EECS598-001
Approximation Algorithms & Hardness of Approximation

Pingbang Hu

September 13, 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

1.1 Computational Problem

29 Aug. 10:30

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 $f(x) = \mathrm{length}(x)$ ($\#$ 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 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 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., $f: X \to \mathbb{R}^* \cup \{0\}$. 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: $f(A(I)) \leq \alpha \cdot \mathsf{OPT}(I)$ for $\alpha \geq 1$
- Max: $f(A(I)) \geq \alpha \cdot \mathsf{OPT}(I)$ for $\alpha \leq 1$

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 cover, clustering, network design, and constraint satisfaction problems.
- 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

2.1 Set Cover

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

Definition 2.1.1 (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 2.1.2 (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.

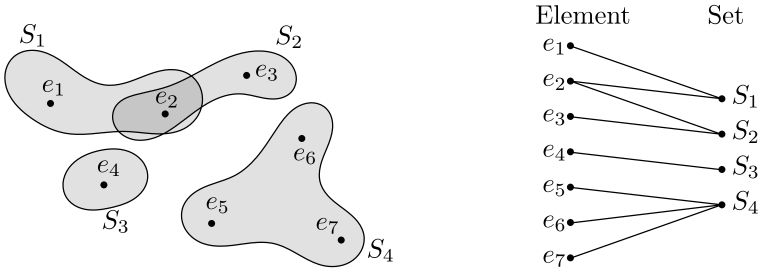


Figure 2.1: Bipartite representation of a set system.

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

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 $\mathcal{S} \coloneqq \{S_i \subseteq U\}_{i=1}^m$ along with a weight function $w \colon \mathcal{S} \to \mathbb{R}^+$, find a covering $\mathcal{S}'$ while minimizing $\sum_{S \in \mathcal{S}'} w(S)$.

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.

2.2 Greedy Method

We first see the algorithm when $w(S) = 1$ for all $S \in \mathcal{S}$.

Algorithm 1: Set cover – Greedy

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

1 $\mathcal{S}' \leftarrow \varnothing$, $i \leftarrow 0$
2 **while** $U \neq \varnothing$ **do** // $O(n)$
3   Choose S_i with maximum $|U \cap S_i|$ // $O(mn)$
4     $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{S_i\}$
5     $U \leftarrow U \setminus S_i$
6   **for** $e \in U \cap S_i$ **do**
7         $y_e \leftarrow w(S_i) / |U \cap S_i|$ // Average costs
8     $i \leftarrow i + 1$
9 **return** $\mathcal{S}'$

We focus on the case that $w(S) = 1$ for all S.

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

Theorem 2.2.1. Algorithm 1 is an H_k-approximation[a] algorithm.

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

Proof. Denote the OPT as $\mathcal{S}^* := \{S_1^*, \ldots, S_\ell^*\}$, 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 $S_1^* =: \{e_1, \ldots, e_k\}$ 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, $y_{e_i} \leq \frac{1}{k-i+1}$.

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

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

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

which finishes the proof. ■

In all, observe that $H_k \leq \ln k + 1$, we see that Algorithm 1 is a $(\ln k)$-approximation algorithm. Also, the weighted version can be easily derived by replacing 1 with the corresponding weight.

Lecture 2: Linear Programming with Set Covers

2.3 Linear Programming Rounding

31 Aug. 10:30

To get a d-approximation algorithm, instead of seeing the greedy algorithm, we first see the LP[1] dual method, which turns out to be exactly the same as the greedy algorithm.

> **As previously seen.** Both linear programming and convex programming can be solved in polynomial time.

Notice that it's more natural to define set cover in terms of ILP (integer LP). Define our integer variables $\{x_i\}_{i\in[n]}$ such that

$$x_i = \begin{cases} 1, & \text{if } S_i \in \mathcal{S}'; \\ 0, & \text{otherwise.} \end{cases}$$

In this way, we have the following ILP formulation for set cover as

$$\begin{aligned} & \min && \sum_i w_i \cdot x_i \\ & && \sum_{S_i \ni e} x_i \geq 1 && \forall i \in U \\ & \text{(IP)} && x_i \in \{0,1\} && \forall i. \end{aligned}$$

But we know that this is an NP-hard problem, so we relax it to be

$$\begin{aligned} & \min && \sum_i w_i \cdot x_i \\ & && \sum_{S_i \ni e} x_i \geq 1 && \forall i \in U \\ & \text{(LP)} && x_i \geq 0 && \forall i. \end{aligned}$$

Write it in a more compact form, we have

$$\begin{aligned} \min \quad & \langle w, x\rangle \\ & Ax \geq \mathbb{1} \\ & x \geq 0 \end{aligned}$$

where $A \in \mathbb{R}^{n\times m}$ such that

$$A_{ij} = \begin{cases} 1, & \text{if } e_i \in S_j; \\ 0, & \text{otherwise.} \end{cases}$$

> **Note.** Note when we do relaxation, we want $x \in \mathrm{fes(IP)} \Rightarrow x \in \mathrm{fea(LP)}$, i.e., $\mathrm{fes(LP)} \supseteq \mathrm{fes(IP)}$. Note that in this case, for a minimization problem, we have
>
> $$f(x) = \mathsf{OPT}_{\mathrm{LP}} \leq \mathsf{OPT}_{\mathrm{IP}}\,.$$

In this case, we see that the most natural way to get an integer solution from the fractional solution obtained from the relaxed LP is to **round** x to integral solution. This leads to the following algorithm.

Algorithm 2: Set cover – LP Rounding

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

1. $x \leftarrow \mathrm{solve(LP)}$ `// Solve the relaxed (LP)`
2. $\mathcal{S}' \leftarrow \{S_i \colon x_i \geq 1/d\}$
3. **return** $\mathcal{S}'$

We now prove the correctness and Algorithm 2's approximation ratio.

[1]See MATH561 for a complete reference.

Lemma 2.3.1. $\mathcal{S}'$ is a covering.

Proof. Fix $e \in U$, let $S_1, \ldots, S_d$ be the sets containing e. We see that

$$\sum_{i=1}^{d} x_i \geq 1 \Rightarrow \exists j \in [d] \text{ s.t. } x_j \geq \frac{1}{d} \Rightarrow S_j \in \mathcal{S}'.$$

■

Theorem 2.3.1. Algorithm 2 is d-approximation algorithm.

Proof. By comparing $w(\mathcal{S}')$ and $\mathsf{OPT}_{\mathrm{LP}} = \sum_{i=1}^{m} x_i w_i$, we see that

$$\mathsf{OPT} \leq \sum_{S_i \in \mathcal{S}'} w_i \leq d \sum_{S_i \in \mathcal{S}'} w_i x_i \leq d \cdot \mathsf{OPT}_{\mathrm{LP}} \leq d \cdot \mathsf{OPT},$$

which implies $\mathsf{OPT}/d \leq \mathsf{OPT}_{\mathrm{LP}} \leq \mathsf{OPT}$.

> **Note.** Note that OPT is assumed to be $\mathsf{OPT}_{\mathrm{IP}}$, i.e., the optimum of the original IP formulation of Problem 2.1.1.

■

Definition 2.3.1 (Intgrality gap)**.** The *integrality gap* as

$$\sup_{\text{input } I} \frac{\mathsf{OPT}(I)}{\mathsf{OPT}_{\mathrm{LP}}(I)}.$$

Remark. We see that the integrality gap of Algorithm 2 is d from Theorem 2.3.1.

2.3.1 Randomized Linear Programming Rounding

And indeed, we can use a more natural way to do the rounding, i.e., respect to the x_i value.

Intuition. If x_i is close to 1, it's reasonable to include it, vice versa.

We see that algorithm first.

Algorithm 3: Set cover – Randomized LP Rounding

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

```
x ← solve(LP)                         // Solve the relaxed (LP)
S ← ∅
for i = 1, ..., m do
    add S_i to S' w.p. x_i            // independently
return S'
```

Now, the question is, how is this $\mathcal{S}'$'s quality? In other words, fix $e \in U$, what's $\Pr(e \text{ is covered})$?

Lemma 2.3.2. $\Pr(e \text{ is covered}) \geq 1 - 1/e \approx 0.63$.

Proof. We bound $\Pr(\overline{e \text{ is covered}})$ instead. Say $S_1, \ldots, S_d$ are the sets containing e, then we see that

$$\Pr(\overline{e \text{ is covered}}) = \prod_{i=1}^{d}(1 - x_i) \leq \prod_{i=1}^{d} e^{-x_i} = e^{-\overbrace{(x_1 + \ldots x_d)}^{\geq 1}} \leq e^{-1}.$$

Note. For every x, we have $1 + x \leq e^x$, and this approximation is close when $|x|$ is small. ■

A standard way to boost the correctness of a randomized algorithm is to run it multiple time, which leads to the following.

Algorithm 4: Set cover – Multi-time Randomized LP Rounding

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

```
x ← solve(LP)                              // Solve the relaxed (LP)
S ← ∅
for t = 1, ..., α do                       // independently
    for i = 1, ..., m do
        add S_i to S' w.p. x_i             // independently
return S'
```

Lemma 2.3.3. With $\alpha = 2\ln n$, $\mathcal{S}'$ returned from Algorithm 4 is a covering w.p. at least $1 - \frac{1}{n}$.

Proof. We have $\Pr(e \text{ is not covered}) \leq e^{-\alpha}$ from independence of each run. Let $\alpha = 2\ln n$, then $\Pr(e \text{ is not covered}) \leq e^{-\alpha} = 1/n^2$. By union bound,

$$\Pr(\text{some elements is not covered}) \leq \sum_{e \in U} \Pr(e \text{ not covered}) \leq n \cdot \frac{1}{n^2} = \frac{1}{n}.$$

This implies w.p. at least $1 - 1/n$, $\mathcal{S}'$ is a covering. ■

In other words, with $\alpha = 2\ln n$, Algorithm 4 is correct with probability at least $1 - 1/n$.

Lemma 2.3.4. With $\alpha = 2\ln n$, $\mathcal{S}'$ returned from Algorithm 4 has an approximation ratio $4\ln n$ w.p. at least $\frac{1}{2}$.[a]

[a]Note that $\mathcal{S}'$ is not necessary a covering.

Proof. Since $\mathbb{E}[w(\mathcal{S}')] \leq \alpha \sum_i w_i x_i = \alpha \,\mathsf{OPT}_{\mathrm{LP}}$, we have $\Pr(w(\mathcal{S}') \geq 2 \cdot \alpha \,\mathsf{OPT}_{\mathrm{LP}}) \leq 1/2$ from Markov inequality. We see that w.p. $\geq 1/2$, $w(\mathcal{S}') \leq 2 \cdot 2\ln n \cdot \mathsf{OPT}_{\mathrm{LP}} \leq 4\ln n \,\mathsf{OPT}$. ■

Theorem 2.3.2. By running Algorithm 4 many times, we get a $(4\ln n)$-approximation algorithm with high probability.[a]

[a]Note that we still need to choose $\mathcal{S}'$.

Proof. Together with Lemma 2.3.3 and Lemma 2.3.4 and using the union bound, the probability of $\mathcal{S}'$ not being a covering or with weight higher than $4\ln n \,\mathsf{OPT}$ is at most $\frac{1}{n} + \frac{1}{2}$, which is less than 1. Hence, by running Algorithm 4 many times (independently), the failing possibility is exponential small. ■

Note. With Theorem 2.3.2, we still need to find a valid covering with the lowest cost, where a valid covering with low enough weight is guaranteed to exist with high probability. Note that this is still a polynomial time algorithm since we know that checking $\mathcal{S}'$ is a covering is just linear.

Remark. Indeed, with some smarter algorithm modified from Algorithm 4, we can get a H_k approximation ratio.

Lecture 3: Covering-Packing Duality and Primal-Dual Method

2.4 Covering-Packing Duality

7 Sep. 10:30

We first define some useful notions.

Definition 2.4.1 (Strongly independent). Given a set system $(U, \mathcal{S})$, we say $C \subseteq U$ is *strongly independent* if there does not exist $S \in \mathcal{S}$ such that $|C \cap S| \geq 2$.

Remark. Then for any strongly independent set $C \subseteq U$, we know that $\mathsf{OPT}_{\mathrm{SC}} \geq |C|$.[a]

[a] SC denotes set cover.

Now, we're trying to find the **strongest witness** of strongly independent set, which suggests we define the following problem.

Problem 2.4.1 (Maximum strongly independent set). Given a set system $(U, \mathcal{S})$, we want to find the largest strongly independent set.

Remark. For any set system, we have $\mathsf{OPT}_{\mathrm{SIS}} \leq \mathsf{OPT}_{\mathrm{SC}}$.[a]

[a] SIS denotes maximum strongly independent set.

As previously seen (LP dual). Recall how we get the dual of a given LP:

$$
\begin{array}{rlcrl}
 & \min\ c^\top x & \qquad & & \max\ y^\top b \\
 & Ax \geq b & & & y^\top A \leq c^\top \\
(P) & x \geq 0 & & (D) & y \geq 0.
\end{array}
$$

Also, recall the weak duality ($\mathsf{OPT}_P \geq \mathsf{OPT}_D$) and strong duality ($\mathsf{OPT}_P = \mathsf{OPT}_D$).

Definition 2.4.2 (Covering LP). A primal LP with $A, b, c \geq 0$ is called a *covering LP*.

Definition 2.4.3 (Packing LP). A dual LP with $A, b, c \geq 0$ is called a *packing LP*.

We now give another LP formulation for the unweighted set cover. Given $\mathcal{S} = \{S_1, \ldots, S_m\}$, $U = \{e_1, \ldots, e_n\}$ and define $A \in \mathbb{R}^{n \times m}$ such that

$$
A_{ij} = \begin{cases} 1, & \text{if } e_i \in S_j; \\ 0, & \text{otherwise.} \end{cases}
$$

Then our LP is defined as

$$
\begin{array}{rlcrl}
 & \min\ \sum_{j=1}^{m} x_j & \qquad & & \max\ \sum_{i=1}^{n} y_i \\
 & Ax \geq \mathbf{1} & & & y^\top A \leq \mathbf{1} \\
(P) & x \geq 0 & & (D) & y \geq 0.
\end{array}
$$

We see that if we restrict $y_i \in \{0, 1\}$, we see that the dual (D) is just Problem 2.4.1. This can be seen via writing the constraint explicitly:

$$
\sum_{i=1}^{n} A_{ij} y_i \leq 1 \Leftrightarrow \sum_{i\colon e_i \in S_j} y_i \leq 1 \text{ for } j \in [m].
$$

And indeed, if we look at the weighted version, we have $\sum_{i\colon e_i \in s_j} y_i \leq w(S_j)$.

Now, recall the claim in Theorem 2.2.1, i.e., $y_{e_i} \leq \frac{w(S_j)}{k-i+1}$. We see that the y_{e_i} are just the dual variables in our setup. Additionally, with the observation that we can do this for any set $S = \{e_1, \ldots, e_k\} \in \mathcal{S}$, we have the following lemma.

Lemma 2.4.1. The variable $y' \coloneqq y/H_k$ is dual-feasible, i.e., it's feasible for (D).

Proof. We see that $y_{e_i} \geq 0$ (and hence y_i) trivially, so we only need to show that

$$\sum_{i=1}^{n} A_{ij} y' = \sum_{i=1}^{n} A_{ij} \frac{y_{e_i}}{H_k} \leq w(S_j)$$

for $j \in [m]$. But this is trivial by plugging in $y_{e_i} \leq \frac{w(S_j)}{k-i+1}$ as shown in Theorem 2.2.1, hence

$$\sum_{i=1}^{n} A_{ij} \frac{y_{e_i}}{H_k} \leq \frac{1}{H_k} \sum_{i=1}^{n} A_{ij} \frac{w(S_j)}{k-i+1} \leq \frac{1}{H_k} \sum_{i=1}^{k} \frac{w(S_j)}{k-i+1} = w(S_j),$$

and we're done.[a] ■

[a]Note that in the above derivation, i is kind of overloading, i.e., e_i corresponding to only some i (confusing, but it's how it is...).

With Lemma 2.4.1, we simply run Algorithm 1 while maintaining y_e for every e, and we're done.

Theorem 2.4.1. Algorithm 1 is an H_k-approximation algorithm in the view of its dual.

Proof. Same as Theorem 2.2.1, but now we have different interpretation. Specifically, if $y' = y/H_k$ is dual-feasible, we know that the corresponding objective value of y' is at most $\mathsf{OPT}_{\mathrm{LP}_D} = \mathsf{OPT}_{\mathrm{LP}_P}$, which is at most $\mathsf{OPT}_{\mathrm{SC}}$ further. Now, since we're dealing with LP, everything is linear includes the objective value, i.e., y is at most $H_k \cdot \mathsf{OPT}_{\mathrm{SC}}$. ■

Remark (Dual fitting)**.** The above method is called *dual fitting*, which is universal as one can easily see. The way to do this is the following.

1. Given an algorithm, distribute the algorithm to $\{y_i\}$.
2. Prove that y/α is dual-feasible.
3. This shows the algorithm is α-approximation algorithm.

2.5 Primal-Dual Method

We first see the general description of the so-called *primal-dual method*.

1. Maintain x (primal solution) and y (dual solution) where x is integral and infeasible, while y is fractional and feasible. Start from $x = y = 0$.
2. **Somehow** increase y until some dual constraints get tight.
3. **Choose** primal variables correspond to tight dual constraints, and update input accordingly.

Remark. We're using dual variables to get a certificate of the lower bound of the optimal problem we're solving.

In terms of set cover, we have the following.

Algorithm 5: Set cover – Primal-Dual

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

```
S' ← ∅, y ← 0
while U ≠ ∅ do
    Choose any e ∈ U
    Raise y_e until some constraints get tight
    S' ← S' ∪ {sets corresponding to tight dual constraints}
    Update U                         // Remove newly covered element in U
return S'
```

Remark. Algorithm 5 is correct and can be implemented efficiently.

Theorem 2.5.1. Algorithm 5 is a d-approximation algorithm.

Proof. Firstly, y is feasible. And we see that

$$w(\mathcal{S}') = \sum_{S \in \mathcal{S}'} w(S) = \sum_{S \in \mathcal{S}'} \sum_{e \in S} y_e \leq d \cdot \sum_{e \in U} y_e \leq d \cdot \mathsf{OPT}_{\mathrm{LP}_D} = d \cdot \mathsf{OPT}_{\mathrm{LP}_P} \leq d \cdot \mathsf{OPT}_{\mathrm{SC}}.$$

■

Lecture 4: Feedback Vertex Set

2.6 Feedback Vertex Set

12 Sep. 10:30

Following the discussion on primal-dual method, we see another covering problem.

2.6.1 Introduction

We consider the following problem.

Problem 2.6.1 (Feedback vertex set)**.** Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a weight function $c \colon \mathcal{V} \to \mathbb{R}^+$, we want to find $F \subseteq \mathcal{V}$ with $\min c(F)$ such that $\mathcal{G}[\mathcal{V} \setminus F]$ has no cycle.[a]

[a] This is equivalent as saying that $\mathcal{G}[\mathcal{V} \setminus F]$ is a forest.

Note (Feedback)**.** The name *feedback* comes from the fact that if there's a cycle in $\mathcal{G}$, then it kind of creates *feedback*.

Note (Edge version)**.** The *edge version* of Problem 2.6.1 can be solved by finding $T \subseteq \mathcal{E}$ be the maximum weight forest,[a] and let $F := \mathcal{E} \setminus T$.

[a] This can be found exactly in polynomial time.

Notation. In this lecture, when talking about cycle, we're referring to the vertices in which. But the meaning can vary from context to context.

Remark. This is a *special case* of Problem 2.1.1.

Proof. Let $\mathcal{C} := \{\text{set of all (simple) cycles}\}$ and consider Problem 2.1.1 on the set system $(\mathcal{C}, \mathcal{V})$, i.e., we want to find $F \subseteq \mathcal{V}$ such that $\forall C \in \mathcal{C}$, $|F \cap C| \geq 1$. ⊛

Note. The naive algorithm by directly applying methods discussed for Problem 2.1.1, we see that since $\min(\log k, d) = \Omega(n)$ for k being the maximum set size (which is $2^{\Omega(n)}$) and $d = n$, the approximation ratio we can get is $\Omega(n)$, which depends on the size of the input.

Now, the goal in this section is to show the following.

Theorem 2.6.1. There exists a 4-approximation algorithm for Problem 2.6.1.

Remark. Actually, there exists a 2-approximation algorithm.

We also have a hardness of Problem 2.6.1.

Theorem 2.6.2. A $(2-\epsilon)$-approximation algorithm doesn't exist for all $\epsilon > 0$ assuming some conjecture.

2.6.2 Cycle Covering LP

The most natural LP which models Problem 2.6.1 is the so-called *cycle covering LP*, which can be defined as

$$\begin{aligned} \min \ & \sum_{v \in \mathcal{V}} c(v) x_v \\ & \sum_{v \in C} x_v \geq 1 \quad \forall \text{ cycle } C \in \mathcal{C} \\ & x \geq 0, \end{aligned}$$

with the variables being $\{x_v\}_{v \in \mathcal{V}}$ such that $x_v = \mathbb{1}_{v \in F}$.

Remark. We see that this cycle covering LP has $2^{\Omega(n)}$ constraints. But we can actually solve this and get an $O(\log n)$-approximation ratio by smartly rounding the solution. And we can show that this approximation ratio is optimal in terms of this particular LP.

2.6.3 Density LP

A more sophisticated LP is the so-called *density LP*, defined as

$$\begin{aligned} \min \ & \sum_{v \in \mathcal{V}} c(v) x_v \\ & \sum_{v \in S} x_v (d_v^S - 1) \geq \lvert E(S) \rvert - \lvert S \rvert + 1 \quad \forall S \subseteq \mathcal{V} \\ & x \geq 0 \end{aligned}$$

with the variables being $\{x_v\}_{v \in \mathcal{V}}$.

Notation. The $E(S)$ denotes the edge set in the induced graph $\mathcal{G}[S] = (S, E(S))$, while d_v^S denotes the degree of v in $\mathcal{G}[S]$.

Intuition. The constraint is equivalent as saying that for every induced graph, $\#e \leq \#v - 1$, i.e., we require it to be a forest. Explicitly, $S \subseteq \mathcal{V}$,

$$\lvert E(S) \rvert - \sum_{v \in S} x_v d_v^S \leq \lvert S \rvert - \sum_{v \in S} x_v - 1.$$

Note that in the constraint, the right-hand side is just a lower-bound of $\#e$.

We see that the above LP is not exactly a covering LP since the coefficients can be negative if a set S is not irreducible.

Definition 2.6.1 (Irreducible). The set $S \subseteq \mathcal{V}$ is *irreducible* if for all $v \in S$, v belongs to some cycles in $G[S]$.

Now, it's clear that by looking at $\mathcal{S} = \{S \subseteq \mathcal{V} \mid S \text{ is irreducible}\}$, we have a covering LP defined as

$$\begin{aligned} \min \quad & \sum_{v\in\mathcal{V}} c(v) x_v \\ & \sum_{v\in S} x_v (d_v^S - 1) \geq |E(S)| - |S| + 1 =: b_S \quad \forall S \in \mathcal{S} \\ & x \geq 0. \end{aligned}$$

We first see why this LP models Problem 2.6.1.

Lemma 2.6.1. The integer version of density LP (denote as IP) is equivalent to Problem 2.6.1.

Proof. If x is feasible for Problem 2.6.1, then x is feasible for the IP. On the other hand, if x is feasible for IP, then for every cycle $C \in \mathcal{C}$, x deletes at least 1 vertex from C. ■

2.6.4 Primal-Dual Method

Now we're ready to solve this LP via primal-dual method. Denote the dual variables as $\{y_S\}_{S\in\mathcal{S}}$, then the dual is

$$\begin{aligned} \max \quad & \sum_{S\in\mathcal{S}} y_S b_S \\ & \sum_{S\ni v} (d_v^S - 1) y_S \leq c(v) \quad \forall v \in \mathcal{V} \\ & y \geq 0. \end{aligned}$$

Note. For the density LP and its dual, the constraint is still exponentially many, and no one knows how to solve this. But the power of primal-dual method is that we don't really solve this, rather, we just maintain two sets of solutions for both primal and dual. Moreover, we can maintain the primal solution in integral, while the dual solution in fractional.

We now have the following algorithm.

Algorithm 6: Feedback vertex set – Primal-Dual

Data: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Result: A minimal feedback vertex set F'

$S \leftarrow \mathcal{V}$, $c' = c$, $y \leftarrow 0$ // $c' \in \mathbb{R}^n$ keeps track of slackness of c

while $S \neq \varnothing$ **do**
  $S \leftarrow$ `reduce`(S) // Compute $\{v \in S : v$ belongs to some cycles in $\mathcal{G}[S]\}$
  $(\alpha, v) \leftarrow \min_{v\in S} c'(v)/(d_v^S - 1)$[a] // y_S gets tight by increasing unit weight
  $y_S \leftarrow \alpha$
  $c'(v) \leftarrow c'(v) - \alpha(d_v^S - 1)$
  $Z \leftarrow \{v \in S : c'(v) = 0\}$
  $F \leftarrow F \cup Z$, $S \leftarrow S \setminus Z$

// Compute a minimal feedback vertex set
$F' \leftarrow F = \{v_1, \dots, v_\ell\}$ // v_1 is deleted first, v_ℓ is deleted last
for $i = \ell, \dots, 1$ **do** // reversed greedy
  if $F' \setminus \{v_i\}$ *is a feedback vertex set for* $\mathcal{G}$ **then**
    $F' \leftarrow F' \setminus \{v_i\}$
return F'

[a] Note that we also get the argument v.

We see that in Algorithm 6, we first use primal-dual method to obtain a feasible feedback vertex set, and then run a reversed greedy algorithm to further ensure we get a good approximation ratio.

Claim. F is a feedback vertex set and y is dual-feasible.

Proof. It should be clear that why F is a feedback vertex set. As for the reason why y is dual-feasible, observe that we have one constraint for each v. After raising y_S for chosen v in line 6 and deduce $c'(v)$ in line 7, v will get removed so the constraint corresponding to v will be satisfied throughout. ⊛

Remark (Reversed greedy). The method we turn F into its minimal is called *reversed greedy*. This just checks that if we remove a vertex v from F' while F' is still feasible, then we just do it. Additionally, we iterate through v in the **reversed** order w.r.t. how v is being added into.

We want to compare the primal cost and the dual cost. The primal cost is

$$c(F) = \sum_{v\in F} c(v) = \sum_{v\in F}\sum_{S\ni v}(d_v^S - 1)y_S = \sum_{S\in\mathcal{S}} y_S \sum_{v\in F\cap S}(d_v^S - 1),$$

while the dual cost is $\sum_{S\in\mathcal{S}} y_S b_S$.

Remark. This is where the primal-dual method is powerful. i.e., by switching the order of summation, if we have some ratio of $\sum_{v\in F\cap S}(d_v^S - 1)$ and b_S for every S, we're done. On caveat is that since S is changing when running Algorithm 6, so the final solution F may not be good for this particular S. We need to guarantee some ratio for this F **for all** S.[a]

[a]At least for S with positive y_S.

Lemma 2.6.2. For all $S\in\mathcal{S}$, if F is *minimal* in S,[a] then we have

$$\sum_{v\in S\cap F}(d_v^S - 1) \leq 4\cdot b_S = 4(|E(S)| - |S| + 1).$$

[a]i.e., in $\mathcal{G}[S]$, no $F' \subsetneq F\cap S$ in feedback vertex set.

Proof. Let's first see a simple case.

Intuition. If the graph is 3-regular, then we see that the left-hand side is $\leq 2\cdot|S|$ by summing over the whole S instead of $S\cap F$, while the right-hand side is $2\cdot|S|+4$ since $|E(S)| = 1.5\,|S|$.

This shows that in a 3-regular graph, deleting every vertex in S is actually 4-approximated. And this intuition generalized to general graph with degree greater than 3.

Since we assume S to be irreducible, so we're not interested in degree 0 or 1 vertices (there are no such vertices in an irreducible S). So the only problematic guy is degree-2 vertex. And the only place a degree-2 vertex can live is in a long path.

Figure 2.2: If there are two $v\in F$, by minimality of F, one of v will be strictly unnecessary to break this path in a cycle.

Note. Observe that we only need to delete at most one vertex in any path, and sometimes this may be loose since we can delete one branch node joining two paths, i.e., deleting 1 nodes for two paths.

Let A be the set of degree 2 vertices, and B be the set of vertices with degree larger than 3. Now, consider line segment in the graph. If ℓ is a line segment,

(a) $|F \cap \ell| \leq 1$, i.e., we delete at most one point in ℓ.

(b) If F contains one of the endpoints of ℓ, then $|F \cap \ell| = 0$.

Since F is minimal, the left-hand side is

$$|A \cap F| + \sum_{v \in B \cap F} (d_v^S - 1) \leq \sum_{v \in B \setminus F} d_v^S/2 + \sum_{v \in B \cap F} (d_v^S - 1) \leq \sum_{v \in B} (d_v^S - 1),$$

where the first inequality comes from the fact that if we delete vertices in A, i.e., in the line segment, then we know we don't delete its end points, and by *distributed* that 1 cost into its two end points, each $1/2$.

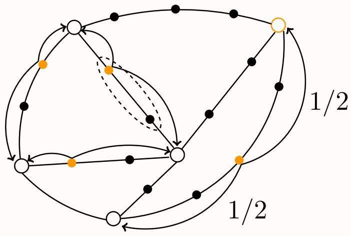


Figure 2.3: Distribute the cost of F.

Similarly, in the right-hand side, the crucial term is

$$|E(S)| - |S| = \sum_{v \in S} (d_v^S/2 - 1) = \sum_{v \in B} (d_v^S/2 - 1)$$

where the last equality holds since for $v \in A$, the summand is just $2/2 - 1 = 0$. It's clear that since $\forall v \in B$, $d_v^S - 1 \leq 4(d_v^S/2 - 1)$, rearranging this inequality gives the result. ■

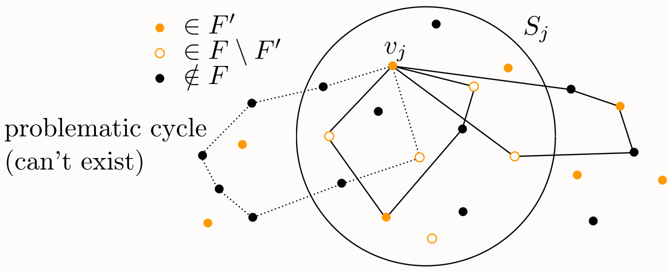
To show Theorem 2.6.1, it's enough to have a minimal F, then the result follows form Lemma 2.6.1. Hence, after obtaining F, Algorithm 6 further convert F into F' and try to obtain a minimal version of F. Clearly, F' is still a feedback vertex set, and the minimality of F' is guaranteed by the following lemma.

Lemma 2.6.3. F' is minimal in every S_i, where S_i is the corresponding S in Algorithm 6 when v_i is deleted.

Proof. Suppose this is not the case. Then there exists $v_j \in F'$ such that in $\mathcal{G}[S_i]$, $(F \cap S_i) \setminus \{v_j\}$ is still a feedback vertex set in $\mathcal{G}[S_j]$. Notice that we only need to consider the case that $i = j$ since $v_j \in S_i$ means $i \geq j$ from how we order them. In this case, $S_j \subseteq S_i$, hence to check the minimality of F' it's enough to just consider the case that $i = j$. Hence, we consider $(F \cap S_j) \setminus \{v_j\}$ instead.

> **Note.** Here we only consider $G[S_j]$, i.e., we want to say that if v_j is not minimal in $G[S_j]$, then v_j should really be deleted even w.r.t. the whole graph.

Now, observe the following picture in step j of line 13 with cycles contained v_j:



Observe that the middle cycles in $G[S_j]$ must exist from our assumption of $(F \cap S_j) \setminus \{v_j\}$ being still a feedback vertex set, i.e., if a cycle exists in $G[S_j]$, then it must contain another nodes other than v_j that's also in F'. But we see that when we consider cycles outside $G[S_j]$, we have the following.

Claim. No vertices outside S_j which is also in $F \setminus F'$ at step j of line 13

Proof. Since S_j is growing, i.e., for $i \leq j$, $S_i \leq S_j$, and we just can't delete something we haven't considered. ⊛

Claim. There are no cycles $C \ni v_j$ such that $C \setminus S_j$ is disjoint from F.

Proof. Observe that there are only two ways for a vertex being deleted from the graph, either $v \in Z$, i.e., its dual constraint is tight, or $v \in S$ is deleted since it prevent S being irreducible. Only the latter case will make $v \notin F$, we see that there's no way such a cycle C exists with all vertices outside S_j are preventing s being irreducible, since this cycle C itself is a cycle... ⊛

This implies $F' \setminus \{v_j\}$ is still a feedback vertex set in $\mathcal{G}$ when $i = j$ in Algorithm 6 since such a problematic cycle can't exist,[a] which contradicts with the minimality of F'. ■

[a]Explicitly, if this exists, then delete v_j will make F' fail to intersect such a cycle.

Finally, we see that we can prove Theorem 2.6.1.

Proof of Theorem 2.6.1. Firstly, Algorithm 6 gives a 4-approximation of the density IP guaranteed by Lemma 2.6.2 and Lemma 2.6.3. Finally, from Lemma 2.6.1, we see that Problem 2.6.1 and the density IP is equivalent, proving the theorem. ■

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.