CSE 632: Analysis of Algorithms II: Combinatorial Optimization and Linear Programming (Fall 2020)

Introduction and Syllabus

Lecturer: Shi Li

Department of Computer Science and Engineering
University at Buffalo
Outline

1 Logistics

2 Contents of course
   - Combinatorial Optimization
   - Linear Programming
CSE 632: Analysis of Algorithms II: Combinatorial Optimization and Linear Programming

- Course Webpage
  - http://www.cse.buffalo.edu/~shil/courses/CSE632/
  - schedule, grading and academic integrity policy, homeworks, slides and notes

- Please sign up course on Piazza
  - https://piazza.com/buffalo/fall2020/cse632
  - announcements, polls, discussions

- Zoom meeting information can be found on Piazza
Course is Delivered 100% Online:

- Zoom
  - Lectures
  - Office hours
  - Request meetings outside lecture and office hour times
- Piazza
  - Announcements
  - Discussion forum
- UBLearns
  - Homework Submission
  - Quizzes
  - Final Exam
CSE 632: Analysis of Algorithms II: Combinatorial Optimization and Linear Programming

- WeFr, 11:30am-12:50pm
- Virtually on Zoom
- Instructor: Shi Li, shil@buffalo.edu
- Office hours: To be decided via poll on Piazza

Prerequisites

- CSE 431/531: Analysis of Algorithms I
- or equivalent course
- or permission from instructor
Online Resources

- There is no required textbook for this course.
- Slides and/or notes will be posted online before a lecture starts.
- Links to related courses and materials.
Grading Criteria

- 50% for 5 homeworks
- 20% for 4 quizzes
- 30% for online final exam
  - 11:45AM - 2:45PM, Tue, Dec 15
  - via UBLearns
Homeworks

5 homeworks, each is worth 10% of total score

You can

- use course materials (slides, lecture notes, linked resources)
- ask questions on Piazza and during office hours
- discuss with classmates
- think before asking questions and discussing with students
- must write down solutions on your own, in your own words
- give names of students you collaborated with

You can not

- copy solutions from other students
- copy solutions from Internet
Final Exam and Quizzes

- Conducted via UBLearns
- Closed-book

Quizzes

- 4 quizzes, each contributing 5% of total score.
- Each quiz contains 10 multiple-choice questions.
- 30 minutes to take each quiz
- Each quiz has a deadline (there is no specific time in which you should take the quiz)

Final Exam

- 30% of total score
- Time: 11:45AM - 2:45PM, Tue, Dec 15
Sanctions for Academic Integrity Violation

- first time offense in a homework/quiz: F for the homework/quiz
- second time offense, or offense during exam:
  - F for the course
  - lose financial support (for PhD students)
  - case recorded at departmental, decanal and university levels
  - suspension or expulsion from university
Outline

1 Logistics

2 Contents of course
   • Combinatorial Optimization
   • Linear Programming
1. Logistics

2. Contents of course
   - Combinatorial Optimization
   - Linear Programming
Combinatorial Optimization

Optimization Problems
- given an instance, find the “best” solution
Optimization Problems

- given an instance, find the “best” solution

Maximization vs Minimization

- minimization problem: minimize the cost of a solution
- maximization problem: maximize the gain of a solution
Combinatorial Optimization

Optimization Problems
- given an instance, find the “best” solution

Maximization vs Minimization
- minimization problem: minimize the cost of a solution
- maximization problem: maximize the gain of a solution

Continuous vs Discrete
- continuous optimization: solutions from a continuous space
- discrete optimization: solutions from a discrete set
### Combinatorial Optimization

**Optimization Problems**
- given an instance, find the “best” solution

**Maximization vs Minimization**
- minimization problem: minimize the cost of a solution
- maximization problem: maximize the gain of a solution

**Continuous vs Discrete**
- continuous optimization: solutions from a continuous space
- discrete optimization: solutions from a discrete set

**Combinatorial optimization**: subset of discrete optimization
Combinatorial optimization: subset of discrete optimization
Combinatorial optimization: subset of discrete optimization

hard to define “combinatorial” formally
Combinatorial Optimization

- **Combinatorial optimization**: subset of discrete optimization
- hard to define “combinatorial” formally

**typical scenario:**
- \( U \): ground set of elements
- a solution is a **subset** of \( U \) satisfying some properties
- find the solution that minimizes the cost, or maximizes the gain.
Most of the time in the course, we deal with **graph problems**.

- given: a graph (undirected or directed), and other parameters
- a solution is a subset of vertices and/or edges satisfying some properties
- goal: find a solution with the minimum cost, or the maximum profit
Examples of graph optimization problems

**shortest path**

**Input:** (undirected or directed) graph $G = (V, E)$, $s, t \in V$, costs on edges

**Output:** Goal: minimum-cost path connecting $s$ to $t$ in $G$
Examples of graph optimization problems

**shortest path**

**Input:** (undirected or directed) graph \( G = (V, E) \),
\[s, t \in V\), costs on edges

**Output:** Goal: minimum-cost path connecting \( s \) to \( t \) in \( G \)
Examples of graph optimization problems

**maximum bipartite matching**

**Input:** bipartite graph \( G = (L \cup R, E) \)

**Output:** a maximum-size matching between \( L \) and \( R \)
Examples of graph optimization problems

maximum bipartite matching

**Input:** bipartite graph $G = (L \cup R, E)$

**Output:** a maximum-size matching between $L$ and $R$
Examples of graph optimization problems

**maximum independent set**

**Input:** undirected graph $G = (V, E)$

**Output:** maximum independent set of $G$
Examples of graph optimization problems

### Maximum Independent Set

**Input:** undirected graph \( G = (V, E) \)

**Output:** maximum independent set of \( G \)
Trivial Algorithm for a Combinatorial Optimization Problem

- **Enumerate** all possible solutions, find the best one

Number of solutions is often exponentially large. Trivial algorithm runs in exponential time. In the course, we focus on polynomial time (efficient) algorithms. Goal: designing efficient algorithms for combinatorial optimization problems.
Trivial Algorithm for a Combinatorial Optimization Problem

- **Enumerate** all possible solutions, find the best one

- Number of solutions is often exponentially large
- Trivial algorithm runs in exponential time
Trivial Algorithm for a Combinatorial Optimization Problem

- **Enumerate** all possible solutions, find the best one

- Number of solutions is often exponentially large
- Trivial algorithm runs in exponential time
- In the course, we focus on polynomial time (efficient) algorithms
Trivial Algorithm for a Combinatorial Optimization Problem

- **Enumerate** all possible solutions, find the best one

- Number of solutions is often exponentially large
- Trivial algorithm runs in exponential time

- In the course, we focus on polynomial time (efficient) algorithms
- Goal: designing efficient algorithms for combinatorial optimization problems
Designing Efficient Algorithms for Combinatorial Optimization Problems

• For problems in P: design exact algorithms
For problems in P: design exact algorithms.
For problems that are NP-hard: design approximation algorithms.
Designing Efficient Algorithms for Combinatorial Optimization Problems

- For problems in P: design **exact algorithms**
- For problems that are NP-hard: design **approximation algorithms**

**Approximation Algorithms**

- do not necessarily output the best solution
- but the solution output is not too bad compared to optimum solution
Tentative Schedule (28 Lectures in Total)

- Introduction (0.5 Lecture)
- Network Flow ($\approx 5$ Lectures)
- Matroid, submodular functions and Greedy Algorithms ($\approx 6$ Lectures)
- Basics of Linear Programming ($\approx 4$ Lectures)
- Exact Linear Program Polytopes ($\approx 6$ Lectures)
- Linear Programming Rounding and Primal-Dual Algorithms ($\approx 6$ Lectures)
- Other Topics (Potentially, Depending on progress)
Goal: send as much flow as possible from $s$ to $t$
Goal: send as much flow as possible from $s$ to $t$
Network Flow

- Goal: send as much flow as possible from $s$ to $t$

- many problems can be reduced to network flow
Maximum Weight Spanning Tree

**Input:** graph with edge weights (weights = profits)

**Output:** the maximum weight spanning tree (or a sub-graph without cycles)
Maximum Weight Spanning Tree

**Input:** graph with edge weights (weights = profits)

**Output:** the maximum weight spanning tree (or a sub-graph without cycles)
Greedy Algorithm for Maximum Weight Spanning Tree

1: \( F \leftarrow \emptyset \)
2: \textbf{repeat}
3: \quad \text{find the most profitable edge } e \in E \setminus F \text{ such that } F \cup \{e\} \text{ does not contain a cycle}
4: \quad F \leftarrow F \cup \{e\}
5: \textbf{until } F \text{ is a spanning tree}
Greedy Algorithm for Maximum Weight Spanning Tree

1:  \( F \leftarrow \emptyset \)
2:  repeat
3:     find the most profitable edge \( e \in E \setminus F \) such that \( F \cup \{e\} \) does not contain a cycle
4:     \( F \leftarrow F \cup \{e\} \)
5:  until \( F \) is a spanning tree
Matroid and Greedy Algorithms

Greedy Algorithm for Maximum Weight Spanning Tree

1: $F \leftarrow \emptyset$
2: repeat
3:    find the most profitable edge $e \in E \setminus F$ such that $F \cup \{e\}$ does not contain a cycle
4:    $F \leftarrow F \cup \{e\}$
5: until $F$ is a spanning tree
Greedy Algorithm for Maximum Weight Spanning Tree

1: \( F \leftarrow \emptyset \)
2: \textbf{repeat}
3: \quad find the most profitable edge \( e \in E \setminus F \) such that \( F \cup \{e\} \) does not contain a cycle
4: \quad \( F \leftarrow F \cup \{e\} \)
5: \textbf{until} \( F \) is a spanning tree
Greedy Algorithm for Maximum Weight Spanning Tree

1: \( F \leftarrow \emptyset \)
2: repeat
3: find the most profitable edge \( e \in E \setminus F \) such that \( F \cup \{e\} \) does not contain a cycle
4: \( F \leftarrow F \cup \{e\} \)
5: until \( F \) is a spanning tree
**Greedy Algorithm for Maximum Weight Spanning Tree**

1: \( F \leftarrow \emptyset \)

2: **repeat**

3: find the most profitable edge \( e \in E \setminus F \) such that \( F \cup \{e\} \) does not contain a cycle

4: \( F \leftarrow F \cup \{e\} \)

5: **until** \( F \) is a spanning tree

---

![Graph](image-url)
Greedy Algorithm for Maximum Weight Spanning Tree

1: \( F \leftarrow \emptyset \)
2: \textbf{repeat}
3: \hspace{0.5cm} \text{find the most profitable edge } e \in E \setminus F \text{ such that } F \cup \{e\} \text{ does not contain a cycle}
4: \hspace{0.5cm} F \leftarrow F \cup \{e\}
5: \hspace{0.5cm} \textbf{until } \ F \text{ is a spanning tree}
Greedy Algorithm for Maximum Weight Spanning Tree

1: \( F \leftarrow \emptyset \)
2: \textbf{repeat}
3: \quad \text{find the most profitable edge } e \in E \setminus F \text{ such that } F \cup \{e\} \text{ does not contain a cycle}
4: \quad F \leftarrow F \cup \{e\}
5: \textbf{until } F \text{ is a spanning tree}
Greedy Algorithm for Maximum Weight Spanning Tree

1: \( F \leftarrow \emptyset \)
2: \textbf{repeat}
3: \quad \text{find the most profitable edge } e \in E \setminus F \text{ such that } F \cup \{e\} \text{ does not contain a cycle}
4: \quad F \leftarrow F \cup \{e\}
5: \textbf{until} \ F \text{ is a spanning tree}
A Generic Problem

**Input:** $E$: ground set, weights on $E$

**Output:** a maximum weight subset $F$ of $E$ that does not contain any forbidden structure

A Generic Greedy Algorithm

1: $F \leftarrow \emptyset$

2: repeat

3: find the most profitable element $e \in E \setminus F$ such that $F \cup \{e\}$ does not contain a forbidden structure

4: $F \leftarrow F \cup \{e\}$

5: until $F$ can not be augmented any more
A Generic Problem

**Input:** \( E \): ground set, weights on \( E \)

**Output:** a maximum weight subset \( F \) of \( E \) that does not contain any forbidden structure

**Q:** When will the algorithm give an optimum solution?
A Generic Problem

**Input:** $E$: ground set, weights on $E$

**Output:** a maximum weight subset $F$ of $E$ that does not contain any forbidden structure

**Q:** When will the algorithm give an optimum solution?

**A:** If the valid subsets of $E$ form a matroid, then the generic algorithm is optimum.
Outline

1. Logistics

2. Contents of course
   - Combinatorial Optimization
   - Linear Programming
Example of A Linear Program

\[ \text{min } 7x_1 + 4x_2 \]
\[ x_1 + x_2 \geq 5 \]
\[ x_1 + 2x_2 \geq 6 \]
\[ 4x_1 + x_2 \geq 8 \]
\[ x_1, x_2 \geq 0 \]
Example of A Linear Program

\[
\begin{align*}
\min & \quad 7x_1 + 4x_2 \\
& x_1 + x_2 \geq 5 \\
& x_1 + 2x_2 \geq 6 \\
& 4x_1 + x_2 \geq 8 \\
& x_1, x_2 \geq 0
\end{align*}
\]
Example of A Linear Program

\[
\begin{align*}
\min & \quad 7x_1 + 4x_2 \\
x_1 + x_2 & \geq 5 \\
x_1 + 2x_2 & \geq 6 \\
4x_1 + x_2 & \geq 8 \\
x_1, x_2 & \geq 0
\end{align*}
\]
Example of A Linear Program

\[ \begin{align*}
\text{min} & \quad 7x_1 + 4x_2 \\
& x_1 + x_2 \geq 5 \\
& x_1 + 2x_2 \geq 6 \\
& 4x_1 + x_2 \geq 8 \\
& x_1, x_2 \geq 0
\end{align*} \]
Example of A Linear Program

\[ \begin{align*}
\text{min} & \quad 7x_1 + 4x_2 \\
& x_1 + x_2 \geq 5 \\
& x_1 + 2x_2 \geq 6 \\
& 4x_1 + x_2 \geq 8 \\
& x_1, x_2 \geq 0
\end{align*} \]
Example of A Linear Program

\[
\begin{align*}
\text{min} & \quad 7x_1 + 4x_2 \\
& x_1 + x_2 \geq 5 \\
& x_1 + 2x_2 \geq 6 \\
& 4x_1 + x_2 \geq 8 \\
& x_1, x_2 \geq 0
\end{align*}
\]

- optimum point:\n\[
x_1 = 1, x_2 = 4
\]
Example of A Linear Program

\[
\begin{align*}
\text{min} & \quad 7x_1 + 4x_2 \\
& x_1 + x_2 \geq 5 \\
& x_1 + 2x_2 \geq 6 \\
& 4x_1 + x_2 \geq 8 \\
& x_1, x_2 \geq 0
\end{align*}
\]

- optimum point:
  \[ x_1 = 1, x_2 = 4 \]
- value = \[ 7 \times 1 + 4 \times 4 = 23 \]
Standard Form of Linear Programming

\[
\begin{align*}
\text{min} \quad & \quad c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
\text{s.t.} \quad & \quad \sum A_{1,1} x_1 + A_{1,2} x_2 + \cdots + A_{1,n} x_n \geq b_1 \\
& \quad \sum A_{2,1} x_1 + A_{2,2} x_2 + \cdots + A_{2,n} x_n \geq b_2 \\
& \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
& \quad \sum A_{m,1} x_1 + A_{m,2} x_2 + \cdots + A_{m,n} x_n \geq b_m \\
& \quad x_1, x_2, \cdots, x_n \geq 0
\end{align*}
\]
Recall combinatorial optimization

- ground set $U$
- solution: a subset $A \subseteq U$ satisfying some properties
- minimize / maximize total cost / gain of $A$
Recall combinatorial optimization
- ground set $U$
- solution: a subset $A \subseteq U$ satisfying some properties
- minimize / maximize total cost / gain of $A$

use $x \in \{0, 1\}^U$ to encode a solution
Recall combinatorial optimization
- ground set $U$
- solution: a subset $A \subseteq U$ satisfying some properties
- minimize / maximize total cost / gain of $A$

use $x \in \{0, 1\}^U$ to encode a solution

problem equivalent to an integer program
Integer Program

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t. } \\
\text{some linear constraints on } x \\
x \in \{0, 1\}^U
\]
Integer Program

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.}
\]

some linear constraints on \( x \)

\( x \in \{0, 1\}^U \)

- solving integer program is \textbf{NP-hard} in general
Linear Programming in Combinatorial Optimization

**Integer Program**

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.}
\]

some linear constraints on \( x \)

\[ x \in \{0, 1\}^U \]

**Linear Program**

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.}
\]

some linear constraints on \( x \)

\[ x \in [0, 1]^U \]

- solving integer program is **NP-hard** in general
Integer Program

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.}
\]

some linear constraints on \( x \)

\[ x \in \{0, 1\}^U \]

- solving integer program is NP-hard in general

Linear Program

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.}
\]

some linear constraints on \( x \)

\[ x \in [0, 1]^U \]

- solving integer program is in P
**Integer Program**

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.}
\]

some linear constraints on \( x \)

\( x \in \{0, 1\}^U \)

- solving integer program is NP-hard in general

**Linear Program**

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.}
\]

some linear constraints on \( x \)

\( x \in [0, 1]^U \)

- solving integer program is in P

**Q:** How does relaxing \( x \in \{0, 1\}^U \) to \( x \in [0, 1]^U \) affect the problem?
Integer Program

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.}
\]

some linear constraints on \( x \)

\( x \in \{0, 1\}^U \)

Linear Program

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.}
\]

some linear constraints on \( x \)

\( x \in [0, 1]^U \)

Q: How does relaxing \( x \in \{0, 1\}^U \) to \( x \in [0, 1]^U \) affect the problem?
Linear Programming in Combinatorial Optimization

**Integer Program**

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.} \quad \text{some linear constraints on } x \\
x \in \{0, 1\}^U
\]

**Linear Program**

\[
\min \sum_{i \in U} c_i x_i, \text{ s.t.} \quad \text{some linear constraints on } x \\
x \in [0, 1]^U
\]

**Q:** How does relaxing \( x \in \{0, 1\}^U \) to \( x \in [0, 1]^U \) affect the problem?

**A:**

- for some problems, linear program \( \equiv \) integer program \( \Rightarrow \) solving the problem **exactly**
- for some other problems, the **gap** is small \( \Rightarrow \) solving the problem **approximately**
Tentative Schedule (28 Lectures in Total)

- Introduction (0.5 Lecture)
- Network Flow (≈ 5 Lectures)
- Matroid, submodular functions and Greedy Algorithms (≈ 6 Lectures)
- Basics of Linear Programming (≈ 4 Lectures)
- Exact Linear Program Polytopes (≈ 6 Lectures)
- Linear Programming Rounding and Primal-Dual Algorithms (≈ 6 Lectures)
- Other Topics (Potentially, Depending on progress)
Tentative Schedule (28 Lectures in Total)

- Introduction (0.5 Lecture)
- Network Flow (≈ 5 Lectures)
- Matroid, submodular functions and Greedy Algorithms (≈ 6 Lectures)
- Basics of Linear Programming (≈ 4 Lectures)
  - linear programming and methods for solving LP
  - Formulating Problems as Linear Programs
  - Linear Programming Duality
  - Application of Linear Programming Duality: Nash Equilibrium
- Exact Linear Program Polytopes (≈ 6 Lectures)
- Linear Programming Rounding and Primal-Dual Algorithms (≈ 6 Lectures)
- Other Topics (Potentially, Depending on progress)
Tentative Schedule (28 Lectures in Total)

- Introduction (0.5 Lecture)
- Network Flow (≈ 5 Lectures)
- Matroid, submodular functions and Greedy Algorithms (≈ 6 Lectures)

- Basics of Linear Programming (≈ 4 Lectures)
- Exact Linear Program Polytopes (≈ 6 Lectures)
  - Bipartite Matching
  - Matroid Polytopes
  - Intersection of Two Matroid Polytopes
  - Non-bipartite Matching Polytopes

- Linear Programming Rounding and Primal-Dual Algorithms (≈ 6 Lectures)
- Other Topics (Potentially, Depending on progress)
Tentative Schedule (28 Lectures in Total)

- Introduction (0.5 Lecture)
- Network Flow (≈ 5 Lectures)
- Matroid, submodular functions and Greedy Algorithms (≈ 6 Lectures)
- Basics of Linear Programming (≈ 4 Lectures)
- Exact Linear Program Polytopes (≈ 6 Lectures)
- Linear Programming Rounding and Primal-Dual Algorithms (≈ 6 Lectures)
  - Randomized Rounding
  - Concentration Bounds
  - Primal-Dual for Weighted Bipartite matching
  - Approximation Algorithms via Primal-Dual
- Other Topics (Potentially, Depending on progress)
Questions?