Fundamentals of Applied Operations Research

Guochuan Zhang

zgc@zju.edu.cn

Research Group on Discrete Optimization and Algorithms

Zhejiang University

Wednesdays, September - December 2021

After-Class Questions

Spanning Tree

- Given an undirected graph $G = (V, E)$, find out as many edge disjoint spanning trees as possible.
- You may compute spanning trees one by one until none exists. Show it works or present a counter-example.

Shortest Path

- Given an undirected graph $G = (V, E)$, each edge e_i is associated with two parameters, namely b_i and c_i.
- A path p is evaluated by a pair (B_p, C_p) as well, where $B_p = \min_{ei \in p} b_i$, and $C_p = \sum_{e_i \in p} c_i$.
- Find an s-t path p with (B_p, C_p), where no path is lexicographically better than p (larger B_p, smaller C_p).

After-Class Questions

Spanning Tree

- Given an undirected graph $G = (V, E)$, find out as many edge disjoint spanning trees as possible.
- You may compute spanning trees one by one until none exists. Show it works or present a counter-example.

Shortest Path

- Given an undirected graph $G = (V, E)$, each edge e_i is associated with two parameters, namely b_i and c_i.
- A path p is evaluated by a pair (B_p, C_p) as well, where $B_p = \min_{ei \in p} b_i$, and $C_p = \sum_{e_i \in p} c_i$.
- Find an s-t path p with (B_p, C_p), where no path is lexicographically better than p (larger B_p, smaller C_p).

After-Class Questions

Spanning Tree

- Given an undirected graph $G = (V, E)$, find out as many edge disjoint spanning trees as possible.
- You may compute spanning trees one by one until none exists. Show it works or present a counter-example.

Shortest Path

- Given an undirected graph $G = (V, E)$, each edge e_i is associated with two parameters, namely b_i and c_i.
- A path p is evaluated by a pair (B_p, C_p) as well, where $B_p = \min_{ei \in p} b_i$, and $C_p = \sum_{e_i \in p} c_i$.
- Find an s-t path p with (B_p, C_p), where no path is lexicographically better than p (larger B_p, smaller C_p).

After-Class Questions

Spanning Tree

- Given an undirected graph $G = (V, E)$, find out as many edge disjoint spanning trees as possible.
- You may compute spanning trees one by one until none exists. Show it works or present a counter-example.

Shortest Path

- Given an undirected graph $G = (V, E)$, each edge e_i is associated with two parameters, namely b_i and c_i.
- A path p is evaluated by a pair (B_p, C_p) as well, where $B_p = \min_{ei \in p} b_i$, and $C_p = \sum_{e_i \in p} c_i$.
- Find an s-t path p with (B_p, C_p), where no path is lexicographically better than p (larger B_p, smaller C_p).

After-Class Questions

Spanning Tree

- Given an undirected graph $G = (V, E)$, find out as many edge disjoint spanning trees as possible.
- You may compute spanning trees one by one until none exists. Show it works or present a counter-example.

Shortest Path

- Given an undirected graph $G = (V, E)$, each edge e_i is associated with two parameters, namely b_i and c_i.
- A path p is evaluated by a pair (B_p, C_p) as well, where $B_p = \min_{ei \in p} b_i$, and $C_p = \sum_{e_i \in p} c_i$.
- Find an s-t path p with (B_p, C_p), where no path is lexicographically better than p (larger B_p, smaller C_p).

Roadmap

Operations Research - the Science of Better

- Explore the methodology for solving a great many of optimization problems with limited resources / information

A List of Topics

- Linear Programming
- Nonlinear Programming
- Integer Programming
- Combinatorial Optimization (Approximation/Online Algorithms)
- Game Theory (optimization with interaction)

Main Issues

- Design and analysis of algorithms/mechanisms

Roadmap

Operations Research - the Science of Better

- Explore the methodology for solving a great many of optimization problems with limited resources / information

A List of Topics

- Linear Programming
- Nonlinear Programming
- Integer Programming
- Combinatorial Optimization (Approximation/Online Algorithms)
- Game Theory (optimization with interaction)

Main Issues

- Design and analysis of algorithms/mechanisms

Roadmap

Operations Research - the Science of Better

- Explore the methodology for solving a great many of optimization problems with limited resources / information

A List of Topics

- Linear Programming
- Nonlinear Programming
- Integer Programming
- Combinatorial Optimization (Approximation/Online Algorithms)
- Game Theory (optimization with interaction)

Main Issues

- Design and analysis of algorithms/mechanisms

Roadmap

Operations Research - the Science of Better

- Explore the methodology for solving a great many of optimization problems with limited resources / information

A List of Topics

- Linear Programming
- Nonlinear Programming
- Integer Programming
- Combinatorial Optimization (Approximation/Online Algorithms)
- Game Theory (optimization with interaction)

Main Issues

- Design and analysis of algorithms/mechanisms

Roadmap

Operations Research - the Science of Better

- Explore the methodology for solving a great many of optimization problems with limited resources / information

A List of Topics

- Linear Programming
- Nonlinear Programming
- Integer Programming
- Combinatorial Optimization (Approximation/Online Algorithms)
- Game Theory (optimization with interaction)

Main Issues

- Design and analysis of algorithms/mechanisms

Roadmap

Operations Research - the Science of Better

- Explore the methodology for solving a great many of optimization problems with limited resources / information

A List of Topics

- Linear Programming
- Nonlinear Programming
- Integer Programming
- Combinatorial Optimization (Approximation/Online Algorithms)
- Game Theory (optimization with interaction)

Main Issues

- Design and analysis of algorithms/mechanisms

Roadmap

Operations Research - the Science of Better

- Explore the methodology for solving a great many of optimization problems with limited resources / information

A List of Topics

- Linear Programming
- Nonlinear Programming
- Integer Programming
- Combinatorial Optimization (Approximation/Online Algorithms)
- Game Theory (optimization with interaction)

Main Issues

- Design and analysis of algorithms/mechanisms

Roadmap

Operations Research - the Science of Better

- Explore the methodology for solving a great many of optimization problems with limited resources / information

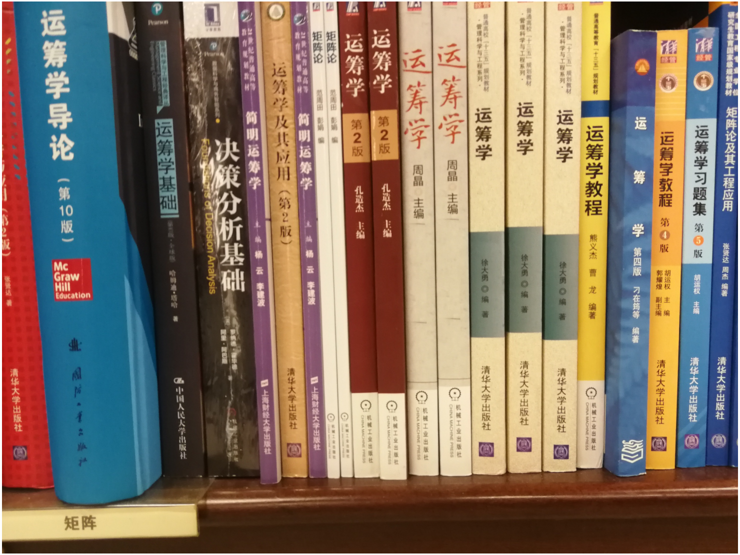
A List of Topics

- Linear Programming
- Nonlinear Programming
- Integer Programming
- Combinatorial Optimization (Approximation/Online Algorithms)
- Game Theory (optimization with interaction)

Main Issues

- Design and analysis of algorithms/mechanisms

Reading Materials



Reading Materials

Books

- Any textbook on Operations Research or Optimization
- Any book on Combinatorial Optimization
  Recommended "Combinatorial Optimization - Algorithms and Complexity, Papademitriou and Steiglitz"
- Any book on Algorithms
  Recommended "Algorithm Design, Tardos and Kleinberg"
- Any book on Game Theory
  Recommended "Algorithmic Game Theory, Nisan, Roughgarden, Tardos, and Vazirani"

Who Are Supposed to Sit Here?

Incentives

- Show great interests
- Have strong math background
- Enjoy finding out the truth
- Get credits (Sure, only if the above are satisfied)

Requirements

- 100% attendance (except for emergency)
- 100% attention
- Being active

Who Are Supposed to Sit Here?

Incentives

- Show great interests
- Have strong math background
- Enjoy finding out the truth
- Get credits (Sure, only if the above are satisfied)

Requirements

- 100% attendance (except for emergency)
- 100% attention
- Being active

Grading Mechanism

In Class

- In-class discussions (quizzes)
- Lecture notes
- Final in-class exercise

After Class

- Homework
- Problem solving in team
- Paper-reading and presentations

Highlights

We are concerned about Problems but not a single Instance

We are doing Re-search but not simple Searches

We are working on Programming but not Coding

We are not only doing something correct but also showing the Correctness

We prefer Intelligence to Artificial Intelligence

Highlights

We are concerned about Problems but not a single Instance

We are doing Re-search but not simple Searches

We are working on Programming but not Coding

We are not only doing something correct but also showing the Correctness

We prefer Intelligence to Artificial Intelligence

Highlights

We are concerned about **Problems** but not a single **Instance**

We are doing **Re-search** but not simple **Searches**

We are working on **Programming** but not **Coding**

We are not only doing something correct but also showing the **Correctness**

We prefer **Intelligence** to **Artificial Intelligence**

Highlights

We are concerned about Problems but not a single Instance

We are doing Re-search but not simple Searches

We are working on Programming but not Coding

We are not only doing something correct but also showing the Correctness

We prefer Intelligence to Artificial Intelligence

Highlights

We are concerned about Problems but not a single Instance

We are doing Re-search but not simple Searches

We are working on Programming but not Coding

We are not only doing something correct but also showing the Correctness

We prefer Intelligence to Artificial Intelligence

Computability

Limits to Computers

- Computers can only carry out algorithms: precise and universally understood sequences of instructions that solve any instances of rigorously defined computational problems
- Are there well-defined mathematical problems for which there are no algorithms? YES! (Alan Turing)
- Undecidable problems do exist, say the Halting problem: given a computer program with its input, will it ever halt?

Computability

Limits to Computers

- Computers can only carry out algorithms: precise and universally understood sequences of instructions that solve any instances of rigorously defined computational problems
- Are there well-defined mathematical problems for which there are no algorithms? YES! (Alan Turing)
- Undecidable problems do exist, say the Halting problem: given a computer program with its input, will it ever halt?

Computability

Limits to Computers

- Computers can only carry out algorithms: precise and universally understood sequences of instructions that solve any instances of rigorously defined computational problems
- Are there well-defined mathematical problems for which there are no algorithms? YES! (Alan Turing)
- Undecidable problems do exist, say the Halting problem: given a computer program with its input, will it ever halt?

Tractability

Time Bounds

- Away from the Turing's time in 1930s, computers nowadays deal with decidable problems. In principal, these problems admit an algorithm for solving every instance
- A new challenge is the running time of an algorithm, namely, the algorithm efficiency

Example

- TSP (the Travelling Salesman Problem): finding a shortest tour (a cycle), visiting each vertex exactly once, on a given weighted complete graph
- The number of possible tours is $(n-1)!/2$

Tractability

Time Bounds

- Away from the Turing's time in 1930s, computers nowadays deal with decidable problems. In principal, these problems admit an algorithm for solving every instance
- A new challenge is the running time of an algorithm, namely, the algorithm efficiency

Example

- TSP (the Travelling Salesman Problem): finding a shortest tour (a cycle), visiting each vertex exactly once, on a given weighted complete graph
- The number of possible tours is $(n-1)!/2$

Tractability

Input Size

- Basically, length of the sequence to encode the instance, the number of symbols in the sequence
- Testing a prime number: check if a given integer is prime?
- Size of a graph

Analysis of Algorithms (I)

- Deriving bounds for the time requirement of an algorithm
- Using the notations of O, Ω and Θ

Polynomial Time Algorithms

- Those running polynomially in the input size

Tractability

Input Size

- Basically, length of the sequence to encode the instance, the number of symbols in the sequence
- Testing a prime number: check if a given integer is prime?
- Size of a graph

Analysis of Algorithms (I)

- Deriving bounds for the time requirement of an algorithm
- Using the notations of O, Ω and Θ

Polynomial Time Algorithms

- Those running polynomially in the input size

Tractability

Input Size

- Basically, length of the sequence to encode the instance, the number of symbols in the sequence
- Testing a prime number: check if a given integer is prime?
- Size of a graph

Analysis of Algorithms (I)

- Deriving bounds for the time requirement of an algorithm
- Using the notations of O, Ω and Θ

Polynomial Time Algorithms

- Those running polynomially in the input size

Tractability

Input Size

- Basically, length of the sequence to encode the instance, the number of symbols in the sequence
- Testing a prime number: check if a given integer is prime?
- Size of a graph

Analysis of Algorithms (I)

- Deriving bounds for the time requirement of an algorithm
- Using the notations of O, Ω and Θ

Polynomial Time Algorithms

- Those running polynomially in the input size

Tractability

Input Size

- Basically, length of the sequence to encode the instance, the number of symbols in the sequence
- Testing a prime number: check if a given integer is prime?
- Size of a graph

Analysis of Algorithms (I)

- Deriving bounds for the time requirement of an algorithm
- Using the notations of O, Ω and Θ

Polynomial Time Algorithms

- Those running polynomially in the input size

P vs NP

Decision Problems

- Decide if there is a solution (Yes/No questions)

Optimization Problems

- Determine an optimal solution

Class P

- A problem with a polynomial time algorithm solving all its instances (solved polynomially by a Turing machine)

Class NP

- If x is a Yes instance of the problem, there exists a certificate for x, whose validity can be checked in polynomial time (solved in polynomial time by a non-deterministic Turing machine)

P vs NP

Decision Problems

- Decide if there is a solution (Yes/No questions)

Optimization Problems

- Determine an optimal solution

Class P

- A problem with a polynomial time algorithm solving all its instances (solved polynomially by a Turing machine)

Class NP

- If x is a Yes instance of the problem, there exists a certificate for x, whose validity can be checked in polynomial time (solved in polynomial time by a non-deterministic Turing machine)

P vs NP

Decision Problems

- Decide if there is a solution (Yes/No questions)

Optimization Problems

- Determine an optimal solution

Class P

- A problem with a polynomial time algorithm solving all its instances (solved polynomially by a Turing machine)

Class NP

- If x is a Yes instance of the problem, there exists a certificate for x, whose validity can be checked in polynomial time (solved in polynomial time by a non-deterministic Turing machine)

The Most Challenging Problem: $P = NP$?

Reduction

- A reduction f from problems B to A: given any instance I of B, $f(I)$ is an instance of A. I is Yes iff $f(I)$ is Yes

NP-Complete

- Problem A is one of the hardest problems in NP
- For any problem $B \in NP$, there is a polynomial time reduction from B to A
- A is NP-complete

NP-Hard

- For any problem $B \in NP$, there is a polynomial time reduction from B to A
- A is NP-hard

The Most Challenging Problem: $P = NP$?

Reduction

- A reduction f from problems B to A: given any instance I of B, $f(I)$ is an instance of A. I is Yes iff $f(I)$ is Yes

NP-Complete

- Problem A is one of the hardest problems in NP
- For any problem $B \in NP$, there is a polynomial time reduction from B to A
- A is NP-complete

NP-Hard

- For any problem $B \in NP$, there is a polynomial time reduction from B to A
- A is NP-hard

The Most Challenging Problem: $P = NP$?

Reduction

- A reduction f from problems B to A: given any instance I of B, $f(I)$ is an instance of A. I is Yes iff $f(I)$ is Yes

NP-Complete

- Problem A is one of the hardest problems in NP
- For any problem $B \in NP$, there is a polynomial time reduction from B to A
- A is NP-complete

NP-Hard

- For any problem $B \in NP$, there is a polynomial time reduction from B to A
- A is NP-hard

The Most Challenging Problem: $P = NP$?

Reduction

- A reduction f from problems B to A: given any instance I of B, $f(I)$ is an instance of A. I is Yes iff $f(I)$ is Yes

NP-Complete

- Problem A is one of the hardest problems in NP
- For any problem $B \in NP$, there is a polynomial time reduction from B to A
- A is NP-complete

NP-Hard

- For any problem $B \in NP$, there is a polynomial time reduction from B to A
- A is NP-hard

The Most Challenging Problem: $P = NP$?

How to Find the first NPC Problem?

- Sounds impossible, as you have to show all problems can be polynomially reduced to a specific problem
- SAT, done by Cook in 1971

How to Find the next NPC Problems?

- Repeat Cook's work? Not necessarily
- Polynomial time reduction is transitive
- Show SAT can be reduced in polynomial time to the problem you expect
- Karp proved 21 NPC problems in 1972

The Most Challenging Problem: $P = NP$?

How to Find the first NPC Problem?

- Sounds impossible, as you have to show all problems can be polynomially reduced to a specific problem
- SAT, done by Cook in 1971

How to Find the next NPC Problems?

- Repeat Cook's work? Not necessarily
- Polynomial time reduction is transitive
- Show SAT can be reduced in polynomial time to the problem you expect
- Karp proved 21 NPC problems in 1972

The Most Challenging Problem: $P = NP$?

How to Find the first NPC Problem?

- Sounds impossible, as you have to show all problems can be polynomially reduced to a specific problem
- SAT, done by Cook in 1971

How to Find the next NPC Problems?

- Repeat Cook's work? Not necessarily
- Polynomial time reduction is transitive
- Show SAT can be reduced in polynomial time to the problem you expect
- Karp proved 21 NPC problems in 1972

The Most Challenging Problem Becomes "Easier"

To Show $P = NP$

- Simply choose a suitable NPC problem and show a polynomial time algorithm

To Show $P \neq NP$

- Simply choose a suitable NPC problem and show it can not be solved in polynomial time

A "Common" Sense

- Most likely $P \neq NP$

The Most Challenging Problem Becomes "Easier"

To Show $P = NP$

- Simply choose a suitable NPC problem and show a polynomial time algorithm

To Show $P \neq NP$

- Simply choose a suitable NPC problem and show it can not be solved in polynomial time

A "Common" Sense

- Most likely $P \neq NP$

The Most Challenging Problem Becomes "Easier"

To Show $P = NP$

- Simply choose a suitable NPC problem and show a polynomial time algorithm

To Show $P \neq NP$

- Simply choose a suitable NPC problem and show it can not be solved in polynomial time

A "Common" Sense

- Most likely $P \neq NP$

The Most Challenging Problem Becomes "Easier"

To Show $P = NP$

- Simply choose a suitable NPC problem and show a polynomial time algorithm

To Show $P \neq NP$

- Simply choose a suitable NPC problem and show it can not be solved in polynomial time

A "Common" Sense

- Most likely $P \neq NP$

Solving a Combinatorial Optimization Problem

Combinatorial Optimization Problem

$$\min f(x) \quad s.t.\ x \in \Omega$$

where Ω is a finite set

Our Concerns

- Efficiency: How fast to obtain a solution?
- Effectiveness: How good the solution is?

Tradeoff

Running Times versus Performance Bounds

Solving a Combinatorial Optimization Problem

Combinatorial Optimization Problem

$$\min f(x) \quad s.t.\ x \in \Omega$$

where Ω is a finite set

Our Concerns

- Efficiency: How fast to obtain a solution?
- Effectiveness: How good the solution is?

Tradeoff

Running Times versus Performance Bounds

Solving a Combinatorial Optimization Problem

Combinatorial Optimization Problem

$$\min f(x) \quad s.t.\ x \in \Omega$$

where Ω is a finite set

Our Concerns

- Efficiency: How fast to obtain a solution?
- Effectiveness: How good the solution is?

Tradeoff

Running Times versus Performance Bounds

Distinguishing Problems

Easy Problems

Those admit a polynomial time algorithm, such as the minimum spanning tree problem, and the matching problem

Hard Problems

Those problems that can only be solved exponentially under the assumption $P \neq NP$

Distinguishing Problems

Easy Problems

Those admit a polynomial time algorithm, such as the minimum spanning tree problem, and the matching problem

Hard Problems

Those problems that can only be solved exponentially under the assumption $P \neq NP$

Research Topics

Complexity

Show a problem is in P by providing a polynomial time algorithm, or prove it is hard under some known assumptions (e.g. $P \neq NP$)

Algorithm Design

- Exact algorithms for easy problems with very low running times
- Exact algorithms for hard problems, that run efficiently in practice
- Approximation algorithms for hard problems, that have good performance bounds
- (Meta-)heuristics for any problem, that work well for some real-world instances

Research Topics

Complexity

Show a problem is in P by providing a polynomial time algorithm, or prove it is hard under some known assumptions (e.g. $P \neq NP$)

Algorithm Design

- Exact algorithms for easy problems with very low running times
- Exact algorithms for hard problems, that run efficiently in practice
- Approximation algorithms for hard problems, that have good performance bounds
- (Meta-)heuristics for any problem, that work well for some real-world instances

Highlighted Topics

Best with a Low Cost

Only exact solution is wanted, but with a reasonable running time

Cheap with a High Quality

Only (sub-, sup-) linear time is allowed, but with an acceptable performance bound

Highlighted Topics

Best with a Low Cost

Only exact solution is wanted, but with a reasonable running time

Cheap with a High Quality

Only (sub-, sup-) linear time is allowed, but with an acceptable performance bound

Coming Back to Hard Problems

Approximation Algorithms

- Constant factor approximation algorithm
- PTAS
- FPTAS
- Absolute approximation algorithm

Hardness

- No polynomial time algorithms
- No FPTAS
- No PTAS
- No constant ratio

Coming Back to Hard Problems

Approximation Algorithms

- Constant factor approximation algorithm
- PTAS
- FPTAS
- Absolute approximation algorithm

Hardness

- No polynomial time algorithms
- No FPTAS
- No PTAS
- No constant ratio

Lecture 1

Optimization Problems

Introduction

Basic Models

- A number of variables (continuous or discrete)
- A feasible set, usually represented by a set of constraints on variables
- An objective function to be optimized

Math Formulation

$$\begin{aligned} \min\ (\max) \quad & f(x) \\ s.t. \quad & x \in \Omega \end{aligned}$$

Feasible Set (I)

$$\Omega : \{h_i(x) = 0,\ i = 1, 2, \ldots, m,\ g_j(x) \leq 0,\ j = 1, 2, \ldots, l\}$$

Introduction

Basic Models

- A number of variables (continuous or discrete)
- A feasible set, usually represented by a set of constraints on variables
- An objective function to be optimized

Math Formulation

$$\begin{array}{rl} \min\ (\max) & f(x) \\ s.t. & x \in \Omega \end{array}$$

Feasible Set (I)

$$\Omega : \{h_i(x) = 0,\ i = 1, 2, \ldots, m,\ g_j(x) \leq 0,\ j = 1, 2, \ldots, l\}$$

Introduction

Basic Models

- A number of variables (continuous or discrete)
- A feasible set, usually represented by a set of constraints on variables
- An objective function to be optimized

Math Formulation

$$\begin{array}{rl} \min\ (\max) & f(x) \\ s.t. & x \in \Omega \end{array}$$

Feasible Set (I)

$$\Omega : \{h_i(x) = 0,\ i = 1, 2, \ldots, m,\ g_j(x) \le 0,\ j = 1, 2, \ldots, l\}$$

Examples

Linear Programming

- Let m and n be positive integers, $b \in Z^m$ and $c \in Z^n$, and A be an $m \times n$ matrix with elements $a_{ij} \in Z$. Then an LP instance is defined as
  $\Omega = \{x : x \in R^n, Ax = b, x \geq 0\}$ and $f = c^T x$

TSP

- Given an integer $n > 0$, and the distance matrix $[d_{ij}]$ between every pair of n points, a tour is a closed path visiting every point exactly once
- $\Omega = \{\text{all cyclic permutation } \pi \text{ on n points}\}$
- The cost function is $f(\pi) = \sum_{j=1}^{n} d_{\pi_j \pi_{j+1}}$, where $\pi_{n+1} = \pi_1$

Examples

Linear Programming

- Let m and n be positive integers, $b \in Z^m$ and $c \in Z^n$, and A be an $m \times n$ matrix with elements $a_{ij} \in Z$. Then an LP instance is defined as
  $\Omega = \{x : x \in R^n, Ax = b, x \geq 0\}$ and $f = c^T x$

TSP

- Given an integer $n > 0$, and the distance matrix $[d_{ij}]$ between every pair of n points, a tour is a closed path visiting every point exactly once
- $\Omega = \{\text{all cyclic permutation } \pi \text{ on n points}\}$
- The cost function is $f(\pi) = \sum_{j=1}^{n} d_{\pi_j \pi_{j+1}}$, where $\pi_{n+1} = \pi_1$

Neighborhoods

Definition

Given an optimization problem with instances (Ω, f), a neighborhood is a mapping

$$N : \Omega \longrightarrow 2^{\Omega}$$

Examples

- If $\Omega = R^n$, the set of points within a fixed Euclidean distance gives a natural neighborhood
- In the TSP, we define a neighborhood 2-change as $N_2(\pi) = \{\tau \in \Omega :$
$\tau \text{ can be obtained from } \pi \text{ by relink four points}\}$
- How about MST?

Neighborhoods

Definition

Given an optimization problem with instances (Ω, f), a neighborhood is a mapping

$$N : \Omega \longrightarrow 2^{\Omega}$$

Examples

- If $\Omega = R^n$, the set of points within a fixed Euclidean distance gives a natural neighborhood
- In the TSP, we define a neighborhood 2-change as $N_2(\pi) = \{\tau \in \Omega :$
  $\tau \text{ can be obtained from } \pi \text{ by relink four points}\}$
- How about MST?

Local and Global Optima

Definitions

- A feasible solution is local optimal with respect to a neighborhood N, if its value is the best among all points in N
- A feasible solution is globally optimal if its value is the best among all points in Ω
- A neighborhood N is exact if its local optimal solution is also global optimal

Examples

- TSP: N_2 is not exact, while N_n is
- MST?

Local and Global Optima

Definitions

- A feasible solution is local optimal with respect to a neighborhood N, if its value is the best among all points in N
- A feasible solution is globally optimal if its value is the best among all points in Ω
- A neighborhood N is exact if its local optimal solution is also global optimal

Examples

- TSP: N_2 is not exact, while N_n is
- MST?

Convex Sets and Functions

Convex Set

- A convex combination of two points $x, y \in R^n$:
  $z = \lambda x + (1 - \lambda) y$, where $0 \leq \lambda \leq 1$
- A set S is convex if it contains all convex combinations of pairs of points $x, y \in S$
- The intersection of any number of convex sets is convex

Examples

- R^n, $\emptyset$, any interval in R
- $\{x : Ax = b, x \geq 0\}$

Convex Sets and Functions

Convex Set

- A convex combination of two points $x, y \in R^n$:
  $z = \lambda x + (1-\lambda)y$, where $0 \leq \lambda \leq 1$
- A set S is convex if it contains all convex combinations of pairs of points $x, y \in S$
- The intersection of any number of convex sets is convex

Examples

- R^n, $\emptyset$, any interval in R
- $\{x : Ax = b, x \geq 0\}$

Convex Sets and Functions

Convex Functions

- Let S be a convex set in R^n (usually $S = R^n$)
- The function $f : S \longrightarrow R$ is convex in S if for any two points $x, y \in S$, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$, where $0 \leq \lambda \leq 1$
- For any $t \in R$, $S_t = \{x : f(x) \leq t, x \in S\}$ is convex
- f is concave if $-f$ is convex

Examples

- A linear function is convex and concave in any convex set S

Convex Sets and Functions

Convex Functions

- Let S be a convex set in R^n (usually $S = R^n$)
- The function $f : S \longrightarrow R$ is convex in S if for any two points $x, y \in S$, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$, where $0 \leq \lambda \leq 1$
- For any $t \in R$, $S_t = \{x : f(x) \leq t, x \in S\}$ is convex
- f is concave if $-f$ is convex

Examples

- A linear function is convex and concave in any convex set S

Convex Programming

Definition

- Minimization of a convex function on a convex set: f is convex and Ω is convex
- Usually Ω is defined by $\{x : g_i(x) \leq 0, i = 1, 2, \ldots, m\}$, where $g_i(x)$ is convex

A Smart Property

- The neighborhood $N_\epsilon(x) = \{y \in \Omega : \text{ and } ||x - y|| \leq \epsilon\}$ is exact for any $\epsilon > 0$
- Local optima are global as well (with respect to the Euclidean distance neighborhood)

Linear Programming

- LP is a special convex programming problem

Convex Programming

Definition

- Minimization of a convex function on a convex set: f is convex and Ω is convex
- Usually Ω is defined by $\{x : g_i(x) \leq 0, i = 1, 2, \ldots, m\}$, where $g_i(x)$ is convex

A Smart Property

- The neighborhood $N_\epsilon(x) = \{y \in \Omega : \text{ and } ||x - y|| \leq \epsilon\}$ is exact for any $\epsilon > 0$
- Local optima are global as well (with respect to the Euclidean distance neighborhood)

Linear Programming

- LP is a special convex programming problem

Convex Programming

Definition

- Minimization of a convex function on a convex set: f is convex and Ω is convex
- Usually Ω is defined by $\{x : g_i(x) \leq 0, i = 1, 2, \ldots, m\}$, where $g_i(x)$ is convex

A Smart Property

- The neighborhood $N_\epsilon(x) = \{y \in \Omega : \text{ and } ||x - y|| \leq \epsilon\}$ is exact for any $\epsilon > 0$
- Local optima are global as well (with respect to the Euclidean distance neighborhood)

Linear Programming

- LP is a special convex programming problem

Let us focus on LP first

Linear Programming Models

Example 1

- There are two products jointly produced by three firms

Firms	Product 1	Product 2	Resources
A	1	0	100
B	0	2	200
C	1	1	150

- Single values of the two products are 1 and 2, respectively
- Make a plan to maximize the total value of products

LP Formulation

$$\begin{array}{ll} \max & x_1 + 2x_2 \\ s.t. & x_1 \leq 100 \\ & 2x_2 \leq 200 \\ & x_1 + x_2 \leq 150 \\ & x_1, x_2 \geq 0 \end{array}$$

Linear Programming Models

Example 1

- There are two products jointly produced by three firms

Firms	Product 1	Product 2	Resources
A	1	0	100
B	0	2	200
C	1	1	150

- Single values of the two products are 1 and 2, respectively
- Make a plan to maximize the total value of products

LP Formulation

$$
\begin{array}{ll}
\max & x_1 + 2x_2 \\
s.t. & x_1 \leq 100 \\
 & 2x_2 \leq 200 \\
 & x_1 + x_2 \leq 150 \\
 & x_1, x_2 \geq 0
\end{array}
$$

Linear Programming Models

Extend to A General Problem

- There are n products jointly produced by m firms
- The j-th product has a value c_j
- The j-th product requires a_{ij} units of resources from the i-th firm
- The i-th firm has a resource amounting to b_i
- Maximize the total value

LP Formulation

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{n} c_j x_j \\
s.t. \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = 1, 2, \ldots, m \\
& x_j \geq 0, j = 1, 2, \ldots, n
\end{aligned}
$$

Linear Programming Models

Extend to A General Problem

- There are n products jointly produced by m firms
- The j-th product has a value c_j
- The j-th product requires a_{ij} units of resources from the i-th firm
- The i-th firm has a resource amounting to b_i
- Maximize the total value

LP Formulation

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{n} c_j x_j \\
s.t. \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = 1, 2, \ldots, m \\
& x_j \geq 0, j = 1, 2, \ldots, n
\end{aligned}
$$

Linear Programming Models

A Simplified Formulation

- $$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$
- $c^T = (c_1, c_2, \ldots, c_n)$, $b = (b_1, b_2, \ldots, b_m)^T$, $A = (a_{ij})_{m \times n}$

Example 2

- The final of EURO Cup is coming soon. Fans are ready for bidding which team will be the champion
- There are n teams in the final
- There are m bids, each of an n-dimensional vector. Namely, bid $b_i = (a_{i1}, \ldots, a_{in})$, where a_{ij} is 1 if bid i supposes team j is the champion, $a_{ij} = 0$, otherwise
- Each bidder i would like to pay π_i for each bet, and he can buy at most q_i bets
- If a bid consists of a champion team (as the game is over), the bidder wins w for each bet
- The dealer decides if accepts the bids and if yes how many bets, so that his benefit is maximized

Linear Programming Models

Formulation

- Let x_i be the number of bets offered to the bidder i.
  $0 \leq x_i \leq q_i$.
- The objective function to maximize is
  $$\sum_{i=1}^{m} \pi_i x_i - \max_{1 \leq j \leq n} \sum_{i=1}^{m} a_{ij} x_i w$$

Linear Model

$$\begin{aligned}
\max \quad & \sum_{i=1}^{m} \pi_i x_i - y \\
& y \geq \sum_{i=1}^{m} a_{ij} x_i w, \quad j = 1, 2, \ldots, n \\
& 0 \leq x_i \leq q_i, \quad i = 1, 2 \ldots, m
\end{aligned}$$

Linear Programming Models

Formulation

- Let x_i be the number of bets offered to the bidder i. $0 \le x_i \le q_i$.
- The objective function to maximize is $\sum_{i=1}^{m} \pi_i x_i - \max_{1 \le j \le n} \sum_{i=1}^{m} a_{ij} x_i w$

Linear Model

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{m} \pi_i x_i - y \\
& y \ge \sum_{i=1}^{m} a_{ij} x_i w, \quad j = 1, 2, \ldots, n \\
& 0 \le x_i \le q_i, \qquad i = 1, 2 \ldots, m
\end{aligned}
$$

Linear Programming Models

Formulation

- Let x_i be the number of bets offered to the bidder i. $0 \leq x_i \leq q_i$.
- The objective function to maximize is
$$\sum_{i=1}^{m} \pi_i x_i - \max_{1 \leq j \leq n} \sum_{i=1}^{m} a_{ij} x_i w$$

Linear Model

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} \pi_i x_i - y \\ & y \geq \sum_{i=1}^{m} a_{ij} x_i w, \quad j = 1, 2, \ldots, n \\ & 0 \leq x_i \leq q_i, \quad i = 1, 2 \ldots, m \end{aligned}$$

Linear Programming Models

Formulation

- Let x_i be the number of bets offered to the bidder i. $0 \le x_i \le q_i$.
- The objective function to maximize is $\sum_{i=1}^{m} \pi_i x_i - \max_{1 \le j \le n} \sum_{i=1}^{m} a_{ij} x_i w$

Linear Model

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{m} \pi_i x_i - y \\
& y \ge \sum_{i=1}^{m} a_{ij} x_i w, \quad j = 1, 2, \ldots, n \\
& 0 \le x_i \le q_i, \quad i = 1, 2 \ldots, m
\end{aligned}
$$