EECS598-001
Approximation Algorithms & Hardness of Approximation

Pingbang Hu

August 30, 2022

Abstract

This is an advanced graduate-level algorithm course taught in University of Michigan by Euiwoong Lee. Topics include both approximation algorithms like covering, clustering, network design, and constraint satisfaction problems (the first half), and also the hardness of approximation algorithms (the second half).

The first half of the course is classical and well-studied, and we'll use Williamson and Shmoys[WS11], Vazirani[Vaz02] as our reference. The second half of the course is still developing, and we'll look into papers by Barak and Steurer[BS14], O'donnell[ODo21], etc.

This course is taken in Fall 2022, and the date on the covering page is the last updated time.

Contents

Chapter 1

Introduction

Lecture 1: Overview, Set Cover

29 Aug. 10:30

1.1 Computational Problem

We're interested in the following optimization problem: Given a problem with an input, we want to either maximize or minimize some objectives. This suggests the following definition.

Definition 1.1.1 (Computational problem)**.** A *computational problem* P is a function from input I to (X, f), where X is the feasible set of I and f is the objective function.

We see that by replacing f with $-f$, we can unify the notion and only consider either minimization or maximization, but we will not bother to do this.

Example (s-t shortest path)**.** The *s-t shortest path* problem P can be formalized as follows. Given input I, it defines

- Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and two vertices $s, t \in \mathcal{V}$.
- Feasible set: $X = \{\text{set of all (simple) paths } s \text{ to } t\}$.
- Objective function: $f \colon X \to \mathbb{R}$ where $f(x) = \text{length}(x)$ ($\#$ of edges of x).

The output of P should be some $x \in X$ (i.e., some valid s-t paths) such that it minimizes $f(x)$.

We see that the computational problem we focus on is an optimization problem, and more specifically, we're interested in combinatorial optimization.

Definition 1.1.2 (Combinatorial optimization)**.** A *combinatorial optimization* problem is a problem where the feasible set X is a finite set.

Example (s-t shortest path)**.** The s-t shortest path problem is an combinatorial optimization problem since given a graph $\mathcal{G}$ with $n = |\mathcal{V}|$, $m = |\mathcal{E}|$, there are at most $n!$ different paths, i.e., $|X| \leq n! < \infty$.

Note. We'll also look into some continuous optimization problem, where X is now infinite (or even uncountable). For example, find $x \in \mathbb{R}$ that minimizes $f(x) = x^2 + 2x + 1$. In this case, $X = \mathbb{R}$ which is uncountable (hence infinite).

1.2 Efficient Algorithms

Given a problem P, we want to solve it fast with algorithms. Before we characterize the speed of an algorithm, we should first define what exactly an algorithm is.

Definition 1.2.1 (Algorithm)**.** Given a problem P and input I (which defines X and f), an *algorithm* A outputs solution $y = A(I)$ such that $y \in X$ and $y = \arg\max_{x \in X} f(x)$ or $\arg\min_{x \in X}$, depending on I.

Definition 1.2.2 (Efficient)**.** We say that an algorithm A is *efficient* if it runs in **polynomial time**.

Remark (Runtime parametrization)**.** The *runtime* of an algorithm A should be parametrized by the size of input I. Formally, given input I represented in s bits, runtime of A on I should be $\text{poly}(s)$ for A to be efficient.

Note. In most cases, there are 1 or 2 parameters that essentially define the size of input.

Example (Graph)**.** A natural representation of a graph with n vertices and m edges are

(a) Adjacency matrix: n^2 numbers.

(b) Adjacency list: $O(m+n)$ numbers.

Example (Set system)**.** A set system with n elements and m sets has a natural representation which uses $O(nm)$ numbers.

Example. If an input I can be represented by s bits, then the runtime of an algorithm can be $O(s \log s)$, $O(s^2)$, or $O(s^{100})$, which are considered as efficient. On the other hand, something like 2^s or $s!$ are not.

Hence, our goal is to get $\text{poly}((n,m))$-time algorithm!

1.3 Approximation Algorithms

We first note that many interesting combinatorial optimization problems are NP-hard, hence it's impossible to find optimum in polynomial time unless P is NP. This suggests one problem: *How well can we do in polynomial time?*

In normal cases, we may assume that objective function value is always positive, i.e., $f\colon X \to \mathbb{R}^* \cup \{0\}$. Then, we have the following definition which characterize the *slackness*.

Definition 1.3.1 (Approximation algorithm)**.** Given an algorithm A, we say A is an *α-approximation algorithm* for a problem P if for every input I of P,

- Min: $f(A(I)) \le \alpha \cdot \mathsf{OPT}(I)$ for $\alpha \ge 1$
- Max: $f(A(I)) \ge \alpha \cdot \mathsf{OPT}(I)$ for $\alpha \le 1$

where we define $\mathsf{OPT}(I)$ as $\max_{x \in X} f(x)$ for maximization, $\min_{x \in X} f(x)$ if minimization.

We see that α characterizes the slackness allowed for our algorithm A. Now, we're ready to look at some interesting problems. Broadly, there are around 10 classes of them which are actively studied:

- We'll see: Covering, Clustering, Network design, Constraint satisfaction.
- We'll not see: graph cuts, Packing, Scheduling, String, etc.

The above list is growing! For example, applications of continuous optimization in combinatorial optimization is getting attention recently. Also, there are around 8 techniques developed, e.g., greedy, local search, LP rounding, SDP rounding, primal-dual, cuts and metrics, etc.

1.4 Hardness

For most problems we saw, we can even say that getting an α-approximation is NP-hard for some $\alpha > 1$. This bound is sometimes tight, but not always, and we'll focus on this part in the second half of this course.

Chapter 2

Covering

Before we jump into any problem formulations, we define a fundamental object in combinatorial optimization, the set system.

Definition 2.0.1 (Set system)**.** Given a ground set Ω (often called *universe*), the *set system* is an order pair $(\Omega, \mathcal{S})$ where $\mathcal{S}$ is a collection of subsets of Ω.

Note. Foa a set system $(\Omega, \mathcal{S})$, we often let $m \coloneqq |\mathcal{S}|$ and $n \coloneqq |\Omega|$.

Definition 2.0.2 (Degree)**.** Given a set system $(\Omega, \mathcal{S})$, the *degree of* $x \in \Omega$, $\deg(x)$, is defined as

$$\deg(x) \coloneqq |\{S \in \mathcal{S} \mid x \in S\}| .$$

Remark (Bipartite representation)**.** Naturally, for a set system, we have a bipartite representation.

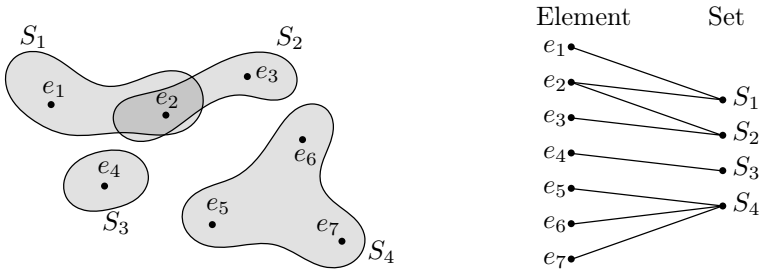

Figure 2.1: Bipartite representation of a set system.

Denote $d \coloneqq \max_{e \in U} \deg(e) \leq m$ and $k \coloneqq \max_{i \in [m]} |S_i| \leq n$, which is just the maximum vertex degree on two sides of the bipartite graph representation of this set system.

Finally, we have the following.

Definition 2.0.3 (Covering)**.** A *covering* $\mathcal{S}' \subseteq \mathcal{S}$ of $(\Omega, \mathcal{S})$ is a (sub)collection of subsets such that $\bigcup_{S \in \mathcal{S}'} S = \Omega$.

2.1 Set Cover

Let's first consider the classical problem called set cover.

Problem 2.1.1 (Set cover). Given a finite set system $(U, \mathcal{S})$ where $\mathcal{S} \coloneqq \{S_i \subseteq U\}_{i=1}^m$, find a covering $\mathcal{S}'$ while minimizing $|\mathcal{S}'|$.

Assuming there always exists at least one covering, we can in fact get two types of non-comparable approximation ratio in terms of k and d. Specifically, we get $\log k$ and d-approximation ratio via either greedy, LP rounding or dual-methods.

2.1.1 Greedy: In terms of k

We first see the algorithm.

Algorithm 1: Greedy I – set cover

Data: A set system $(U, \mathcal{S})$
Result: A covering $\mathcal{S}'$

```
1 S' ← ∅, i ← 0
2 while U ≠ ∅ do                                   // O(n)
3     Choose S_i with maximum |U ∩ S_i|            // O(mn)
4     S' ← S' ∪ {S_i}
5     U ← U \ S_i
6     for e ∈ U ∩ S_i do
7         y_e ← 1/|U ∩ S_i|                        // Average costs
8     i ← i + 1
9 return S'
```

Remark. It's clear that Algorithm 1 is a polynomial time algorithm, also, the output $\mathcal{S}'$ is always a valid covering.

Theorem 2.1.1. Algorithm 1 is an H_k-approximation[a] algorithm.

[a] H_k is the so-called *harmonic number*, which is defined as $\sum_{i=1}^k 1/i \le \ln k + 1$.

Proof. Denote the OPT as $\mathcal{S}^* \coloneqq \{S_1^*, \dots, S_\ell^*\}$, and first note that the average cost y_e essentially maintains $\sum_{e \in U} y_e = |\mathcal{S}'|$, hence we just need to bound y_e w.r.t. S^*. To do this, for any $S^* \in \mathcal{S}^*$, say $S_1^* =: \{e_1, \dots, e_k\}$ where we number e_i in terms of the order of which being deleted, i.e., e_1 is deleted first from U (line 5), etc.

Note. S_1^* can have less than k element, but in that case similar argument will follow. Also, if some elements are deleted at the same time, we just order them arbitrarily.

Then, we have the following claim.

Claim. For all e_i, $y_{e_i} \le \frac{1}{k-i+1}$.

Proof. Consider the iteration when e_i was picked by S', i.e., $|U \cap S'| \ge |U \cap S_1^*| \ge k - i + 1$, then by definition (line 5) we have $y_{e_i} = \frac{1}{|U \cap S'|} \le \frac{1}{|U \cap S_1^*|} \le \frac{1}{k-i+1}$. ⊛

We immediately see that whenever the optimal solution pays 1 (for choosing S_1^* for instance), Algorithm 1 pays at most H_k since $\sum_{e_i \in S_1^*} y_{e_i} \le \sum_{i=1}^k \frac{1}{k-i+1} = H_k$, or more formally,

$$|\mathcal{S}'| = \sum_{e \in U} y_e \le \sum_{S_i^* \in \mathcal{S}^*} \underbrace{\sum_{e \in S_i^*} y_e}_{\le H_k} \le \ell \cdot H_k = H_k \cdot |\mathsf{OPT}|,$$

which finishes the proof. ■

In all, observe that $H_k \le \ln k + 1$, we see that Algorithm 1 is a $(\ln k)$-approximation algorithm.

Appendix

Bibliography

[BS14] Boaz Barak and David Steurer. *Sum-of-squares proofs and the quest toward optimal algorithms*. 2014. DOI: 10.48550/ARXIV.1404.5236. URL: https://arxiv.org/abs/1404.5236.

[ODo21] Ryan O'Donnell. *Analysis of Boolean Functions*. 2021. DOI: 10.48550/ARXIV.2105.10386. URL: https://arxiv.org/abs/2105.10386.

[Vaz02] V.V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2002. ISBN: 9783540653677. URL: https://books.google.com/books?id=EILqAmzKgYIC.

[WS11] D.P. Williamson and D.B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. ISBN: 9781139498173. URL: https://books.google.com/books?id=Cc%5C_Fdqf3bBgC.