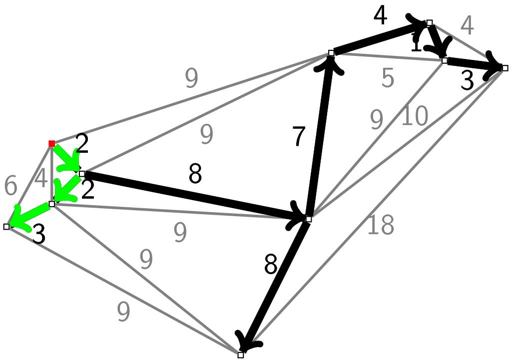


Figure: 2-approximation algorithm

2-approx algorithm

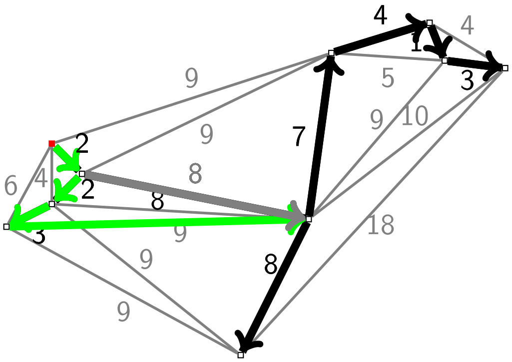


Figure: 2-approximation algorithm

2-approx algorithm

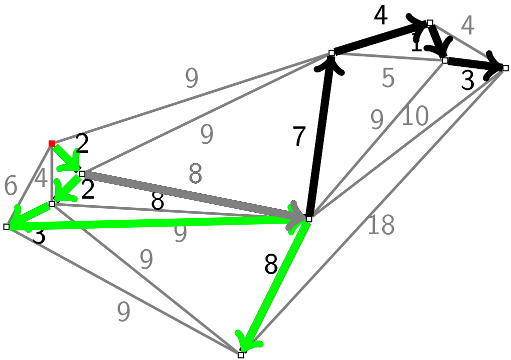


Figure: 2-approximation algorithm

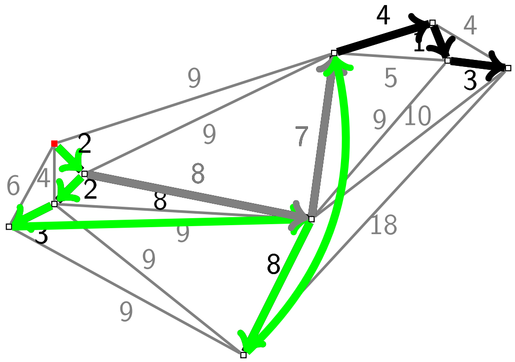


Figure: 2-approximation algorithm

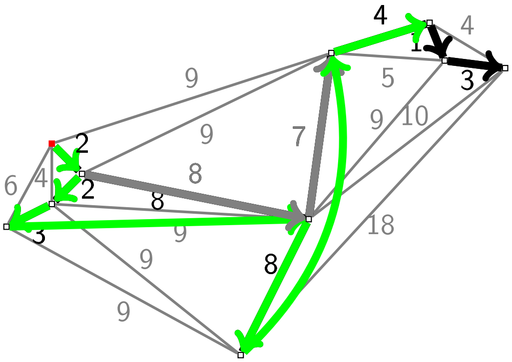


Figure: 2-approximation algorithm

2-approx algorithm

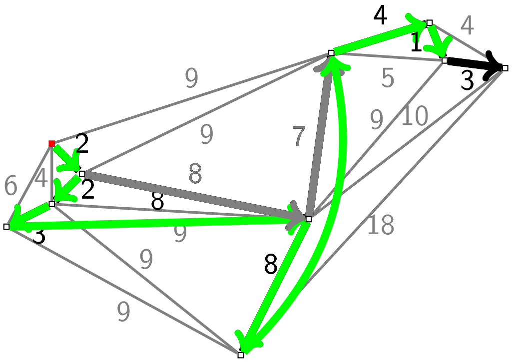


Figure: 2-approximation algorithm

2-approx algorithm

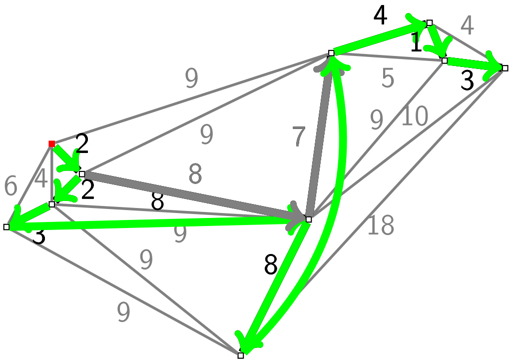


Figure: 2-approximation algorithm

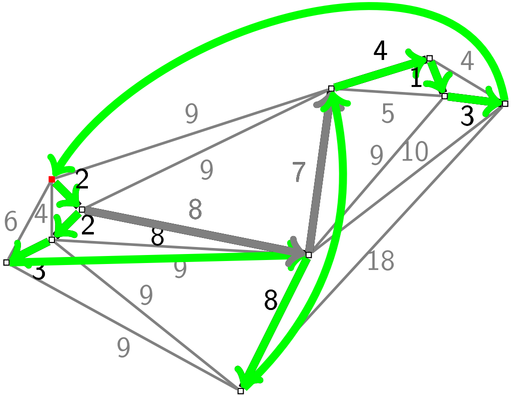


Figure: 2-approximation algorithm

Approximation ratio

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

Approximation ratio

The solution given by this heuristics 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.

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.

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)

Simulated Annealing

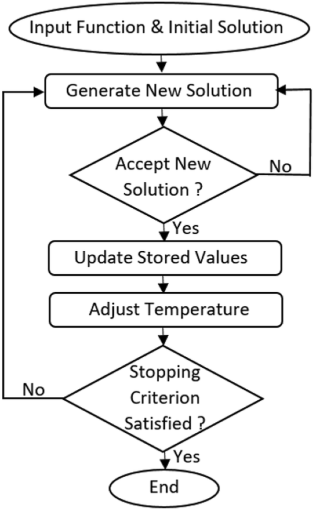


Figure: Simulated Annealing Flowchart

Tabu Search

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

Tabu Search

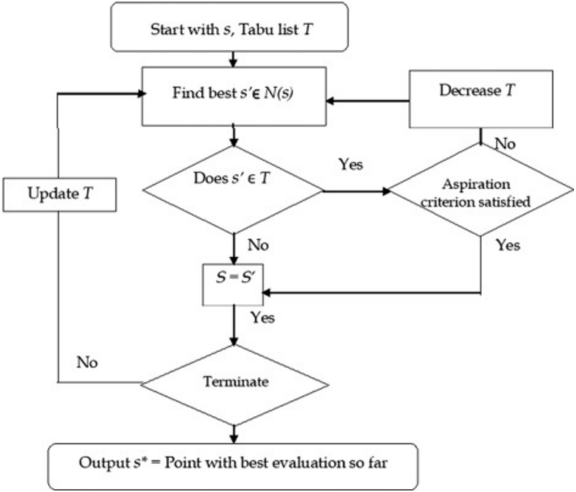


Figure: Tabu Search Flowchart

Genetic Algorithm

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

Genetic Algorithm

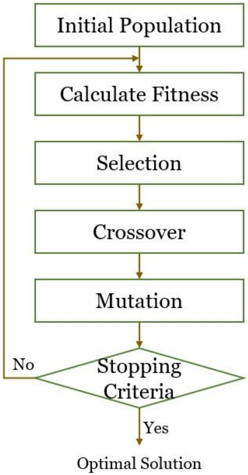


Figure: Genetic Algorithm Flowchart

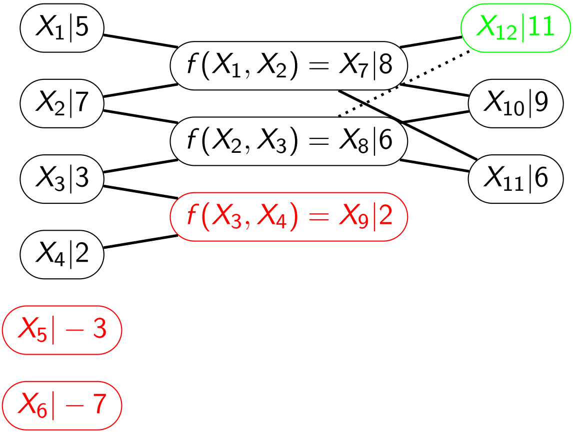


Figure: Illustration of a genetic algorithm

Reformulations

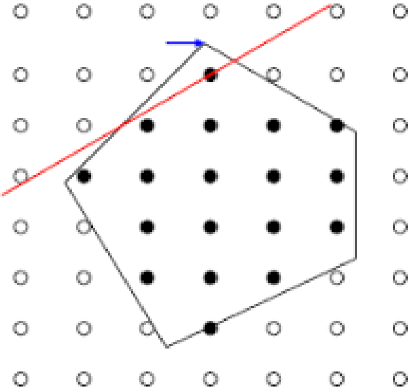


Figure: Integer programs admit infinite formulations

Reformulations

Formulations closer to their linear relaxation are better. Why is that?

Reformulations

Formulations closer to their linear relaxation are better. Why is that?

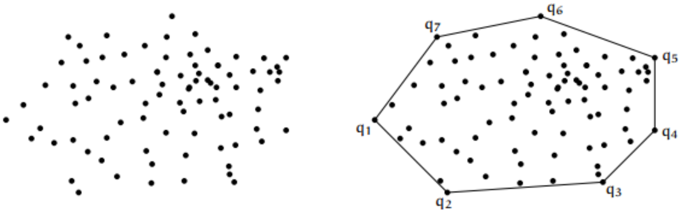
The provided bound is better, can prune branch and bound tree earlier.

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.

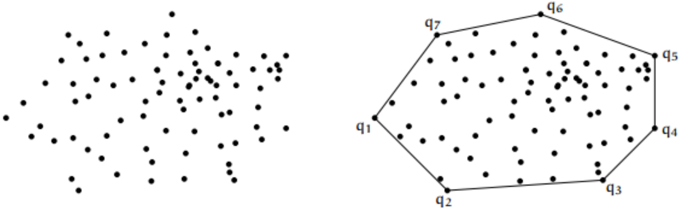
Perfect Formulation



(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

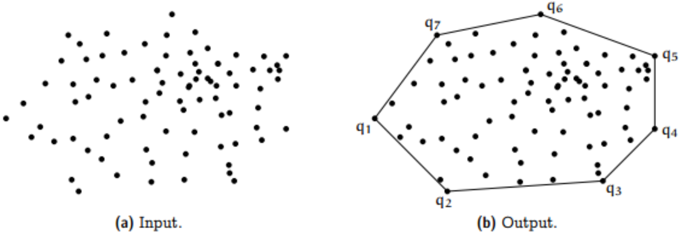


Figure: Linear relaxation equal to convex hull

Which points do we need to check? Why?

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

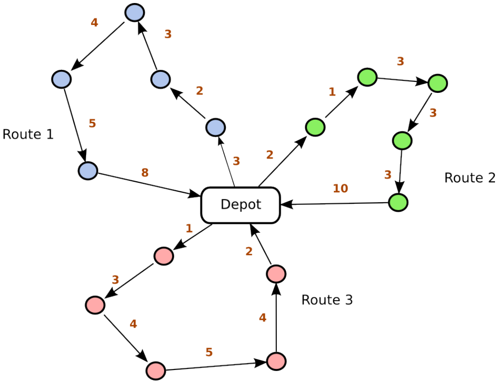


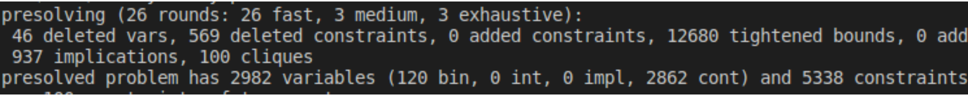
Figure: The vehicle routing problem has a lot of symmetry

Presolving

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

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 (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

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.

Logical Constraints

With binary variables, we can model some logical constraints.

With binary variables, we can model some logical constraints.

$\neg x$	

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.

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

Logical Constraints

With binary variables, we can model some logical constraints.

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

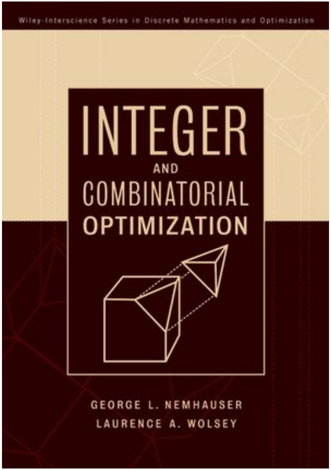
Logical Constraints

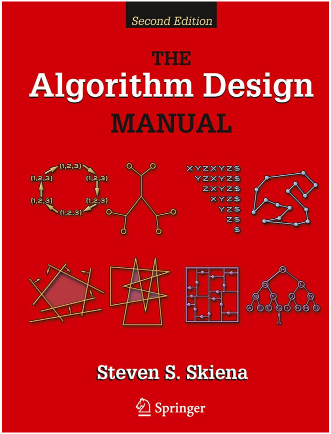
With binary variables, we can model some logical constraints.

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

Table: Formulating logical expressions with integer programming

Bibliography





Suggestions

Light read on integer programming: Link

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

Exercises

Decomposition Methods

Sometimes difficult problems have a structure that can be explored.

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

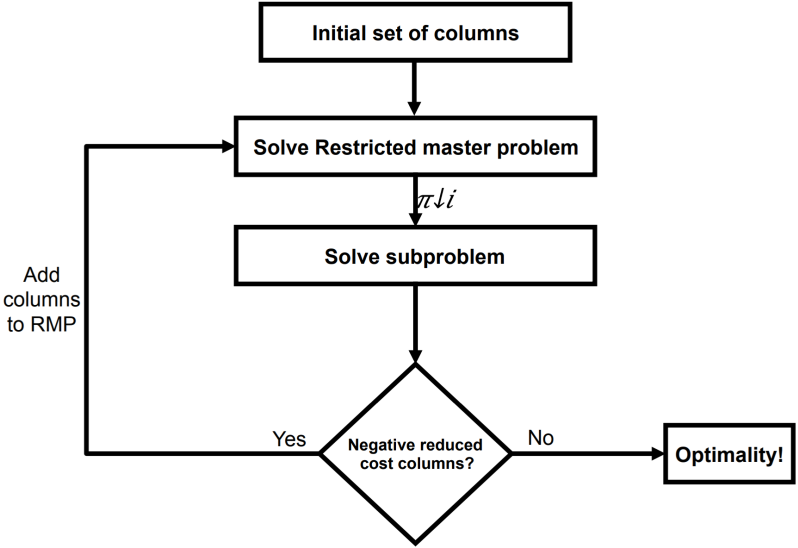
Column Generation

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



Master Problem

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

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:

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

Cutting Stock Revisited

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

How do solutions look like?

Master problem

Pricing Problem

Pricing Problem

This is precisely the knapsack problem!

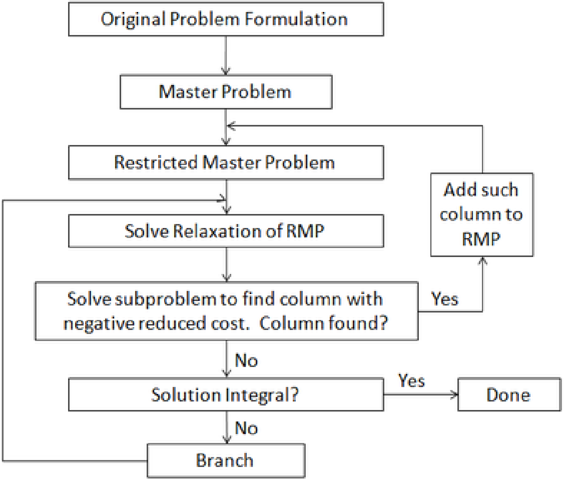
Column generation Tips

- The pricing problem should be easy.
- Solve the pricing problem heuristically. How many times does it need to be solved to optimality?
- Add multiple columns per iteration
- Remove columns from the RMP that have been inactive for many iterations

PySCIPOpt

Branch-and-Price

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



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

When does this work?

(Weak) Minkowsi-Weyl Theorem

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)

DW-decomposition master problem

DW-decomposition pricing problem

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.

Benders' Decomposition

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

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

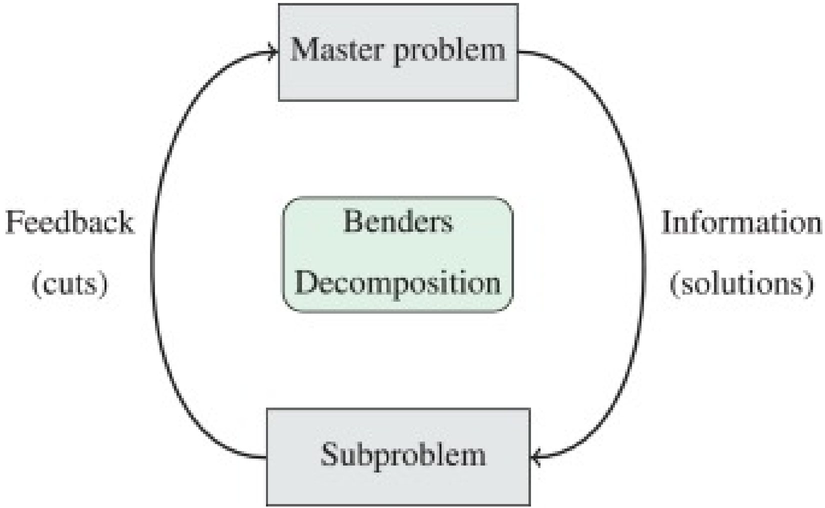


Figure: Benders' Decomposition Idea

Benders' Decomposition

The previous problem has the following dual:

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

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

Optimize the problem for a fixed y.

Inner Problem

Use the solution from the outer problem to choose better ys in the outer problem.

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* $b^\intercal y \leq c^\intercal x^*$

Optimality cuts

If the inner problem is feasible, we can derive bounds for the outer problem.

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

Feasibility cuts

If the inner problem is unbounded, we know the fixed y cannot be chosen.

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

Suggestions

Light read on column-generation: Link

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

Exercises