EECS598-001
Approximation Algorithms & Hardness of Approximation

Pingbang Hu

October 3, 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 $f(x) = \text{length}(x)$ ($\#$ 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 a combinatorial optimization problem since given a graph $\mathcal{G}$ with $n = \lvert\mathcal{V}\rvert$, $m = \lvert\mathcal{E}\rvert$, there are at most $n!$ different paths, i.e., $\lvert X\rvert \leq n! < \infty$.

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 $\mathrm{poly}(s)$ 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 $\mathrm{poly}((n, m))$-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: $f(A(I)) \leq \alpha \cdot \mathsf{OPT}(I)$ 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. For a set system $(\Omega, \mathcal{S})$, we often let $m := |\mathcal{S}|$ and $n := |\Omega|$.

Definition 2.1.2 (Degree). Given a set system $(\Omega, \mathcal{S})$, the *degree of* $x \in \Omega$, $\deg(x)$, is defined as

$$\deg(x) := \left|\{S \in \mathcal{S} \mid x \in S\}\right|.$$

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

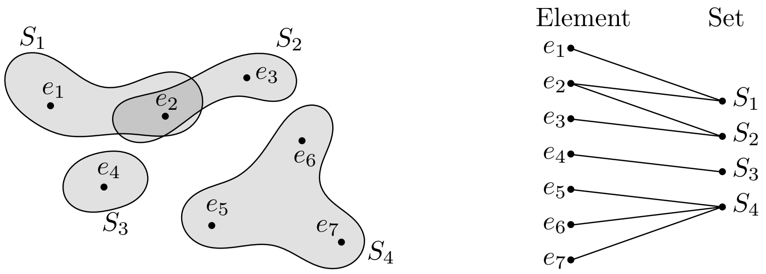


Figure 2.1: Bipartite representation of a set system.

Denote $d := \max_{e \in U} \deg(e) \leq m$ and $k := \max_{i \in [m]} |S_i| \leq n$, 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 $\mathcal{S} := \{S_i \subseteq U\}_{i=1}^m$ 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 2.1: 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   **for** $e \in U \cap S_i$ **do**
5         $y_e \leftarrow w(S_i) / |U \cap S_i|$ // Average costs
6     $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{S_i\}$
7     $U \leftarrow U \setminus S_i$
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 2.1 is a polynomial time algorithm, also, the output $\mathcal{S}'$ is always a valid covering.

Theorem 2.2.1. Algorithm 2.1 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. $\mathcal{S}^*$. 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 7), 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 7) 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 2.1 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 2.1 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

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

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

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

Write it in a more compact form, we have

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

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 $x \in \mathrm{fes}(\mathrm{IP}) \Rightarrow x \in \mathrm{fea}(\mathrm{LP})$, i.e., $\mathrm{fes}(\mathrm{LP}) \supseteq \mathrm{fes}(\mathrm{IP})$. 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 2.2: Set cover – LP Rounding

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

1. $x \leftarrow \mathrm{solve}(\mathrm{LP})$ `// 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 2.2'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 2.2 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). Given an integer programming, the *integrality gap* between OPT and $\mathsf{OPT}_{\mathrm{LP}}$ of its LP relaxation is defined as

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

Remark. We see that the integrality gap of Algorithm 2.2 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 2.3: Set cover – Randomized LP Rounding

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

$x \leftarrow \text{solve(LP)}$ // Solve the relaxed (LP)
$\mathcal{S} \leftarrow \varnothing$
for $i = 1, \ldots, m$ **do**
  add S_i to $\mathcal{S}'$ w.p. x_i // independently
return $\mathcal{S}'$

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

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

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 2.4: 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 2.4 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 2.4 is correct with probability at least $1 - 1/n$.

Lemma 2.3.4. With $\alpha = 2\ln n$, $\mathcal{S}'$ returned from Algorithm 2.4 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 2.4 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 2.4 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 2.4, we can get an 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 $\mathsf{OPT}_{\text{SC}} \geq |C|$.[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 $\mathsf{OPT}_{\text{SIS}} \leq \mathsf{OPT}_{\text{SC}}$.[a]

[a] SIS denotes maximum strongly independent set.

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

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

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

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

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 2.1 while maintaining y_e for every e, and we're done.

Theorem 2.4.1. Algorithm 2.1 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 2.5: Set cover – Primal-Dual

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

$\mathcal{S}' \leftarrow \varnothing$, $y \leftarrow 0$
while $U \neq \varnothing$ **do**
    Choose any $e \in U$
    Raise y_e until some constraints get tight
    $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{$sets corresponding to tight dual constraints$\}$
    Update U `// Remove newly covered element in` U
return $\mathcal{S}'$

Remark. Algorithm 2.5 is correct and can be implemented efficiently.

Theorem 2.5.1. Algorithm 2.5 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. Achieving $(2 - \epsilon)$-approximation algorithm if NP-hard for all $\epsilon > 0$ assuming the unique games conjecture.

Proof. See Homework 1. ■

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

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

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.[a] And we can show that this approximation ratio is optimal in terms of this particular LP.

[a] See homework 1.

2.6.3 Density LP

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

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

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, $S \subseteq \mathcal{V}$,

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

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

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

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

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

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 2.6: Feedback vertex set – Primal-Dual

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

1 $S \leftarrow \mathcal{V}$, $c' = c$, $y \leftarrow 0$ // $c' \in \mathbb{R}^n$ keeps track of slackness of c
2
3 **while** $S \neq \varnothing$ **do**
4 $\quad S \leftarrow \texttt{reduce}(S)$ // Compute $\{v \in S \colon v$ belongs to some cycles in $\mathcal{G}[S]\}$
5 $\quad (\alpha, v) \leftarrow \min_{v \in S} c'(v)/(d_v^S - 1)$[a] // y_S gets tight by increasing unit weight
6 $\quad y_S \leftarrow \alpha$
7 $\quad c'(v) \leftarrow c'(v) - \alpha(d_v^S - 1)$
8 $\quad Z \leftarrow \{v \in S \colon c'(v) = 0\}$
9 $\quad F \leftarrow F \cup Z$, $S \leftarrow S \setminus Z$
10
11 // Compute a minimal feedback vertex set
12 $F' \leftarrow F = \{v_1, \dots, v_\ell\}$ // v_1 is deleted first, v_ℓ is deleted last
13 **for** $i = \ell, \dots, 1$ **do** // reversed greedy
14 $\quad$ **if** $F' \setminus \{v_i\}$ *is a feedback vertex set for* $\mathcal{G}$ **then**
15 $\qquad F' \leftarrow F' \setminus \{v_i\}$
16 **return** F'

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

We see that in Algorithm 2.6, 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 2.6, 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.

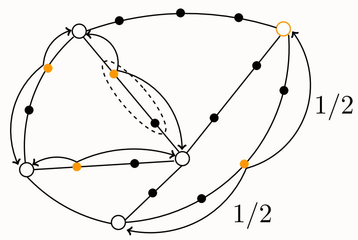


Figure 2.3: Distribute the cost of F.

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. ■

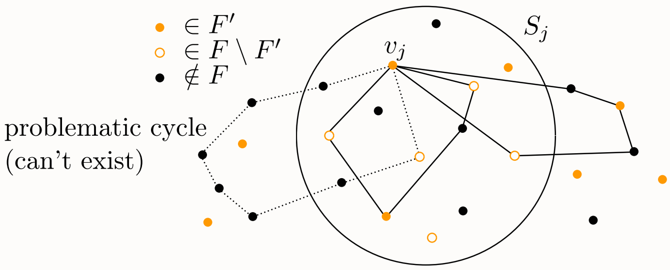
To show Theorem 2.6.1, it's enough to have a minimal F, then the result follows form Lemma 2.6.1. Hence, after obtaining F, Algorithm 2.6 further convert F into F' and try to obtain a minimal version of F. Clearly, F' is still a feedback vertex set, and the minimality of F' is guaranteed by the following lemma.

Lemma 2.6.3. F' is minimal in every S_i, where S_i is the corresponding S in Algorithm 2.6 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 2.6 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 2.6 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. ■

Chapter 3

Clustering

Lecture 5: Facility Location

14 Sep. 10:30

3.1 Introduction

The problem we're interested in is called the clustering problem.

Problem 3.1.1 (Clustering). Given n objects, partition them into k groups such that

- *Similar* objects are in same group
- *Different* objects are in different group.

Note. We see that Problem 3.1.1 is vague in terms of the definition, which is because this is more like a class of problems. We'll see different notions of *similar* and *different* later when we consider more explicit problems.

In particular, the notion of metric is useful.

Definition 3.1.1 (Metric). Given a set X, a function $d\colon X \times X \to \mathbb{R}^{+} \cup \{0\}$ is called a *metric* if

(a) $d(\cdot, \cdot) \geq 0$ and $d(i, j) = 0$ if and only if $i = j$.[a]

(b) $d(i, j) = d(j, i)$ for all $i, j \in X$.

(c) $d(i, j) + d(j, k) \geq d(i, k)$ for all $i, j, k \in X$.

[a] We didn't mention this in lectures, but in math community this should also be included.

Remark (Metric space). Though we didn't formally introduce, but the pair (X, d) of X and a metric d on X is sometimes called a *metric space.*

3.2 Facility Location

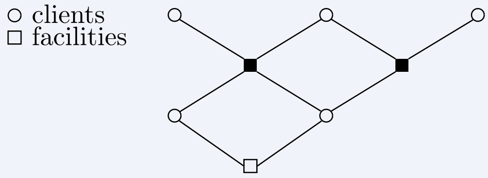
Let's first look at the problem.

Problem 3.2.1 (Facility location). Given a metric space (X, d) and $P, Q \subseteq X$, $f \in \mathbb{R}^{+}$ where P is the set of clients, Q is the set of (possible) facilities, we want to open $Q' \subseteq Q$ such that it minimizes

$$\sum_{i\in P} \min_{j\in Q'} d(i, j) + f\left|Q'\right|,$$

where we interpret the first summation as connection cost, the second term as opening cost.

Example. Consider the following example.



If $f = 1$ and we open the black facilities, then the cost is $2 + 5 = 7$ assuming unit length.

We now write down the LP of Problem 3.2.1. Denote variables $\{y_j\}_{j\in Q}$ and $\{x_{ij}\}_{i\in P, j\in Q}$. Then the LP can be written as

$$
\begin{aligned}
& \min \sum_{ij} d(i,j)x_{ij} + \sum_j y_j \cdot f & & \\
& \sum_j x_{ij} \geq 1 & \forall i \in P \qquad & (\alpha_i) \\
& x_{ij} \leq y_j \Leftrightarrow y_j - x_{ij} \geq 0 & \forall i,j \qquad & (\beta_{ij}) \\
(P) \quad & x, y \geq 0. & &
\end{aligned}
$$

Denote the dual variables as α_i and β_{ij}, the dual is

$$
\begin{aligned}
& \max \sum_i \alpha_i & & \\
& \alpha_i - \beta_{ij} \leq d(i,j) & \forall i,j \qquad & (x_{ij}) \\
& \sum_i \beta_{ij} \leq f & \forall j \qquad & (y_i) \\
(D) \quad & \alpha, \beta \geq 0. & &
\end{aligned}
$$

Remark. If (α, β) is feasible, redefine $\beta_{ij} := \max(0, \alpha_i - d(i,j))$, it's still feasible and will not affect the objective value. We see that we can drop β and only look at α.

We can then define the following useful notion called cluster.

Definition 3.2.1 (Cluster). A *cluster* $C := (j, P')$ is the order pair for $j \in Q$ and $P' \subseteq P$, where the cost $c(C)$ is calculated by directing all $i \in P'$ to j, i.e., $c(C) = f + \sum_{i\in P'} d(i,j)$.

Notation. We denote the set of all clusters C by $\mathcal{C}$.

Remark (Just set cover!). We see that Problem 3.2.1 is equivalent to set cover on $(P, \mathcal{C})$.

Proof. If we write down the LP for set cover on $(P, \mathcal{C})$, we have

$$
\begin{aligned}
& \min \sum_{C\in\mathcal{C}} c(C) \cdot y_C & & \max \sum_{i\in P} \alpha_i \\
& \sum_{C\ni i} y_C \geq 1 \quad \forall i \in P & & \sum_{i\in C} \alpha_i \leq c(C) \quad \forall C \in \mathcal{C} \\
(P) \quad & y \geq 0 & (D) \quad & \alpha \geq 0,
\end{aligned}
$$

which is equivalent to what we have as above. ⊛

But observe that the number of clusters is $|Q| \cdot 2^{|P|}$, hence directly solve either (P) or (D) is not feasible. In this case, we can use the primal-dual method.

3.2.1 Primal-Dual Method

Let's first see the primal-dual algorithm on (P) and (D) derived above.

Algorithm 3.1: Facility location – Primal-Dual

Data: A set of clients $P \subseteq X$, a set of (possible) facilities $Q \subseteq X$, facility cost f
Result: A set of opened facilities $Q' \subseteq Q$

1 $S \leftarrow \varnothing$, $Q' \leftarrow \varnothing$, $\alpha \leftarrow 0$ `//` S`:connected clients,` O`:open facilities`
2
3 **while** $S \neq P$ **do**
4   **while** *True* **do**
5   increase all $\{\alpha_i\}_{i\in P\setminus S}$ by a *unit*
6   **if** *some* $j \in Q \setminus Q'$ *s.t.* $\sum_{i\in P} \beta_{ij} = f$ **then** `//` j `gets tight (open)`
7   **break**
8   **else if** *some* $i \in P \setminus S$ *s.t.* $\alpha_i \geq d(i,j)$ **then** `//` i `can connect to` $j \in Q'$
9   **break**
10     $Q' \leftarrow$ {tight facilities} `// Update` Q'
11     $S \leftarrow$ {clients connected to Q'} `// Update` S
12
13 `// Trim down` Q'
14 $G = (Q', E := \{(j,j') \colon \exists i \in P \text{ such that } \alpha_i > d(i,j), \alpha_i > d(i,j'), j, j' \in Q'\})$
15 Compute Q'' s.t. $\forall j \in Q'$, either $j \in Q''$ or $\exists j' \in Q''$ s.t. $(j,j') \in E$ `// max independent set`
16 **return** Q''

Note. line 6 and line 8 can happen in the same time.

Intuition. We're basically increasing the cost i willing to pay and stop (in the second while loop) when i finally connect to j. Or one can also interpret α_i as the time i connects to some facilities j.

This directly relates to the fact that for all i, j, if i, j are connected, then $d(i,j) \leq \alpha_i$, which is exactly the spirit of the primal-dual method since we want to argue the upper-bound in terms of α. But before that, we need to argue that α is actually feasible in order to make this bound valid.

Lemma 3.2.1. α is dual-feasible in Algorithm 3.1.

Proof. Firstly, α start from 0 which is feasible. Now, for α_i violates the constraints $\sum_{i\in C} \alpha \leq c(C) = f + \sum_{i \in P'} d(i,j)$, there are two possibilities, but both are handled in Algorithm 3.1. Specifically, line 6 and line 8:

- In line 6: This corresponds to some j gets opened, we then need to make sure that no α_i will pay toward j for its open cost f. But this is clear since whoever i is paying non-zero amounts to j for its f, i immediately connect to j and will be clicked out from $P \setminus S$, meaning that their dual α_i will not be increased anymore.
- In line 8: This corresponds to when i want to connect (willing to pay non-zero amount to) an already opened j. But we see that whenever i willing to pay for an already opened j, we immediately connect them and so j gets nothing (hence will not be violated) while i just pays for the distance to go to j.

In all, throughout Algorithm 3.1, α is feasible. ■

Note (Trim down). Just like Algorithm 2.6, after getting the initial solution Q', we'll soon see in the analysis section that it's kind of wasteful, so we trim it down to obtain a better solution.

3.2.2 Analysis

We first do a naive analysis, i.e., try to bound the connected cost and opening cost for Q' obtained in Algorithm 3.1 before line 12, which turns out to be not working. The problem is not on connected cost, since as noted above, $d(i,j) \leq \alpha_i$ so the connection cost is at most $\sum_i \alpha_i$.

Remark. Bound the opening cost naively can't guarantee a constant approximation factor.

Proof. To bound opening cost, we see that

$$\text{opening cost } = f\left|Q'\right| = \sum_{j\in Q'} f = \sum_j \sum_i \beta_{ij} = \sum_i \sum_{j\in Q'} \beta_{ij}.$$

Observe that since $\beta_{ij} = \max(0, \alpha_i - d(i,j)) \leq \alpha_i$, hence if we can guarantee for each i, it only pays for one j, then we will get a 2-approximation. But this might not be the case since we don't have control of how many j that i is paying. ⊛

Let's first introduce some notions in order to analyze Algorithm 3.1.

Notation (Connecting witness). The first open facility connected to i is called the *connecting witness* $w(i) \in Q$ for every $i \in P$.

Notation (Contributing). We say (i,j) is *contributing* if $\alpha_i > d(i,j)$, i.e., $\beta_{ij} > 0$.[a]

[a] We now have a strict inequality, i.e., i is now paying some non-trivial amount to j.

Note that the problem in the naive solution happens when a client i pays multiple facilities j. And a simple idea is to close some facilities j such that every client pay at most 1 facility.

Intuition. If i is contributes to two facilities j and j', we close down one of them basically since this is where the problem comes from. This is exactly how we trim down Q': by considering $G = (Q', E)$ such that $(j,j') \in E$ if and only if $\exists i \in P$ that contributes to both j and j', taking maximal independent set of G exactly makes i paying to only one j.

Note. In this case, we take care of opening cost, but the connected cost might be worse, so we basically turn to bound another quantity while still keep one term simple to bound.

Notation (Directed connected). We say $i \in P$ is *directed connected* if $j \in Q''$ such that (i,j) is connected ($\alpha_i \geq d(i,j)$). For these i, divide α_i into $\alpha_i^f := \beta_{ij}$ and $\alpha_i^c := d(i,j)$, i.e., $\alpha_i = \alpha_i^f + \alpha_i^c$.

Notation (Indirected connected). We say i is *indirectly connected* if i is not directed connected,[a] and like in directed connected, $\alpha_i =: \alpha_i^f + \alpha_i^c$ where $\alpha_i^f = 0$, $\alpha_i^c = \alpha_i$.

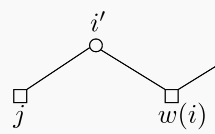


Figure 3.1: When i is indirected connected.

[a] i.e., there exists j such that $(j, w(i)) \in E$, hence there exists i' such that (i', j) and $(i', w(i))$ contributing.

Now, we can bound the opening cost $f\left|Q''\right|$ for Q'' more carefully. It's now

$$f\left|Q''\right| = \sum_{j\in Q''} \sum_i \beta_{ij} = \sum_{j\in Q''} \sum_{\text{d.c. } i} \beta_{ij} = \sum_{\text{d.c. } i} \left[\sum_{j\in Q''} \beta_{ij}\right] = \sum_{\text{d.c. } i} \alpha_i^f.$$

As for connected cost, we see that if i is directed connected, $d(i,j) \leq \alpha_i^c$, while if i is indirected connected, it's not so clear. However, we have the following.

Claim. If i is indirected connected, then $d(i,j) \leq 3\alpha_i$.

Proof. Note that $(j, w(i)) \in E$ and $d(i,j) \leq \alpha_i + 2\alpha_{i'}$ by looking at Figure 3.1, hence it's sufficient to prove $\alpha_{i'} \leq \alpha_i$. To do this, for some facility ℓ, define t_ℓ to be the time ℓ open in line 6, and α_i be the time i connected in line 8. We see that

- If (i, ℓ) are contributing, then $\alpha_i \leq t_\ell$.
- If $\ell = w(i)$, then $t_\ell \leq \alpha_i$.

Combining these together, we have $\alpha_{i'} \leq t_{w(i)} \leq \alpha_i$. ⊛

Finally, we have the following.

Theorem 3.2.1. Algorithm 3.1 is a 3-approximation algorithm.

Proof. The cost of Q'' produce by Algorithm 3.1 is just the connected cost of plus the opening cost of Q'', which can be bounded as

$$\text{final cost} = \text{connected cost} + \text{opening cost} \leq \sum_i 3\alpha_i^c + \sum_i \alpha_i^f \leq 3\sum_i \alpha_i \leq 3\,\mathsf{OPT},$$

which shows that it is a 3-approximation algorithm. ■

Note. Notice that in the above proof, since we know that the opening cost is exactly $\sum_i \alpha_i^f$, and hence even if we pay 3 times of the opening cost, we still get a 3-approximation algorithm.

Remark. Algorithm 3.1 is a very basic algorithm which can be used even as a black-box for other clustering problems. We'll revisit this later and consider other metrics and see what can we improve.

Lecture 6: Facility Location with LMP Approximation

3.2.3 Hardness

19 Sep. 10:30

For Problem 3.2.1, we have the following.

(a) 1.488-approximation [Li13]

(b) 1.463-approximation is NP-hard [GK99]

Turns out that specifically for Problem 3.2.1, we can have a more refine notion of approximation ratio defined below.

Definition 3.2.2 (LMP approximation)**.** An algorithm ALG which solves facility location is called *γ-Lagrangian multiplier preserving approximation* (LMP-approximation) if

$$\frac{\mathrm{conn}(\mathrm{ALG})}{\gamma} + \mathrm{open}(\mathrm{ALG}) \leq \sum_i \alpha_i$$

for some $\gamma > 0$.[a]

[a] The opening cost is just $k'f$ if ALG opens k' centers.

Remark. The notion of LMP approximation is due to Lagrangian multiplier in the field of optimization, where the dual variables are treated as a Lagrangian multipliers. And Definition 3.2.2 says that we're not approximating $k'f$ at all, hence it's *preserving*.

And indeed, we now have a more refined characterization about Algorithm 3.1.

Corollary 3.2.1. Algorithm 3.1 is a 3-LMP approximation algorithm.

Remark (SOTA). If we look at the SOTA result in terms of LMP, we have the following.

(a) 3-LMP approximation [JV01]

(b) 2-LMP approximation [JMS02]

(c) $1.99\ldots9$-LMP approximation [Coh+22]

(d) 1.73-LMP approximation[a] is NP-hard [JMS02]

[a]The number comes from $1 + 2/e$.

3.2.4 Greedy Method

Let's take another look at Problem 3.2.1 and see it as an instance of Problem 2.1.1 where the universe is all the clients P, while the collection of sets are pairs of facility and its connected clients, i.e., clusters. Then, it's natural to consider using a similar algorithm as Algorithm 2.1 to solve this.

Algorithm 3.2: Facility location – Greedy

Data: A set of clients $P \subseteq X$, a set of (possible) facilities $Q \subseteq X$, facility cost f[a]
Result: A set of opened facilities $Q' \subseteq Q$

```
S ← ∅, Q' ← ∅
while S ≠ P do
    choose (j, T) ∈ Q × 𝒫(P \ X) with minimum c((j, T))/|T|
    Q' ← Q' ∪ {j}
    S ← S ∪ T
return Q'
```

[a]We didn't use it explicitly in the algorithm since we hide it in the cost function $c(\cdot)$.

This is just Algorithm 2.1, hence we have H_n-approximation. But as we have seen in Theorem 3.2.1, we have achieved a constant approximation ratio for Problem 3.2.1. Hence, we should be able to do better based on Algorithm 3.2.

Remark. If we modify Algorithm 3.2 such that for all (j, T), if j is open, then we define the cost of this cluster as

$$c((j, T)) := \frac{\sum_{i \in T} d(i, j)}{|T|}.$$

We'll achieve 1.861-approximation, but the analysis is complex.

Instead, we're going to see other variations based on Algorithm 3.2.

First Modification

We see observe that $c((j, T))/|T|$ is increasing in Algorithm 3.2. Also, if $\alpha := c((j, T))/|T|$, then for all $i \in T$, $d(i, j) \leq \alpha$ where we interpret this as i *pays* α_i to cover the connection cost $d(i, j)$ and the opening cost $\alpha_i - d(i, j)$ of j. Following this intuition, if we change line 6 in Algorithm 3.1 (with only first phase) such that the summation is over $P \setminus S$, it becomes exactly Algorithm 3.2. proof!

Algorithm 3.3: Facility location – Greedy Modification I

Data: A set of clients $P \subseteq X$, a set of (possible) facilities $Q \subseteq X$, facility cost f
Result: A set of opened facilities $Q' \subseteq Q$

1 $S \leftarrow \varnothing$, $Q' \leftarrow \varnothing$, $\alpha \leftarrow 0$ // S:connected clients, Q':open facilities
2
3 **while** $S \neq P$ **do**
4   **while** *True* **do**
5   increase all $\{\alpha_i\}_{i \in P \setminus S}$ by a *unit*
6   **if** *some* $j \in Q \setminus Q'$ *s.t.* $\sum_{i \in P \setminus S} \beta_{ij} = f$ **then** // j gets tight (open)
7   **break**
8   **else if** *some* $i \in P \setminus S$ *s.t.* $\alpha_i \geq d(i, j)$ **then** // i can connect to $j \in Q'$
9   **break**
10     $Q' \leftarrow Q' \cup \{j\}$ // Update Q'
11     $S \leftarrow S \cup \{i \in P \setminus S : \alpha_i \geq d(i, j)\}$ // Update S
12 **return** Q'

Remark. Since line 6 and line 8 can happen simultaneously, while what we just said assumes the opposite, so we need to further modify Algorithm 3.1 in line 10 and line 11.

Second Modification

Another potential modification gives us a 1.61-approximation. We essentially allow $i \in S$ to *switch* in Algorithm 3.3, i.e., after i connects to j, if j' is closer to i later, i can *offer* with $d(i, j) - d(i, j')$ to other facilities.

Algorithm 3.4: Facility location – Greedy Modification II

Data: A set of clients $P \subseteq X$, a set of (possible) facilities $Q \subseteq X$, facility cost f
Result: A set of opened facilities $Q' \subseteq Q$

1 $S \leftarrow \varnothing$, $Q' \leftarrow \varnothing$, $\alpha \leftarrow 0$ // S:connected clients, Q':open facilities
2
3 **while** $S \neq P$ **do**
4   **while** *True* **do**
5   increase all $\{\alpha_i\}_{i \in P \setminus S}$ by a *unit*
6   **if** *some* $j \in Q \setminus Q'$ *s.t.* $\sum_{i \in S} (d(i, w(i)) - d(i, j))^+ + \sum_{i \in P \setminus S} \beta_{ij} = f$[a] **then**
7   **break**
8   **if** *some* $i \in P \setminus S$ *s.t.* $\alpha_i \geq d(i, j)$ **then** // i can connect to $j \in Q'$
9   **break**
10     $Q' \leftarrow Q' \cup \{j\}$ // Update Q'
11     $S \leftarrow S \cup \{i \in P \setminus S : \alpha_i \geq d(i, j)\}$ // Update S
12 **return** Q'

[a] We define $a^+ := \max(a, 0)$ and also, $w(i)$ is now the *current* facility i is connected to.

Third Modification

If we run Algorithm 3.4 with facility cost being $\hat{f} := 2f$, we can have a 2-LMP approximation algorithm as follows.

Algorithm 3.5: Facility location – Greedy Modification III

Data: A set of clients $P \subseteq X$, a set of (possible) facilities $Q \subseteq X$, facility cost f
Result: A set of opened facilities $Q' \subseteq Q$

$S \leftarrow \varnothing$, $Q' \leftarrow \varnothing$, $\alpha \leftarrow 0$ // S:connected clients, Q':open facilities

while $S \neq P$ **do**
  while *True* **do**
    increase all $\{\alpha_i\}_{i\in P\setminus S}$ by a *unit*
    if *some* $j \in Q \setminus Q'$ *s.t.* $\sum_{i\in S}(d(i,w(i)) - d(i,j))^+ + \sum_{i\in P\setminus S}\beta_{ij} = \hat{f}$[a] **then**
      break
    if *some* $i \in P \setminus S$ *s.t.* $\alpha_i \geq d(i,j)$ **then** // i can connect to $j \in Q'$
      break
  $Q' \leftarrow Q' \cup \{j\}$ // Update Q'
  $S \leftarrow S \cup \{i \in P \setminus S \colon \alpha_i \geq d(i,j)\}$ // Update S
return Q'

[a] We define $a^+ \coloneqq \max(a,0)$ and also, $w(i)$ is now the *current* facility i is connected to.

It's clear that in Algorithm 3.5, the connection cost plus 2 times the opening cost is $\sum_{i\in P}\alpha_i$ from how we design the algorithm by changing the facility cost from f to $\hat{f} \coloneqq 2f$. Now, a crucial lemma is the following.

Lemma 3.2.2. (α', β') is dual feasible, where $\alpha' \coloneqq \alpha/2$, $\beta'_{ij} \coloneqq (\alpha'_i - d(i,j))^+$.

Proof. It's sufficient to consider $j \in Q$ and prove that $\sum_{i\in P'}\beta'_{ij} \leq f$ where $P' \coloneqq \left\{i \colon \beta'_{ij} > 0\right\} = [n]$ where we're overloading n here. Let's order α_i such that $\alpha_1 \leq \ldots \leq \alpha_n$ where α_i is the time i when i is *first connected*.

Claim. For all $i, k \in P'$ such that $\alpha_k \leq \alpha_i$, at time (right before) α_i, offer from k to j[a] is at most $\alpha_i - d(i,j) - 2d(k,j)$ for any $j \in Q$.

[a] We assume k currently (or is going to) connects to j'.

Proof. We see that if $\alpha_i = \alpha_k$, the offer is just $(\alpha_k - d(i,j))^+$. Otherwise, we have $\alpha_k < \alpha_i$. If $\alpha_i > d(k,j') + d(k,j) + d(i,j)$, we immediately get a contradiction since from triangle inequality, $\alpha_i > d(i,j')$, i.e., i already connect to j'. Hence,

$$\alpha_i \leq d(k,j') + d(k,j) + d(i,j).$$

Then, the offer from k to j is $(d(k,j') - d(k,j))^+ \geq \alpha_i - d(i,j) - 2d(k,j)$. ⊛

Observe that for all $i \in [n]$, we have

$$\sum_{k=1}^{i-1}(\alpha_i - d(i,j) - 2d(k,j)) + \sum_{k=i}^{n}(\alpha_i - d(k,j)) \leq \hat{f} \tag{3.1}$$

by considering the total offer from k to j at time (right before) α_i. Now, we add Equation 3.1 for all $i \in [n]$, we have

$$n\sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}(i-1)d(i,j) - 2\sum_{k=1}^{n}(n-k)d(k,j) - \sum_{i=1}^{n}k \cdot d(k,j) \leq n\hat{f} = 2nf.$$

Since the summation over k is just indexes, we can change it to i, hence

$$
\begin{aligned}
2nf &\geq n\sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}(i-1)d(i,j) - 2\sum_{i=1}^{n}(n-i)d(i,j) - \sum_{i=1}^{n} i\cdot d(i,j) \\
&\geq n\sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n} id(i,j) - 2\sum_{i=1}^{n}(n-i)d(i,j) - \sum_{i=1}^{n} i\cdot d(i,j) = n\sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n} 2nd(i,j)
\end{aligned}
$$

where we turn the factor $(i-1)$ into i and gather the terms together. Clean up a bit, we have

$$
n\sum_{i=1}^{n}\alpha_i - 2n\sum_{i=1}^{n} d(i,j) \leq 2nf \Leftrightarrow \frac{\sum_{i=1}^{n}\alpha_i}{2} - \sum_{i=1}^{n} d(i,j) \leq f,
$$

finishing the proof. ■

From Lemma 3.2.2, we immediately have the following.

Theorem 3.2.2. Algorithm 3.5 is a 2-LMP approximation algorithm w.r.t. the original f.

Lecture 7: k-Median and LMP Approximation

3.3 k-Median

21 Sep. 10:30

Let's look at another clustering problem.

Problem 3.3.1 (k-median)**.** Given a metric space (X,d) and $P,Q\subseteq X$ with $k\in\mathbb{N}$, find $Q'\subseteq Q$ with $|Q'|=k$ which minimizes $\sum_{i\in P}\min_{j\in Q'} d(i,j)$.

The natural linear programming for Problem 3.3.1 is the following. Consider $\{x_{ij}\}_{i\in P, j\in Q}$ and $\{y_j\}_{j\in Q}$, then

$$
\begin{aligned}
\min\ & \sum_{ij} x_{ij}d(i,j) & & \\
& \sum_{j} x_{ij} \geq 1 & \forall i\in P \qquad & (\alpha_i) \\
& x_{ij} \leq y_j & \forall i\in P, j\in Q \qquad & (\beta_{ij}) \\
& \sum_{j} y_j \leq k & & (f)^{1} \\
& x,y\geq 0 & &
\end{aligned}
$$

Intuition. We interpret x_{ij} as follows: if $x_{ij}=1$, then i belongs to j. And $y_j=1$ if j is the actual median we choose (i.e., in Q'). As for constraints, both $\sum_j x_{ij}\geq 1$ and $\sum_j y_j\leq k$ are clear, while for $x_{ij}\leq y_j$, we see that it can't be the case that $x_{ij}=1$ while $y_j=0$, i.e., we can't have the case that x_{ij} belongs to j while j isn't even in Q'.

[1] Notice that compare to Problem 3.2.1, f here is a variable but not a given facility cost! The reason why we do this will be clear soon.

The dual is then

$$
\begin{aligned}
\max \quad & \sum_i \alpha_i - kf \\
& \sum_i \beta_{ij} \leq d(i,j) && \forall i \in P, j \in Q \\
& \sum_i \beta_{ij} \leq f && \forall j \in Q \\
& \alpha, \beta \geq 0
\end{aligned}
$$

> **Note.** Notice that this is exactly the dual as Problem 3.2.1, except that we now have an additional $-kf$ term in the objective function. Although f is not included in the statement of Problem 3.3.1, by denoting one of the dual variable f, we get a similar formulation compare to Problem 3.2.1.

Due to the similarity between Problem 3.3.1 and Problem 3.2.1, we can try to use Algorithm 3.5 which solves Problem 3.2.1 with 2-LMP guarantee. But note that in Problem 3.2.1, we need to specify f. Suppose we guessed f, and we run a γ-LMP approximation algorithm and somehow get $k' = k$. Then we have

$$
\frac{\mathrm{conn}(\mathrm{ALG})}{\gamma} \leq \sum_i \alpha_i - kf \leq \mathsf{OPT}_{k\text{-med.}},
$$

i.e., this is a γ-approximation algorithm. So now, the task is to guess f such that the algorithm gives exactly k centers.

3.3.1 Bipoint Solution

Turns out that we don't have ideas about the relation between k and f, the only thing we know is that if $f \to \infty$, k decreases, other than that it behaves quite arbitrary.

> **Remark.** The relation between k and f indeed highly depends on what algorithm we use. But at least for Algorithm 3.5, nobody knows anything in this case.

Given this fact, just randomly guess one f doesn't work. A new idea is then to maintain two solutions (or interval) $[f^2, f^1]$ such that $f^2 \leq f^1$,[2] where

- at f^2, the algorithm opens $k^2 \geq k$ facilities;
- at f^1, the algorithm opens $k^1 \leq k$ facilities.

Then, a naive approach is to use binary search and get $f^2 \leq f^1$ such that

$$
\left| f^1 - f^2 \right| \leq \frac{\epsilon\,\mathsf{OPT}}{n}.
$$

Notice that the whole point of doing binary search is because we assume that if $k^2 \geq k$ at f^2 and $k_1 \leq k$ at f^1, then we can find an $f^* \in [f^2, f^1]$ such that we get exactly $k^* = k$ at f^*.

> **Remark** (Caveat of achieving k)**.** This is probably not the case for Algorithm 3.2 (2-LMP) since the decision is quite sequential; but if we use Algorithm 3.1 (2-LMP), since there are lots of maximal independent sets, so by doing **a lot more work**, we can actually achieve this.

Now, assume that we have continuity of the relation between k and f by carefully designing our (γ-LMP) algorithm, then $\exists a \in [0,1]$ and $b := 1 - a$ such that $k := ak^1 + bk^2$ where $k^1 \leq k \leq k^2$. Denote C^i as the connection cost $\mathrm{conn}(f^i)$ of f^i such that $C^1 \geq C^2$, we have

$$
\begin{cases}
C^1 + \gamma k^1 f^1 \leq \gamma \sum_i \alpha_i^1, & (\times a) \\
C^2 + \gamma kr2 f^2 \leq \gamma \sum_i \alpha_i^2, & (\times b)
\end{cases}
$$

[2] We start from $f^2 = 0$ and $f^1 = \infty$, where we set f^1 arbitrary large.

hence,

$$aC^1 + bC^2 \leq \gamma \left(a \sum_i \alpha_i^1 + b \sum_i \alpha_i^2 - ak^1 f^1 - bk^2 f^2 \right) \leq \gamma \underbrace{\left(\sum_i \alpha_i - kf \right)}_{\leq \mathsf{OPT}_{k\text{-med.}}} + \underbrace{\gamma k \left| f^1 - f^2 \right|}_{\leq \epsilon\, \mathsf{OPT}_{k\text{-med.}}},$$

where we set $\alpha := a\alpha^1 + b\alpha^2$ and $f := \max(f^1, f^2)$.

Note. To make sure $\sum_i \alpha_i - kf \leq \mathsf{OPT}_{k\text{-med.}}$, we need to check that (α, f) is dual-feasible for Problem 3.3.1.

Proof. The feasibility comes from the fact that the first two constraints of Problem 3.3.1 are linear, so they're automatically satisfied. The only non-trivial constraint is $\sum_i \beta_{ij} \leq f$, but since we choose f to be the maximum, it'll be more satisfied. ⊛

Definition 3.3.1 (Bipoint solution). Given F^1, F^2 with $\left|F^1\right| = k^1$, $\left|F^2\right| = k^2$ and $k = ak^1 + bk^2$ for $a, b \in [0, 1]$ and $a + b = 1$, the *bipoint solution*, denoted as $aF^1 + bF^2$, satisfies

$$aC^1 + bC^2 \leq \gamma \cdot \mathsf{OPT}_{k\text{-med.}}.$$

3.3.2 Bipoint Rounding

From Definition 3.3.1, it's natural to do the so-called bipoint rounding.

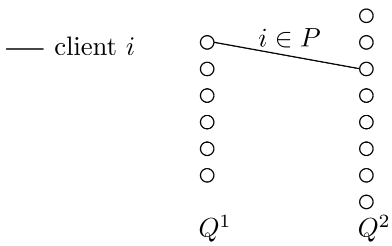
Definition 3.3.2 (δ-bipoint rounding). Given solutions F^1 and F^2, a solution F with $|F| = k$ such that

$$\operatorname{conn}(F) \leq \delta \cdot (aC^1 + bC^2) = \delta \cdot \operatorname{conn}(aF^1 + bF^2)$$

is the so-called *δ-bipoint rounding solution*.

Note. If we have a δ-bipoint rounding of a γ-LMP algorithm solution, then we automatically have an approximation ratio of $\delta \cdot \gamma$ for this bipoint rounding solution.

Back to Problem 3.3.1, we see that we can actually get a 2-bipoint rounding as follows. Consider we create a bipartite graph with $Q^1, Q^2 \subseteq Q$ being two sides of the graph. Then for each $i \in P$, i is connected to the closest facility in Q^1, and also another closest facility in Q^2, so we can create an edge between these two facilities.



Now, for a fixed $i \in P$, let $d_j := d(i, Q^j)$ for $j = 1, 2$, we want to compare our designed final cost to $aC^1 + bC^2$, so for this fixed i, we want to make sure i *pays* not much more than $ad_1 + bd_2$.

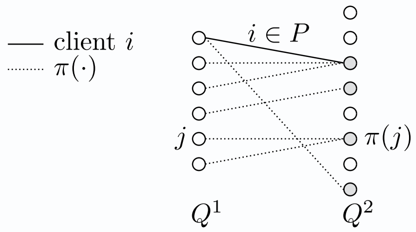
Intuition. We see that a natural rounding algorithm is the following: for an $i \in P$, if its closest facility in Q^1 is opened while its closest facility in Q^2 is not opened, we may just direct i to the opened one in Q^1, same for the other case. Now, if both facilities are opened, then we direct i to the facility in F^1 with probability a, while to the facility in Q^2 with probability $1 - a = b$.

Remark. The problem of the above algorithm is that we don't have control about the total number of the final open facilities: it can be the case that at the end we open every facility in Q^2, which is k^2, not k. So we need to sometimes direct i to other facilities (in Q^1) that is not closest to which.

For $j \in Q^1$, let $\pi(j)$ be the closest facility in Q^2 to j, and let Q^* be the image of such a map π, i.e., $Q^* = \left\{j' \in Q^2 \colon j' = \pi(j) \text{ for some } j \in Q^1\right\}$.

Note. We may assume $|Q^*| = k^1$.

Proof. Clearly, $|Q^*| \leq k^1$. And if $|Q^*| < k^1$, we add arbitrary centers so that $|Q^*| = k^1$.



For example, the initial image size above is only 4, we need to add 2 more arbitrary centers into Q^*. ⊛

To open the facilities as what we want, consider the following rounding algorithm.

Algorithm 3.6: k-Median – 2-Bipoint Rounding

Data: A set of clients $P \subseteq X$, a set of (possible) facilities $Q \subseteq X$, $a \in (0,1)$, $\epsilon \in (0,1)$, $k \in \mathbb{N}$
Result: A set of opened facilities $Q' \subseteq Q$ with $|Q'| = k$

1 $(Q^1, Q^2) \leftarrow$`binary-search(`P, Q, ϵ`)` // achieve $\left|f^1 - f^2\right| \leq \epsilon\,\mathsf{OPT}/n$
2
3 $Q' \leftarrow \varnothing$, $k^1 \leftarrow \left|Q^1\right|$, $k^2 \leftarrow \left|Q^2\right|$, $Q^* \leftarrow \left\{j' \in Q^2 \colon j' = \pi(j) \text{ for some } j \in Q^1\right\}$
4
5 **for** $j \in Q^1$ **do**
6   **if** `rand(`0, 1`)`$\leq a$ **then** // open Q^1 w.p.a
7         $Q' \leftarrow Q' \cup \{j\}$
8   **else** // open Q^* w.p.$1-a$
9         $Q' \leftarrow Q' \cup \{\pi(j)\}$
10
11 // still need to open $k - k^1$ more
12 $Q' \leftarrow Q' \cup \left\{(k - k^1) \text{ random } j \in Q^2 \setminus Q^* \right\}$
13 **return** Q'

Remark. Algorithm 3.6 is a randomized algorithm which will **always** open k facilities. The randomness comes from the cost, i.e., we can analyze its cost in expectation.

Intuition. Algorithm 3.6 is kind of *mimicking* what we want, since

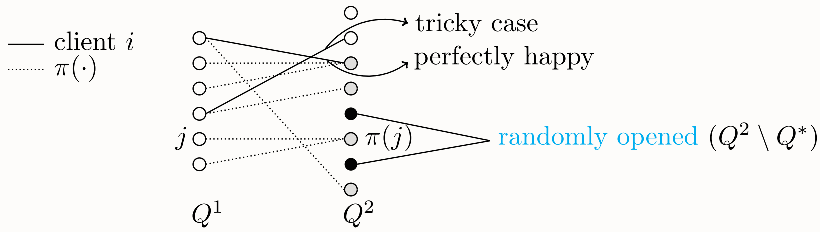
- $j \in Q^1$, $\Pr(j \text{ open}) = a$
- $j \in Q^*$, $\Pr(j \text{ open}) = 1 - a = b$
- $j \in Q^2 \setminus Q^*$, $\Pr(j \text{ open}) = \frac{k-k^1}{k^2-k^1} = b$

Theorem 3.3.1. Algorithm 3.6 is a 2-bipoint algorithm (in expectation).

Proof. Let's analyze a bit careful. Fixing an $i \in P$, and denote its closest facility in Q^1 as j^1, and the closest facility in Q^2 as j^2. If j^1 is not opened, then we know $\pi(j^1)$ is opened for sure in line 8.

We see that

- If j^2 is in Q^*, then we know i will be direct to either j^1 or j^2 in line 5, i.e., i is perfectly happy since it can go to one of the closest facility.
- The tricky case is when j^2 is not in Q^*.
  - If j^1 is opened, i can still go to j^1 without problem.
  - If j^1 is also not opened, we know that $\pi(j^1)$ will be opened in line 8. In this worst case, we just direct i to $\pi(j^1)$ and the distance will be $i \to j^1 \to \pi(j^1)$, which is bounded by $d_1 + d(j^1, \pi(j^1))$. But observe that $d(j^1, \pi(j^1)) \leq d_1 + d_2$, so we have $2d_1 + d_2$.



In all, we have the following.[a]

	Distance	Probability
j^2 open	d_2	b
j^2 not open, j^1 open	d_1	$\geq (a-b)^+ =: M$
none of j^1, j^2 open	$2d_1 + d_2$	$\leq 1 - b - M$

Then, the expected cost is just[b]

$$\mathbb{E}\left[i\text{'s connection cost}\right] \leq bd_2 + Md_1 + (1-b-M)(2d_1+d_2),$$

and we now have two cases.

- If $b \geq a$, then $b \geq 1/2$, $M = 0$ and

$$\mathbb{E}\left[i\text{'s connected cost}\right] \leq b \cdot d_2 + (1-b)(2d_1+d_2) = 2ad_1 + d_2 \leq 2(ad_1 + bd_2).$$

- If $a > b$, then $a > 1/2$, $M = a - b$ and

$$\mathbb{E}\left[i\text{'s connected cost}\right] \leq b \cdot d_2 + (a-b)d_1 + b(2d_1+d_2) = d_1(a+b) + d_2(b+b) \leq 2(ad_1 + bd_2).$$

This shows Algorithm 3.6 is a 2-bipoint algorithm in expectation, proving the result. ■

[a] This is a slightly worse result since we force i to go to j^2 if j^2 is opened, but actually, i can go to j^1 if j^1 is opened too with shorter distance. But this still gives us a good enough bound.

[b] Since the final case is always worse than the second case, it is legal to assume that the second case has the minimum probability and the final has the maximum for the expectation bound to hold.

Remark (SOTA). The SOTA result specifically for Problem 3.3.1 is summarized as follows.

Primal-Dual 3-LMP $\xrightarrow{\text{Conversion}}$ 3-approximation

Greedy 2-LMP $\longrightarrow$ 2-bipoint rounding $\longrightarrow$ 4-approximation

Dual Fitting [Coh+22] 1.9...9-LMP $\longrightarrow$ 1.3...3-bipoint rounding[a] $\longrightarrow$ 2.67-approximation

But we'll see that by changing the problem a bit, like consider squaring the distance in the objective of Problem 3.3.1 (which is the k-mean problem), we can get 9-approximation by Primal-Dual, while the lower path doesn't tell us anything, which is so fragile.

[a]This will return $k+c$ centers, where c is an absolute constant. There's a way to transform this solution back to k centers without loosing any performance.

Note (Derandomized). It's possible to derandomized Algorithm 3.6.

Lecture 8: Local Search for k-Median

We'll now see a completely different algorithm which solve Problem 3.3.1 with $(3+\epsilon)$-approximation ratio by local search. 26 Sep. 10:30

3.3.3 Local Search

The idea is to iteratively improve the current solution. We first see the algorithm.

Algorithm 3.7: k-Median – Local Search

Data: A set of clients $P \subseteq X$, a set of (possible) facilities $Q \subseteq X$, $k \in \mathbb{N}$, width w
Result: A set of opened facilities $Q' \subseteq Q$ with $|Q'| = k$

1 $Q' \leftarrow$ arbitrary k centers in Q
2 **while** $\exists Q''$ *s.t.* $|Q''| = k$ *and* $\mathrm{cost}(Q'') < \mathrm{cost}(Q')$ *and* $|Q' \triangle Q''| \leq w$[a] **do**
3 $\quad Q' \leftarrow Q''$
4 **return** Q'

[a]The symmetric difference $A \triangle B$ is defined as $A \triangle B \coloneqq (A \setminus B) \cup (B \setminus A)$.

Remark (Runtime). In line 2, each iteration in Algorithm 3.7 takes $(n+m)^{O(w)}$ time for $n \coloneqq |P|$ and $m \coloneqq |Q|$. But we have no control of how many iterations Algorithm 3.7 might take since we might decrease the cost by a little each time hence we might fall into exponentially many updates. To solve this, we can ask for

$$\mathrm{cost}(Q'') < (1-\epsilon)\,\mathrm{cost}(Q')$$

instead to make sure we decrease a reasonable amount each time, which guarantees that we can bound the number of iterations by

$$\log_{\frac{1}{1-\epsilon}}\left(\frac{\mathrm{cost}(\text{starting } Q')}{\mathsf{OPT}}\right).$$

Analysis

Firstly, note that for any solution Q' output from Algorithm 3.7, we have that there exists no Q'' such that $|Q' \triangle Q''| \leq w$, $|Q''| = k$ and $\mathrm{cost}(Q'') < \mathrm{cost}(Q')$.

Note (Local optimum). We say this Q' is a *local optimum*.

Let $Q^* \subseteq Q$ be the optimal solution, and without loss of generality (by duplicating facilities), assume $Q' \cap Q^* = \varnothing$. We define something called swap.

Notation (Swap). A *swap* $S \subseteq Q' \cup Q^*$ satisfies $|S \cap Q'| = |S \cap Q^*| \leq w/2$.

Note. From local optimality of Q', for any swap S, $\mathrm{cost}(Q') \leq \mathrm{cost}(Q' \triangle S)$.

Now, consider constructing swaps $S_1, \ldots, S_t$ with weights $p_1, \ldots, p_t \in \mathbb{R}^+$ such that $\mathrm{cost}(Q') \leq$

$\text{cost}(Q' \triangle S_i)$ for all i, we have

$$\sum_{i=1}^{t} p_i \cdot (\text{cost}(Q') - \text{cost}(Q' \triangle S_i)) \leq 0. \tag{3.2}$$

Our goal is to show that Equation 3.2 implies $\text{cost}(Q') \leq \alpha \cdot \text{cost}(Q^*)$ for some $\alpha \in \mathbb{R}^+$. To do this, we require the set of swaps to have the following properties.

(a) For all $j \in Q^*$, $\sum_{S_i \ni j} p_i = 1$, and let $p' := \max_{j \in Q'} \sum_{S_i \ni j} p_i$.

(b) For all $j \in Q^*$, let $\pi(j) \in Q'$ be the facility closest to j. Then if S_i contains $j \in Q'$, $\pi^{-1}(j) \subseteq S_i$.[3]

The existence of such swaps family is ensured by the following lemma.

Lemma 3.3.1. There exists a family of swaps $S_1, \ldots, S_t$ with weights $p_1, \ldots, p_t$ such that $\forall j \in Q^*$, $\sum_{S_i \ni j} p_i = 1$ and if $j \in S_i$, $\pi^{-1}(j) \subseteq S_i$ with $p' = \max_{j \in Q'} \sum_{S_i \ni j} p_i = 1 + 2/w$.

Proof. For all $j \in Q'$, we call j

- *big*: if $\left|\pi^{-1}(j)\right| > w/2$.
- *small*: if $\left|\pi^{-1}(j)\right| \in [1, w/2]$.
- *lonely*: if $\left|\pi^{-1}(j)\right| = 0$.

Then for each small or big j, we create a *group* G_j that contains $\pi^{-1}(j)$, j and $\left|\pi^{-1}(j)\right| - 1$ lonely facilities (denote as $R_j \subseteq Q'$). We see that $|G_j| = 2\left|\pi^{-1}(j)\right|$, and we can ensure each lonely facility belongs to exactly 1 group, i.e., $\exists G_1, \ldots G_r$ such that each facility belongs to exactly 1 group. It's now clear that how we should create swaps and their corresponding weight:

(a) For small j, let G_j be a swap with weight 1.

(b) For big j, let $w' := \left|\pi^{-1}(j)\right|$, then for any $S \subseteq \pi^{-1}(j)$ and $T \subseteq R_j$ with $|S| = |T| = w/2$, we let $(S \cup T)$ be a swap with weight $1/\left(\binom{w'-1}{w/2-1} \cdot \binom{w'-1}{w/2}\right)$.[a]

Since for every $j^* \in Q^*$, there is only one group containing j^*, to verify $\sum_{S_i \ni j^*} p_i = 1$, we see that

(a) j^* is containing in G_j for j small: In this case, we have one swap, i.e., G_j itself with weight 1.

(b) j^* is containing in G_j for j big: In this case, since every such swap created inside G_j contains j^* and has uniform weight, it sums up to 1.

Finally, we want to show that $p' = \max_{j \in Q'} \sum_{S_i \ni j} p_i = 1 + 2/w$. But this is also easy since given $j \in Q'$, the summation is inside G_j, and in particular, S_i is inside G_j as well. Then

(a) j is small: Only swap is G_j itself with weight 1.

(b) j is big: j can't even be in one swap, hence the sum is 0.

(c) j is lonely: In this case, we have

$$\sum_{S_i \ni j} p_i = \frac{1}{\binom{w'-1}{w/2-1}\binom{w'-1}{w/2}} \cdot \binom{w'}{w/2}\binom{w'-2}{w/2-1} = \frac{w'}{w/2}\frac{w/2}{w'-1} = \frac{w'}{w'-1} \leq 1 + \frac{2}{w}.$$

Taking the maximum, we have $p' = 1 + 2/w$ as desired. ■

[a] Notice that since j is big, so j can't be in any swap, so we have only $w' - 1$ to choose from.

With Lemma 3.3.1, we're ready to prove the following.

[3] In particular, if $\left|\pi^{-1}(j)\right| > w/2$, then j is not contained in any swap.

Theorem 3.3.2. Algorithm 3.7 is a $(3+\epsilon)$-approximation algorithm for arbitrary small $\epsilon > 0$.

Proof. Fix $i \in P$, we analyze how it contributes to the left-hand side of Equation 3.2. Let $j' \in Q'$ and $j^* \in Q^*$ be facilities closest to i, and $d'_i := d(i, j')$, $d^*_i = d(i, j^*)$, then for every S_ℓ, we have

(a) $S_\ell \ni j^*$. Then contribution of i to $\mathrm{cost}(Q') - \mathrm{cost}(S_\ell \triangle Q')$ is at least $d'_i - d^*_i$.[a]

(b) $S_\ell \ni j'$. By the second property, either

- $\pi(j^*) \in S_\ell$: this implies $j^* \in S_\ell$, which falls back to the first case.
- $\pi(j^*) \notin S_\ell$: the contribution of i to $\mathrm{cost}(Q') - \mathrm{cost}(S_\ell \triangle Q')$ is at least $d'_i - (2d^*_i + d'_i) = -2d^*_i$.[b]

(c) Otherwise, contribution of i to $\mathrm{cost}(Q') - \mathrm{cost}(S_\ell \triangle Q')$ is at least 0.[c]

In all, we see that the first case has total weight 1 from the first property, while (b)−(a) has total weight $\leq p'$, hence Equation 3.2 implies

$$\sum_{i \in P} [(d'_i - d^*_i) \cdot 1 - (2d^*_i) \cdot p'] \leq \sum_{\ell=1}^{t} p_\ell (\mathrm{cost}(Q') - \mathrm{cost}(S_\ell \triangle Q')) \leq 0,$$

which is equivalent to say

$$\mathrm{cost}(Q') - (1 - 2p')\,\mathsf{OPT} \leq 0,$$

so we get a $(1+2p')$-approximation ratio. Furthermore, From Lemma 3.3.1, we have $p' = 1 + 2/w$, hence we can achieve $1 + 2(1 + 2/w) = 3 + 4/w$-approximation ratio. Given $\epsilon > 0$, by setting $w := 4/\epsilon$, we're done. ∎

[a] i can go to j^*.
[b] i can go to $\pi(j^*)$.
[c] i can stay with j'.

Lecture 9: Euclidean k-Median

Consider the following problem. 28 Sep. 10:30

Problem 3.3.2 (Euclidean k-median). Given a metric space $(X, d) = (\mathbb{R}^\ell, \|\cdot\|_2)$ and $P, Q \subseteq X$ with $k \in \mathbb{N}$, find $Q' \subseteq Q$ with $|Q'| = k$ which minimizes $\sum_{i \in P} \min_{j \in Q'} d(i, j)$.

We see that Problem 3.3.2 is like Problem 3.3.1, but with a specific metric space, i.e., $(\mathbb{R}^\ell, \|\cdot\|_2)$.

Note. We assume that ℓ is large. If it's not the case, then actually for all $\epsilon > 0$, there exists a $(1+\epsilon)$-approximation algorithm with running time $2^{2^{O(\ell)}} \cdot \mathrm{poly}(n)$. Hence, if ℓ is small (or constant), we can use this algorithm. But if it's not, what we're going to see is better.

It's natural to ask that whether we can solve Problem 3.3.2 by the similar algorithm which solves Problem 3.2.1 and Problem 3.3.1. In particular, we're going to use Algorithm 3.1.

As previously seen (Dual LP for Problem 3.2.1). Recall the dual of Problem 3.2.1

$$\begin{aligned} & \max \sum_i \alpha_i \\ & \alpha_i - \beta_{ij} \leq d(i, j) && \forall i, j \\ & \sum_i \beta_{ij} \leq f && \forall j \\ (D) \quad & \alpha, \beta \geq 0 \end{aligned}$$

and the Algorithm 3.1.

Again, we think of α_i as the time that i is connected, and t_j be the time that j is open in Algorithm 3.1, and the only thing we change is the phase two, i.e., how we trim down the solution. We now see the algorithm, which essentially achieves $\rho := (1+\delta)$-LMP approximation where $\delta := \sqrt{8/3} \approx 1.633\ldots$[4]

Algorithm 3.8: Euclidean k-Median – Primal-Dual

Data: A set of clients $P \subseteq X$, a set of (possible) facilities $Q \subseteq X$, facility cost f
Result: A set of opened facilities $Q' \subseteq Q$

$S \leftarrow \varnothing$, $Q' \leftarrow \varnothing$, $\alpha \leftarrow 0$ // S:connected clients, O:open facilities

while $S \neq P$ **do**
  while *True* **do**
    increase all $\{\alpha_i\}_{i \in P \setminus S}$ by a *unit*
    if *some* $j \in Q \setminus Q'$ *s.t.* $\sum_{i \in P} \beta_{ij} = f$ **then** // j gets tight (open)
      break
    else if *some* $i \in P \setminus S$ *s.t.* $\alpha_i \geq d(i,j)$ **then** // i can connect to $j \in Q'$
      break
  $Q' \leftarrow \{\text{tight facilities}\}$ // Update Q'
  $S \leftarrow \{\text{clients connected to } Q'\}$ // Update S

// Trim down Q'
$G = (Q', E := \{(j,j') \colon \exists i \in P \text{ such that } d(j,j') \leq \delta \cdot \min(t_j, t_{j'}), j, j' \in Q'\})$
Compute Q'' s.t. $\forall j \in Q'$, either $j \in Q''$ or $\exists j' \in Q''$ s.t. $(j,j') \in E$ // max independent set
return Q''

Intuition. The problematic part is when i contributing to too many facilities at once, but we'll see that this can't happen if we're considering Euclidean metric.

Remark. As before, let $w(i) \in Q'$ for all i such that $\alpha \geq t_{w(i)}$, i.e., $w(i)$ is the connected witness of i. Then we have the following.

(a) α is dual-feasible.

(b) If $\beta_{ij} > 0$, then $\alpha_i \leq t_j$.

(c) For all i, $\exists w(i) \in Q'$ such that $\alpha_i \geq t_{w(i)}$.

We now do the analysis similarly. Fix a client $i \in P$, then observe that given $S = Q'' \cap \{j \colon \beta_{ij} > 0\}$, if $\delta = 2$, then $|S| \leq 1$.[5] We see that

(a) If $|S| = 1$, $S = \{j\}$. We see that $\operatorname{conn}(i) \leq d(i,j)$ and $\operatorname{open}(i) = \alpha_i - d(i,j)$, so
$$\operatorname{conn}(i) + \operatorname{open}(i) \leq d(i,j) + (\alpha_i - d(i,j)) \leq \alpha_i.$$

(b) If $|S| = 0$, then $\operatorname{open}(i) = 0$ and either $w(i) \in Q''$, or $j' \in Q''$ such that $(w(i), j') \in E$. In any case, $\operatorname{conn}(i) \leq d(i,j') \leq d(i, w(i)) + d(w(i), j')$, hence
$$\operatorname{conn}(i) + \operatorname{open}(i) \leq d(i,j') \leq \alpha_i + \delta t_{w(i)} \leq (1+\delta)\alpha_i.$$

Generally, our goal is to prove that for all i,
$$\frac{\operatorname{conn}(i)}{\rho} + \operatorname{open}(i) \leq \alpha_i, \tag{3.3}$$
which implies
$$\frac{\operatorname{conn}}{\rho} + |Q''| f \leq \sum_i \alpha_i,$$
i.e., we get a ρ-LMP approximation algorithm.

[4] We'll see why we have this artificial number soon.
[5] Since if both β_j and $\beta_{j'}$ is greater than 0, then $d(j,j') \leq 2\alpha_i \leq 2\min(t_j, t_{j'})$. This means j and j' will have an edge but from the property of max independent set, one of them will not be included.

Note. Specifically, Equation 3.3 is equivalent to

$$\frac{\min_{j\in S} d(i,j)}{\rho} + \sum_{j\in S}(\alpha_i - d(i,j)) \leq \alpha_i.$$

Furthermore, in the case of $\delta = 2$, we see that we can set $\rho \coloneqq 1+\delta = 3$.

Remark. We see that we get the exactly 3-LMP approximation for $\delta = 2$ case! Notice that in this case, since $|S| \leq 1$ as we noted, Algorithm 3.8 and Algorithm 3.1 are equivalent.

Claim. For $\delta \coloneqq \sqrt{8/3}$ and $S = Q'' \cap \{j \colon \beta_{ij} > 0\}$, $|S| \leq 3$.

Proof. We first see two tricks, which we call them the k*-means magic formulas*. Let $i' = \sum_{j\in S} j / |S|$. Then

$$\begin{aligned}
\sum_{j\in S} \|j-i\|^2 &= \sum_{j\in S} \langle j-i+i'-i', j-i+i'-i'\rangle \\
&= \sum_{j\in S}\left(\|j-i\|^2 + \|i'-i\|^2 + 2\langle j-i', i'-i\rangle\right) \\
&= \sum_{j\in S}\|j-i'\|^2 + |S|\,\|i'-i\|^2 .
\end{aligned}$$

Remark (Fact)**.** One can actually show that i' (i.e., the geometric mean) is the optimal solution for k-means, and if we choose i rather than i' to be the center, the deviation from OPT is exactly $|S|\,\|i'-i\|^2$.

Also,

$$\begin{aligned}
\sum_{j,j'\in S} \|j-j'\|^2 &= \sum_{j,j'\in S} \langle j-j'+i'-i', j-j'+i'-i'\rangle \\
&= \sum_{j,j'\in s}\left(\|j-i'\|^2 + \|j'-i'\|^2 + 2\langle j-i', i'-j'\rangle\right) \\
&= 2\,|S| \cdot \sum_{j\in S}\|j-i'\|^2 .
\end{aligned}$$

From the above tricks, we have

$$|S|\,\alpha_i^2 \geq \sum_{j\in S}\|j-i\|^2 \geq \frac{1}{2\,|S|}\sum_{j,j'\in S}\|j-j'\|^2 > \frac{(s-1)\delta^2\alpha_i^2}{2},$$

where the last inequality follows from $\|j-j'\| > \delta\cdot\min(t_j, t_{j'}) \geq \delta\cdot\alpha_i$. Then, we have

$$|S|\,\alpha_i^2 > \frac{(s-1)\delta^2\alpha_i^2}{2} \Rightarrow |S|\left(\frac{\delta^2}{2}-1\right) < \frac{\delta^2}{2} \Rightarrow |S| < \frac{\delta^2}{\delta^2-2} = 4$$

by plugging in $\delta = \sqrt{8/3}$, hence $|S| \leq 3$ by integrality. ⊛

Now, given the fact that we already handle the case that $|S| = 0$ and $|S| = 1$, we now consider the cases that $|S| = 2$ and $|S| = 3$. If $|S| = 2$, let $S = \{j_1, j_2\}$, then $(\alpha_i - d(i,j_1)) + (\alpha_i - d(i,j_2)) \leq (2-\delta)\alpha_i$. Since $\mathrm{conn}(i) \leq \alpha_i$,

$$\frac{d(i,j^*)}{\rho} + \sum_{j\in S}(\alpha_i - d(i,j)) \leq \alpha_i\left(\frac{1}{\rho}+2-\delta\right) \leq \alpha_i, \tag{3.4}$$

where the last inequality follows from $1/\rho + 2 - \delta \leq 1 \Leftrightarrow 1/\rho \leq \delta - 1$, which is satisfied by $\rho \coloneqq 1+\delta = 1+\sqrt{8/3}$.

If $|S| = 3$, let $S = \{j_1, j_2, j_3\}$. Now, instead of looking at a more complicated geometric structure and try to optimize it, we simply add Equation 3.4 three times for (j_1, j_2), (j_2, j_3) and (j_1, j_3), we have $2\sum_{j\in S}(\alpha_i - d(i,j)) \leq 3(2-\delta)\alpha_i$ hence

$$\frac{d(i,j^*)}{\rho} + \sum_{j\in S}(\alpha_i - d(i,j)) \leq \alpha_i\left(\frac{1}{\rho} + \frac{3(2-\delta)}{2}\right) \leq \alpha_i$$

since $1/\rho + 3(2-\delta)/2 \leq 1 \Leftrightarrow 1/\rho \leq (3\delta-4)/2$, which is satisfied by $\rho \coloneqq 1+\delta = 1+\sqrt{8/3}$

Remark (SOTA). Compare general metric Problem 3.3.1 and Euclidean metric Problem 3.3.2, we have the following.

2.41-LMP[a] $\xrightarrow{\text{Conversion}}$ 2.41-approximation

Euclidean Primal-Dual 2.63-LMP $\xrightarrow{\text{Conversion}}$ 2.63-approximation

Primal-Dual 3-LMP $\xrightarrow{\text{Conversion}}$ 3-approximation

Dual Fitting [Coh+22] $1.9\ldots9$-LMP $\longrightarrow$ $1.3\ldots3$-bipoint rounding $\longrightarrow$ 2.67-approximation

Local Search 3-LMP $\longrightarrow$ 2-bipoint rounding $\longrightarrow$ 4-approximation

Noticeably, 2.41-LMP approximation is $1+\sqrt{2}$, which is exactly the threshold behavior in Euclidean metric we're building our intuition upon.

[a] 2.40 is the SOTA.

Chapter 4

Traveling Salesman Problem

Lecture 10: Spanning Tree

Instead of discussing general network design problems, we focus on traveling salesman problem specifically. And turns out that although this is a good old problem in TCS, but still, lots of improvement is done in the past decade. Turns out, most of the improvement is based on the understanding of spanning tree, specifically, how to sample a good enough random spanning tree. 3 Oct. 10:30

4.1 Spanning Tree

We first look at the definition of a spanning tree.

Definition 4.1.1 (Spanning tree). A *spanning tree* T of a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an induced subgraph of $\mathcal{G}$ which spans $\mathcal{G}$, i.e., $V(T) = \mathcal{V}$ and $E(T) \subseteq \mathcal{E}$.

Remark. A spanning tree of a connected graph $\mathcal{G}$ can also be defined as a maximal set of edges of $\mathcal{G}$ that contains no cycle, or as a minimal set of edges that connect all vertices.

Then, we're interested in the following problem.

Problem 4.1.1 (Minimum spanning tree). Given a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and an edge-weight function $w\colon \mathcal{E} \to \mathbb{R}^+$, find a spanning tree T which minimizes $w(T)$.

There are lots of different algorithms which solve Problem 4.1.1, e.g., Prim's algorithm, Kruskal's Algorithm, etc. in undergraduate algorithm courses. But turns out that by looking at the LP formulation of this problem, we get some non-trivial result.

4.1.1 LP Formulation

Denote the variables as $\{x_e\}_{e\in\mathcal{E}}$, where we interpret $x_e = 1$ if e is in the final spanning tree, otherwise if it's 0, then e is not in the final spanning tree. One natural formulation is the following:

$$
\begin{aligned}
\min\ & \sum_{e\in\mathcal{E}} x_e w(e) & \\
& \sum_{e\in\partial S} x_e \geq 1 & \forall S \subseteq \mathcal{V} \\
& x \geq 0. &
\end{aligned}
$$

Notation. If $S \subseteq \mathcal{V}$, then we denote $\partial S = E(S, \overline{S})$ be the edges between S and $\overline{S}$.

Intuition. The second constraint is trying to model that for every cut set $S \subseteq \mathcal{V}$, our spanning tree need to include at least one edge from the boundary, i.e., ∂S.

But turns out that this formulation will give us an integrality gap of 2, since for a cycle graph, just by choosing half of the edges, i.e., $x_e = 1/2$ for all $e \in \mathcal{E}$, the constraints are satisfied while we know we need to include all but one edge to form a valid spanning tree.

Remark. There are ways to strengthen the second constraints by looking at **directed** spanning trees rather than the usual undirected ones to give us an LP which solves Problem 4.1.1 exactly.

We see that the problems arise from the fact that there are not enough edges to span $\mathcal{G}$, so we now require it explicitly in our LP formulation. Furthermore, to ensure there are no cycles, for any $S \subseteq \mathcal{V}$, we again make sure that the total edges we have is less than $|S| - 1$. Then, we have the following.

$$
\begin{aligned}
\min \ & \sum_{e\in\mathcal{E}} x_e w(e) \\
& \sum_{e\in\mathcal{E}} x_e = n-1 \\
& \sum_{e\in E(S,\overline{S})} x_e \leq |S| - 1 \forall S \subseteq \mathcal{V} \\
& x \geq 0.
\end{aligned}
\tag{4.1}
$$

This is not solvable just by throwing this into an LP solver since there are exponentially many constraints! Regardless, we note the following.

Remark (Separation oracle). Given a linear program (P) with $x \in \mathbb{R}^n$ as variables, a *separation oracle* is an algorithm which outputs

- Yes if x is feasible.
- No with *the violating constraint* if x is not feasible.

And if we have a polynomial time separation oracle, we can solve any LP in polynomial time by using the ellipsoid algorithm.

Now, we just state that there's a separation oracle for the above LP, so we can solve it in polynomial time and get a fractional solution $\{x_e\}_{e\in\mathcal{E}}$. So our next task is to round it into an integral one.

4.1.2 Pipage Rounding

Now, call $S \subseteq \mathcal{V}$ *tight* if $\sum_{e\in E(S,\overline{S})} x_e = |S| - 1$.

Lemma 4.1.1 (Uncrossing). If S and T are tight with $S \cap T \neq \varnothing$, both $S \cup T$ and $S \cap T$ are tight.

Proof. Observe that since S and T are tight and $S \cup T$ and $S \cap T$ are cuts as well (hence satisfy the constraints),

$$
\begin{aligned}
(|S|-1) + (|T|-1) &= \sum_{e\in E(S)} x_e + \sum_{e\in E(T)} x_e \\
&\leq \sum_{e\in E(S\cup T)} x_e + \sum_{e\in E(S\cap T)} x_e \leq (|S\cup T| - 1) + (|S\cap T| - 1),
\end{aligned}
$$

with the fact that $|S| + |T| = |S \cup T| + |S \cap T|$,[a] hence everything is equal. ■

[a] Consider every possible edges between $S \setminus T$, $T \setminus S$, $S \cap T$ and $\overline{S \cup T}$.

Finally, we call a tight T *integral* if and only if for all $e \in E(T)$, $x_e \in \{0, 1\}$; and a tight T *fractional* if there exists $e \neq f \in E(T)$ such that x_e and x_f are fractional.[1]

[1] We can equivalently require only one x_e being fractional, but since T is tight, there'll another $f \neq e$ such that x_f is fractional as well.

Deterministic Pipage Rounding

We first see the deterministic rounding algorithm.

Algorithm 4.1: Minimum spanning tree – Pipage-Rounding

Data: A connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a weight function $w \colon \mathcal{E} \to \mathbb{R}^+$
Result: A minimum spanning tree T

1 $x \leftarrow$ `LP-Solver(`*Equation 4.1*`)` // with separation oracle
2
3 // Pipage Rounding
4 **while** $x \notin \mathbb{N}^m$ **do** // not integral
5 $\quad T \leftarrow$ minimal tight fraction set // inclusion-wise minimal[a]
6 $\quad f, g \leftarrow$ fractional edges // $f, g \in E(T)$
7 $\quad$ **if** $w(f) > w(g)$ **then** // ensure $w(f) \leq w(g)$
8 $\quad\quad$ `swap(`f, g`)`
9 $\quad$ **while** *increase* x_f *and decrease* x_g *by a unit* **do** // by solving Equation 4.2
10 $\quad\quad$ **if** x_f *or* x_g *becomes integral* **then**
11 $\quad\quad\quad$ **break**
12 $\quad\quad$ **else if** $\exists T' \subsetneq T$ *is tight* **then**
13 $\quad\quad\quad$ **break**
14
15 $T \leftarrow$ `Subgraph(`$\mathcal{G}$,x`)` // construct a spanning tree
16 **return** T

[a] i.e., $\nexists T' \subsetneq T$ tight fractional set.

Remark (Implementation detail). There are two non-trivial steps in Algorithm 4.1.

- line 9: This continuous process is done by taking δ from solving the following LP as the total unit we should increase/decrease:

$$\begin{aligned} \max\ & \delta \\ & y = x + \delta e_f - \delta e_g \\ & \sum_{e \in E(S)} y - e \leq |S| - 1 \quad \forall S \subseteq \mathcal{V} \\ & 0 \leq y \leq 1, \end{aligned} \tag{4.2}$$

  where e_i is the unit vector with 1 at entry i. Again, this is in the similar form as Equation 4.1, and there's a separation oracle which solves this LP in polynomial-time.

- line 5: Start from the whole vertex set $\mathcal{V}$, and we simply look at f which is none-integral edge and ask can we increase it or not, i.e., we ask the separation oracle for Equation 4.2, and if there's a smaller tight fraction set inside T, $\delta > 0$ strictly, and we just keep searching in this way. We'll see what does this mean exactly in Lemma 4.1.2.

Our goal now is to show that during the pipage rounding:

(a) x remains feasible;

(b) $\sum_e x_e w(e)$ remains unchanged (does not increase).

To show x remains feasible, first note that the non-tight sets are handled (captured) in line 12, as for tight sets, we have Lemma 4.1.2.

Lemma 4.1.2. All tight sets remain tight after running line 9.

Proof. The only way for a tight set becomes over-tight is when we increase x_f in line 9, an already tight set U becomes over-tight. But if this is the case and U is violated, then $U \ni f$ and $U \not\ni g$ and

U is tight, we have $U \cap T$ is tight from Lemma 4.1.1, contradicting the minimality of T ↯ ■

Remark. From the proof above, we see that we can now find minimal T by increasing a fractional x_f: if some set U is not violated, then we know $T \cap U$ is tight, then we just keep nesting and get the minimal one.

Now, we just need to show Algorithm 4.1 terminates in polynomial time.

Lemma 4.1.3. Algorithm 4.1 in a polynomial time algorithm.

Proof. Observe that

(a) line 10 can only happen m times: at most m edges can be fractional at first, and after one becomes integral, it remains integral.

(b) line 12 can only happen n times: at most n nodes can be in T at first, and when line 12 is triggered, the size of T decreases by at least 1 and never goes up.

In all, we see that Algorithm 4.1 is a polynomial time algorithm. ■

Note. Notice that in line 12, we require $T' \subsetneq T$, and if we trigger this, in the next iteration when choosing T in line 5, we'll need to choose a strictly smaller T compare to the last iteration[a] in order to make Lemma 4.1.3 valid.

[a]This is guaranteed by Lemma 4.1.2 since we know the only we change is x_f and x_g, and if some new T' can become tight, it has non-empty intersection with T and hence as the remark, we can find such a T'.

We see that this implies the following.

Theorem 4.1.1. Algorithm 4.1 solves Problem 4.1.1 exactly in polynomial time.

Proof. Firstly, Algorithm 4.1 is a polynomial time algorithm from Lemma 4.1.3. Also, since Equation 4.1 is an LP-relaxation of Problem 4.1.1 while we know that ■

Randomized Pipage Rounding

We first see the algorithm.

Algorithm 4.2: Minimum spanning tree – Randomized pipage-rounding

Data: A connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a weight function $w\colon \mathcal{E} \to \mathbb{R}^+$
Result: A minimum spanning tree T

$x \leftarrow$ `LP-Solver(`*Equation 4.1*`)` // with separation oracle

while $x \notin \mathbb{N}^m$ **do** // not integral
  $T \leftarrow$ minimal tight fraction set // inclusion-wise minimal[a]
  $f, g \leftarrow$ fractional edges // $f, g \in E(T)$
  if $w(f) > w(g)$ **then** // ensure $w(f) \leq w(g)$
    `swap(`f, g`)`
  $a \leftarrow \max_a x_f \leftarrow x_f + a, x_g \leftarrow x_g - a$ feasible // $a > 0$
  $b \leftarrow \max_b x_f \leftarrow x_f - b, x_g \leftarrow x_g + b$ feasible // $b > 0$
  if `rand(`$(0,1)$`)`$< \frac{b}{a+b}$ **then** // w.p. $\frac{b}{a+b}$
    $x_f \leftarrow x_f + a$, $x_g \leftarrow x_g - a$
  else // w.p. $\frac{a}{a+b}$
    $x_f \leftarrow x_f - b$, $x_g \leftarrow x_g + b$

$T \leftarrow$ `Subgraph(`$\mathcal{G}$,x`)` // construct a spanning tree
return T

[a]i.e., $\nexists T' \subsetneq T$ tight fractional set.

As in the deterministic version, the same proof can show that x is feasible, and the number of iteration will be less than $m \cdot n$, hence it's a polynomial time algorithm. Remarkably, we have the following.

Theorem 4.1.2. Algorithm 4.2 solves Problem 4.1.1 exactly.

Proof. To show that the cost is good enough, note that in one iteration, $\mathbb{E}\left[x^{\text{end}}\right] = x^{\text{start}}$, then

$$\mathbb{E}\left[x^{\text{final}}\right] = x^{\text{LP}},$$

hence any possible x^{final} satisfies

$$\sum_{e\in\mathcal{E}} x_e^{\text{final}} w(e) = \sum_{e\in\mathcal{E}} x_e^{\text{LP}} w(e),$$

hence we get a 1-approximation algorithm, i.e., Algorithm 4.2 solves Problem 4.1.1 exactly. ■

Remark. We can interpret x^{final} as the distribution of spanning trees, i.e., we have

$$\mathbb{E}\left[x^{\text{final}}\right] = x^{\text{LP}} \Leftrightarrow \forall e \in \mathcal{E}, \Pr(e \in T) = x_e^{\text{LP}},$$

where the probability depends on the randomness introduce in Algorithm 4.2, i.e., x_e^{final}.

4.2 Negative Correlation

The above remark is just for one edge, and actually, Algorithm 4.2 produces a negative correlated distribution. Firstly, if x_e^{final} are independent, then

$$\mathbb{E}\left[\prod_{e\in S} x_e^{\text{final}}\right] = \Pr(S \subseteq T) = \prod_{e\in S} \Pr(e \in T) = \prod_{e\in S} x_e^{\text{LP}}.$$

But since we know that x_e^{final} are not independent for sure since they depend on a sequence of steps executed by Algorithm 4.2, it's non-trivial to analyze. We now see the main result in this section.

Theorem 4.2.1 (Negative correlation). For all $S \subseteq \mathcal{E}$,

$$\mathbb{E}\left[\prod_{e\in S} x_e^{\text{final}}\right] = \Pr(S \subseteq T) \leq \prod_{e\in S} \Pr(e \in T) = \prod_{e\in S} x_e^{\text{LP}}.$$

Proof. Let y^i be x after i^{th} iteration maintained by Algorithm 4.2, it's sufficient to show

$$\mathbb{E}\left[\prod_{e\in S} y_e^{i+1} \mid y^i\right] \leq \prod_{e\in S} y_e^i$$

since if this holds, say Algorithm 4.2 runs M iterations in total, then

$$\mathbb{E}\left[\prod_{e\in S} x_e^{\text{final}}\right] = \mathbb{E}\left[\prod_{e\in S} y_e^{M}\right] = \mathbb{E}\left[\prod_{e\in S} y_e^{M} \mid y^{M-1}\right] \leq \prod_{e\in S} y_e^{M-1},$$

any by taking expectation again iteratively, we obtain the desired result down to $\prod_{e\in S} y_e^0$. Now, consider that in the i^{th} iteration of Algorithm 4.2, for f, g picked in line 5:

(i) $f, g \notin S$: trivially holds.

(ii) $f \in S$, $g \notin S$:[a] we have $\mathbb{E}\left[\prod_{e\in S} y_e^{i+1} \mid y^i\right] = \prod_{e\in S\setminus\{f\}} y_e^i \cdot \mathbb{E}\left[y_f^{i+1} \mid y^i\right] = \prod_{e\in S} y_e^i$ where $\mathbb{E}\left[y_f^{i+1} \mid y^i\right] = y_f^i$ is the designed from Algorithm 4.2.

(iii) $f, g \in S$. Suffices to compare $\mathbb{E}\left[y_f^{i+1} \cdot y_g^{i+1} \mid y^i\right]$ and $y_f^i \cdot y_g^i$, and the goal is to show $\leq$.

(a) $\mathbb{E}\left[(y_f^{i+1} + y_g^{i+1})^2 \mid y^i\right] = (y_f^i + y_g^i)^2$ since $y_f^{i+1} + y_g^{i+1} = y_f^i + y_g^i$ almost surely.

(b) $\mathbb{E}\left[(y_f^{i+1} - y_g^{i+1})^2 \mid y^i\right] \geq (y_f^i + y_g^i)^2$ since the variance of any random variable is non-negative.

We see that by subtracting them, we have $\mathbb{E}\left[y_f^{i+1} \cdot y_g^{i+1} \mid y^i\right] \leq y_f^i \cdot y_g^i$ as desired.

In all cases, the hypothesis for i holds, hence the theorem is proved. ■

[a] And also $g \in S$ and $f \notin S$, since they're symmetric.

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.

[Coh+22] Vincent Cohen-Addad et al. *Breaching the 2 LMP Approximation Barrier for Facility Location with Applications to k-Median.* 2022. DOI: 10.48550/ARXIV.2207.05150. URL: https://arxiv.org/abs/2207.05150.

[GK99] Sudipto Guha and Samir Khuller. "Greedy Strikes Back: Improved Facility Location Algorithms". In: *Journal of Algorithms* 31.1 (1999), pp. 228–248. ISSN: 0196-6774. DOI: https://doi.org/10.1006/jagm.1998.0993. URL: https://www.sciencedirect.com/science/article/pii/S0196677498909932.

[JMS02] Kamal Jain, Mohammad Mahdian, and Amin Saberi. "A New Greedy Approach for Facility Location Problems". In: STOC '02. Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 731–740. ISBN: 1581134959. DOI: 10.1145/509907.510012. URL: https://doi.org/10.1145/509907.510012.

[JV01] Kamal Jain and Vijay V. Vazirani. "Approximation Algorithms for Metric Facility Location and K-Median Problems Using the Primal-Dual Schema and Lagrangian Relaxation". In: *J. ACM* 48.2 (Mar. 2001), pp. 274–296. ISSN: 0004-5411. DOI: 10.1145/375827.375845. URL: https://doi.org/10.1145/375827.375845.

[Li13] Shi Li. "A 1.488 approximation algorithm for the uncapacitated facility location problem". In: *Information and Computation* 222 (2013). 38th International Colloquium on Automata, Languages and Programming (ICALP 2011), pp. 45–58. ISSN: 0890-5401. DOI: https://doi.org/10.1016/j.ic.2012.01.007. URL: https://www.sciencedirect.com/science/article/pii/S0890540112001459.

[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.