Advanced Algorithms

Computer Science, ETH Zürich

Mohsen Ghaffari

These notes will be updated regularly. Please read critically; there are typos throughout, but there might also be mistakes. Feedback and comments would be greatly appreciated and should be emailed to aroeyskoe@inf.ethz.ch.

Last update: September 20, 2023

Contents

Notation and useful inequalities

Commonly used notation

- $\mathcal{P}$: class of decision problems that can be solved on a deterministic sequential machine in polynomial time with respect to input size
- $\mathcal{NP}$: class of decision problems that can be solved non-deterministically in polynomial time with respect to input size. That is, decision problems for which "yes" instances have a proof that can be verified in polynomial time.
- $\mathcal{A}$: usually denotes the algorithm we are discussing about
- $\mathcal{I}$: usually denotes a problem instance
- ind.: independent / independently
- w.p.: with probability
- w.h.p: with high probability
  We say event X holds *with high probability* (w.h.p.) if

$$\Pr[X] \geq 1 - \frac{1}{poly(n)}$$

  say, $\Pr[X] \geq 1 - \frac{1}{n^c}$ for some constant $c \geq 2$.
- L.o.E.: linearity of expectation
- u.a.r.: uniformly at random
- Integer range $[n] = \{1, \ldots, n\}$
- $e \approx 2.718281828459$: the base of the natural logarithm

Useful distributions

Bernoulli Coin flip w.p. p. Useful for indicators

$$\begin{aligned}\Pr[X = 1] &= p \\ \mathbb{E}[X] &= p \\ \mathrm{Var}(X) &= p(1-p)\end{aligned}$$

Binomial Number of successes out of n trials, each succeeding w.p. p; Sample *with replacement* out of n items, p of which are successes

$$\begin{aligned}\Pr[X = k] &= \binom{n}{k} p^k (1-p)^{n-k} \\ \mathbb{E}[X] &= np \\ \mathrm{Var}(X) &= np(1-p) \leq np\end{aligned}$$

Geometric Number of Bernoulli trials until one success

$$\begin{aligned}\Pr[X = k] &= (1-p)^{n-1} p \\ \mathbb{E}[X] &= \frac{1}{p} \\ \mathrm{Var}(X) &= \frac{1-p}{p^2}\end{aligned}$$

Hypergeometric Number of successes in n draws *without replacement*, from a population of N items in which K are successful:

$$\begin{aligned}\Pr[X = k] &= \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}} \\ \mathbb{E}[X] &= n \cdot \frac{K}{N} \\ \mathrm{Var}(X) &= n \cdot \frac{K}{N} \cdot \frac{N-K}{N} \cdot \frac{N-n}{N-1}\end{aligned}$$

Exponential Parameter: λ; Written as $X \sim \mathrm{Exp}(\lambda)$

$$\begin{aligned}\Pr[X = x] &= \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \\ \mathbb{E}[X] &= \frac{1}{\lambda} \\ \mathrm{Var}(X) &= \frac{1}{\lambda^2}\end{aligned}$$

Remark If $x_1 \sim \text{Exp}(\lambda_1), \ldots, x_n \sim \text{Exp}(\lambda_n)$, then

- $\min\{x_1, \ldots, x_n\} \sim \text{Exp}(\lambda_1 + \cdots + \lambda_n)$
- $\Pr[k \mid x_k = \min\{x_1, \ldots, x_n\}] = \frac{\lambda_k}{\lambda_1 + \cdots + \lambda_n}$

Useful inequalities

- $(\frac{n}{k})^k \leq \binom{n}{k} \leq (\frac{en}{k})^k$
- $\binom{n}{k} \leq n^k$
- $\lim_{n\to\infty}(1 - \frac{1}{n})^n = e^{-1}$
- $\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$
- $(1 - x) \leq e^{-x}$, for any x
- $(1 + 2x) \geq e^x$, for $x \in [0, 1]$
- $(1 + \frac{x}{2}) \geq e^x$, for $x \in [-1, 0]$
- $(1 - x) \geq e^{-x-x^2}$, for $x \in (0, \frac{1}{2})$
- $\frac{1}{1-x} \leq 1 + 2x$ for $x \leq \frac{1}{2}$

Theorem (Linearity of Expectation)**.**

$$\mathbb{E}(\sum_{i=1}^{n} a_i X_i) = \sum_{i=1}^{n} a_i \mathbb{E}(X_i)$$

Theorem (Variance)**.**

$$\mathbb{V}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$

Theorem (Variance of a Sum of Random Variables)**.**

$$\mathbb{V}(aX + bY) = a^2\mathbb{V}(X) + b^2\mathbb{V}(Y) + 2ab\text{Cov}(X, Y)$$

Theorem (AM-GM inequality)**.** *Given n numbers $x_1, \ldots, x_n$,*

$$\frac{x_1 + \cdots + x_n}{n} \geq (x_1 * \cdots * x_n)^{1/n}$$

The equality holds if and only if $x_1 = \cdots = x_n$.

Theorem (Markov's inequality)**.** *If X is a nonnegative random variable and $a > 0$, then*

$$\Pr[X \geq a] \leq \frac{\mathbb{E}(X)}{a}$$

Theorem (Chebyshev's inequality)**.** *If X is a random variable (with finite expected value μ and non-zero variance σ^2), then for any $k > 0$,*

$$\Pr[|X - \mu| \geq k\sigma] \leq \frac{1}{k^2}$$

Theorem (Bernoulli's inequality)**.** *For every integer $r \geq 0$ and every real number $x \geq -1$,*

$$(1 + x)^r \geq 1 + rx$$

Theorem (Chernoff bound)**.** *For independent Bernoulli variables $X_1, \ldots, X_n$, let $X = \sum_{i=1}^{n} X_i$. Then,*

$$\Pr[X \geq (1 + \epsilon) \cdot \mathbb{E}(X)] \leq \exp(\frac{-\epsilon^2 \mathbb{E}(X)}{3}) \qquad \textit{for } 0 < \epsilon$$
$$\Pr[X \leq (1 - \epsilon) \cdot \mathbb{E}(X)] \leq \exp(\frac{-\epsilon^2 \mathbb{E}(X)}{2}) \qquad \textit{for } 0 < \epsilon < 1$$

By union bound, for $0 < \epsilon < 1$, we have

$$\Pr[|X - \mathbb{E}(X)| \geq \epsilon \cdot \mathbb{E}(X)] \leq 2 \exp(\frac{-\epsilon^2 \mathbb{E}(X)}{3})$$

Remark 1 There is actually a tighter form of Chernoff bounds:

$$\forall \epsilon > 0, \Pr[X \geq (1 + \epsilon)\mathbb{E}(X)] \leq (\frac{e^\epsilon}{(1 + \epsilon)^{1+\epsilon}})^{\mathbb{E}(X)}$$

Remark 2 We usually apply Chernoff bound to show that the probability of bad approximation is low by picking parameters such that $2 \exp(\frac{-\epsilon^2 \mathbb{E}(X)}{3}) \leq \delta$, then negate to get $\Pr[|X - \mathbb{E}(X)| \leq \epsilon \cdot \mathbb{E}(X)] \geq 1 - \delta$.

Theorem (Probabilistic Method)**.** *Let $(\Omega, \mathcal{A}, P)$ be a probability space,*

$$\Pr[\omega] > 0 \Longleftrightarrow \exists \omega \in \Omega$$

Combinatorics taking k elements out of n:

- no repetition, no ordering: $\binom{n}{k}$
- no repetition, ordering: $\frac{n!}{(n-k)!}$
- repetition, no ordering: $\binom{n+k-1}{k}$
- repetition, ordering: n^k

Part I

Basics of Approximation Algorithms

Chapter 1

Greedy algorithms

Unless $\mathcal{P} = \mathcal{NP}$, we do not expect efficient algorithms for $\mathcal{NP}$-hard problems. However, we are often able to design efficient algorithms that give solutions that are provably close/approximate to the optimum.

Definition 1.1 (α-approximation)**.** *An algorithm $\mathcal{A}$ is an α-approximation algorithm for a* minimization *problem with respect to cost metric c if for any problem instance I and for some optimum solution OPT,*

$$c(\mathcal{A}(I)) \leq \alpha \cdot c(OPT(I))$$

Maximization problems are defined similarly with $c(OPT(I)) \leq \alpha \cdot c(\mathcal{A}(I))$.

1.1 Minimum set cover & vertex cover

Consider a universe $\mathcal{U} = \{e_1, \ldots, e_n\}$ of n elements, a collection of subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$ of m subsets of $\mathcal{U}$ such that $\mathcal{U} = \bigcup_{i=1}^{m} S_i$, and a non-negative[1] cost function $c : \mathcal{S} \to \mathbb{R}^+$. If $S_i = \{e_1, e_2, e_5\}$, then we say S_i *covers* elements e_1, e_2, and e_5. For any subset $T \subseteq \mathcal{S}$, define the cost of T as the cost of all subsets in T. That is,

$$c(T) = \sum_{S_i \in T} c(S_i)$$

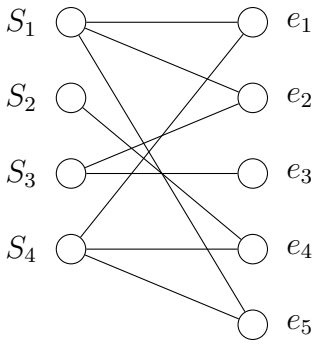
Definition 1.2 (Minimum set cover problem)**.** *Given a universe of elements $\mathcal{U}$, a collection of subsets $\mathcal{S}$, and a non-negative cost function $c : \mathcal{S} \to \mathbb{R}^+$, find a subset $S^* \subseteq \mathcal{S}$ such that:*

(i) S^* is a set cover: $\bigcup_{S_i \in S^*} S_i = \mathcal{U}$

(ii) $c(S^*)$, the cost of S^*, is minimized

[1] If a set costs 0, then we can just remove all the elements covered by it for free.

Example



Suppose there are 5 elements e_1, e_2, e_3, e_4, e_5, 4 subsets S_1, S_2, S_3, S_4, and the cost function is defined as $c(S_i) = i^2$. Even though $S_3 \cup S_4$ covers all vertices, this costs $c(\{S_3, S_4\}) = c(S_3) + c(S_4) = 9 + 16 = 25$. One can verify that the minimum set cover is $S^* = \{S_1, S_2, S_3\}$ with a cost of $c(S^*) = 14$. Notice that we want a minimum cover with respect to c and not the number of subsets chosen from $\mathcal{S}$ (unless c is uniform cost).

1.1.1 A greedy minimum set cover algorithm

Since finding the minimum set cover is $\mathcal{NP}$-complete, we are interested in algorithms that give a good approximation for the optimum. [Joh74] describes a greedy algorithm GreedySetCover and proved that it gives an H_n-approximation[2]. The intuition is as follows: Spread the cost $c(S_i)$ amongst the vertices that are newly covered by S_i. Denoting the price-per-item by $\texttt{ppi}(S_i)$, we greedily select the set that has the lowest `ppi` at each step unitl we have found a set cover.

Algorithm 1 GreedySetCover$(\mathcal{U}, \mathcal{S}, c)$

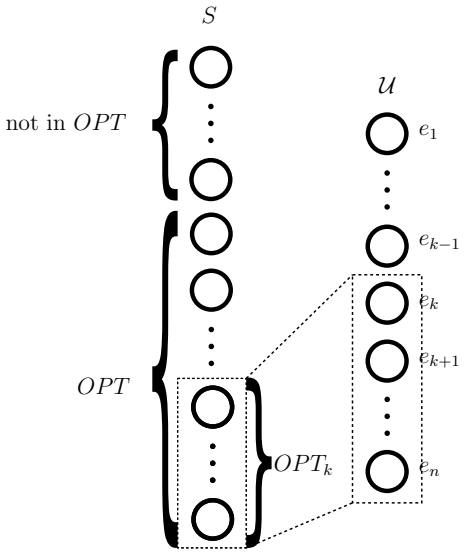
$T \leftarrow \emptyset$ ▷ Selected subset of $\mathcal{S}$
$C \leftarrow \emptyset$ ▷ Covered vertices
while $C \neq \mathcal{U}$ **do**
  $S_i \leftarrow \arg\min_{S_i \in S \setminus T} \frac{c(S_i)}{|S_i \setminus C|}$ ▷ Pick set with lowest price-per-item
  $T \leftarrow T \cup \{S_i\}$ ▷ Add S_i to selection
  $C \leftarrow C \cup S_i$ ▷ Update covered vertices
end while
return T

[2] $H_n = \sum_{i=1}^{n} \frac{1}{i} = \ln(n) + \gamma \leq \ln(n) + 0.6 \in \mathcal{O}(\log(n))$, where γ is the Euler-Mascheroni constant. See https://en.wikipedia.org/wiki/Euler-Mascheroni_constant.

Consider a run of GreedySetCover on the earlier example. In the first iteration, $\texttt{ppi}(S_1) = 1/3$, $\texttt{ppi}(S_2) = 4$, $\texttt{ppi}(S_3) = 9/2$, $\texttt{ppi}(S_4) = 16/3$. So, S_1 is chosen. In the second iteration, $\texttt{ppi}(S_2) = 4$, $\texttt{ppi}(S_3) = 9$, $\texttt{ppi}(S_4) = 16$. So, S_2 was chosen. In the third iteration, $\texttt{ppi}(S_3) = 9$, $\texttt{ppi}(S_4) = \infty$. So, S_3 was chosen. Since all vertices are now covered, the algorithm terminates (coincidentally to the minimum set cover). Notice that ppi for the unchosen sets change according to which vertices remain uncovered. Furthermore, once one can simply ignore S_4 when it no longer covers any uncovered vertices.

Theorem 1.3. GreedySetCover *is an H_n-approximation algorithm.*

Proof. By construction, GreedySetCover terminates with a valid set cover T. It remains to show that $c(T) \leq H_n \cdot c(OPT)$ for any minimum set cover OPT. Upon relabelling, let $e_1, \ldots, e_n$ be the elements in the order they are covered by GreedySetCover. Define $\texttt{price}(e_i)$ as the price-per-item associated with e_i at the time e_i was purchased during the run of the algorithm. Consider the moment in the algorithm where elements $C_{k-1} = \{e_1, \ldots, e_{k-1}\}$ are already covered by some sets $T_k \subset T$. T_k covers no elements in $\{e_k, \ldots, e_n\}$. Since there is a cover[3] of cost at most $c(OPT)$ for the remaining $n-k+1$ elements, there must be an element $e^* \in \{e_k, \ldots, e_n\}$ whose price $\texttt{price}(e^*)$ is at most $\frac{c(OPT)}{n-k+1}$.



We formalize this intuition with the argument below. Since OPT is a set cover, there exists a subset $OPT_k \subseteq OPT$ that covers $e_k \ldots e_n$. Sup-

[3] OPT is a valid cover (though probably not minimum) for the remaining elements.

pose $OPT_k = \{O_1, \ldots, O_p\}$ where $O_i \in S \; \forall i \in [p]$. We make the following observations:

1. Since no element in $\{e_k, \ldots, e_n\}$ is covered by T_k, $O_1, \ldots, O_p \in \mathcal{S} \setminus T_k$.
2. Because some elements may be covered more than once,

$$\begin{aligned} n - k + 1 &= |\mathcal{U} \setminus C_{k-1}| \\ &\leq |O_1 \cap (\mathcal{U} \setminus C_{k-1})| + \cdots + |O_p \cap (\mathcal{U} \setminus C_{k-1})| \\ &= \sum_{j=1}^{p} |O_j \cap (\mathcal{U} \setminus C_{k-1})| \end{aligned}$$

3. By definition, for each $j \in \{1, \ldots, p\}$, $\texttt{ppi}(O_j) = \frac{c(O_j)}{|O_j \cap (\mathcal{U} \setminus C_{k-1})|}$.

Since the greedy algorithm will pick a set in $\mathcal{S} \setminus T_k$ with the lowest price-per-item, $\texttt{price}(e_k) \leq \texttt{ppi}(O_j)$ for all $j \in \{1, \ldots, p\}$. Substituting this expression into the last equation and rearranging the terms we get:

$$c(O_j) \geq \texttt{price}(e_k) \cdot |O_j \cap (\mathcal{U} \setminus C_{k-1})|, \forall j \in \{1, \ldots, p\} \tag{1.1}$$

Summing over all p sets, we have

$$\begin{aligned} c(OPT) &\geq c(OPT_k) && \text{Since } OPT_k \subseteq OPT \\ &= \sum_{j=1}^{p} c(O_j) && \text{Definition of } c(OPT_k) \\ &\geq \texttt{price}(e_k) \cdot \sum_{j=1}^{p} |O_j \cap (\mathcal{U} \setminus C_{k-1})| && \text{By Equation (1.1)} \\ &\geq \texttt{price}(e_k) \cdot |\mathcal{U} \setminus C_{k-1}| && \text{By observation 2} \\ &= \texttt{price}(e_k) \cdot (n - k + 1) \end{aligned}$$

Rearranging, $\texttt{price}(e_k) \leq \frac{c(OPT)}{n-k+1}$. Summing over all elements, we have:

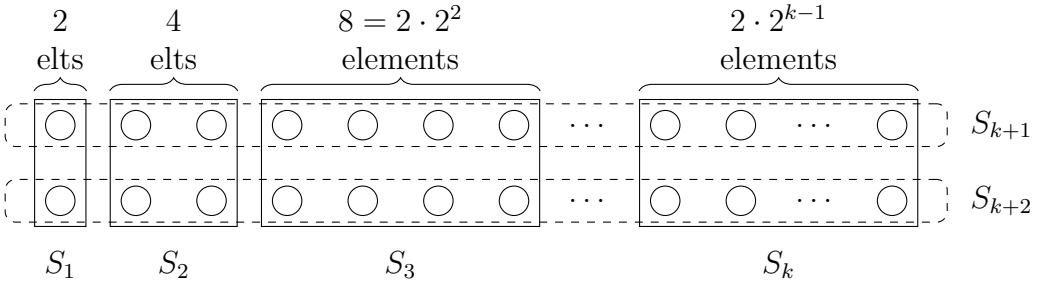
$$c(T) = \sum_{S \in T} c(S) = \sum_{k=1}^{n} \texttt{price}(e_k) \leq \sum_{k=1}^{n} \frac{c(OPT)}{n - k + 1} = c(OPT) \cdot \sum_{k=1}^{n} \frac{1}{k} = H_n \cdot c(OPT)$$

□

Remark By construction, $\texttt{price}(e_1) \leq \cdots \leq \texttt{price}(e_n)$.

Next we provide an example to show this bound is indeed tight.

Tight bound example for GreedySetCover Consider $n = 2 \cdot (2^k - 1)$ elements, for some $k \in \mathbb{N} \setminus \{0\}$. Partition the elements into groups of size $2 \cdot 2^0, 2 \cdot 2^1, 2 \cdot 2^2, \ldots, 2 \cdot 2^{k-1}$. Let $\mathcal{S} = \{S_1, \ldots, S_k, S_{k+1}, S_{k+2}\}$. For $1 \leq i \leq k$, let S_i cover the group of size $2 \cdot 2^{i-1} = 2^i$. Let S_{k+1} and S_{k+2} cover half of each group (i.e. $2^k - 1$ elements each) such that $S_{k+1} \cap S_{k+2} = \emptyset$.



Suppose $c(S_i) = 1, \forall i \in \{1, \ldots, k+2\}$. The greedy algorithm will pick S_k, then S_{k-1}, ..., and finally S_1. This is because $2 \cdot 2^{k-1} > n/2$ and $2 \cdot 2^i > (n - \sum_{j=i+1}^{k-1} 2 \cdot 2^j)/2$, for $1 \leq i \leq k-1$. This greedy set cover costs $k = \mathcal{O}(\log(n))$. Meanwhile, the minimum set cover is $S^* = \{S_{k+1}, S_{k+2}\}$ with a cost of 2.

A series of works by Lund and Yannakakis [LY94], Feige [Fei98], and Dinur [DS14, Corollary 1.5] showed that it is $\mathcal{NP}$-hard to always approximate set cover to within $(1 - \epsilon) \cdot \ln |\mathcal{U}|$, for any constant $\epsilon > 0$.

Theorem 1.4 ([DS14, Corollary 1.5])**.** *It is $\mathcal{NP}$-hard to always approximate set cover to within $(1 - \epsilon) \cdot \ln |\mathcal{U}|$, for any constant $\epsilon > 0$.*

Proof. See [DS14, Corollary 1.5] □

1.1.2 Special cases

In this section, we show that one may improve the approximation factor from H_n if we have further assumptions on the set cover instance. Viewing a set cover instance as a bipartite graph between sets and elements, let $\Delta = \max_{i \in \{1,\ldots,m\}} \texttt{degree}(S_i)$ and $f = \max_{i \in \{1,\ldots,n\}} \texttt{degree}(e_i)$ represent the maximum degree of the sets and elements respectively. Consider the following two special cases of set cover instances:

1. All sets are small. That is, Δ is small.

2. Every element is covered by few sets. That is, f is small.

Special case: Small Δ

Theorem 1.5. GreedySetCover *is a H_Δ-approximation algorithm.*

Proof. Suppose $OPT = \{O_1, \ldots, O_p\}$, where $O_i \in S \ \forall i \in [p]$. Consider a set $O_i = \{e_{i,1}, \ldots, e_{i,d}\}$ with $\texttt{degree}(O_i) = d \leq \Delta$. Without loss of generality, suppose that the greedy algorithm covers $e_{i,1}$, then $e_{i,2}$, and so on. For $1 \leq k \leq d$, when $e_{i,k}$ is covered, $\texttt{price}(e_{i,k}) \leq \frac{c(O_i)}{d-k+1}$ (It is an equality if the greedy algorithm also chose O_i to first cover $e_{i,k}, \ldots, e_{i,d}$). Hence, the greedy cost of covering elements in O_i (i.e. $e_{i,1}, \ldots, e_{i,d}$) is at most

$$\sum_{k=1}^{d} \frac{c(O_i)}{d-k+1} = c(O_i) \cdot \sum_{k=1}^{d} \frac{1}{k} = c(O_i) \cdot H_d \leq c(O_i) \cdot H_\Delta$$

Summing over all p sets to cover all n elements, we have $c(T) \leq H_\Delta \cdot c(OPT)$. □

Remark We apply the same greedy algorithm for small Δ but analyzed in a more localized manner. Crucially, in this analysis, we always work with the exact degree d and only use the fact $d \leq \Delta$ *after* summation. Observe that $\Delta \leq n$ and the approximation factor equals that of Theorem 1.3 when $\Delta = n$.

Special case: Small f

We first look at the case when $f = 2$, show that it is related to another graph problem, then generalize the approach for general f.

Vertex cover as a special case of set cover

Definition 1.6 (Minimum vertex cover problem)**.** *Given a graph $G = (V, E)$, find a subset $S \subseteq V$ such that:*

(i) S is a vertex cover: $\forall e = \{u, v\} \in E$, $u \in S$ or $v \in S$

(ii) $|S|$, the size of S, is minimized

We next argue that each instance of minimum vertex cover can be seen as an instance of minimum set cover problem with $f = 2$, and (more importantly for our approximation algorithm) any instance of minimum set cover problem with $f = 2$ can be reduced to an instance of minimum vertex cover.

When $f = 2$ and $c(S_i) = 1, \forall S_i \in \mathcal{S}$, the minimum vertex cover can be seen as an instance of minimum set cover. Given an instance I of minimum vertex cover $I = \langle G = (V, E) \rangle$ we build an instance $I^* = \langle \mathcal{U}^*, \mathcal{S}^* \rangle$ of minimum set cover as follows:

- Each edge $e_i \in E$ in G becomes an element e'_i in $\mathcal{U}^*$
- Each vertex $v_j \in V$ in G becomes an element S_j in $\mathcal{S}^*$ and $e'_i \in S_j \iff e_i$ is adjacent to $v_j \in I$

Notice that every element $e_i \in \mathcal{U}$ will be in exactly 2 elements of $\mathcal{S}$, for every edge is adjacent to exactly two vertices. Hence, I^* has $f = 2$.

Moreover, we can reduce the minimum set cover problem with $f = 2$ to an instance of a minimum vertex cover. For each element that appears in only one set, simply take the set that includes it. And repeat this until we have removed all the elements that appear in exactly one set. At this point, we are left with sets and elements such that each element appears in exactly two sets. Then, we can view this as a simple graph by thinking of the sets as the vertices and each element as an edge between between the two vertices corresponding to the two sets that contain the edge.

One way to obtain a 2-approximation to minimum vertex cover (and hence 2-approximation for this special case of set cover) is to use a maximal matching.

Definition 1.7 (Maximal matching problem)**.** *Given a graph $G = (V, E)$, find a subset $M \subseteq E$ such that:*

(i) M is a matching: Distinct edges $e_i, e_j \in M$ do not share an endpoint

(ii) M is maximal: $\forall e_k \notin M$, $M \cup \{e_k\}$ is not a matching

A related concept to maximal matching is maxi*mum* matching, where one tries to maximize the set of M. By definition, any maximum matching is also a maximal matching, but the converse is not necessarily true. Consider a path of 6 vertices and 5 edges. Both the set of blue edges $\{\{a, b\}, \{c, d\}, \{e, f\}\}$ and the set of red edges $\{\{b, c\}, \{d, e\}\}$ are valid maximal matchings, where the maximum matching is the former.

Remark Any maximal matching is a 2-approximation of maximum matching.

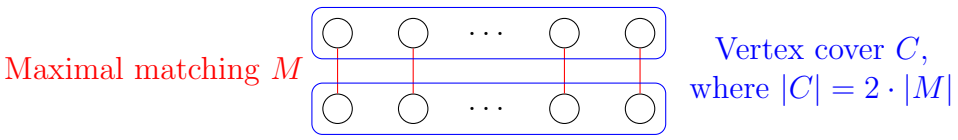
GreedyMaximalMatching is a greedy maximal matching algorithm. The algorithm greedily adds any available edge e_i that is not yet incident to M, then excludes all edges that are adjacent to e_i.

Algorithm 2 GREEDYMAXIMALMATCHING(V, E)

```
M ← ∅                                          ▷ Selected edges
C ← ∅                                          ▷ Set of incident vertices
while E ≠ ∅ do
    e_i = {u, v} ← Pick any edge from E
    M ← M ∪ {e_i}                              ▷ Add e_i to the matching
    C ← C ∪ {u, v}                             ▷ Add endpoints to incident vertices
    Remove all edges in E that are incident to u or v
end while
return M
```

Theorem 1.8. *The set of incident vertices C at the end of* GREEDYMAXIMALMATCHING *is a 2-approximation for minimum vertex cover.*

[h]



Proof. Suppose, for a contradiction, that GREEDYMAXIMALMATCHING terminated with a set C that is not a vertex cover. Then, there exists an edge $e = \{u, v\}$ such that $u \notin C$ and $v \notin C$. If such an edge exists, $e = \{u, v\} \in E$ then $M' = M \cup \{e\}$ would have been a matching with $|M'| > |M|$ and GREEDYMAXIMALMATCHING would not have terminated. This is a contradiction, hence C is a vertex cover.

Consider the matching M. Any vertex cover has to include at least one endpoint for each edge in M, hence the minimum vertex cover OPT has at least $|M|$ vertices (i.e. $|OPT| \geq |M|$). By picking C as our vertex cover, $|C| = 2 \cdot |M| \leq 2 \cdot |OPT|$. Therefore, C is a 2-approximation. □

We now generalize beyond $f = 2$ by considering hypergraphs. Hypergraphs are a generalization of graphs in which an edge can join any number of vertices. Formally, a hypergraph $H = (X, E)$ consists of a set X of vertices/elements and a set E of hyperedges where each hyperedge is an element of $\mathcal{P}(X) \setminus \emptyset$ (where $\mathcal{P}$ is the powerset of X). The minimum vertex cover problem and maximal matching problem are defined similarly on a hypergraph.

Remark A hypergraph $H = (X, E)$ can be viewed as a bipartite graph with partitions X and E, with an edge between element $x \in X$ and hyperedge $e \in E$ if $x \in e$ in H.

Example Suppose $H = (X, E)$ where $X = \{a, b, c, d, e\}$ and $E = \{\{a, b, c\}, \{b, c\}, \{a, d, e\}\}$. A minimum vertex cover of size 2 would be $\{a, c\}$ (there are multiple vertex covers of size 2). Maximal matchings would be $\{\{a, b, c\}\}$ and $\{\{b, c\}, \{a, d, e\}\}$, where the latter is the maximum matching.

Claim 1.9. *Generalizing* GreedyMaximalMatching *to compute a maximal matching in the hypergraph by greedily picking hyperedges yields an f-approximation algorithm for minimum vertex cover.*

Sketch of Proof Let C be the set of all vertices involved in the greedily selected hyperedges. In a similar manner as the proof in Theorem 1.8, C can be showed to be an f-approximation. □

Chapter 2

Approximation schemes

In the last chapter, we described simple greedy algorithms that approximate the optimum for minimum set cover, maximal matching and minimum vertex cover within a constant factor of the optimum solution. We now want to devise algorithms, which come arbitrarily close to the optimum solution. For that purpose we formalize the notion of efficient $(1+\epsilon)$-approximation algorithms for minimization problems, a la [Vaz13].

Let I be an instance from the problem of interest (e.g. minimum set cover). Denote $|I|$ as the size of the problem instance in bits, and $|I_u|$ as the size of the problem instance in unary. For example, if the input is a number x of at most n bits, then $|I| = \log_2(x) = \mathcal{O}(n)$ while $|I_u| = \mathcal{O}(2^n)$. This distinction of "size of input" will be important when we discuss the knapsack problem later.

Definition 2.1 (Polynomial time approximation scheme (PTAS))**.** *For a given cost metric c, an optimal algorithm OPT and a parameter ϵ, an algorithm $\mathcal{A}_\epsilon$ is a PTAS for a minimization problem if*

- $c(\mathcal{A}_\epsilon(I)) \leq (1+\epsilon) \cdot c(OPT(I))$
- $\mathcal{A}_\epsilon$ *runs in* $\text{poly}(|I|)$ *time*

Note that ϵ is a parameter of the algorithm, and is not considered as input. Thus the runtime for PTAS may depend arbitrarily on ϵ. If we define ϵ as an input parameter for the algorithm, we can obtain a stricter definition, namely that of fully polynomial time approximation schemes (FPTAS). Assuming $\mathcal{P} \neq \mathcal{NP}$, FPTAS is the best one can hope for on $\mathcal{NP}$-hard problems.

Definition 2.2 (Fully polynomial time approximation scheme (FPTAS))**.** *For a given cost metric c, an optimal algorithm OPT and input parameter ϵ, an algorithm $\mathcal{A}$ is an FPTAS for a minimization problem if*

- *For any* $\epsilon > 0$, $c(\mathcal{A}(I)) \leq (1+\epsilon) \cdot c(OPT(I))$
- $\mathcal{A}$ *runs in* $\text{poly}(|I|, \frac{1}{\epsilon})$ *time*

As before, one can define $(1-\epsilon)$-approximations, PTAS, and FPTAS for maximization problems similarly.

2.1 Knapsack

Definition 2.3 (Knapsack problem)**.** *Consider a set* $\mathcal{S}$ *with* n *items. Each item* i *has* $\texttt{size}(i) \in \mathbb{Z}^+$ *and* $\texttt{profit}(i) \in \mathbb{Z}^+$*. Given a budget* B*, find a subset* $S^* \subseteq S$ *such that:*

(i) Selection S^* *fits budget:* $\sum_{i \in S^*} \texttt{size}(i) \leq B$

(ii) Selection S^* *has maximum value:* $\sum_{i \in S^*} \texttt{profit}(i)$ *is maximized*

Let $p_{max} = \max_{i \in \{1,\ldots,n\}} \texttt{profit}(i)$ denote the highest profit for an item. Also, notice that any item which has $\texttt{size}(i) > B$ cannot be chosen, due to the size constraint, and therefore we can it. In $\mathcal{O}(n)$ time, we can remove any such item and relabel the remaining ones as items 1, 2, 3, ... Thus, without loss of generality, we can assume that $\texttt{size}(i) \leq B, \forall i \in \{1, \ldots, n\}$.

Observe that $p_{max} \leq \texttt{profit}(OPT(I))$ because we can always pick at least one item, namely the highest valued one.

Example Denote the i-th item by $i : \langle \texttt{size}(i), \texttt{profit}(i) \rangle$. Consider an instance with $\mathcal{S} = \{1 : \langle 10, 130 \rangle, 2 : \langle 7, 103 \rangle, 3 : \langle 6, 91 \rangle, 4 : \langle 4, 40 \rangle, 5 : \langle 3, 38 \rangle\}$ and budget $B = 10$. Then, the best subset $S^* = \{2 : \langle 7, 103 \rangle, 5 : \langle 3, 38 \rangle\} \subseteq S$ yields a total profit of $103 + 38 = 141$.

2.1.1 An exact algorithm via dynamic programming

The maximum achievable profit is at most np_{max}, as we can have at most n items, each having profit at most p_{max}. Define the `size` of a subset as the sum of the `size` of the sets involved. Using dynamic programming (DP), we can form an n-by-(np_{max}) matrix M where $M[i, p]$ is the smallest `size` of a subset chosen from $\{1, \ldots, i\}$ such that the total `profit` equals p. Trivially, set $M[1, 0] = 0$. To handle boundaries, define $M[i, j] = \infty$ for $j \leq 0$. Then, we compute $M[i+1, p]$ as follows:

- If $\texttt{profit}(i+1) > p$, then we cannot pick item $i+1$.
  So, $M[i+1, p] = M[i, p]$.

- If $\texttt{profit}(i+1) \leq p$, then we may pick item $i+1$.
  So, $M[i+1,p] = \min\{M[i,p], \texttt{size}(i+1) + M[i, p - \texttt{profit}(i+1)]\}$.

Since each cell can be computed in $\mathcal{O}(1)$ using DP via the above recurrence, matrix M can be filled in $\mathcal{O}(n^2 p_{max})$ and S^* may be extracted from the table $M[.,.]$: we find the maximum value $j \in [p_{max}, np_{max}]$ for which $M[n,j] < B$ and we back-track from there to extract the optimal set S^*.

Remark This dynamic programming algorithm is *not* a PTAS because $\mathcal{O}(n^2 p_{max})$ can be exponential in input problem size $|I|$. Namely the number p_{max} which appears in the runtime is encoded by $\log_2(p_{max})$ bits in the input, thus is of order at most $\mathcal{O}(n)$. However the actual value can be of exponential size. As such, we say that this Dynamic Programming provides a *pseudo-polynomial time algorithm.*

2.1.2 FPTAS via profit rounding

Algorithm 3 FPTAS-KNAPSACK$(\mathcal{S}, B, \epsilon)$

$k \leftarrow \max\{1, \lfloor \frac{\epsilon}{n} p_{max} \rfloor\}$ ▷ Choice of k to be justified later
for $i \in \{1, \ldots, n\}$ **do**
  $\texttt{profit}'(i) = \lfloor \frac{\texttt{profit}(i)}{k} \rfloor$ ▷ Round and scale the profits
end for
Run DP in Section 2.1.1 with B, $\texttt{size}(i)$, and re-scaled $\texttt{profit}'(i)$.
return Items selected by DP

FPTAS-KNAPSACK pre-processes the problem input by rounding down to the nearest multiple of k and then, since every value is now a multiple of k, scaling down by a factor of k. FPTAS-KNAPSACK then calls the DP algorithm described in Section 2.1.1. Since we scaled down the profits, the new maximum profit is $\frac{p_{max}}{k}$, hence the DP now runs in $\mathcal{O}(\frac{n^2 p_{max}}{k})$.

To obtain a FPTAS, we pick $k = \max\{1, \lfloor \frac{\epsilon p_{max}}{n} \rfloor\}$ so that FPTAS-KNAPSACK is a $(1-\epsilon)$-approximation algorithm and runs in $\text{poly}(n, \frac{1}{\epsilon})$.

Theorem 2.4. FPTAS-KNAPSACK *is a FPTAS for the knapsack problem.*

Proof. Suppose we are given a knapsack instance $I = (\mathcal{S}, B)$. Let $\texttt{loss}(i)$ denote the decrease in value by using rounded $\texttt{profit}'(i)$ for item i. By the profit rounding definition, for each item i,

$$\texttt{loss}(i) = \texttt{profit}(i) - k \cdot \lfloor \frac{\texttt{profit}(i)}{k} \rfloor \leq k$$

Then, over all n items,

$$\begin{aligned}
\sum_{i=1}^{n} \texttt{loss}(i) &\leq nk && \texttt{loss}(i) \leq k \text{ for any item } i \\
&< \epsilon \cdot p_{max} && \text{Since } k = \lfloor \frac{\epsilon}{n} p_{max} \rfloor \\
&\leq \epsilon \cdot \texttt{profit}(OPT(I)) && \text{Since } p_{max} \leq \texttt{profit}(OPT(I))
\end{aligned}$$

Thus, $\texttt{profit}(\text{FPTAS-KNAPSACK}(I)) \geq (1-\epsilon) \cdot \texttt{profit}(OPT(I))$. Furthermore, FPTAS-KNAPSACK runs in $\mathcal{O}(\frac{n^2 p_{max}}{k}) = \mathcal{O}(\frac{n^3}{\epsilon}) \in \text{poly}(n, \frac{1}{\epsilon})$. □

Remark $k = 1$ when $p_{max} \leq \frac{n}{\epsilon}$. In that case, no rounding occurs and the DP finds the exact solution in $\mathcal{O}(n^2 p_{max}) \in \mathcal{O}(\frac{n^3}{\epsilon}) \in \text{poly}(n, \frac{1}{\epsilon})$ time.

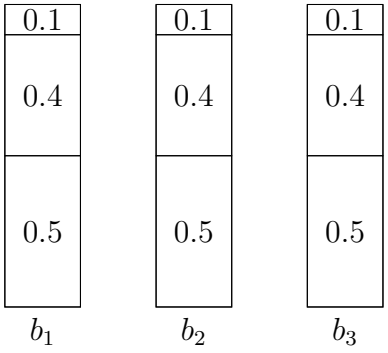
Example Recall the earlier example where budget $B = 10$ and $\mathcal{S} = \{1 : \langle 10, 130\rangle, 2 : \langle 7, 103\rangle, 3 : \langle 6, 91\rangle, 4 : \langle 4, 40\rangle, 5 : \langle 3, 38\rangle\}$. For $\epsilon = \frac{1}{2}$, one would set $k = \max\{1, \lfloor \frac{\epsilon p_{max}}{n} \rfloor\} = \max\{1, \lfloor \frac{130/2}{5} \rfloor\} = 13$. After rounding, we have $\mathcal{S}' = \{1 : \langle 10, 10\rangle, 2 : \langle 7, 7\rangle, 3 : \langle 6, 7\rangle, 4 : \langle 4, 3\rangle, 5 : \langle 3, 2\rangle\}$. The optimum subset from $\mathcal{S}'$ is $\{3 : \langle 6, 7\rangle, 4 : \langle 4, 3\rangle\}$ which translates to a total profit of $91 + 40 = 131$ in the original problem. As expected, $131 = \texttt{profit}(\text{FPTAS-KNAPSACK}(I)) \geq (1 - \frac{1}{2}) \cdot \texttt{profit}(OPT(I)) = 70.5$.

2.2 Bin packing

Definition 2.5 (Bin packing problem)**.** *Given a set $\mathcal{S}$ with n items where each item i has $\texttt{size}(i) \in (0, 1]$, find the minimum number of unit-sized bins (i.e. bins of size 1) that can hold all n items.*

For any problem instance I, let $OPT(I)$ be an optimal bin assignment and $|OPT(I)|$ be the corresponding minimum number of bins required. One can see that $\sum_{i=1}^{n} \texttt{size}(i) \leq |OPT(I)|$.

Example Consider $\mathcal{S} = \{0.5, 0.1, 0.1, 0.1, 0.5, 0.4, 0.5, 0.4, 0.4\}$, where $|\mathcal{S}| = n = 9$. Since $\sum_{i=1}^{n} \texttt{size}(i) = 3$, at least 3 bins are needed. One can verify that 3 bins suffice: $b_1 = b_2 = b_3 = \{0.5, 0.4, 0.1\}$. Hence, $|OPT(\mathcal{S})| = 3$.



2.2.1 First-fit: A 2-approximation algorithm

FirstFit processes items one-by-one, creating new bins if an item cannot fit into one of the existing bins. For a unit-sized bin b, we use $\texttt{size}(b)$ to denote the sum of the `size` of items that are put into b, and define $\texttt{free}(b) = 1 - \texttt{size}(b)$.

Algorithm 4 FirstFit($\mathcal{S}$)

$B \to \emptyset$ ▷ Collection of bins
for $i \in \{1, \ldots, n\}$ **do**
  if $\texttt{size}(i) \leq \texttt{free}(b)$ for some bin $b \in B$. **then**
    Pick the smallest such b.
    $\texttt{free}(b) \leftarrow \texttt{free}(b) - \texttt{size}(i)$ ▷ Put item i into existing bin b
  else
    $B \leftarrow B \cup \{b'\}$ ▷ Put item i into a fresh bin b'
    $\texttt{free}(b') = 1 - \texttt{size}(i)$
  end if
end for
return B

Lemma 2.6. *Using* FirstFit*, at most one bin is less than half-full. That is,* $|\{b \in B : \texttt{size}(b) \leq \frac{1}{2}\}| \leq 1$*, where* B *is the output of* FirstFit*.*

Proof. Suppose, for contradiction, that there are two bins b_i and b_j such that $i < j$, $\texttt{size}(b_i) \leq \frac{1}{2}$ and $\texttt{size}(b_j) \leq \frac{1}{2}$. Then, FirstFit could have put all items in b_j into b_i, and would not have created b_j. This is a contradiction. □

Theorem 2.7. FirstFit *is a 2-approximation algorithm for bin packing.*

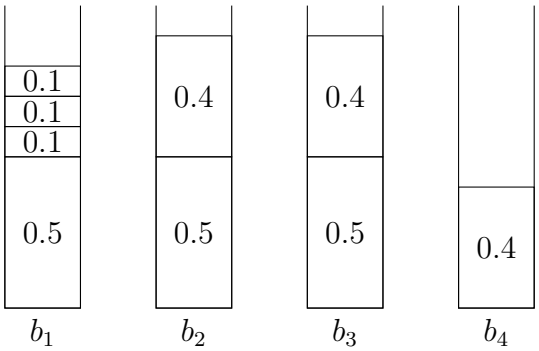
Proof. Suppose FirstFit terminates with $|B| = m$ bins. By Lemma 2.6, $\sum_{i=1}^{n} \texttt{size}(i) > \frac{m-1}{2}$, as $m-1$ bins are at least half-full. Since $\sum_{i=1}^{n} \texttt{size}(i) \leq$

$|OPT(I)|$, we have

$$m - 1 < 2 \cdot \sum_{i=1}^{n} \texttt{size}(i) \leq 2 \cdot |OPT(I)|$$

That is, $m \leq 2 \cdot |OPT(I)|$ since both m and $|OPT(I)|$ are integers. □

Recall example with $\mathcal{S} = \{0.5, 0.1, 0.1, 0.1, 0.5, 0.4, 0.5, 0.4, 0.4\}$. FIRSTFIT will use 4 bins: $b_1 = \{0.5, 0.1, 0.1, 0.1\}, b_2 = b_3 = \{0.5, 0.4\}, b_4 = \{0.4\}$. As expected, $4 = |\text{FIRSTFIT}(\mathcal{S})| \leq 2 \cdot |OPT(\mathcal{S})| = 6$.



Remark If we first sort the item weights in non-increasing order, then one can show that running FIRSTFIT on the sorted item weights will yield a $\frac{3}{2}$-approximation algorithm for bin packing. See footnote for details[1].

It is natural to wonder whether we can do better than a $\frac{3}{2}$-approximation. Unfortunately, unless $\mathcal{P} = \mathcal{NP}$, we cannot do so efficiently. To prove this, we show that if we can efficiently derive a $(\frac{3}{2} - \epsilon)$-approximation for bin packing, then the partition problem (which is $\mathcal{NP}$-hard) can be solved efficiently.

Definition 2.8 (Partition problem)**.** *Given a multiset $\mathcal{S}$ of (possibly repeated) positive integers $x_1, \ldots, x_n$, is there a way to partition $\mathcal{S}$ into $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\sum_{x \in \mathcal{S}_1} x = \sum_{x \in \mathcal{S}_2} x$?*

Theorem 2.9. *It is $\mathcal{NP}$-hard to solve bin packing with an approximation factor better than $\frac{3}{2}$.*

[1] Curious readers can read the following lecture notes for proof on First-Fit-Decreasing:
http://ac.informatik.uni-freiburg.de/lak_teaching/ws11_12/combopt/notes/bin_packing.pdf
https://dcg.epfl.ch/files/content/sites/dcg/files/courses/2012%20-%20Combinatorial%20Optimization/12-BinPacking.pdf

Proof. Suppose some polytime algorithm $\mathcal{A}$ solves bin packing with a $(\frac{3}{2} - \epsilon)$-approximation for $\epsilon > 0$. Given an instance of the partition problem with $\mathcal{S} = \{x_1, \ldots, x_n\}$, let $X = \sum_{i=1}^{n} x_i$. Define a bin packing instance $\mathcal{S}' = \{\frac{2x_1}{X}, \ldots, \frac{2x_n}{X}\}$. Since $\sum_{x \in \mathcal{S}'} x = 2$, at least two bins are required. By construction, one can bipartition $\mathcal{S}$ if and only if only two bins are required to pack $\mathcal{S}'$. Since $\mathcal{A}$ gives a $(\frac{3}{2} - \epsilon)$-approximation, if OPT on $\mathcal{S}'$ returns 2 bins, then $\mathcal{A}$ on $\mathcal{S}'$ will return also $\lfloor 2 \cdot (\frac{3}{2} - \epsilon) \rfloor = 2$ bins. Therefore, as $\mathcal{A}$ solves bin-packing with a $(\frac{3}{2} - \epsilon)$-approximation in polytime, we would get an algorithm for solving the partition problem in polytime. Contradiction. □

The above rules out the possibility of a proper PTAS for bin packing as a $1 + \varepsilon$ approximation of 2, for $\varepsilon < 0.5$, would be strictly less than 3. Another way to view this negative result is to say that we need to allow the approximation algorithm to have at least an additive +1 loss, in comparison to the optimum. But we can still aim for an approximation that is within a $1+\varepsilon$ factor of optimum modulo an additive +1 error, i.e., achieving a number of bins that is at most $OPT(1 + \varepsilon) + 1$.

In the following sections, we work towards this goal, with a runtime that is exponential in $\frac{1}{\epsilon}$. To do this, we first consider two simplifying assumptions and design algorithms for them. Then, we see how to adapt the algorithm and remove these two assumptions.

2.2.2 Special case 1: Exact solving with $\mathcal{A}_\epsilon$

In this section, we make the following two assumptions:

Assumption (1) All items have at least size ϵ, for some $\epsilon > 0$.

Assumption (2) There are only k different possible sizes (k is a constant).

Define $M = \lceil \frac{1}{\epsilon} \rceil$. By assumption 1, there are at most M items in a bin. In addition, define $R = \binom{M+k}{M}$. By assumption 2, there are at most R items arrangements in one bin. Since at most n bins are needed, the total number of bin configurations is at most $\binom{n+R}{R} \leq (n + R)^R = \mathcal{O}(n^R)$. Since k and ϵ are constant, R is also constant and one can enumerate over all possible bin configurations (denote this algorithm as $\mathcal{A}_\epsilon$) to *exactly* solve bin packing, in this special case, in $\mathcal{O}(n^R) \in \text{poly}(n)$ time.

Remark 1 The number of configurations are computed by solving combinatorics problems of the following form: If x_i defines the number of items of the i^{th} possible size, how many non-negative integer solutions are there to $x_1 + \cdots + x_k \leq M$? This type of problems can be solved by counting how

many ways there are to put n indistinguishable balls into k distinguishable bins and is generally known under *stars and bars*.[2]

Remark 2 The number of bin configurations is computed out of n bins (i.e., 1 bin for each item). One may use less than n bins, but this upper bound suffices for our purposes.

2.2.3 Special case 2: PTAS

In this section, we remove the second assumption and require only:

Assumption (1) All items have at least size ϵ, for some $\epsilon > 0$.

Our goal is to reuse the exact algorithm $\mathcal{A}_\epsilon$ on a slightly modified problem instance J that satisfies both assumptions. For this, we partition the items into k non-overlapping groups of $Q \leq \lfloor n\epsilon^2 \rfloor$ elements each. To obtain a constant number of different sizes, we round the sizes of all items in a group to the largest size of that group, resulting in at most k different item sizes. We can now call $\mathcal{A}_\epsilon$ on J to solve the modified instance exactly in polynomial time. Since J only *rounds up* sizes, $\mathcal{A}_\epsilon(J)$ will yield a satisfying bin assignment for instance I, with possibly "spare slack". The entire procedure is described in PTAS-BinPacking.

Algorithm 5 PTAS-BinPacking$(I = \mathcal{S}, \epsilon)$

$k \leftarrow \lceil \frac{1}{\epsilon^2} \rceil$
$Q \leftarrow \lfloor n\epsilon^2 \rfloor$
Partition n items into k non-overlapping groups, each with $\leq Q$ items
for $i \in \{1, \ldots, k\}$ **do**
  $i_{max} \leftarrow \max_{\text{item } j \text{ in group } i} \texttt{size}(j)$
  for item j in group i **do**
    $\texttt{size}(j) \leftarrow i_{max}$
  end for
end for
Denote the modified instance as J
return $\mathcal{A}_\epsilon(J)$

It remains to show that the solution to the modified instance $OPT(J)$ yields a $(1+\epsilon)$-approximation of $OPT(I)$. For this, consider another modified instance J' that is defined analogously to J only with *rounded down* item

[2] See slides 22 and 23 of http://www.cs.ucr.edu/~neal/2006/cs260/piyush.pdf for illustration of $\binom{M+k}{M}$ and $\binom{n+R}{R}$.

sizes. Thus, since we rounded down item sizes in J', we have $|OPT(J')| \leq |OPT(I)|$.

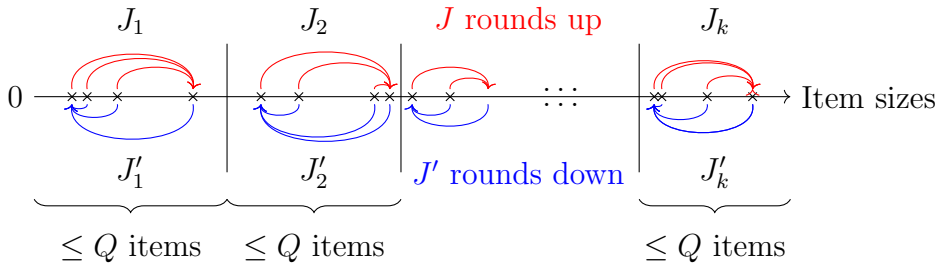


Figure 2.1: Partition items into k groups, each with $\leq Q$ items; Label groups in ascending sizes; J rounds up item sizes, J' rounds down item sizes.

Lemma 2.10. $|OPT(J)| \leq |OPT(J')| + Q$

Proof. Label the k groups in J by $J_1, \ldots, J_k$ where the items in J_i have smaller sizes than the items in J_{i+1}. Label the k groups in J' similarly. See Figure 2.1. For $i = \{1, \ldots, k-1\}$, since the smallest item in J'_{i+1} has size at least as large as the largest item in J_i, any valid packing for J'_i serves as a valid packing for the J_{i-1}. For J_k (the largest $\leq Q$ items of J), we use separate bins for each of its items (hence the additive Q term). □

Lemma 2.11. $|OPT(J)| \leq |OPT(I)| + Q$

Proof. By Lemma 2.10 and the fact that $|OPT(J')| \leq |OPT(I)|$. □

Theorem 2.12. PTAS-BinPacking *is an* $(1+\epsilon)$*-approximation algorithm for bin packing with assumption (1).*

Proof. By Assumption (1), all item sizes are at least ϵ, so $|OPT(I)| \geq n\epsilon$. Then, $Q = \lfloor n\epsilon^2 \rfloor \leq \epsilon \cdot |OPT(I)|$. Apply Lemma 2.11. □

2.2.4 General case

We now consider the general case where we do not make any assumptions on the problem instance I. First, we lower bound the minimum item size by putting aside all items with size smaller than $\min\{\frac{1}{2}, \frac{\epsilon}{2}\}$, thus allowing us to use PTAS-BinPacking. Then, we add back the small items in a greedy manner with FirstFit to complete the packing.

Theorem 2.13. Full-PTAS-BinPacking *uses* $\leq (1+\epsilon)|OPT(I)|+1$ *bins*

Algorithm 6 FULL-PTAS-BINPACKING$(I = \mathcal{S}, \epsilon)$

$\epsilon' \leftarrow \min\{\frac{1}{2}, \frac{\epsilon}{2}\}$ ▷ See analysis why we chose such an ϵ'
$X \leftarrow$ Items with size $< \epsilon'$ ▷ Ignore small items
$P \leftarrow$ PTAS-BINPACKING$(\mathcal{S} \setminus X, \epsilon')$ ▷ By Theorem 2.12, ▷ $|P| = (1+\epsilon') \cdot |OPT(\mathcal{S} \setminus X)|$
$P' \leftarrow$ Using FIRSTFIT, add items in X to P ▷ Handle small items
return Resultant packing P'

Proof. If FIRSTFIT does not open a new bin, the theorem trivially holds. Suppose FIRSTFIT opens a new bin (using m bins in total), then we know that at least $(m-1)$ bins are strictly more than $(1-\epsilon')$-full.

$$\begin{aligned} |OPT(I)| &\geq \sum_{i=1}^{n} \texttt{size}(i) && \text{Lower bound on } |OPT(I)| \\ &> (m-1)(1-\epsilon') && \text{From above observation} \end{aligned}$$

Hence,

$$\begin{aligned} m &< \frac{|OPT(I)|}{1-\epsilon'} + 1 && \text{Rearranging} \\ &< |OPT(I)| \cdot (1+2\epsilon') + 1 && \text{Since } \frac{1}{1-\epsilon'} \leq 1+2\epsilon', \text{ for } \epsilon' \leq \frac{1}{2} \\ &\leq (1+\epsilon) \cdot |OPT(I)| + 1 && \text{By choice of } \epsilon' = \min\{\frac{1}{2}, \frac{\epsilon}{2}\} \end{aligned}$$

□

2.3 Minimum makespan scheduling

Definition 2.14 (Minimum makespan scheduling problem)**.** *Given n jobs, let $I = \{p_1, \ldots, p_n\}$ be the set of processing times, where job i takes p_i units of time to complete. Find an assignment for the n jobs to m identical machines such that the completion time (i.e. makespan) is minimized.*

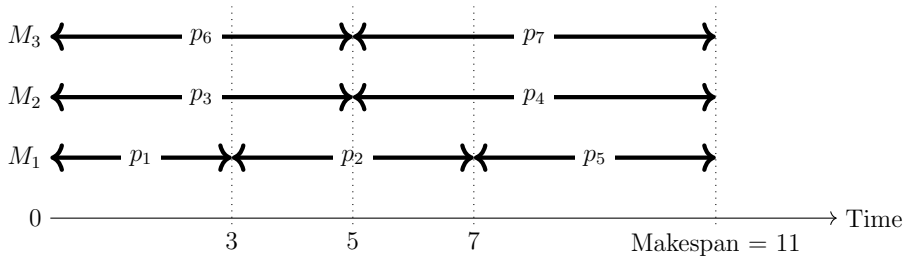
For any problem instance I, let $OPT(I)$ be an optimal job assignment and $|OPT(I)|$ be the corresponding makespan. One can see that:

- $p_{max} = \max_{i \in \{1,\ldots,n\}} p_i \leq |OPT(I)|$
- $\frac{1}{m} \sum_{i=1}^{n} p_i \leq |OPT(I)|$

Denote $L(I) = \max\{p_{max}, \frac{1}{m}\sum_{i=1}^{n} p_i\}$ as the larger lower bound. Then, $L(I) \leq |OPT(I)|$.

Remark To prove approximation factors, it is often useful to relate to lower bounds of $|OPT(I)|$.

Example Suppose we have 7 jobs with processing times $I = \{p_1 = 3, p_2 = 4, p_3 = 5, p_4 = 6, p_5 = 4, p_6 = 5, p_7 = 6\}$ and $m = 3$ machines. Then, the lower bound on the makespan is $L(I) = \max\{6, 11\} = 11$. This is achievable by allocating $M_1 = \{p_1, p_2, p_5\}$, $M_2 = \{p_3, p_4\}$, $M_3 = \{p_6, p_7\}$.



Graham [Gra66] is a 2-approximation greedy algorithm for the minimum makespan scheduling problem. With slight modifications, we improve it to ModifiedGraham, a $\frac{4}{3}$-approximation algorithm. Finally, we end off the section with a PTAS for minimum makespan scheduling.

2.3.1 Greedy approximation algorithms

Algorithm 7 Graham($I = \{p_1, \ldots, p_n\}, m$)

```
M_1, ..., M_m ← ∅                          ▷ All machines are initially free
for i ∈ {1, ..., n} do
    j ← argmin_{j∈{1,...,m}} Σ_{p∈M_j} p   ▷ Pick the least loaded machine
    M_j ← M_j ∪ {p_i}                      ▷ Add job i to this machine
end for
return M_1, ..., M_m
```

Theorem 2.15. Graham *is a 2-approximation algorithm.*

Proof. Suppose the last job that finishes (which takes p_{last} time) running was assigned to machine M_j. Define $t = (\sum_{p \in M_j} p) - p_{\text{last}}$ as the makespan of machine M_j before the last job was assigned to it. That is,

$$|\text{Graham}(I)| = t + p_{\text{last}}$$

As GRAHAM assigns greedily to the least loaded machine, all machines take at least t time, hence

$$t \leq \frac{1}{m} \sum_{i=1}^{n} p_i \leq |OPT(I)|.$$

as $\frac{1}{m} \sum_{i=1}^{n} p_i$ is the average of work done on each machine. Since $p_{\text{last}} \leq p_{max} \leq |OPT(I)|$, we have $|\text{GRAHAM}(I)| = t + p_{\text{last}} \leq 2 \cdot |OPT(I)|$. □

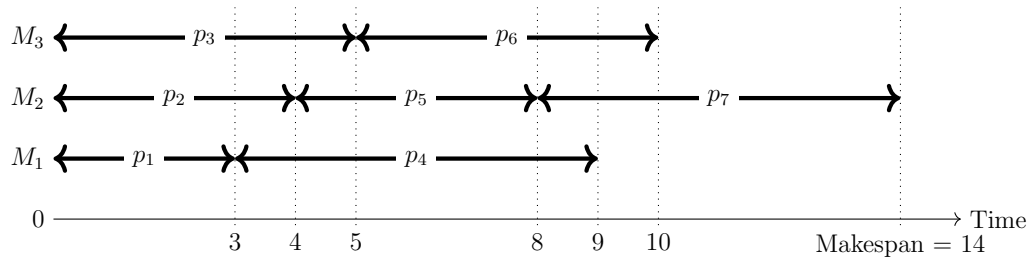
Corollary 2.16. *$|OPT(I)| \leq 2 \cdot L(I)$, where $L(I) = \max\{p_{max}, \frac{1}{m} \sum_{i=1}^{n} p_i\}$.*

Proof. From the proof of Theorem 2.15, we have $|\text{GRAHAM}(I)| = t + p_{\text{last}}$ and $t \leq \frac{1}{m} \sum_{i=1}^{n} p_i$. Since $|OPT(I)| \leq |\text{GRAHAM}(I)|$ and $p_{\text{last}} \leq p_{max}$, we have

$$|OPT(I)| \leq \frac{1}{m} \sum_{i=1}^{n} p_i + p_{max} \leq 2 \cdot L(I)$$

□

Recall the example with $I = \{p_1 = 3, p_2 = 4, p_3 = 5, p_4 = 6, p_5 = 4, p_6 = 5, p_7 = 6\}$ and $m = 3$. GRAHAM will schedule $M_1 = \{p_1, p_4\}$, $M_2 = \{p_2, p_5, p_7\}$, $M_3 = \{p_3, p_6\}$, yielding a makespan of 14. As expected, $14 = |\text{GRAHAM}(I)| \leq 2 \cdot |OPT(I)| = 22$.



Remark The approximation for GRAHAM is loose because we have no guarantees on p_{last} beyond $p_{\text{last}} \leq p_{max}$. This motivates us to order the job timings in descending order (see MODIFIEDGRAHAM).

Algorithm 8 MODIFIEDGRAHAM($I = \{p_1, \ldots, p_n\}, m$)

$I' \leftarrow I$ in descending order
return GRAHAM(I', m)

Let p_{last} be the last job that finishes running. We consider the two cases $p_{\text{last}} > \frac{1}{3} \cdot |OPT(I)|$ and $p_{\text{last}} \leq \frac{1}{3} \cdot |OPT(I)|$ separately for the analysis.

Lemma 2.17. *If* $p_{\text{last}} > \frac{1}{3} \cdot |OPT(I)|$, *then* $|\textsc{ModifiedGraham}(I)| = |OPT(I)|$.

Proof. For $m \geq n$, $|\textsc{ModifiedGraham}(I)| = |OPT(I)|$ by trivially putting one job on each machine. For $m < n$, without loss of generality[3], we can assume that every machine has a job.

Suppose, for a contradiction, that $|\textsc{ModifiedGraham}(I)| > |OPT(I)|$. Then, there exists a sequence of jobs with descending sizes $I = \{p_1, \ldots, p_n\}$ such that the last, smallest job p_n causes $\textsc{ModifiedGraham}(I)$ to have a makespan larger than $OPT(I)$[4]. That is, $|\textsc{ModifiedGraham}(I \setminus \{p_n\})| \leq |OPT(I)|$ and $p_{\text{last}} = p_n$. Let $\mathcal{C}$ be the configuration of machines after ModifiedGraham assigned $\{p_1, \ldots, p_{n-1}\}$.

Observation 1 In $\mathcal{C}$, each machine has either 1 or 2 jobs.

If there exists machine M_i with ≥ 3 jobs, M_i will take $> |OPT(I)|$ time because all jobs take $> \frac{1}{3} \cdot |OPT(I)|$ time. This contradicts the assumption $|\textsc{ModifiedGraham}(I \setminus \{p_n\})| \leq |OPT(I)|$.

Let us denote the jobs that are alone in $\mathcal{C}$ as *heavy jobs*, and the machines they are on as *heavy machines*.

Observation 2 In $OPT(I)$, all heavy jobs are alone.

By assumption on p_n, we know that assigning p_n to any machine (in particular, the heavy machines) in $\mathcal{C}$ causes the makespan to exceed $|OPT(I)|$. Since p_n is the smallest job, no other job can be assigned to the heavy machines otherwise $|OPT(I)|$ cannot attained by $OPT(I)$.

Suppose there are k heavy jobs occupying a machine each in $OPT(I)$. Then, there are $2(m-k)+1$ jobs (two non-heavy jobs per machine in $\mathcal{C}$, and p_n) to be distributed across $m-k$ machines. By the pigeonhole principle, at least one machine M^* will get ≥ 3 jobs in $OPT(I)$. However, since the smallest job p_n takes $> \frac{1}{3} \cdot |OPT(I)|$ time, M^* will spend $> |OPT(I)|$ time. This is a contradiction. □

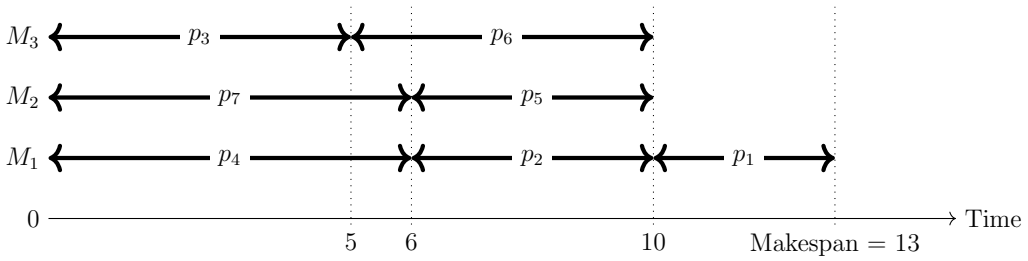
Theorem 2.18. ModifiedGraham *is a* $\frac{4}{3}$*-approximation algorithm.*

Proof. By similar arguments as per Theorem 2.15, $|\textsc{ModifiedGraham}(I)| = t + p_{\text{last}} \leq \frac{4}{3} \cdot |OPT(I)|$ when $p_{\text{last}} \leq \frac{1}{3} \cdot |OPT(I)|$. Meanwhile, when $p_{\text{last}} > \frac{1}{3} \cdot |OPT(I)|$, $|\textsc{ModifiedGraham}(I)| = |OPT(I)|$ by Lemma 2.17. □

[3]Suppose there is a machine M_i without a job, then there must be another machine M_j with more than 1 job (by pigeonhole principle). Shifting one of the jobs from M_j to M_i will not increase the makespan.

[4]If adding p_j for some $j < n$ already causes $|\textsc{ModifiedGraham}(\{p_1, \ldots, p_j\})| > |OPT(I)|$, we can truncate I to $\{p_1, \ldots, p_j\}$ so that $p_{\text{last}} = p_j$. Since $p_j \geq p_n > \frac{1}{3} \cdot |OPT(I)|$, the antecedent still holds.

Recall the example with $I = \{p_1 = 3, p_2 = 4, p_3 = 5, p_4 = 6, p_5 = 4, p_6 = 5, p_7 = 6\}$ and $m = 3$. Putting I in decreasing sizes, $I' = \langle p_4 = 6, p_7 = 6, p_3 = 5, p_6 = 5, p_2 = 4, p_5 = 4, p_1 = 3\rangle$ and ModifiedGraham will schedule $M_1 = \{p_4, p_2, p_1\}$, $M_2 = \{p_7, p_5\}$, $M_3 = \{p_3, p_6\}$, yielding a makespan of 13. As expected, $13 = |\text{ModifiedGraham}(I)| \leq \frac{4}{3} \cdot |OPT(I)| = 14.666\ldots$



2.3.2 PTAS for minimum makespan scheduling

Recall that any makespan scheduling instance (I, m) has a lower bound $L(I) = \max\{p_{max}, \frac{1}{m}\sum_{i=1}^{n} p_i\}$. From Corollary 2.16, we know that $|OPT(I)| \in [L(I), 2L(I)]$. Let $\text{Bin}(I, t)$ be the minimum number of bins of size t that can hold all jobs. By associating job processing times with item sizes, and scaling bin sizes up by a factor of t, we can relate $\text{Bin}(I, t)$ to the bin packing problem. One can see that $\text{Bin}(I, t)$ is monotonically decreasing in t and $|OPT(I)|$ is the minimum t such that $\text{Bin}(I, t) = m$. Hence, to get a $(1+\epsilon)$-approximate schedule, it suffices to find a $t \leq (1 + \epsilon) \cdot |OPT(I)|$ such that $\text{Bin}(I, t) \leq m$.

Given t, PTAS-Makespan transforms a makespan scheduling instance into a bin packing instance, then solves for an approximate bin packing to yield an approximate scheduling. By ignoring small jobs (jobs of size $\leq \epsilon t$) and rounding job sizes down to the closest power of $(1 + \epsilon) : t\epsilon \cdot \{1, (1 + \epsilon), \ldots, (1+\epsilon)^h = \epsilon^{-1}\}$, exact bin packing $\mathcal{A}_\epsilon$ with size t bins is used yielding a packing P. To get a bin packing for the original job sizes, PTAS-Makespan follows P's bin packing but uses bins of size $t(1+\epsilon)$ to account for the rounded down job sizes. Suppose jobs 1 and 2 with sizes p_1 and p_2 were rounded down to p'_1 and p'_2, and P assigns them to a same bin (i.e., $p'_1 + p'_2 \leq t$). Then, due to the rounding process, their original sizes should also fit into a size $t(1+\epsilon)$ bin since $p_1 + p_2 \leq p'_1(1+\epsilon) + p'_2(1+\epsilon) \leq t(1+\epsilon)$. Finally, small jobs are handled using FirstFit. Let $\alpha(I, t, \epsilon)$ be the final bin configuration produced by PTAS-Makespan on parameter t and $|\alpha(I, t, \epsilon)|$ be the number of bins used. Since $|OPT(I)| \in [L, 2L]$, there will be a $t \in \{L, L + \epsilon L, L + 2\epsilon L, \ldots, 2L\}$ such that $|\alpha(I, t, \epsilon)| \leq \text{Bin}(I, t) \leq m$ bins (see Lemma 2.19 for the first

Algorithm 9 PTAS-MAKESPAN($I = \{p_1, \ldots, p_n\}, m$)

$L = \max\{p_{max}, \frac{1}{m}\sum_{i=1}^{n} p_i\}$
for $t \in \{L, L + \epsilon L, L + 2\epsilon L, L + 3\epsilon L, \ldots, 2L\}$ **do**
  $I' \leftarrow I \setminus \{\text{Jobs with sizes} \leq \epsilon t\} := I \setminus X$ ▷ Ignore small jobs
  $h \leftarrow \lceil \log_{1+\epsilon}(\frac{1}{\epsilon}) \rceil$ ▷ To partition $(\epsilon t, t]$ into powers of $(1 + \epsilon)$
  for $p_i \in I'$ **do**
    $k \leftarrow$ Largest $j \in \{0, \ldots, h\}$ such that $p_i \geq t\epsilon(1 + \epsilon)^j$
    $p_i \leftarrow t\epsilon(1 + \epsilon)^k$ ▷ Round down job size
  end for
  $P \leftarrow \mathcal{A}_\epsilon(I')$ ▷ Use $\mathcal{A}_\epsilon$ from Section 2.2.2 with size t bins
  $\alpha(I, t, \epsilon) \leftarrow$ Use bins of size $t(1 + \epsilon)$ to emulate P on original sizes
  $\alpha(I, t, \epsilon) \leftarrow$ Using FIRSTFIT, add items in X to $\alpha(I, t, \epsilon)$
  if $\alpha(I, t, \epsilon)$ uses $\leq m$ bins **then**
    return Assign jobs to machines according to $\alpha(I, t, \epsilon)$
  end if
end for

inequality). Note that running binary search on t also works, but we only care about poly-time.

Lemma 2.19. *For any $t > 0$, $|\alpha(I, t, \epsilon)| \leq \text{Bin}(I, t)$.*

Proof. If FIRSTFIT does not open a new bin, then $|\alpha(I, t, \epsilon)| \leq \text{Bin}(I, t)$ since $\alpha(I, t, \epsilon)$ uses an additional $(1 + \epsilon)$ buffer on each bin. If FIRSTFIT opens a new bin (say, totalling b bins), then there are at least $(b - 1)$ produced bins from $\mathcal{A}_\epsilon$ (exact solving on rounded down items of size $> \epsilon t$) that are more than $(t(1 + \epsilon) - \epsilon t) = t$-full. Hence, *any* bin packing algorithm must use strictly more than $\frac{(b-1)t}{t} = b - 1$ bins. In particular, $\text{Bin}(I, t) \geq b = |\alpha(I, t, \epsilon)|$. □

Theorem 2.20. PTAS-MAKESPAN *is a $(1+\epsilon)$-approximation for the minimum makespan scheduling problem.*

Proof. Let $t^* = |OPT(I)|$ and t_α be the minimum $t \in \{L, L + \epsilon L, L + 2\epsilon L, \ldots, 2L\}$ such that $|\alpha(I, t, \epsilon)| \leq m$. It follows that $t_\alpha \leq t^* + \epsilon L$. Since $L \leq |OPT(I)|$ and since we consider bins of final size $t_\alpha(1+\epsilon)$ to accomodate for the original sizes, we have $|\text{PTAS-MAKESPAN}(I)| = t_\alpha(1 + \epsilon) \leq (t^* + \epsilon L)(1 + \epsilon) \leq (1 + \epsilon)^2 \cdot |OPT(I)|$. For $\epsilon \in [0, 1]$ we have $(1 + \epsilon)^2 \leq (1 + 3\epsilon)$ and thus the statement follows. □

Theorem 2.21. PTAS-MAKESPAN *runs in poly$(|I|, m)$ time.*

Proof. There are at most $\max\{\frac{p_{max}}{\epsilon}, \frac{1}{m\epsilon}\sum_{i=1}^{n} p_i\} \in \mathcal{O}(\frac{1}{\epsilon})$ values of t to try. Filtering small jobs and rounding remaining jobs takes $\mathcal{O}(n)$. From the previous lecture, $\mathcal{A}_\epsilon$ runs in $\mathcal{O}(\frac{1}{\epsilon} \cdot n^{\frac{h+1}{\epsilon}})$ and FIRSTFIT runs in $\mathcal{O}(nm)$. □

Chapter 3

Randomized approximation schemes

In this chapter, we study the class of algorithms which extends FPTAS by allowing randomization.

Definition 3.1 (Fully polynomial randomized approximation scheme (FPRAS))**.** *For cost metric c, an algorithm $\mathcal{A}$ is a FPRAS if*

- *For any $\epsilon > 0$,* $\Pr[|c(\mathcal{A}(I)) - c(OPT(I))| \leq \epsilon \cdot c(OPT(I))] \geq \frac{3}{4}$
- $\mathcal{A}$ *runs in* $\text{poly}(|I|, \frac{1}{\epsilon})$

Intuition An FPRAS computes, with a high enough probability, a solution which is not too far from the optimal one in a reasonable time.

Remark The probability $\frac{3}{4}$ above is somewhat arbitrary. We can then easily amplify the success probability. In particular, for any desired $\delta > 0$, we can invoke $\mathcal{O}(\log \frac{1}{\delta})$ independent copies of the algorithm $\mathcal{A}$ and then return the median. The median is a correct estimation with probability greater than $1 - \delta$. This is known as *probability amplification* (see section 9.1).

3.1 DNF counting

Definition 3.2 (Disjunctive Normal Form (DNF))**.** *A formula F on n Boolean variables $x_1, \ldots, x_n$ is said to be in a Disjunctive Normal Form (DNF) if*

- *$F = C_1 \vee \cdots \vee C_m$ is a disjunction (that is, logical OR) of clauses*

- $\forall i \in [m]$, *a clause* $C_i = l_{i,1} \wedge \cdots \wedge l_{i,|C_i|}$ *is a conjunction (that is, logical AND) of literals*
- $\forall i \in [n]$, *a literal* $l_i \in \{x_i, \neg x_i\}$ *is either the variable* x_i *or its negation.*

Let $\alpha : [n] \to \{0,1\}$ *be a truth assignment to the* n *variables. Formula* F *is said to be* satisfiable *if there exists a* satisfying assignment α *such that* F *evaluates to true under* α *(i.e.* $F[\alpha] = 1$*).*

Any clause with both x_i and $\neg x_i$ is trivially false. As they can be removed in a single scan of F, we assume that F does not contain such trivial clauses.

Example Let $F = (x_1 \wedge \neg x_2 \wedge \neg x_4) \vee (x_2 \wedge x_3) \vee (\neg x_3 \wedge \neg x_4)$ be a Boolean formula on 4 variables, where $C_1 = x_1 \wedge \neg x_2 \wedge \neg x_4$, $C_2 = x_2 \wedge x_3$ and $C_3 = \neg x_3 \wedge \neg x_4$. Drawing the truth table, one sees that there are 9 satisfying assignments to F, one of which is $\alpha(1) = 1, \alpha(2) = \alpha(3) = \alpha(4) = 0$.

Remark Another common normal form for representing Boolean formulas is the *Conjunctive Normal Form* (CNF). Formulas in CNF are conjunctions of disjunctions (as compared to disjunctions of conjunctions in DNF). In particular, one can determine in polynomial time whether a DNF formula is satisfiable but it is $\mathcal{NP}$-complete to determine if a CNF formula is satisfiable.

In this section, we are interested in the number of satisfying assignment for each given DNF. Suppose F is a Boolean formula in DNF. Let $f(F) = |\{\alpha : F[\alpha] = 1\}|$ be the number of satisfying assignments to F. If we let $S_i = \{\alpha : C_i[\alpha] = 1\}$ be the set of satisfying assignments to clause C_i, then we see that $f(F) = |\bigcup_{i=1}^{m} S_i|$. We are interested in polynomial-time algorithms for computing or approximating $|f(F)|$. In the above example, $|S_1| = 2$, $|S_2| = 4$, $|S_3| = 4$, and $f(F) = 9$.

In the following, we present two failed attempts to compute $f(F)$ and then present DNF-Count, a FPRAS for DNF counting via sampling.

3.1.1 Failed attempt 1: Principle of Inclusion-Exclusion

By definition of $f(F) = |\bigcup_{i=1}^{m} S_i|$, one may be tempted to apply the Principle of Inclusion-Exclusion to expand:

$$|\bigcup_{i=1}^{m} S_i| = \sum_{i=1}^{m} |S_i| - \sum_{i<j} |S_i \cap S_j| + \dots$$

However, there are exponentially many terms and there exist instances where truncating the sum yields arbitrarily bad approximation.

3.1.2 Failed attempt 2: Sampling (wrongly)

Suppose we pick k assignments uniformly at random (u.a.r.). Let X_i be the indicator variable whether the i-th assignment satisfies F, and $X = \sum_{i=1}^{k} X_i$ be the total number of satisfying assignments out of the k sampled assignments. A u.a.r. assignment is satisfying with probability $\frac{f(F)}{2^n}$. By linearity of expectation, $\mathbb{E}(X) = k \cdot \frac{f(F)}{2^n}$. Unfortunately, since we only sample $k \in \text{poly}(n, \frac{1}{\epsilon})$ assignments, and as $\frac{k}{2^n}$ can be exponentially small, it can be quite likely (e.g., probability much larger than $1 - 1/\text{poly}(n)$) that none of our sampled assignments is a satisfying assignment: in such a case, we cannot infer much about the number of satisfying assignments using only $\text{poly}(n)$ samples. We would need exponentially many samples for $\mathbb{E}(X)$ to be a good estimate of $f(F)$. Thus, this approach will *not* yield a FPRAS for DNF counting.

3.1.3 An FPRAS for DNF counting via sampling

Consider an m-by-$f(F)$ boolean matrix M where

$$M[i,j] = \begin{cases} 1 & \text{if assignment } \alpha_j \text{ satisfies clause } C_i \\ 0 & \text{otherwise} \end{cases}$$

Remark We are trying to estimate $f(F)$ and thus will never be able to build the matrix M. It is used here as an explanation of why this attempt works.

	α_1	α_2	$\ldots$	$\alpha_{f(F)}$
C_1	0	1	$\ldots$	0
C_2	1	1	$\ldots$	1
C_3	0	0	$\ldots$	0
$\ldots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_m	0	1	$\ldots$	1

Table 3.1: Visual representation of the matrix M. Red 1's indicate the topmost clause C_i satisfied for each assignment α_j

Let $|M|$ denote the total number of 1's in M; it is the sum of the number of clauses satisfied by each assignment that satisfies F. Recall that S_i is the set of assignments that satisfy C_i. Since $|S_i| = 2^{n-|C_i|}$, $|M| = \sum_{i=1}^{m} |S_i| = \sum_{i=1}^{m} 2^{n-|C_i|}$.

We are now interested in the number of "topmost" 1's in the matrix, where "topmost" is defined column-wise. As every column represents a satisfying assignment, at least one clause must be satisfied for each assignment and this proves that there are exactly $f(F)$ "topmost" 1's in the matrix M (i.e. one by column).

DNF-Count estimates the fraction of "topmost" 1's in M, then returns this fraction times $|M|$ as an estimate of $f(F)$.

To estimate the fraction of "topmost" 1's:

- Pick a clause according to its length: shorter clauses are more likely.
- Uniformly select a satisfying assignment for the picked clause by flipping coins for variables not in the clause.
- Check if the assignment satisfies any clauses with a smaller index.

Algorithm 10 DNF-Count(F, ϵ)

```
X ← 0                                   ▷ Empirical number of "topmost" 1's sampled
for k = 9m/ε² times do
    C_i ← Sample one of m clauses, where Pr[C_i chosen] = 2^{n−|C_i|} / |M|
    α_j ← Sample one of 2^{n−|C_i|} satisfying assignments of C_i
    IsTopmost ← True
    for l ∈ {1, ..., i − 1} do          ▷ Check if α_j is "topmost"
        if C_l[α] = 1 then              ▷ Checkable in O(n) time
            IsTopmost ← False
        end if
    end for
    if IsTopmost then
        X ← X + 1
    end if
end for
return |M|·X / k
```

Lemma 3.3. DNF-Count *samples a '1' in the matrix M uniformly at random at each step.*

Proof. Recall that the total number of 1's in M is $|M| = \sum_{i=1}^{m} |S_i| = \sum_{i=1}^{m} 2^{n-|C_i|}$.

$$\begin{aligned}
\Pr[C_i \text{ and } \alpha_j \text{ are chosen}] &= \Pr[C_i \text{ is chosen}] \cdot \Pr[\alpha_j \text{ is chosen} | C_i \text{ is chosen}] \\
&= \frac{2^{n-|C_i|}}{\sum_{i=1}^{m} 2^{n-|C_i|}} \cdot \frac{1}{2^{n-|C_i|}} \\
&= \frac{1}{\sum_{i=1}^{m} 2^{n-|C_i|}} \\
&= \frac{1}{|M|}
\end{aligned}$$

□

Lemma 3.4. *In* DNF-Count, $\Pr\left[\left|\frac{|M| \cdot X}{k} - f(F)\right| \leq \epsilon \cdot f(F)\right] \geq \frac{3}{4}$.

Proof. Let X_i be the indicator variable whether the i-th sampled assignment is "topmost", where $p = \Pr[X_i = 1]$. By Lemma 3.3, $p = \Pr[X_i = 1] = \frac{f(F)}{|M|}$. Let $X = \sum_{i=1}^{k} X_i$ be the empirical number of "topmost" 1's. Then, $\mathbb{E}(X) = kp$ by linearity of expectation. By picking $k = \frac{9m}{\epsilon^2}$,

$$\begin{aligned}
&\Pr\left[\left|\frac{|M| \cdot X}{k} - f(F)\right| \geq \epsilon \cdot f(F)\right] \\
&= \Pr\left[\left|X - \frac{k \cdot f(F)}{|M|}\right| \geq \frac{\epsilon \cdot k \cdot f(F)}{|M|}\right] && \text{Multiply by } \frac{k}{|M|} \\
&= \Pr\left[|X - kp| \geq \epsilon kp\right] && \text{Since } p = \frac{f(F)}{|M|} \\
&\leq 2\exp\left(-\frac{\epsilon^2 kp}{3}\right) && \text{By Chernoff bound} \\
&= 2\exp\left(-\frac{3m \cdot f(F)}{|M|}\right) && \text{Since } k = \frac{9m}{\epsilon^2} \text{ and } p = \frac{f(F)}{|M|} \\
&\leq 2\exp(-3) && \text{Since } |M| \leq m \cdot f(F) \\
&\leq \frac{1}{4}
\end{aligned}$$

Negating, we get:

$$\Pr\left[\left|\frac{|M| \cdot X}{k} - f(F)\right| \leq \epsilon \cdot f(F)\right] \geq 1 - \frac{1}{4} = \frac{3}{4}$$

□

Lemma 3.5. DNF-Count *runs in* $poly(F, \frac{1}{\epsilon}) = poly(n, m, \frac{1}{\epsilon})$ *time.*

Proof. There are $k \in \mathcal{O}(\frac{m}{\epsilon^2})$ iterations. In each iteration, we spend $\mathcal{O}(m+n)$ sampling C_i and α_j, and $\mathcal{O}(nm)$ for checking if a sampled α_j is "topmost". In total, DNF-Count runs in $\mathcal{O}(\frac{m^2 n(m+n)}{\epsilon^2})$ time. □

Theorem 3.6. DNF-Count *is a FPRAS for DNF counting.*

Proof. By Lemmas 3.4 and 3.5. □

3.2 Network reliability

Let's consider an undirected graph $G = (V, E)$, where each edge $e \in E$ has a certain probability of failing, namely p_e, independent of other edges. In the *network reliability* problem, we are interested in calculating or estimating the probability that the network becomes disconnected. For simplicity we study the symmetrical case, where $p_e = p$ for every edge $e \in E$.

As a side remark, we note that we are aiming to obtain a $(1 \pm \varepsilon)$-approximation of the quantity $P := \Pr[G \text{ is disconnected}]$. Such an approximation is not necessarily a $(1 \pm \varepsilon)$-approximation of the probability of the converse, $1 - P = \Pr[G \text{ is connected}]$ and vice versa, since P may be very close to 1.

Observation 3.7. *For an edge cut of size k edges, the probability of its disconnection is p^k.*

Reduction to DNF counting: If the graph had only a few cuts, there would be a natural and easy way to formulate the problem as a variant of DNF counting: each edge would be represented by a variable, and every clause would correspond to a cut postulating that all of its edges have failed. The probability of disconnecting can then be inferred from the fraction of satisfying assignments, when each variable is true with probability p and false otherwise. We note that the latter can be computed by an easy extension of the DNF counting algorithm discussed in the previous section.

Unfortunately, there are exponentially many cuts in a general graph and this method is thus inefficient. We discuss two cases based on the minimum cut size c. The reduction we present here is due to Karger [Kar01].

- When $p^c \geq \frac{1}{n^4}$, this means that the probability of the network disconnection is rather high. As mentioned previously, this is not the case of a

large interest, since the motivation behind studying this problem is the understanding how to build reliable networks, and network with rather small cut is not. Nevertheless, since the probability of disconnection is rather high, Monte-Carlo method of sampling subsets of edges and checking whether they are cuts is sufficient, since we only need $\tilde{\mathcal{O}}(n^4)$ samples to achieve concentration.

- When $p^c \leq \frac{1}{n^4}$, we show that the large cuts do not contribute to the probability of disconnection too much, and therefore they can be safely ignored. Recall that the number of cuts of size αc for $\alpha \geq 1$ is at most $\mathcal{O}(n^{2\alpha})$[1]. When one selects a threshold $\gamma = \max\{\mathcal{O}(\log_n \varepsilon^{-1}), \Theta(1)\}$ on the cut size, we can express the contribution of large cuts – those of size γc and higher — to the overall probability as

$$\int_{\gamma}^{\infty} n^{2\alpha} \cdot p^{\alpha c} \, d\alpha < \varepsilon p^c \ .$$

  Hence, the error introduced by ignoring large cuts is at most a ε factor of the lower bound on the probability of failure, i.e., p^c. Thus, we can ignore those large cuts.

  The number of cuts smaller than γc is clearly a polynomial of n and therefore the reduction to DNF-counting is efficient. The only remaining thing is finding those – Karger's contraction algorithm provides us with a way of sampling those cuts and we can thus use Coupon collector to enumerate those [2].

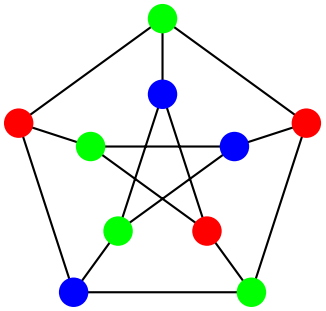
3.3 Counting graph colorings

Definition 3.8 (Graph coloring)**.** *Let $G = (V, E)$ be a graph on $|V| = n$ vertices and $|E| = m$ edges. Denote the maximum degree as Δ. A valid q-coloring of G is an assignment $c : V \to \{1, \ldots, q\}$ such that adjacent vertices have different colors. i.e., If u and v are adjacent in G, then $c(u) \neq c(v)$.*

Example (3-coloring of the Petersen graph)

[1] If you haven't seen this fact before (e.g., in the context of Karger's randomized contraction algorithm in your undergraduate algorithmic classes), take this as a black-box claim for now. You will see the proof later in Chapter 13.

[2] As you might have seen in your undergraduate classes, e.g., in the course Algorithms, Probability, and Computing



For $q \geq \Delta + 1$, one can obtain a valid q-coloring by sequentially coloring a vertex with available colors greedily. In this section, we show a FPRAS for counting $f(G)$, the number of graph coloring of a given graph G, under the assumption that we have $q \geq 2\Delta + 1$ colors.

3.3.1 Sampling a coloring uniformly

When $q \geq 2\Delta + 1$, the Markov chain approach in SampleColor allows us to sample a random coloring in $\mathcal{O}(n \log \frac{n}{\epsilon})$ steps.

Algorithm 11 SampleColor$(G = (V, E), \epsilon)$

Greedily color the graph
for $k = \mathcal{O}(n \log \frac{n}{\epsilon})$ times **do**
  Pick a random vertex v uniformly at random from V
  Pick u.a.r. an available color ▷ Different from the colours in $N(v)$
  Color v with new color ▷ May end up with same color
end for
return Coloring

Claim 3.9. *For $q \geq 2\Delta + 1$, the distribution of colorings returned by* SampleColor *is ϵ-close to a uniform distribution on all valid q-colorings.*

Notes on the Proof. A full proof of this claim is beyond the scope of this course. Let us still provide some helpful explanations: The coloring in the course of the algorithm can be modelled as a Markov chain. Moreover, this chain is aperiodic and irreducible: Firstly, the chain is aperiodic because the color of a vertex can be retained while resampling (since it is not used in any of its neighbours). The chain is irreducible because we can transform between any two colorings using at most $2n$ steps. Since the chain is aperiodic and irreducible, it converges to its stationary distribution. Now, as the chain is clearly symmetric, the stationary distribution is uniform among all possible colorings with q colors. Finally, it is known that the chain is rapidly mixing

and has mixing time at most $\mathcal{O}(n \log n)$. Therefore, after $k = \mathcal{O}(n \log \frac{n}{\epsilon})$ times resamplings, the distribution of the coloring will be ϵ-close to a uniform distribution on all valid q-colorings. □

3.3.2 FPRAS for $q \geq 2\Delta + 1$ and $\Delta \geq 2$

Fix an arbitrary ordering of edges in E. For $i = \{1, \ldots, m\}$, let $G_i = (V, E_i)$ be a graph such that $E_i = \{e_1, \ldots, e_i\}$ is the set of the first i edges. Define $\Omega_i = \{c : c \text{ is a valid coloring for } G_i\}$ as the set of all valid colorings of G_i, and denote $r_i = \frac{|\Omega_i|}{|\Omega_{i-1}|}$.

We will estimate the number of graph colorings as

$$f(G) = |\Omega_m| = |\Omega_0| \cdot \frac{|\Omega_1|}{|\Omega_0|} \ldots \frac{|\Omega_m|}{|\Omega_{m-1}|} = |\Omega_0| \cdot \Pi_{i=1}^m r_i = q^n \cdot \Pi_{i=1}^m r_i$$

One can see that $\Omega_i \subseteq \Omega_{i-1}$ as removal of e_i in G_{i-1} can only increase the number of valid colorings. Furthermore, suppose $e_i = \{u, v\}$. Then, $\Omega_{i-1} \setminus \Omega_i = \{c : c(u) = c(v)\}$. That is, colorings that are in Ω_{i-1} but not in Ω_i are exactly those that assign the same color to both endpoints v and u of the edge e. We can argue that $\frac{|\Omega_i|}{|\Omega_{i-1}|}$ cannot be too small. In particular, for any coloring in $\Omega_{i-1} \setminus \Omega_i$, we can associate (in an injective manner) many different colorings in Ω_i: Fix the coloring of, say the lower-indexed vertex, u. Then, there are $\geq q - \Delta \geq 2\Delta + 1 - \Delta = \Delta + 1$ possible recolorings of v in G_i. Hence,

$$\begin{aligned}
& |\Omega_i| && \geq (\Delta + 1) \cdot |\Omega_{i-1} \setminus \Omega_i| \\
\iff & |\Omega_i| && \geq (\Delta + 1) \cdot (|\Omega_{i-1}| - |\Omega_i|) \\
\iff & |\Omega_i| + (\Delta + 1) \cdot |\Omega_i| && \geq (\Delta + 1) \cdot |\Omega_{i-1}| \\
\iff & (\Delta + 2) \cdot |\Omega_i| && \geq (\Delta + 1) \cdot |\Omega_{i-1}| \\
\iff & \frac{|\Omega_i|}{|\Omega_{i-1}|} && \geq \frac{\Delta + 1}{\Delta + 2}
\end{aligned}$$

This implies that $r_i = \frac{|\Omega_i|}{|\Omega_{i-1}|} \geq \frac{\Delta+1}{\Delta+2} \geq \frac{3}{4}$ since $\Delta \geq 2$.

Since $f(G) = |\Omega_m| = |\Omega_0| \cdot \frac{|\Omega_1|}{|\Omega_0|} \ldots \frac{|\Omega_m|}{|\Omega_{m-1}|} = |\Omega_0| \cdot \Pi_{i=1}^m r_i = q^n \cdot \Pi_{i=1}^m r_i$, if we can find a good estimate for each r_i with high probability, then we have a FPRAS for counting the number of valid graph colorings for G.

We now define Color-Count(G, ϵ) (algorithm 12) as an algorithm that estimates the number of valid colorings of graph G using $q \geq 2\Delta + 1$ colors.

Lemma 3.10. *For all* $i \in \{1, \ldots, m\}$, $\Pr\left[|\widehat{r_i} - r_i| \leq \frac{\epsilon}{2m} \cdot r_i\right] \geq 1 - \frac{1}{4m}$.

Algorithm 12 COLOR-COUNT(G, ϵ)

$\widehat{r_1}, \ldots, \widehat{r_m} \leftarrow 0$ ▷ Estimates for r_i
for $i = 1, \ldots, m$ **do**
  for $k = \frac{128m^3}{\epsilon^2}$ times **do**
    $c \leftarrow$ Sample coloring of G_{i-1} ▷ Using SAMPLECOLOR
    if c is a valid coloring for G_i **then**
      $\widehat{r_i} \leftarrow \widehat{r_i} + \frac{1}{k}$ ▷ Update empirical count of $r_i = \frac{|\Omega_i|}{|\Omega_{i-1}|}$
    end if
  end for
end for
return $q^n \Pi_{i=1}^m \widehat{r_i}$

Proof. Let X_j be the indicator variable whether the j-th sampled coloring for Ω_{i-1} is a valid coloring for Ω_i, where $p = \Pr[X_j = 1]$. From above, we know that $p = \Pr[X_j = 1] = \frac{|\Omega_i|}{|\Omega_{i-1}|} \geq \frac{3}{4}$. Let $X = \sum_{j=1}^k X_j$ be the empirical number of colorings that is valid for both Ω_{i-1} and Ω_i, captured by $k \cdot \widehat{r_i}$. Then, $\mathbb{E}(X) = kp$ by linearity of expectation. Picking $k = \frac{128m^3}{\epsilon^2}$,

$$\begin{aligned}
\Pr\left[|X - kp| \geq \frac{\epsilon}{2m} kp\right] &\leq 2\exp\left(-\frac{(\frac{\epsilon}{2m})^2 kp}{3}\right) && \text{By Chernoff bound} \\
&= 2\exp\left(-\frac{32mp}{3}\right) && \text{Since } k = \frac{128m^3}{\epsilon^2} \\
&\leq 2\exp(-8m) && \text{Since } p \geq \frac{3}{4} \\
&\leq \frac{1}{4m} && \text{Since } \exp(-x) \leq \frac{1}{x} \text{ for } x > 0
\end{aligned}$$

Dividing by k and negating, we have:

$$\Pr\left[|\widehat{r_i} - r_i| \leq \frac{\epsilon}{2m} \cdot r_i\right] = 1 - \Pr\left[|X - kp| \geq \frac{\epsilon}{2m} kp\right] \geq 1 - \frac{1}{4m}$$

□

Lemma 3.11. COLOR-COUNT *runs in poly*$(F, \frac{1}{\epsilon})$ = *poly*$(n, m, \frac{1}{\epsilon})$ *time.*

Proof. There are m r_i's to estimate. Each estimation has $k \in \mathcal{O}(\frac{m^3}{\epsilon^2})$ iterations. In each iteration, we spend $\mathcal{O}(n \log \frac{n}{\epsilon})$ time to sample a coloring c of G_{i-1} and $\mathcal{O}(m)$ time to check if c is a valid coloring for G_i. In total, COLOR-COUNT runs in $\mathcal{O}(mk(n \log \frac{n}{\epsilon} + m)) = \text{poly}(n, m, \frac{1}{\epsilon})$ time. □

Theorem 3.12. Color-Count *is a FPRAS for counting the number of valid graph colorings when* $q \geq 2\Delta + 1$ *and* $\Delta \geq 2$.

Proof. By Lemma 3.11, Color-Count runs in $\text{poly}(n, m, \frac{1}{\epsilon})$ time. Since $1 + x \leq e^x$ for all real x, we have $(1 + \frac{\epsilon}{2m})^m \leq e^{\frac{\epsilon}{2}} \leq 1 + \epsilon$. The last inequality[3] is because $e^x \leq 1 + 2x$ for $0 \leq x \leq 1.25643$. On the other hand, Bernoulli's inequality tells us that $(1 - \frac{\epsilon}{2m})^m \geq 1 - \frac{\epsilon}{2} \geq 1 - \epsilon$. We know from the proof of Lemma 3.10, $\Pr[|\widehat{r_i} - r_i| > \frac{\epsilon}{2m} \cdot r_i] \leq \frac{1}{4m}$ for any estimate r_i. Therefore, by a union bound, we have

$$\begin{aligned}\Pr\left[|q^n \Pi_{i=1}^m \widehat{r_i} - f(G)| > \epsilon f(G)\right] &\leq \sum_{i=1}^m \Pr\left[|\widehat{r_i} - r_i| > \frac{\epsilon}{2m} \cdot r_i\right] \\ &\leq m \cdot \frac{1}{4m} = \frac{1}{4}\end{aligned}$$

Hence, $\Pr\left[|q^n \Pi_{i=1}^m \widehat{r_i} - f(G)| \leq \epsilon f(G)\right] \geq 3/4$. □

Remark Recall from Claim 3.9 that SampleColor actually gives an approximate uniform coloring. A more careful analysis can absorb the approximation of SampleColor under Color-Count's ϵ factor.

[3] See https://www.wolframalpha.com/input/?i=e%5Ex+%3C%3D+1%2B2x

Chapter 4

Rounding Linear Program Solutions

Linear programming (LP) and integer linear programming (ILP) are versatile models but with different solving complexities — LPs are solvable in polynomial time while ILPs are $\mathcal{NP}$-hard.

Definition 4.1 (Linear program (LP))**.** *The canonical form of an LP is*

$$\begin{aligned} \textit{minimize} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \textit{subject to} \quad & A\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq 0 \end{aligned}$$

where $\boldsymbol{x}$ is the vector of n variables (to be determined), $\boldsymbol{b}$ and $\boldsymbol{c}$ are vectors of (known) coefficients, and A is a (known) matrix of coefficients. $\boldsymbol{c}^T\boldsymbol{x}$ and $obj(\boldsymbol{x})$ are the objective function and objective value of the LP respectively. For an optimal variable assignment $\boldsymbol{x}^*$, $obj(\boldsymbol{x}^*)$ is the optimal value.

ILPs are defined similarly with the additional constraint that variables take on integer values. As we will be relaxing ILPs into LPs, to avoid confusion, we use y for ILP variables to contrast against the x variables in LPs.

Definition 4.2 (Integer linear program (ILP))**.** *The canonical form of an ILP is*

$$\begin{aligned} \textit{minimize} \quad & \boldsymbol{c}^T \boldsymbol{y} \\ \textit{subject to} \quad & A\boldsymbol{y} \geq \boldsymbol{b} \\ & \boldsymbol{y} \geq 0 \\ & \boldsymbol{y} \in \mathbb{Z}^n \end{aligned}$$

where $\boldsymbol{y}$ is the vector of n variables (to be determined), $\boldsymbol{b}$ and $\boldsymbol{c}$ are vectors of (known) coefficients, and A is a (known) matrix of coefficients. $\boldsymbol{c}^T\boldsymbol{y}$ and $obj(\boldsymbol{y})$ are the objective function and objective value of the LP respectively. For an optimal variable assignment $\boldsymbol{y}^*$, $obj(\boldsymbol{y}^*)$ is the optimal value.

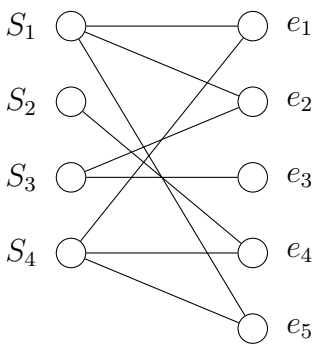
Remark We can define LPs and ILPs for maximization problems similarly. One can also solve maximization problems with a minimization LPs using the same constraints but negated objective function. The optimal value from the solved LP will then be the negation of the maximized optimal value.

In this chapter, we illustrate how one can model set cover and multi-commodity routing as ILPs, and how to perform rounding to yield approximations for these problems. As before, Chernoff bounds will be a useful inequality in our analysis toolbox.

4.1 Minimum set cover

Recall the minimum set cover problem and the example from Section 1.1.

Example



Suppose there are $n = 5$ vertices and $m = 4$ subsets $S = \{S_1, S_2, S_3, S_4\}$, where the cost function is defined as $c(S_i) = i^2$. Then, the minimum set cover is $S^* = \{S_1, S_2, S_3\}$ with a cost of $c(S^*) = 14$.

In Section 1.1, we saw that a greedy selection of sets that minimizes the price-per-item of remaining sets gave an H_n-approximation for set cover. Furthermore, in the special cases where $\Delta = \max_{i\in[m]} \text{degree}(S_i)$ and $f = \max_{i\in[n]} \text{degree}(x_i)$ are small, one can obtain H_Δ-approximation and f-approximation respectively.

We now show how to formulate set cover as an ILP, reduce it into a LP, and how to round the solutions to yield an approximation to the original set cover instance. Consider the following ILP:

$\mathbf{ILP}_{\text{Set cover}}$

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m} y_i \cdot c(S_i) && \triangleleft \text{ Cost of chosen set cover} \\
\text{subject to} \quad & \sum_{i:e_j \in S_i} y_i \geq 1 \quad \forall j \in [n] && \triangleleft \text{ Every item } e_j \text{ is covered} \\
& y_i \in \{0,1\} \quad \forall i \in [m] && \triangleleft \text{ Indicator whether set } S_i \text{ is chosen}
\end{aligned}$$

Upon solving $\text{ILP}_{\text{Set cover}}$, the set $\{S_i : i \in [n] \wedge y_i^* = 1\}$ is the optimal solution for a given set cover instance. However, as solving ILPs is $\mathcal{NP}$-hard, we consider relaxing the integral constraint by replacing binary y_i variables by real-valued/fractional $x_i \in [0,1]$. Such a relaxation will yield the corresponding LP:

$\mathbf{LP}_{\text{Set cover}}$

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m} x_i \cdot c(S_i) && \triangleleft \text{ Cost of chosen fractional set cover} \\
\text{subject to} \quad & \sum_{i:e_j \in S_i} x_i \geq 1 \quad \forall j \in [n] && \triangleleft \text{ Every item } e_j \text{ is fractionally covered} \\
& 0 \leq x_i \leq 1 \quad \forall i \in [m] && \triangleleft \text{ Relaxed indicator variables}
\end{aligned}$$

Since LPs can be solved in polynomial time, we can find the optimal fractional solution to $\text{LP}_{\text{Set cover}}$ in polynomial time.

Observation As the set of solutions of $\text{ILP}_{\text{Set cover}}$ is a subset of $\text{LP}_{\text{Set cover}}$, $obj(\boldsymbol{x}^*) \leq obj(\boldsymbol{y}^*)$.

Example The corresponding ILP for the example set cover instance is:

$$\begin{aligned}
\text{minimize} \quad & y_1 + 4y_2 + 9y_3 + 16y_4 && \\
\text{subject to} \quad & y_1 + y_4 \geq 1 && \triangleleft \text{ Sets covering } e_1 \\
& y_1 + y_3 \geq 1 && \triangleleft \text{ Sets covering } e_2 \\
& y_3 \geq 1 && \triangleleft \text{ Sets covering } e_3 \\
& y_2 + y_4 \geq 1 && \triangleleft \text{ Sets covering } e_4 \\
& y_1 + y_4 \geq 1 && \triangleleft \text{ Sets covering } e_5 \\
& \forall i \in \{1,\ldots,4\}, y_i \in \{0,1\} &&
\end{aligned}$$

After relaxing:

$$\begin{aligned}
\text{minimize} \quad & x_1 + 4x_2 + 9x_3 + 16x_4 \\
\text{subject to} \quad & x_1 + x_4 \geq 1 \\
& x_1 + x_3 \geq 1 \\
& x_3 \geq 1 \\
& x_2 + x_4 \geq 1 \\
& x_1 + x_4 \geq 1 \\
& \forall i \in \{1, \ldots, 4\}, 0 \leq x_i \leq 1 \quad \triangleleft \text{ Relaxed indicator variables}
\end{aligned}$$

Solving it using a LP solver[1] yields: $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 0$. Since the solved $\boldsymbol{x}^*$ are integral, $\boldsymbol{x}^*$ is also the optimal solution for the original ILP. In general, the solved $\boldsymbol{x}^*$ may be fractional, which does not immediately yield a set selection.

We now describe two ways to round the fractional assignments $\boldsymbol{x}^*$ into binary variables $\boldsymbol{y}$ so that we can interpret them as proper set selections.

4.1.1 (Deterministic) Rounding for small f

We round $\boldsymbol{x}^*$ as follows:

$$\forall i \in [m], \text{ set } y_i = \begin{cases} 1 & \text{if } x_i^* \geq \frac{1}{f} \\ 0 & \text{else} \end{cases}$$

Theorem 4.3. *The rounded $\boldsymbol{y}$ is a feasible solution to $ILP_{\text{Set cover}}$.*

Proof. Since $\boldsymbol{x}^*$ is a feasible (not to mention, optimal) solution for $\text{LP}_{\text{Set cover}}$, in each constraint, there is at least one x_i^* that is greater or equal to $\frac{1}{f}$. Hence, every element is covered by some set y_i in the rounding. □

Theorem 4.4. *The rounded $\boldsymbol{y}$ is a f-approximation to $ILP_{\text{Set cover}}$.*

Proof. By the rounding, $y_i \leq f \cdot x_i^*, \forall i \in [m]$. Therefore,

$$obj(\boldsymbol{y}) \leq f \cdot obj(\boldsymbol{x}^*) \leq f \cdot obj(\boldsymbol{y}^*)$$

□

[1] Using Microsoft Excel. See tutorial: http://faculty.sfasu.edu/fisherwarre/lp_solver.html
Or, use an online LP solver such as: http://online-optimizer.appspot.com/?model=builtin:default.mod

4.1.2 (Randomized) Rounding for general f

If f is large, having a f-approximation algorithm from the previous subsection may be unsatisfactory. By introducing randomness in the rounding process, we show that one can obtain a $\ln(n)$-approximation (in expectation) with arbitrarily high probability through probability amplification.

Consider the following rounding procedure:

1. Interpret each x_i^* as probability for picking S_i. That is, $\Pr[y_i = 1] = x_i^*$.
2. For each i, independently set y_i to 1 with probability x_i^*.

Theorem 4.5. $\mathbb{E}(obj(\boldsymbol{y})) = obj(\boldsymbol{x}^*)$

Proof.

$$\begin{aligned}
\mathbb{E}(obj(\boldsymbol{y})) &= \mathbb{E}(\sum_{i=1}^{m} y_i \cdot c(S_i)) \\
&= \sum_{i=1}^{m} \mathbb{E}(y_i) \cdot c(S_i) && \text{By linearity of expectation} \\
&= \sum_{i=1}^{m} \Pr(y_i = 1) \cdot c(S_i) && \text{Since each } y_i \text{ is an indicator variable} \\
&= \sum_{i=1}^{m} x_i^* \cdot c(S_i) && \text{Since } \Pr(y_i = 1) = x_i^* \\
&= obj(\boldsymbol{x}^*)
\end{aligned}$$

□

Although the rounded selection to yield an objective cost that is close to the optimum (in expectation) of the LP, we need to consider whether all constraints are satisfied.

Theorem 4.6. *For any $j \in [n]$, item e_j is not covered w.p. $\leq e^{-1}$.*

Proof. For any $j \in [n]$,

$$\begin{aligned}
\Pr[\text{Item } e_j \text{ not covered}] &= \Pr[\sum_{i:e_j \in S_i} y_i = 0] \\
&= \Pi_{i:e_j \in S_i}(1 - x_i^*) && \text{Since the } y_i \text{ are chosen independently} \\
&\leq \Pi_{i:e_j \in S_i} e^{-x_i^*} && \text{Since } (1 - x) \leq e^{-x} \\
&= e^{-\sum_{i:e_j \in S_i} x_i^*} \\
&\leq e^{-1}
\end{aligned}$$

The last inequality holds because the optimal solution $\boldsymbol{x}^*$ satisfies the j^{th} constraint in the LP that $\sum_{i:e_j \in S_i} x_i^* \geq 1$. □

Since $e^{-1} \approx 0.37$, we would expect the rounded $\boldsymbol{y}$ not to cover several items. However, one can amplify the success probability by considering independent roundings and taking the union (See ApxSetCoverILP).

Algorithm 13 ApxSetCoverILP$(\mathcal{U}, \mathcal{S}, c)$

```
ILP_Set cover ← Construct ILP of problem instance
LP_Set cover ← Relax integral constraints on indicator variables y to x
x* ← Solve LP_Set cover
T ← ∅                                          ▷ Selected subset of S
for k · ln(n) times (for any constant k > 1) do
    for i ∈ [m] do
        y_i ← Set to 1 with probability x*_i
        if y_i = 1 then
            T ← T ∪ {S_i}                      ▷ Add to selected sets T
        end if
    end for
end for
return T
```

Similar to Theorem 4.4, we can see that $\mathbb{E}(obj(T)) \leq (k \cdot \ln(n)) \cdot obj(\boldsymbol{y}^*)$. Furthermore, Markov's inequality tells us that the probability of $obj(T)$ being z times larger than its expectation is at most $\frac{1}{z}$.

Theorem 4.7. ApxSetCoverILP *gives a valid set cover w.p.* $\geq 1 - n^{1-k}$.

Proof. For all $j \in [n]$,

$$\begin{aligned}\Pr[\text{Item } e_j \text{ not covered by } T] &= \Pr[e_j \text{ not covered by } \textit{all } k\ln(n) \text{ roundings}] \\ &\leq (e^{-1})^{k\ln(n)} \\ &= n^{-k}\end{aligned}$$

Taking union bound over all n items,

$$\Pr[T \text{ is not a valid set cover}] \leq \sum_{i=1}^{n} n^{-k} = n^{1-k}$$

So, T is a valid set cover with probability $\geq 1 - n^{1-k}$. □

Note that the success probability of $1 - n^{1-k}$ can be further amplified by taking several *independent* samples of ApxSetCoverILP, then returning the lowest cost valid set cover sampled. With z samples, the probability that *all* repetitions fail is less than $n^{z(1-k)}$, so we succeed w.p. $\geq 1 - n^{z(1-k)}$.

4.2 Minimizing congestion in multi-commodity routing

A multi-commodity routing (MCR) problem involves routing multiple (s_i, t_i) flows across a network with the goal of minimizing congestion, where congestion is defined as the largest ratio of flow over capacity of any edge in the network. In this section, we discuss two variants of the multi-commodity routing problem. In the first variant (special case), we are given the set of possible paths $\mathcal{P}_i$ for each (s_i, t_i) source-target pairs. In the second variant (general case), we are given only the network. In both cases, [RT87] showed that one can obtain an approximation of $\mathcal{O}(\frac{\log(m)}{\log\log(m)})$ with high probability.

Definition 4.8 (Multi-commodity routing problem)**.** *Consider a directed graph $G = (V, E)$ where $|E| = m$ and each edge $e = (u, v) \in E$ has a capacity $c(u, v)$. The in-set/out-set of a vertex v is denoted as $in(v) = \{(u, v) \in E : u \in V\}$ and $out(v) = \{(v, u) \in E : u \in V\}$ respectively. Given k triplets (s_i, t_i, d_i), where $s_i \in V$ is the source, $t_i \in V$ is the target, and $d_i \geq 0$ is the demand for the i^{th} commodity respectively, denote $f(e, i) \in [0, 1]$ as the fraction of d_i that is flowing through edge e. The task is to minimize the congestion parameter λ by finding paths p_i for each $i \in [k]$, such that:*

(i) (Valid sources): $\sum_{e \in out(s_i)} f(e, i) - \sum_{e \in in(s_i)} f(e, i) = 1, \forall i \in [k]$

(ii) (Valid sinks): $\sum_{e \in in(t_i)} f(e, i) - \sum_{e \in out(t_i)} f(e, i) = 1, \forall i \in [k]$

(iii) (Flow conservation): For each commodity $i \in [k]$,

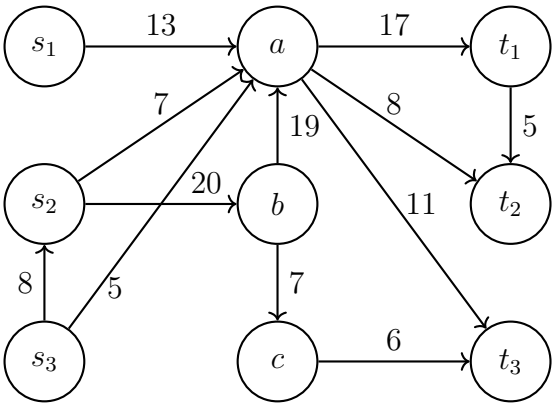
$$\sum_{e \in out(v)} f(e, i) - \sum_{e \in in(v)} f(e, i) = 0, \quad \forall e \in E, \forall v \in V \setminus \{s_i \cup t_i\}$$

(iv) (Single path): All demand for commodity i passes through a single path p_i (no repeated vertices).

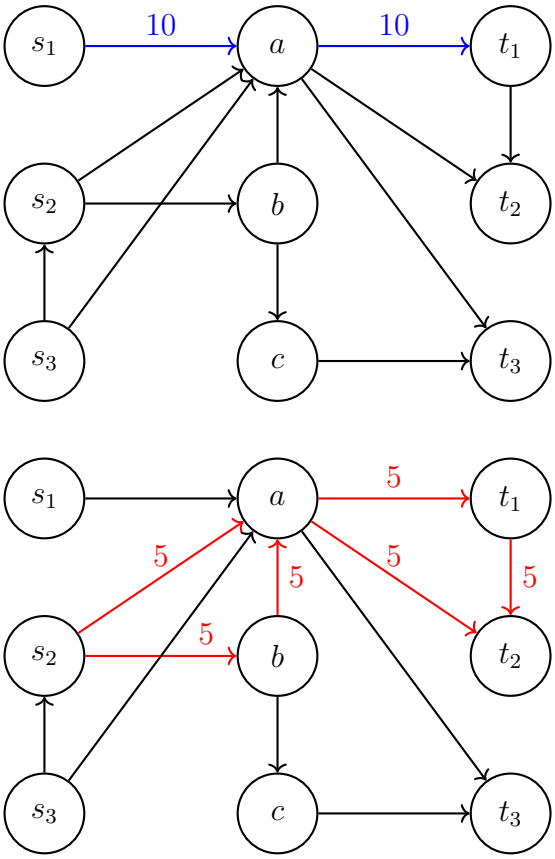
(v) (Congestion factor): $\forall e \in E, \sum_{i=1}^{k} d_i \mathbb{1}_{e \in p_i} \leq \lambda \cdot c(e)$*, where indicator* $\mathbb{1}_{e \in p_i} = 1 \iff e \in p_i$.

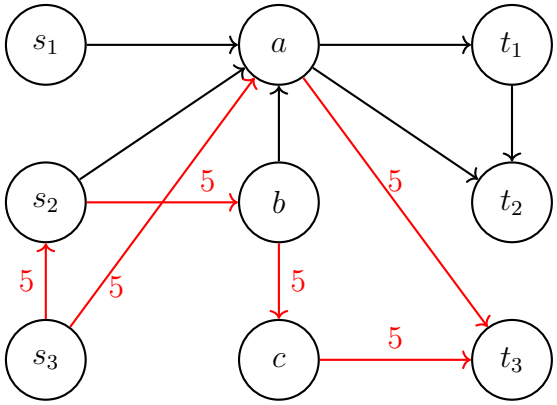
(vi) (Minimum congestion): λ is minimized.

Example Consider the following flow network with $k = 3$ commodities with edge capacities as labelled:

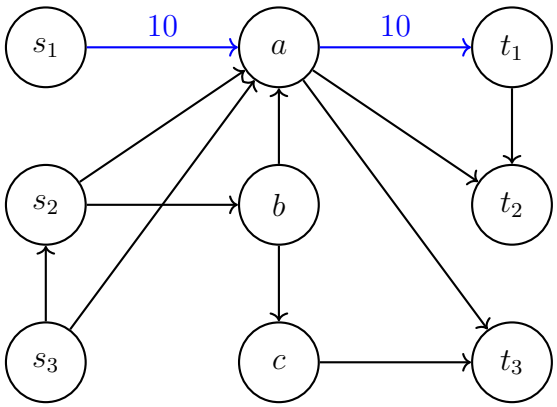


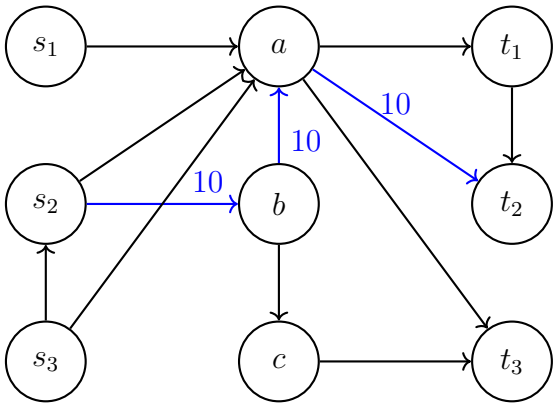
For demands $d_1 = d_2 = d_3 = 10$, there exists a flow assignment such that the total demands flowing on each edge is below its capacity:

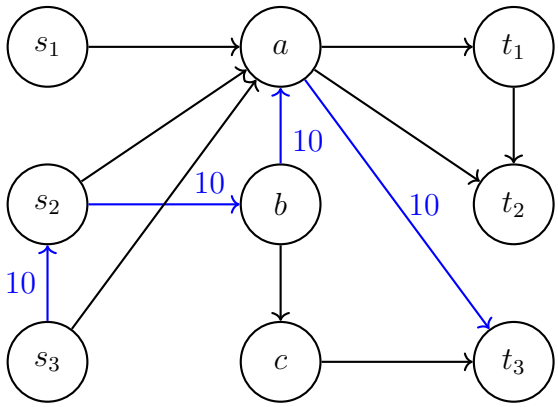




Although the assignment attains congestion $\lambda = 1$ (due to edge (s_3, a)), the path assignments for commodities 2 and 3 violate the property of "single path". Forcing all demand of each commodity to flow through a single path, we have a minimum congestion of $\lambda = 1.25$ (due to edges (s_3, s_2) and (a, t_2)):







4.2.1 Special case: Given sets of $s_i - t_i$ paths $\mathcal{P}_i$

For each commodity $i \in [k]$, we are to select a path p_i from a given set of valid paths $\mathcal{P}_i$, where each edge in all paths in $\mathcal{P}_i$ has capacities $\geq d_i$. Because we intend to pick a single path for each commodity to send *all* demands through, constraints (i)-(iii) of MCR are fulfilled trivially. Using $y_{i,p}$ as indicator variables whether path $p \in \mathcal{P}_i$ is chosen, we can model the following ILP:

ILP$_{\text{MCR-Given-Paths}}$

$$\begin{aligned}
&\text{minimize} && \lambda && && \triangleleft\ (1)\\
&\text{subject to} && \sum_{i=1}^{k} \Big(d_i \cdot \sum_{p \in \mathcal{P}_i, e \in p} y_{i,p} \Big) \leq \lambda \cdot c(e) && \forall e \in E && \triangleleft\ (2)\\
& && \sum_{p \in \mathcal{P}_i} y_{i,p} = 1 && \forall i \in [k] && \triangleleft\ (3)\\
& && y_{i,p} \in \{0,1\} && \forall i \in [k], p \in \mathcal{P}_i && \triangleleft\ (4)
\end{aligned}$$

◁ **(1)** Congestion parameter λ

◁ **(2)** Congestion factor relative to selected paths

◁ **(3)** Exactly one path chosen from each $\mathcal{P}_i$

◁ **(4)** Indicator variable for path $p \in \mathcal{P}_i$

Relax the integral constraint on $y_{i,p}$ to $x_{i,p} \in [0,1]$ and solve the corresponding LP. Define $\lambda^* = obj(\text{LP}_{\text{MCR-Given-Paths}})$ and denote $\boldsymbol{x}^*$ as a fractional path

selection that achieves λ^*. To obtain a valid path selection, for each commodity $i \in [k]$, pick path $p \in \mathcal{P}_i$ with weighted probability $\frac{x^*_{i,p}}{\sum_{p \in \mathcal{P}_i} x^*_{i,p}} = x^*_{i,p}$. Note that by constraint (3), $\sum_{p \in \mathcal{P}_i} x^*_{i,p} = 1$.

Remark 1 For a fixed i, a path is selected *exclusively* (only one!) (cf. set cover's roundings where we may pick multiple sets for an item).

Remark 2 The weighted sampling is independent across different commodities. That is, the choice of path amongst $\mathcal{P}_i$ does not influence the choice of path amongst $\mathcal{P}_j$ for $i \neq j$.

Theorem 4.9. $\Pr[obj(\boldsymbol{y}) \geq \frac{2c \log m}{\log \log m} \max\{1, \lambda^*\}] \leq \frac{1}{m^{c-1}}$

Proof. Fix an arbitrary edge $e \in E$. For each commodity i, define an indicator variable $Y_{e,i}$ whether edge e is part of the chosen path for commodity i. By randomized rounding, $\Pr[Y_{e,i} = 1] = \sum_{p \in \mathcal{P}_i, e \in p} x^*_{i,p}$. Denoting $Y_e = \sum_{i=1}^{k} d_i \cdot Y_{e,i}$ as the total demand on edge e in *all* k chosen paths,

$$
\begin{aligned}
\mathbb{E}(Y_e) &= \mathbb{E}(\sum_{i=1}^{k} d_i \cdot Y_{e,i}) \\
&= \sum_{i=1}^{k} d_i \cdot \mathbb{E}(Y_{e,i}) && \text{By linearity of expectation} \\
&= \sum_{i=1}^{k} d_i \sum_{p \in \mathcal{P}_i, e \in p} x_{i,p} && \text{Since } \Pr[Y_{e,i} = 1] = \sum_{p \in \mathcal{P}_i, e \in p} x_{i,p} \\
&\leq \lambda^* \cdot c(e) && \text{By MCR constraint and optimality of the solved LP}
\end{aligned}
$$

For every edge $e \in E$, applying[2] the tight form of Chernoff bounds with $(1+\epsilon) = \frac{2 \log n}{\log \log n}$ on variable $\frac{Y_e}{c(e)}$ gives

$$\Pr[\frac{Y_e}{c(e)} \geq \frac{2c \log m}{\log \log m} \max\{1, \lambda^*\}] \leq \frac{1}{m^c}$$

Finally, take union bound over all m edges. □

[2]See Corollary 2 of https://courses.engr.illinois.edu/cs598csc/sp2011/Lectures/lecture_9.pdf for details.

4.2.2 General: Given only a network

In the general case, we may not be given path sets $\mathcal{P}_i$ and there may be exponentially many $s_i - t_i$ paths in the network. However, we show that one can still formulate an ILP and round it (slightly differently) to yield the same approximation factor. Consider the following:

ILP$_{\text{MCR-Given-Network}}$

$$\begin{aligned}
&\text{minimize} && \lambda && && \triangleleft\ (1)\\
&\text{subject to} && \sum_{e\in out(s_i)} f(e,i) - \sum_{e\in in(s_i)} f(e,i) = 1 && \forall i \in [k] && \triangleleft\ (2)\\
& && \sum_{e\in in(t_i)} f(e,i) - \sum_{e\in out(t_i)} f(e,i) = 1 && \forall i \in [k] && \triangleleft\ (3)\\
& && \sum_{e\in out(v)} f(e,1) - \sum_{e\in in(v)} f(e,1) = 0 && \forall e \in E, && \triangleleft\ (4)\\
& && && \forall v \in V \setminus \{s_1 \cup t_1\} &&\\
& && \vdots && && \vdots\\
& && \sum_{e\in out(v)} f(e,k) - \sum_{e\in in(v)} f(e,k) = 0 && \forall e \in E, && \triangleleft\ (4)\\
& && && \forall v \in V \setminus \{s_k \cup t_k\} &&\\
& && \sum_{i=1}^{k} \Big(d_i \cdot \sum_{p\in\mathcal{P}_i, e\in p} y_{i,p}\Big) \le \lambda \cdot c(e) && \forall e \in E && \text{As before}\\
& && \sum_{p\in\mathcal{P}_i} y_{i,p} = 1 && \forall i \in [k] && \text{As before}\\
& && y_{i,p} \in \{0,1\} && \forall i \in [k], p \in \mathcal{P}_i && \text{As before}
\end{aligned}$$

◁ **(1)** Congestion parameter λ

◁ **(2)** Valid sources

◁ **(3)** Valid sinks

◁ **(4)** Flow conservation

Relax the integral constraint on $y_{i,p}$ to $x_{i,p} \in [0,1]$ and solve the corresponding LP. To extract the path candidates $\mathcal{P}_i$ for each commodity, perform flow decomposition[3]. For each extracted path p_i for commodity i, treat the minimum

[3]See https://www.youtube.com/watch?v=zgutyzA9JM4&t=1020s (17:00 to 29:50) for a recap on flow decomposition.

$\min_{e \in p_i} f(e,i)$ on the path as the selection probability (as per $x_{e,i}$ in the previous section). By selecting the path p_i with probability $\min_{e \in p_i} f(e,i)$, one can show by similar arguments as before that $\mathbb{E}(obj(\boldsymbol{y})) \leq obj(\boldsymbol{x}^*) \leq obj(\boldsymbol{y}^*)$.

4.3 Scheduling on unrelated parallel machines

We will now discuss a generalization of the makespan problem. In order to get a good approximation, we will again use an LP formulation. However, this time, randomized rounding will not be sufficient, and we will instead use a more elaborate form of rounding, by utilizing the combinatorial structure of the problem.

Setting. The setting is close to what we discussed in the past: We have a collection J of n jobs/tasks, and we have a set M of m machines to process them. Previously, each job had a fixed size regardless of which machine processes it. Now we think about a more general case: $t_{ij} \geq 0$ is the time for machine j to process job i. The t_{ij} can be arbitrary. This means that some machines can be a lot better than others on given jobs, and a lot worse on others. This is what we mean when we speak of "unrelated" machines.

How can we formulate this problem as a Linear Program?

Naive LP. The most obvious way is to take x_{ij} to be an indicator for assigning job i to machine j and then optimizing the following objective:

$$\begin{aligned} &\min t \\ \text{s.t. } &\forall \text{machine } j, \sum_{i=1}^{n} t_{ij} \cdot x_{ij} \leq t \\ \text{and } &\forall \text{job } i, \sum_{j=1}^{m} x_{ij} \geq 1 \\ \text{and } &\forall \text{job i}, \text{machine } j,\ x_{ij} \geq 0 \end{aligned}$$

Here, t is an additional variable giving an upper bound for the finishing time of the last job.

The problem with this LP is that the best fractional solution and the best integral solution can be far apart, i.e., the LP has a large "*integrality gap*". Namely, the fractional solution to the LP is allowed to distribute a

big job among different machines. In particular, consider an instance with a single large job with the same processing time on all machines. The integral solution needs to assign this job to one of the machines, while the fractional solution can evenly split it among all m machines. Therefore, we can lose a factor as big as m.

Note that we get a correct solution for both the fractional and the integral case. The problem is that we want to relate the two solutions in order to prove a small approximation factor, and this is not possible for the given LP.

Improved LPs. We will now change the LP a bit. Suppose somebody tells us that the processing time t is at most a certain λ. We will find a way to check this claim up to an approximation factor. Once we have the upper bound λ, only some of the assignments make sense, namely

$$S_\lambda = \{(i,j) \mid t_{ij} \leq \lambda\}.$$

If a single job i takes more than λ time on a given machine j, we cannot schedule it on this machine at all. The set S_λ contains all assignments of single jobs to single machines that are not ruled out in this way. We can now write an LP on $|S_\lambda|$ variables that is specific to a given value of λ.

$$\begin{aligned}
&\text{LP}(\lambda): \\
&\forall \text{machine } j, \sum_{(i,j)\in S_\lambda} t_{ij} \cdot x_{ij} \leq \lambda \\
&\forall \text{job } i, \sum_{(i,j)\in S_\lambda} x_{ij} \geq 1 \\
&\forall (i,j) \in S_\lambda,\ x_{ij} \geq 0
\end{aligned}$$

This time, we just want to check for feasibility. We have constraints (defining a polytope), but there is no objective function. Using binary search[4], we can find the smallest λ^* for which we can find a fractional solution of LP(λ^*). Note that it is easy to initialize the binary search, as there are trivial lower and upper bounds on λ^*, for example 0 and the sum of all processing times.

Now, somebody gives us a fractional solution of LP(λ^*): Suppose this solution is $\mathbf{x}^* \in [0,1]^{|S_\lambda|}$. Instead of just assuming that $\mathbf{x}^*$ is an arbitrary solution, we will also assume that $\mathbf{x}^*$ is a vertex of the polytope.[5]

[4] Feasibility of LP(λ) is a monotone property in λ, as for $\lambda \leq \lambda'$, a solution of LP(λ) can be extended to a solution of LP(λ').

[5] Also known as: a basic feasible solution, an extreme point, a generator of the polytope, a solution that cannot be written as a convex combination of other solutions, etc.

Rounding Algorithm. We want to round $\mathbf{x}^*$ to an integral assignment of jobs to machines. This is a place where the rounding will be not very direct. However, there are some assignments for which rounding is obvious, in the sense that $x_{ij} = 1$. For those cases, we can just assign the job i to machine j. So in the following we can assume that all variables have fractional values. The support of $\mathbf{x}^*$ forms a graph H on jobs and machines. (The edge (i, j) is present iff $x^*_{ij} \in (0, 1)$.) This is a bipartite graph with jobs on one side and machines on the other side.

We will prove that there is a perfect matching in H, and that we obtain a 2-approximation by combining the obvious assignments with this perfect matching. More explicitly, the algorithm is this:

1. For edges (i, j) such that $x_{ij} = 1$, assign job i to machine j. Let I be the set of jobs assigned in this step. ($I \subseteq J$.)

2. Let H be the bipartite graph on jobs and machines where job i and machine j are connected iff $x_{i,j} \in (0, 1)$.

3. Find a matching in H that is perfect for the remaining jobs F. ($F = J \setminus I$.)

4. For each matching edge (i, j), assign job i to machine j.

Jobs assigned in step 1 take at most time λ^* to complete. With the matching, each machine gets at most one more job, whose cost is also at most λ^* (by definition of the set S^*_λ). Therefore, we can construct a solution with cost at most $2 \cdot \lambda^*$. As λ^* is a lower bound on the optimal cost, this proves that the algorithm is a 2-approximation. This is a much more careful rounding method than randomized rounding.

Correctness of Rounding. It remains to be argued that the perfect matching exists.

Definition 4.10. *A* pseudo-forest *is a graph for which each connected component is a pseudo-tree, where a pseudo-tree is either a tree or a tree with an additional edge*

Claim: H is a bipartite pseudo-forest, where each leaf is a machine. It is easy to prove that such bipartite pseudo-forests admit a matching that is perfect for the jobs.

Lemma 4.11. *A bipartite pseudo-forest whose leaves are machines contains a matching that covers all the jobs.*

Proof. We argue separately for each connected component. If a connected component has a leaf that is not adjacent to a cycle, it is always possible to pick a leaf not adjacent to a cycle such that if we remove it and its (unique) neighbour, the produced components consist of at most one pseudo-tree with machines as leaves and possibly a few isolated machines.

We can repeatedly pick a machine that is such a leaf and match it to its neighbour. We then delete both the matched job and its machine, and further, we delete all machines that have become isolated. The resulting component is again a bipartite pseudo-tree whose leaves are machines. If we repeat this process until no more steps can be taken, we are left with a graph that is an even cycle, possibly with a few leaves attached. As those leaves are machines, we can ignore them and use one of the two perfect matchings of the cycle to assign the remaining jobs. By doing this for all components, we can assign all jobs to machines. □

We still need to prove that H is in fact a pseudo-forest.

Lemma 4.12. *The graph H is a pseudo-forest.*

Proof. We first prove that H is a pseudo-tree if it is connected, and then we show how to extend the argument to the case where H has multiple connected components.

H connected. If H is connected, it suffices to show that $|E(H)| \leq |V(H)|$. We will use the fact that $\mathbf{x}^*$ is a vertex of the polytope. Let $r = |S_{\lambda^*}|$ be the number of variables. As $\mathbf{x}^*$ is a vertex, there must exist r linearly independent tight constraints. Among those r constraints, there can be at most m constraints on machines and n constraints on job assignments (c.f. the LP). Therefore, at least $r-(n+m)$ constraints of the form $x_{ij} \geq 0$ must be tight. Hence the number of variables that are non-zero in $\mathbf{x}^*$ is at most $r-(r-(n+m)) = n+m$. In particular, the number of fractional variables, corresponding to edges in $E(H)$, is at most $n+m = |V(H)|$.

(Aside: Recall that we defined I as the set of integrally-assigned jobs, while F is the set of fractionally-assigned jobs. As all jobs fall into exactly one of those categories, we have $|I|+|F| = n$. Each integral job i is associated to at least one non-zero variable $x^*_{ij_0}$, while a fractional job is associated to at least two non-zero variables $x^*_{ij_1}$ and $x^*_{ij_2}$ (because of the constraint that $\sum_j x_{ij} \geq 1$). We derive the inequality $|I| + 2 \cdot |F| \leq n+m$ and can conclude that $|I| \geq n-m$. This means that if the number of machines is small, many jobs will be assigned non-fractionally.)

H disconnected. Now we extend the argument to cover the case where H has multiple connected components. Given some such component H', we can restrict the solution $\mathbf{x}^*$ to this component, by ignoring all variables corresponding to assignments of single jobs to single machines that are not both in $V(H')$. We call this restricted vector $\mathbf{x}'^*$. Claim: $\mathbf{x}'^*$ is a vertex of the polytope obtained by only considering variables associated to edges connecting vertices in $V(H')$ and writing the analogue of the LP(λ^*) constraints for them. Proof: Otherwise, we can pick two feasible solutions $\mathbf{x_1'^*}$ and $\mathbf{x_2'^*}$ of the restricted LP such that $\mathbf{x}'^* = \frac{1}{2}(\mathbf{x_1'^*} + \mathbf{x_2'^*})$. We can then extend $\mathbf{x_1'^*}$ and $\mathbf{x_2'^*}$ to solutions for the unrestricted LP by filling in the missing components from $\mathbf{x}^*$. The resulting solutions have $\mathbf{x}^*$ as their middle point, which contradicts the assumption that $\mathbf{x}^*$ is a vertex of the polytope associated to the unrestricted LP. Therefore, the reasoning from above applies independently to all connected components of H and H is a pseudo-forest. □

All leaves of H are machines, because fractionally-assigned jobs have at least two neighbours. Using the two lemmas, we conclude that the rounding algorithm is correct.

Part II

Selected Topics in Approximation Algorithms

Chapter 5

Distance-preserving tree embedding

Many hard graph problems become easy if the graph is a tree: in particular, some $\mathcal{NP}$-hard problems are known to admit exact polynomial time solutions on trees, and for some other problems, we can obtain much better approximations on tree. Motivated by this fact, one hopes to design the following framework for a general graph $G = (V, E)$ with distance metric $d_G(u, v)$ between vertices $u, v \in V$:

1. Construct a tree T
2. Solve the problem on T efficiently
3. Map the solution back to G
4. Argue that the transformed solution from T is a good approximation for the exact solution on G.

Ideally, we want to build a tree T such that $d_G(u, v) \leq d_T(u, v)$ and $d_T(u, v) \leq c \cdot d_G(u, v)$, where c is the *stretch of the tree embedding*. Unfortunately, such a construction is hopeless[1].

Instead, we relax the hard constraint $d_T(u, v) \leq c \cdot d_G(u, v)$ and consider a distribution over a collection of trees $\mathcal{T}$, so that

- (Over-estimates cost): $\forall u, v \in V, \forall T \in \mathcal{T}, d_G(u, v) \leq d_T(u, v)$
- (Over-estimate by not too much): $\forall u, v \in V, \mathbb{E}_{T \in \mathcal{T}}[d_T(u, v)] \leq c \cdot d_G(u, v)$

[1] For a cycle G with n vertices, the excluded edge in a constructed tree will cause the stretch factor $c \geq n - 1$. Exercise 8.7 in [WS11]

- ($\mathcal{T}$ is a probability space): $\sum_{T \in \mathcal{T}} \Pr[T] = 1$

Bartal [Bar96, Theorem 8] gave a construction for probabilistic tree embedding with poly-logarithmic stretch factor c. He also proved in [Bar96, Theorem 9] that a stretch factor $c \in \Omega(\log n)$ is required for general graphs. A construction that yields $c \in \mathcal{O}(\log n)$, in expectation, was subsequently found by Fakcharoenphol, Talwar, and Rao [FRT03].

5.1 A tight probabilistic tree embedding construction

In this section, we describe a probabilistic tree embedding construction due to [FRT03] with a stretch factor $c = \mathcal{O}(\log n)$. For a graph $G = (V, E)$, let the distance metric $d_G(u, v)$ be the distance between two vertices $u, v \in V$ and denote $diam(C) = \max_{u,v \in C} d_G(u, v)$ as the maximum distance between any two vertices $u, v \in C$ for any subset of vertices $C \subseteq V$. In particular, $diam(V)$ refers to the diameter of the whole graph. In the following, let $B(v, r) := \{u \in V : d_G(u, v) \leq r\}$ denote the ball of radius r around vertex v.

5.1.1 Idea: Ball carving

To sample an element of the collection $\mathcal{T}$ we will recursively split our graph using a technique called ball carving.

Definition 5.1 (Ball carving)**.** *Given a graph $G = (V, E)$, a subset $C \subseteq V$ of vertices and upper bound D, where $diam(C) = \max_{u,v \in C} d_G(u, v) \leq D$, partition C into $C_1, \ldots, C_l$ such that*

(A) $\forall i \in \{1, \ldots, l\}, \max_{u,v \in C_i} d_G(u, v) \leq \frac{D}{2}$

(B) $\forall u, v \in V$, $\Pr[u \text{ and } v \text{ not in same partition}] \leq \alpha \cdot \frac{d_G(u,v)}{D}$, *for some α*

Before using ball carving to construct a tree embedding with expected stretch α, we show that a reasonable value $\alpha \in \mathcal{O}(\log n)$ can be achieved.

5.1.2 Ball carving construction

The following algorithm concretely implements *ball carving* and thus gives a split of a given subset of the graph that satisfies properties (A) and (B) as defined.

Algorithm 14 BallCarving$(G = (V, E), C \subseteq V, D)$

if $|C| = 1$ **then**
  return The only vertex in C
else ▷ Say there are n vertices, where $n > 1$
  $\theta \leftarrow$ Uniform random value from the range $[\frac{D}{8}, \frac{D}{4}]$
  Pick a random permutation π on C ▷ Denote π_i as the i^{th} vertex in π
  for $i \in [n]$ **do**
    $V_i \leftarrow B(\pi_i, \theta) \setminus \bigcup_{j=1}^{i-1} B(\pi_j, \theta)$ ▷ $V_1, \ldots, V_n$ is a partition of C
  end for
  return Non-empty sets $V_1, \ldots, V_l$ ▷ V_i can be empty
end if ▷ i.e. $V_i = \emptyset \iff \forall v \in B(\pi_i, \theta), [\exists j < i, v \in B(\pi_j, \theta)]$

Notation Let $\pi : C \to \mathbb{N}$ be an ordering of the vertices C. For vertex $v \in C$, denote $\pi(v)$ as v's position in π and π_i as the i^{th} vertex. That is, $v = \pi_{\pi(v)}$.

Example $C = \{A, B, C, D, E, F\}$ and $\pi(A) = 3, \pi(B) = 2, \pi(C) = 5, \pi(D) = 1, \pi(E) = 6, \pi(F) = 4$. Then π gives an ordering of these vertices as (D, B, A, F, C, E) denoted as π. $E = \pi_6 = \pi_{\pi(E)}$.

Figure 5.1 illustrates the process of ball carving on a set of vertices $C = \{N_1, N_2, \cdots, N_8\}$.

Claim 5.2. BallCarving(G, C, D) *returns partition* $V_1, \ldots, V_l$ *such that*

$$diam(V_i) = \max_{u,v \in V_i} d_G(u, v) \leq \frac{D}{2}$$

for all $i \in \{1, \ldots, l\}$.

Proof. Since $\theta \in [\frac{D}{8}, \frac{D}{4}]$, all constructed balls have diameters $\leq \frac{D}{4} \cdot 2 = \frac{D}{2}$. □

Definition 5.3 (Ball cut)**.** *A ball* $B(u, r)$ *is cut if* BallCarving *puts the vertices in* $B(u, r)$ *in at least two different partitions. We say* V_i *cuts* $B(u, r)$ *if there exist* $w, y \in B(u, r)$ *such that* $w \in V_i$ *and* $y \notin V_i$.

Lemma 5.4. *For any vertex* $u \in C$ *and radius* $r \in \mathbb{R}^+$,

$$\Pr[B(u, r) \text{ is cut in BallCarving}(G,C,D)] \leq \mathcal{O}(\log n) \cdot \frac{r}{D}$$

Proof. Let θ be the randomly chosen ball radius and π be the random permutation on C in BallCarving. We give another ordering of vertices according to the increasing order of their distances to $B(u, r)$. The distance of a fixed point w to the ball $B(u, r)$ is the distance of w to the closest point in $B(u, r)$:

$v_1, v_2, \ldots, v_n$, such that $d_G(B(u, r), v_1) \leq d_G(B(u, r), v_2) \leq \cdots \leq d_G(B(u, r), v_n)$.

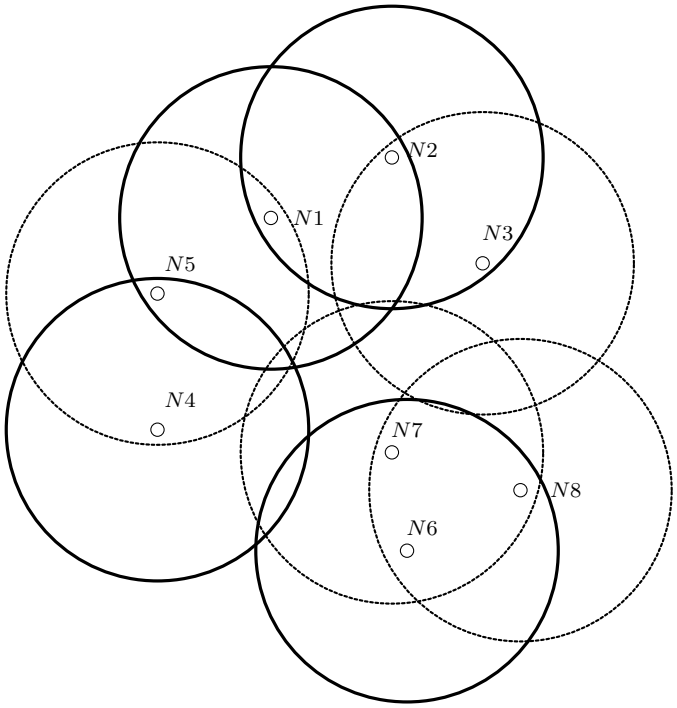


Figure 5.1: Ball carving on a set of vertices $C = \{N_1, N_2, \cdots, N_8\}$. The ordering of nodes is given by a random permutation π. In $Ball(N_1)$ there are vertices N_1, N_2, N_5. So $V_1 = \{N_1, N_2, N_5\}$. In $Ball(N_2)$ there is only N_3 not been carved by the former balls. So $V_2 = \{N_3\}$. All of vertices in $Ball(N_3)$ have been carved. So $V_3 = \phi$. In $Ball(N_4)$, only N_4 has not been carved. $V_4 = \{N_4\}$. In $Ball(N_5)$ all of vertices have been carved. $V_5 = \phi$. $Ball(N_6)$ carves N_6, N_7, N_8, so $V_6 = \{N_6, N_7, N_8\}$. Similar to N_3, N_5, $V_7 = \phi$ and $V_8 = \phi$. Thus C is partitioned into sets $\{N_1, N_2, N_5\}$, $\{N_3\}$, $\{N_4\}$ and $\{N_6, N_7, N_8\}$.

Observation 5.5. *If V_i is the* first *partition that cuts $B(u,r)$, a* necessary condition *is that in the random permutation π, v_i appears before any v_j with $j < i$. (i.e. $\pi(v_i) < \pi(v_j), \forall 1 \leq j < i$).*

Proof. Consider the largest $1 \leq j < i$ such that $\pi(v_j) < \pi(v_i)$:

- If $B(u,r) \cap B(v_j,\theta) = \emptyset$, then $B(u,r) \cap B(v_i,\theta) = \emptyset$. Since $B(u,r) \cap B(v_j,\theta) = \emptyset \iff \forall u' \in B(u,r), u' \notin B(v_j,\theta) \iff \forall u' \in B(u,r), d_G(u',v_j) > \theta \iff d_G(B(u,r),v_j) > \theta$. Also, we know $d_G(B(u,r),v_i) \geq d_G(B(u,r),v_j) > \theta$. None of $B(u,r)$'s vertices will be in $B(v_i,\theta)$, neither in V_i.

- If $B(u,r) \subseteq B(v_j,\theta)$, then vertices in $B(u,r)$ would have been removed before v_i is considered.

- If $B(u,r) \cap B(v_j,\theta) \neq \emptyset$ and $B(u,r) \not\subseteq B(v_j,\theta)$, then V_i is not the first partition that cuts $B(u,r)$ since V_j (or possibly an earlier partition) has already cut $B(u,r)$.

In any case, if there is a $1 \leq j < i$ such that $\pi(v_j) < \pi(v_i)$, V_i does not cut $B(u,r)$. □

Observation 5.6. $\Pr[V_i \textit{ cuts } B(u,r)] \leq \frac{2r}{D/8}$

Proof. We ignore all the other partitions, only considering the sufficient condition for a partition to cut a ball. V_i cuts $B(u,r)$ means $\exists u_1 \in B(u,r), s.t. u_1 \in B(v_i,\theta) \cap \exists u_2 \in B(u,r), s.t. u_2 \notin B(v_i,\theta)$.

- $\exists u_1 \in B(u,r), s.t. u_1 \in B(v_i,\theta) \Rightarrow d_G(u,v_i) - r \leq d_G(u_1,v_i) \leq \theta$.

- $\exists u_2 \in B(u,r), s.t. u_2 \notin B(v_i,\theta) \Rightarrow d_G(u,v_i) + r \geq d_G(u_2,v_i) \geq \theta$.

We get the bounds of $\theta : \theta \in [d_G(u,v_i) - r, d_G(u,v_i) + r]$. Since θ is uniformly chosen from $[\frac{D}{8}, \frac{D}{4}]$,

$$\Pr[\theta \in [d_G(u,v_i) - r, d_G(u,v_i) + r]] \leq \frac{(d_G(u,v_i) + r) - (d_G(u,v_i) - r)}{D/4 - D/8} = \frac{2r}{D/8}$$

Therefore, $\Pr[V_i \text{ cuts } B(u,r)] \leq \Pr[\theta \in [d_G(u,v_i) - r, d_G(u,v_i) + r]] \leq \frac{2r}{D/8}$. □

Thus,

$$
\begin{aligned}
&\Pr[B(u,r) \text{ is cut}] \\
&= \Pr[\bigcup_{i=1}^{n} \text{Event that } V_i \text{ first cuts } B(u,r)] \\
&\leq \sum_{i=1}^{n} \Pr[V_i \text{ first cuts } B(u,r)] && \text{Union bound} \\
&= \sum_{i=1}^{n} \Pr[\pi(v_i) = \min_{j \leq i} \pi(v_j)] \Pr[V_i \text{ cuts } B(u,r)] && \text{Require } v_i \text{ to appear first} \\
&= \sum_{i=1}^{n} \frac{1}{i} \cdot \Pr[V_i \text{ cuts } B(u,r)] && \text{By random permutation } \pi \\
&\leq \sum_{i=1}^{n} \frac{1}{i} \cdot \frac{2r}{D/8} && diam(B(u,r)) \leq 2r,\ \theta \in [\frac{D}{8}, \frac{D}{4}] \\
&= 16 \frac{r}{D} H_n && H_n = \sum_{i=1}^{n} \frac{1}{i} \\
&\in \mathcal{O}(\log(n)) \cdot \frac{r}{D}
\end{aligned}
$$

□

Claim 5.7. BallCarving(G) *returns partition* $V_1, \ldots, V_l$ *such that*

$$\forall u, v \in V, \Pr[u \text{ and } v \text{ not in same partition}] \leq \alpha \cdot \frac{d_G(u,v)}{D}$$

Proof. Let $r = d_G(u,v)$, then v is on the boundary of $B(u,r)$.

$$
\begin{aligned}
&\Pr[u \text{ and } v \text{ not in same partition}] \\
&\leq \Pr[B(u,r) \text{ is cut in BallCarving}] \\
&\leq \mathcal{O}(\log n) \cdot \frac{r}{D} && \text{By Lemma 5.4} \\
&= \mathcal{O}(\log n) \cdot \frac{d_G(u,v)}{D} && \text{Since } r = d_G(u,v)
\end{aligned}
$$

Note: $\alpha = \mathcal{O}(\log n)$ as previously claimed. □

5.1.3 Construction of T

Using ball carving, ConstructT recursively partitions the vertices of a given graph until there is only one vertex remaining. At each step, the upper

bound D indicates the maximum distance between the vertices of C. The first call of ConstructT starts with $C = V$ and $D = diam(V)$. Figure 5.2 illustrates the process of building a tree T from a given graph G.

Algorithm 15 ConstructT$(G = (V, E), C \subseteq V, D)$

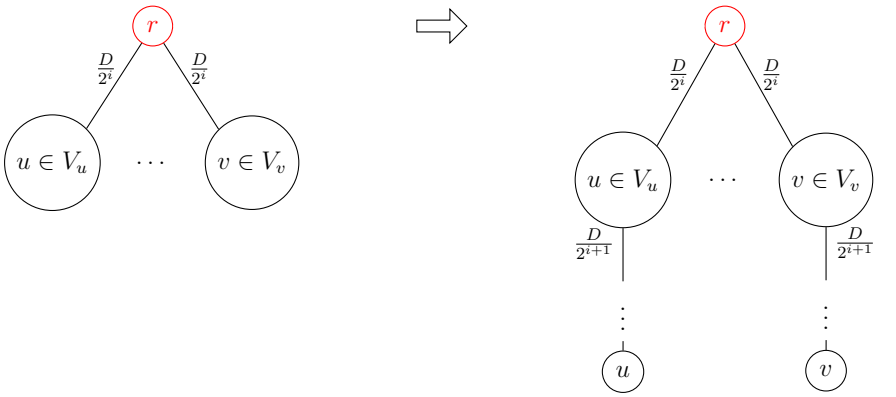
if $|C| = 1$ **then**
  return The only vertex in C ▷ Return an actual vertex from $V(G)$
else
  $V_1, \dots, V_l \leftarrow$ BallCarving(G, C, D) ▷ $\max_{u,v,\in V_i} d_G(u, v) \leq \frac{D}{2}$
  Create auxiliary vertex r ▷ r is root of current subtree
  for $i \in \{1, \dots, l\}$ **do**
    $r_i \leftarrow$ ConstructT$(G, V_i, \frac{D}{2})$
    Add edge $\{r, r_i\}$ with weight D
  end for
  return Root of subtree r ▷ Return an auxiliary vertex r
end if

Lemma 5.8. *For any two vertices $u, v \in V$ and $i \in \mathbb{N}$, if T separates u and v at level i, then $\frac{2D}{2^i} \leq d_T(u, v) \leq \frac{4D}{2^i}$, where $D = diam(V)$.*

Proof. If T splits u and v at level i, then the path from u to v in T has to include two edges of length $\frac{D}{2^i}$, hence $d_T(u, v) \geq \frac{2D}{2^i}$. To be precise,

$$\frac{2D}{2^i} \leq d_T(u, v) = 2 \cdot (\frac{D}{2^i} + \frac{D}{2^{i+1}} + \cdots) \leq \frac{4D}{2^i}$$

See picture — r is the auxiliary node at level i which splits nodes u and v.



□

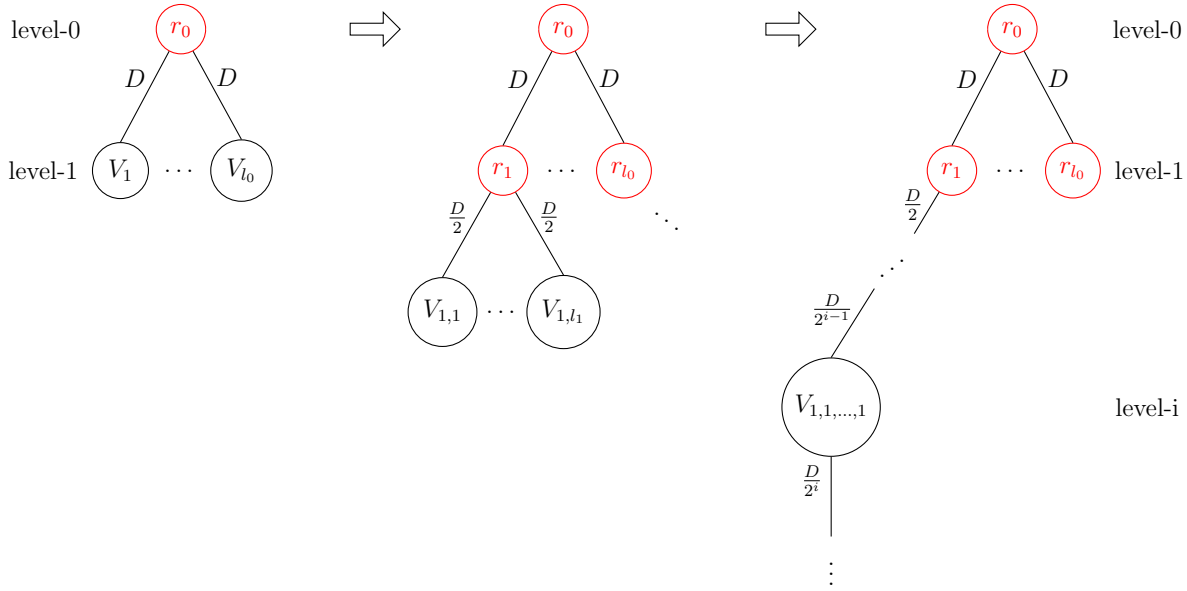


Figure 5.2: Recursive ball carving with $\lceil \log_2(D) \rceil$ levels. Red vertices are auxiliary nodes that are not in the original graph G. Denoting the root as the 0^{th} level, edges from level i to level $i+1$ have weight $\frac{D}{2^i}$.

Remark If $u, v \in V$ separate *before* level i, then $d_T(u,v)$ must still include the two edges of length $\frac{D}{2^i}$, hence $d_T(u,v) \geq \frac{2D}{2^i}$.

Claim 5.9. ConstructT$(G, C = V, D = diam(V))$ *returns a tree* T *such that*

$$d_G(u,v) \leq d_T(u,v)$$

Proof. Consider $u, v \in V$. Say $\frac{D}{2^i} \leq d_G(u,v) \leq \frac{D}{2^{i-1}}$ for some $i \in \mathbb{N}$. By property (A) of ball carving, T will separate them at, or before, level i. By Lemma 5.8, $d_T(u,v) \geq \frac{2D}{2^i} = \frac{D}{2^{i-1}} \geq d_G(u,v)$. □

Claim 5.10. ConstructT$(G, C = V, D = diam(V))$ *returns a tree* T *such that*

$$\mathbb{E}[d_T(u,v)] \leq 4\alpha \log(D) \cdot d_G(u,v)$$

Proof. Consider $u, v \in V$. Define $\mathcal{E}_i$ as the event that "vertices u and v get separated at the i^{th} level", for $i \in \mathbb{N}$. By recursive nature of ConstructT, the subset at the i^{th} level has distance at most $\frac{D}{2^i}$. So, property (B) of ball carving tells us that $\Pr[\mathcal{E}_i] \leq \alpha \cdot \frac{d_G(u,v)}{D/2^i}$. Then,

$$\begin{aligned}
\mathbb{E}[d_T(u,v)] &= \sum_{i=0}^{\log(D)-1} \Pr[\mathcal{E}_i] \cdot [d_T(u,v), \text{ given } \mathcal{E}_i] && \text{Definition of expectation} \\
&\leq \sum_{i=0}^{\log(D)-1} \Pr[\mathcal{E}_i] \cdot \frac{4D}{2^i} && \text{By Lemma 5.8} \\
&\leq \sum_{i=0}^{\log(D)-1} (\alpha \cdot \frac{d_G(u,v)}{D/2^i}) \cdot \frac{4D}{2^i} && \text{Property (B) of ball carving} \\
&= 4\alpha \log(D) \cdot d_G(u,v) && \text{Simplifying}
\end{aligned}$$

□

If we apply Claim 5.7 with Claim 5.10, we get

$$\mathbb{E}[d_T(u,v)] \leq \mathcal{O}(\log(n)\log(D)) \cdot d_G(u,v)$$

We can remove the $\log(D)$ factor, and prove that the tree embedding built by the algorithm has stretch factor $c = \mathcal{O}(\log n)$. For that, we need a tighter analysis of the ball carving process, by only considering vertices that may cut $B(u, d_G(u,v))$ instead of all n vertices, in each level of the recursive partitioning. This sharper analysis is presented as a separate section below. See Theorem 5.13 in Section 5.1.4.

5.1.4 Sharper Analysis of Tree Embedding

If we apply Claim 5.7 with Claim 5.10, we get $\mathbb{E}[d_T(u,v)] \leq \mathcal{O}(\log(n)\log(D)) \cdot d_G(u,v)$. To remove the $\log(D)$ factor, so that stretch factor $c = \mathcal{O}(\log n)$, a tighter analysis is needed by only considering vertices that may cut $B(u, d_G(u,v))$ instead of all n vertices.

Tighter analysis of ball carving

Fix arbitrary vertices u and v. Let $r = d_G(u,v)$. Recall that θ is chosen uniformly at random from the range $[\frac{D}{8}, \frac{D}{4}]$. A ball $B(v_i, \theta)$ can cut $B(u,r)$ only when $d_G(u,v_i) - r \leq \theta \leq d_G(u,v_i) + r$. In other words, one only needs to consider vertices v_i such that $\frac{D}{8} - r \leq \theta - r \leq d_G(u,v_i) \leq \theta + r \leq \frac{D}{4} + r$.

Lemma 5.11. *For $i \in \mathbb{N}$, if $r > \frac{D}{16}$, then $\Pr[B(u,r)$ is cut$] \leq \frac{16r}{D}$*

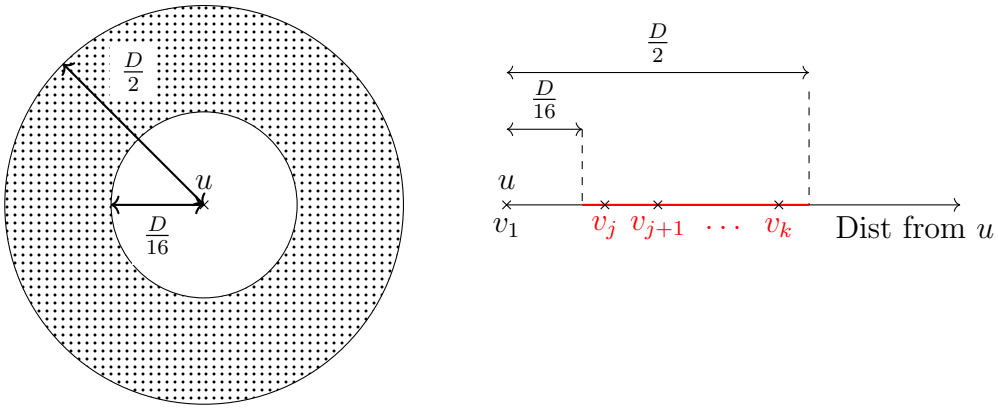
Proof. If $r > \frac{D}{16}$, then $\frac{16r}{D} > 1$. As $\Pr[B(u,r)$ is cut at level $i]$ is a probability ≤ 1, the claim holds. □

Remark Although lemma 5.11 is not a very useful inequality (since any probability ≤ 1), we use it to partition the value range of r so that we can say something stronger in the next lemma.

Lemma 5.12. *For $i \in \mathbb{N}$, if $r \leq \frac{D}{16}$, then*

$$\Pr[B(u,r) \text{ is cut}] \leq \frac{r}{D}\mathcal{O}(\log(\frac{|B(u,D/2)|}{|B(u,D/16)|}))$$

Proof. V_i cuts $B(u,r)$ only if $\frac{D}{8} - r \leq d_G(u,v_i) \leq \frac{D}{4} + r$, we have $d_G(u,v_i) \in [\frac{D}{16}, \frac{5D}{16}] \subseteq [\frac{D}{16}, \frac{D}{2}]$.



Suppose we arrange the vertices in ascending order of distance from u: $u = v_1, v_2, \ldots, v_n$. Denote:

- $j - 1 = |B(u, \frac{D}{16})|$ as the number of nodes that have distance $\leq \frac{D}{16}$ from u
- $k = |B(u, \frac{D}{2})|$ as the number of nodes that have distance $\leq \frac{D}{2}$ from u

We see that only vertices $v_j, v_{j+1}, \ldots, v_k$ have distances from u in the range $[\frac{D}{16}, \frac{D}{2}]$. Pictorially, only vertices in the shaded region could possibly cut $B(u,r)$. As before, let $\pi(v)$ be the ordering in which vertex v appears in

random permutation π. Then,

$$
\begin{aligned}
&\Pr[B(u,r) \text{ is cut}] \\
&= \Pr[\bigcup_{i=j}^{k} \text{Event that } V_i \text{ cuts } B(u,r)] && \text{Only } V_j, V_{j+1}, \ldots, V_k \text{ can cut} \\
&\leq \sum_{i=j}^{k} \Pr[\pi(v_i) < \min_{z<i}\{\pi(v_z)\}] \cdot \Pr[V_i \text{ cuts } B(u,r)] && \text{Union bound} \\
&= \sum_{i=j}^{k} \frac{1}{i} \cdot \Pr[V_i \text{ cuts } B(u,r)] && \text{By random permutation } \pi \\
&\leq \sum_{i=j}^{k} \frac{1}{i} \cdot \frac{2r}{D/8} && diam(B(u,r)) \leq 2r, \theta \in [\frac{D}{8}, \frac{D}{4}] \\
&= \frac{r}{D}(H_k - H_j) && \text{where } H_k = \sum_{i=1}^{k} \frac{1}{i} \\
&\in \frac{r}{D}\mathcal{O}(\log(\frac{|B(u, D/2)|}{|B(u, D/16)|})) && \text{since } H_k \in \Theta(\log(k))
\end{aligned}
$$

□

Plugging into ConstructT

Recall that ConstructT is a recursive algorithm which handles graphs of diameter $\leq \frac{D}{2^i}$ at level i. For a given pair of vertices u and v, there exists $i^* \in \mathbb{N}$ such that $\frac{D}{2^{i^*}} \leq r = d_G(u,v) \leq \frac{D}{2^{i^*-1}}$. In other words, $\frac{D}{2^{i^*-4}}\frac{1}{16} \leq r \leq \frac{D}{2^{i^*-5}}\frac{1}{16}$. So, lemma 5.12 applies for levels $i \in [0, i^* - 5]$ and lemma 5.11 applies for levels $i \in [i^* - 4, \log(D) - 1]$.

Theorem 5.13. $\mathbb{E}[d_T(u,v)] \in \mathcal{O}(\log n) \cdot d_G(u,v)$

Proof. As before, let $\mathcal{E}_i$ be the event that "vertices u and v get separated at the i^{th} level. For $\mathcal{E}_i$ to happen, the ball $B(u,r) = B(u, d_G(u,v))$ must be cut at level i, so $\Pr[\mathcal{E}_i] \leq \Pr[B(u,r)$ is cut at level $i]$.

$$
\begin{aligned}
&\mathbb{E}[d_T(u,v)] \\
&= \sum_{i=0}^{\log(D)-1} \Pr[\mathcal{E}_i] \cdot \Pr[d_T(u,v), \text{ given } \mathcal{E}_i] && (1)\\
&\leq \sum_{i=0}^{\log(D)-1} \Pr[\mathcal{E}_i] \cdot \frac{4D}{2^i} && (2)\\
&= \sum_{i=0}^{i^*-5} \Pr[\mathcal{E}_i] \cdot \frac{4D}{2^i} + \sum_{i=i^*-4}^{\log(D)-1} \Pr[\mathcal{E}_i] \cdot \frac{4D}{2^i} && (3)\\
&\leq \sum_{i=0}^{i^*-5} \frac{r}{D/2^i} \mathcal{O}(\log(\frac{|B(u,D/2^{i+1})|}{|B(u,D/2^{i+4})|})) \cdot \frac{4D}{2^i} + \sum_{i=i^*-4}^{\log(D)-1} \Pr[\mathcal{E}_i] \cdot \frac{4D}{2^i} && (4)\\
&\leq \sum_{i=0}^{i^*-5} \frac{r}{D/2^i} \mathcal{O}(\log(\frac{|B(u,D/2^{i+1})|}{|B(u,D/2^{i+4})|})) \cdot \frac{4D}{2^i} + \sum_{i=i^*-4}^{\log(D)-1} \frac{16r}{D/2^{i^*-4}} \cdot \frac{4D}{2^i} && (5)\\
&= 4r \cdot \sum_{i=0}^{i^*-5} \mathcal{O}(\log(\frac{|B(u,D/2^{i+1})|}{|B(u,D/2^{i+4})|})) + \sum_{i=i^*-4}^{\log(D)-1} 4 \cdot 2^{i^*-i} \cdot r && (6)\\
&\leq 4r \cdot \sum_{i=0}^{i^*-5} \mathcal{O}(\log(\frac{|B(u,D/2^{i+1})|}{|B(u,D/2^{i+4})|})) + 2^7 r && (7)\\
&= 4r \cdot \mathcal{O}(\log(n)) + 2^7 r && (8)\\
&\in \mathcal{O}(\log n) \cdot r
\end{aligned}
$$

(1) Definition of expectation

(2) By Lemma 5.8

(3) Split into cases: $\frac{D}{2^{i^*-4}} \frac{1}{16} \leq r \leq \frac{D}{2^{i^*-5}} \frac{1}{16}$

(4) By Lemma 5.12

(5) By Lemma 5.11 with respect to $D/2^{i^*-4}$

(6) Simplifying

(7) Since $\sum_{i=i^*-4}^{\log(D)-1} 2^{i^*-i} \leq 2^5$

(8) $\log(\frac{x}{y}) = \log(x) - \log(y)$ and $|B(u,\infty)| \leq n$

□

5.1.5 Removing auxiliary nodes from tree T

Note in Figure 5.2 that we introduce auxiliary vertices in our tree construction. We next would like to build a tree T without additional vertices (i.e. such that $V(T) = V(G)$). In this section, the pseudo-code CONTRACT explains how to remove the auxiliary vertices. It remains to show that the produced tree still preserves desirable properties of a tree embedding.

Algorithm 16 CONTRACT(T)

while T has an edge (u, w) such that $u \in V$ and w is an auxiliary node **do**
    Contract edge (u, w) by merging subtree rooted at u into w
    Identify the new node as u
end while
Multiply weight of every edge by 4
return Modified T'

Claim 5.14. CONTRACT *returns a tree T' such that*

$$d_T(u, v) \leq d_{T'}(u, v) \leq 4 \cdot d_T(u, v)$$

Proof. Suppose auxiliary node w, at level i, is the closest common ancestor for two arbitrary vertices $u, v \in V$ in the original tree T. Then,

$$d_T(u, v) = d_T(u, w) + d_T(w, v) = 2 \cdot (\sum_{j=i}^{\log D} \frac{D}{2^j}) \leq 4 \cdot \frac{D}{2^i}$$

Since we do not contract actual vertices, at least one of the (u, w) or (v, w) edges of weight $\frac{D}{2^i}$ will remain. Multiplying the weights of all remaining edges by 4, we get $d_T(u, v) \leq 4 \cdot \frac{D}{2^i} = d_{T'}(u, v)$.

Suppose we only multiply the weights of $d_T(u, v)$ by 4, then $d_{T'}(u, v) = 4 \cdot d_T(u, v)$. Since we contract edges, $d_{T'}(u, v)$ can only decrease, so $d_{T'}(u, v) \leq 4 \cdot d_T(u, v)$. □

Remark Claim 5.14 tells us that one can construct a tree T' without auxiliary variables by incurring an additional constant factor overhead.

5.2 Application: Buy-at-bulk network design

Definition 5.15 (Buy-at-bulk network design problem)**.** *Consider a graph $G = (V, E)$ with edge lengths l_e for $e \in E$. Let $f : \mathbb{R}^+ \to \mathbb{R}^+$ be a sub-additive*

cost function. That is, $f(x+y) \leq f(x) + f(y)$*. Given* k *commodity triplets* (s_i, t_i, d_i)*, where* $s_i \in V$ *is the source,* $t_i \in V$ *is the target, and* $d_i \geq 0$ *is the demand for the* i^{th} *commodity, find a capacity assignment on edges* c_e *(for all edges) such that*

- $\sum_{e \in E} f(c_e) \cdot l_e$ *is minimized*
- $\forall e \in E, c_e \geq$ *Total flow passing through it*
- *Flow conservation is satisfied and every commodity's demand is met*

Remark If f is linear (e.g. $f(x+y) = f(x) + f(y)$), one can obtain an optimum solution by finding the shortest path $s_i \to t_i$ for each commodity i, then summing up the required capacities for each edge.

Algorithm 17 NetworkDesign($G = (V, E)$)

```
c_e = 0, ∀e ∈ E                                  ▷ Initialize capacities
T ← ConstructT(G)          ▷ Build probabilistic tree embedding T of G
T ← Contract(T)                          ▷ V(T) = V(G) after contraction
for i ∈ {1, ..., k} do                                ▷ Solve problem on T
    P^T_{s_i,t_i} ← Find shortest s_i − t_i path in T      ▷ It is unique in a tree
    for Edge {u, v} of P^T_{s_i,t_i} in T do
        P^G_{u,v} ← Find shortest u − v path in G
        c_e ← c_e + d_i, for each edge in e ∈ P^G_{u,v}
    end for
end for
return {e ∈ E : c_e}
```

Let us denote $I = (G, f, \{s_i, t_i, d_i\}_{i=1}^k)$ as the given instance. Let $OPT(I, G)$ be the optimal solution on G. The general idea of our algorithm NetworkDesign is first transforming the original graph G into a tree T by probabilistic tree embedding method, contracting the tree as T', then finding an optimal solution on T' and map it back to graph G. Let $A(I, G)$ be the solution produced by our algorithm on graph G. Denote the costs as $|OPT(I, G)|$ and $|A(I, T)|$ respectively.

We now compare the solutions $OPT(I, G)$ and $A(I, T)$ by comparing edge costs $(u, v) \in E$ in G and tree embedding T. For the three claims below, we provide just proof sketches, without diving into the notation-heavy calculations. Please refer to Section 8.6 in [WS11] for the formal arguments.

Claim 5.16. $|A(I, G)|$ *using edges in* $G \leq |A(I, T)|$ *using edges in* T*.*

Proof. (Sketch) This follows from two facts: (1) For any edge $xy \in T$, all of the paths sent in $A(I,T)$ along edge xy are now sent along the shortest path connecting x and y in G, which by the first property of the tree embedding has length at most equal to the length of the xy edge. (2) Sevaral paths of , corresponding to different edges in the tree T, might end up being routed through the same graph G edge e. But by subadditivity, the cost on edge e is at most the summation of the costs of those paths. □

Claim 5.17. $|A(I,T)|$ *using edges in* $T \leq |OPT(I,T)|$ *using edges in* T.

Proof. (Sketch) Since shortest path in a tree is unique, $A(I,T)$ is optimum for T. So, any other flow assignment has to incur higher edge capacities. □

Claim 5.18. $\mathbb{E}[|OPT(I,T)|$ *using edges in* $T] \leq \mathcal{O}(\log n) \cdot |OPT(I,G)|$

Proof. (Sketch) Using subadditivity, we can upper bound the cost of $OPT(I,T)$ by the summation over all edges $e \in G$ of the cost for the capacity of this edge in the optimal solution $OPT(I,G)$ multiplied by the length of the path connecting the two endpoints of e in the tree T. We know that T stretches edges by at most a factor of $\mathcal{O}(\log n)$, in expectation. Hence, the cost is in expectation upper bounded by the summation over all edges $e \in G$ of the cost for the capacity of this edge in the optimal solution $OPT(I,G)$ multiplied by the length of the edge e in G. The latter is simply the cost of $OPT(I,G)$. □

By the three claims above, NetworkDesign gives a $\mathcal{O}(\log n)$-approximation to the buy-at-bulk network design problem, in expectation.

Chapter 6

L1 metric embedding & sparsest cut

In this section, we see how viewing graphs as geometric objects helps us to solve some cut problems. You can find more on this topic in the seminal work of Linial, London, and Rabinovich [LLR95].

6.1 Warm up: Min s-t Cut

In this section we will study the minimum s-t cut problem in undirected graphs. Given a graph $G = (V, E)$ we define a cut as a partition of the vertices in two sets$(S, V \setminus S)$. We will typically work with connected graphs, in this situation any non-trivial cut $(S \neq \emptyset, V \setminus S \neq \emptyset)$ will have some edges going across the cut. The minimum s-t cut problem searches for cuts that separate a source $s \in V$ from a target $t \in V$ minimizing the capacity of the edges across the cut.
An intuitive way of thinking of the minimum s-t cut problem is considering the capacities as prices to pay to remove the edges and trying to disconnect the node s from t while minimizing the price paid.
The minimum s-t cut problem can be solved in polynomial time by formulating it as a linear programming. We will follow a slightly different approach, solving a relaxation of the problem and constructing a minimum cost cut using the solution of the relaxed problem. The solution that we present is not computationally better than solving the minimum s-t cut directly but presents a framework that can be generalized to more complex cut problems.

The mathematical formulation of the minimum cut problem is:

$$\min_{\text{cut } S\subset V} \sum_{\substack{e=\{u,v\}\in E \\ u\in S,\ v\in V\setminus S}} c_e = \min_{\text{cut } S\subset V} \sum_{e=\{u,v\}\in E} c_e|\mathbb{1}_S(u) - \mathbb{1}_S(v)| \tag{6.1}$$

where $\mathbb{1}_S(u) = \begin{cases} 1, & u \in S \\ 0, & else \end{cases}$ is the indicator function of the set S and $|\mathbb{1}_S(u) - \mathbb{1}_S(v)| = 1$ when u and v are not on the same side of the cut. Note that for any cut $(S, V \setminus S)$, $|\mathbb{1}_S(u) - \mathbb{1}_S(v)|$ defines a pseudo-metric.

Definition 6.1. *A pseudo-metric on a set $\mathcal{S}$ is a function $d : \mathcal{S} \times \mathcal{S} \longrightarrow \mathbb{R}$ such that*

(A) identity: $\forall s \in \mathcal{S},\ d(s,s) = 0$

(B) non-negativity: $\forall s,t \in \mathcal{S},\ d(s,t) \geq 0$

(C) symmetry: $\forall s,t \in \mathcal{S},\ d(s,t) = d(t,s)$

(D) triangular inequality: $\forall s,t,u \in \mathcal{S},\ d(s,u) \leq d(s,t) + d(t,u)$

A pseudo-metric is a generalization of the notion of a metric where there is no requirement that $d(s,t) = 0 \Rightarrow s = t$

Since $|\mathbb{1}_S(u) - \mathbb{1}_S(v)|$ defines a pseudo-metric that separates the source and the target, $|\mathbb{1}_S(s) - \mathbb{1}_S(t)| = 1$ we can relax the minimization problem allowing any pseudo-metric that satisfies $d(s,t) = 1$. This minimization problem can be formulated as the following linear program:

$$\begin{aligned} \min_{d_{uv}} \quad & \sum_{e=\{u,v\}\in E} c_e d_{uv} & \\ \textit{subject to} \quad & d_{uv} \geq 0 & \forall u,v \in V \\ & d_{uv} = d_{vu} & \forall u,v \in V \\ & d_{uw} \leq d_{uv} + d_{vw} & \forall u,v,w \in V \\ & d_{st} = 1 & \end{aligned}$$

With the solution of the linear program d^* we can find the minimum s-t cut.

Claim 6.2. *For the minimum s-t cut problem, the relaxed optimization problem has the same optimum value as the original problem.*

$$\sum_{e\in E} c_e d^*(e) = \sum_{e=\{u,v\}\in E} c_e|\mathbb{1}_{S^*}(u) - \mathbb{1}_{S^*}(v)|$$

Moreover, given any optimal pseudo-metric d^*, we can explicitly construct a cut S^* that achieves the optimal value of the original problem.

Proof. First we note that since d^* is a relaxation of the original problem, for any cut $(S, V \setminus S)$ it holds that

$$\sum_{e \in E} c_e d^*(e) \leq \sum_{e=\{u,v\} \in E} c_e |\mathbb{1}_S(u) - \mathbb{1}_S(v)|$$

To prove the equality it suffices to find a cut S^* that satisfies

$$\sum c_e d^*(u,v) \geq \sum c_e |\mathbb{1}_{S^*}(u) - \mathbb{1}_{S^*}(v)|$$

Such cut S^* will be a minimum s-t cut. To find it we first observe that since d^* is the optimum of the linear program it must satisfy the constraint $d^*(s,t) = 1$. Then we order the vertices according to their distance to the source, defining $v_i \in V$ as the ith farthest vertex to the source and $x_i = d^*(s, v_i)$ it's corresponding distance to the source. We also define the increments $y_i = x_{i+1} - x_i$.
Now we can define the *natural cuts* as the cuts that separate the vertices according to their distance to the source, $S_i = \{v \in V \ | d^*(s,v) \leq x_i\}$. Figure 6.1 shows the vertices and the natural cuts.
We now show that one of the natural cuts achieves the optimum and there-

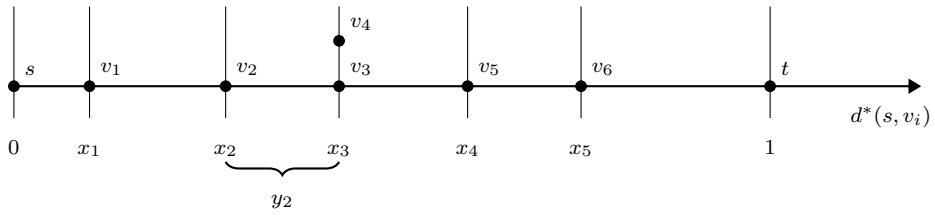


Figure 6.1: The natural cuts x_i and the corresponding increments y_i

fore is a minimum s-t cut. First we observe that for any edge $e = \{v_i, v_j\}$ with $x_i \leq x_j$ *w.l.g.* the triangular inequality gives us $d(s,v_j) \leq d(s,v_i) + d(v_i,v_j)$, hence

$$d(e) = d(v_i,v_j) \geq d(s,v_j) - d(s,v_i) = x_j - x_i = \sum_{\text{cuts } k \text{ that are crossed by the edge } e} y_k$$

Therefore we can write:

$$\begin{aligned}\sum_{e\in E} c_e d^*(e) &\geq \sum_{e\in E} c_e \sum_{\text{cuts } k \text{ that are crossed by the edge } e} y_k \\ &= \sum_{k\text{-th cut}} y_k \sum_{\text{edges } e \text{ that cross the } k\text{-th cut}} c_e \\ &\geq \min_{k \text{ cuts}} \left\{ \sum_{\text{edges } e \text{ that cross the } k\text{-th cut}} c_e \right\} \sum_{k\text{-th cut}} y_k\end{aligned}$$

Since $d^*(s,t) = 1$ and the vertices are ordered, $\sum_{k=0}^{n-1} y_k = x_n \geq x_t = 1$, hence

$$\begin{aligned}&\geq \min_{k \text{ cuts}} \left\{ \sum_{\text{edges } e \text{ that cross the } k\text{-th cut}} c_e \right\} \\ &= \sum_{e=\{u,v\}\in E} c_e |\mathbb{1}_{S^*}(u) - \mathbb{1}_{S^*}(v)|\end{aligned}$$

Where S^* is defined as the natural cut with the lowest capacity, i.e. it minimizes $\sum_{\text{edges } e \text{ that cross the } k\text{-th cut}} c_e$. Therefore the cut S^* achieves a lower cost than the cost of the relaxed problem and is a minimum s-t cut. □

6.2 Sparsest Cut via L1 Embedding

We now move to the sparsest cut problem, which is finding a cut $S \subseteq V$ that minimizes the following objective function:

$$\mathcal{L}(S) = \frac{|E(S \setminus V, S)|}{|S||V \setminus S|} \tag{6.2}$$

where $E(S, V \setminus S)$ denotes the set of edges that cross the cut $\{e = (u,v) \in E | u \in S \text{ and } v \in V \setminus S\}$.

Observation 6.3 (Sparse cuts in complete graphs)**.** *For complete graphs, the number of edges between any cut S and $V \setminus S$ is exactly $|S| \cdot |V \setminus S|$, so $\mathcal{L}(S) = 1$.*

We can interpret the objective function (6.2) as comparison between the cut S in the given graph and the cut S in a complete graph on the same vertices. We also note that sparse cuts penalize cuts where the two sets have

very different sizes.

The sparsest cut problem is NP-hard. In this section we will provide a polynomial time approximation using ideas closely related to the ideas exposed in the previous section. We will prove the following theorem.

Theorem 6.4 (Sparsest cut approximation)**.** *Let $S_{OPT} \subseteq V$ be an optimal solution to the sparsest cut problem. There is a cut $S \subseteq V$, which can be computed in polynomial time such that*

$$\mathcal{L}(S) \leq \mathcal{O}(\log(n))\mathcal{L}(S_{OPT}) \tag{6.3}$$

with high probability.

We start by formulating the sparsest cut problem with indicator functions:

$$\mathcal{L}(S) = \frac{\sum_{\{u,v\}\in E} |\mathbb{1}_S(u) - \mathbb{1}_S(v)|}{\sum_{u,v\in V} |\mathbb{1}_S(u) - \mathbb{1}_S(v)|}. \tag{6.4}$$

Now, as we did in the minimum s-t cut problem, we will relax the problem. We observe that for any cut $(S, V \setminus S)$ the expression $d_S(u,v) = \frac{|\mathbb{1}_S(u)-\mathbb{1}_S(v)|}{\sum_{u,v\in V}|\mathbb{1}_S(u)-\mathbb{1}_S(v)|}$ defines a pseudo-metric and satisfies $\sum_{u,v\in V} d_S(u,v) = \frac{|\mathbb{1}_S(u)-\mathbb{1}_S(v)|}{\sum_{u,v\in V}|\mathbb{1}_S(u)-\mathbb{1}_S(v)|} = 1$. Therefore we will relax the problem to any pseudo-metric that satisfies $\sum_{u,v\in V} d_{uv} = 1$. We write the relaxed problem as the following linear program:

Definition 6.5 (Sparsest cut relaxation)**.**

$$\begin{aligned}
&\text{minimise} && \sum_{e=\{u,v\}\in E} d_{uv} && && (6.5)\\
&\text{subject to} && d_{uv} \geq 0 && \forall u,v \in V,\\
& && d_{uv} = d_{vu} && \forall u,v \in V,\\
& && d_{uw} \leq d_{uv} + d_{vw} && \forall u,v,w \in V,\\
& && \sum_{u,v\in V} d_{uv} = 1
\end{aligned}$$

Now from the optimal pseudo-metric d^* we would like to obtain a cut that approximates the sparsest cut. Finding such cut from d^* is not easy, therefore we approximate the optimal pseudo-metric d^* with the $L1$ distance of an $\mathbb{R}^k$ space for some k. This approximation can be done using the following embedding result.

Theorem 6.6 (L1 embedding of a graph)**.** *Let d be an arbitrary pseudo-metric on V, e.g. Then there exists an integer $k = \mathcal{O}(\log^2(n))$[1] and a map $f : V \to \mathbb{R}^k$ which preserves distances up to a $\log(n)$ factor:*

$$\frac{d^*(u,v)}{\Theta(\log(n))} \leq \|f(v) - f(u)\|_1 \leq d^*(u,v) \tag{6.6}$$

for all $u, v \in V$ with high probability. Moreover, f can be computed in polynomial time.

The $L1$ embedding theorem will be proved in next section and can be assumed as a black-box for now. The approximation of the optimal pseudo-metric d^* by a $L1$ metric will allow us to find a cut $(S^*, V \setminus S^*)$ with lower cost than the cost given by the $L1$ metric, this cut will have an "aproximatively lower" cost than the cost of the relaxed problem. To find the cut we first prove this useful lemma.

Lemma 6.7. *for any non-negative numbers $a_1, b_1, a_2, b_2 \ldots, a_k, b_k \geq 0$ with $\sum_{i=1}^k b_i \neq 0$ we have:*

$$\frac{\sum_{i=1}^k a_i}{\sum_{i=1}^k b_i} \geq \min_{j=1}^{k} \frac{a_j}{b_j}.$$

Proof. For $k = 1$ this clearly holds, so suppose it holds for some $k \geq 1$. We assume that $b_{k+1} \neq 0$ and $\sum_{i=1}^k b_i \neq 0$, otherwise the inequality quickly follows from non-negativity of $a_1, \ldots, a_{k+1}$. Then

$$\begin{aligned}
\frac{a_{k+1} + \sum_{i=1}^k a_i}{b_{k+1} + \sum_{i=1}^k b_i} &\geq \frac{\frac{a_{k+1}}{b_{k+1}} + \frac{1}{b_{k+1}} \frac{\sum_{i=1}^k a_i}{\sum_{i=1}^k b_i} \sum_{i=1}^k b_i}{1 + \sum_{i=1}^k \frac{b_i}{b_{k+1}}} \\
&\geq \frac{\min_{j=1}^{k+1} \frac{a_j}{b_j} + \min_{j=1}^{k} \frac{a_j}{b_j} \sum_{i=1}^k \frac{b_i}{b_{k+1}}}{1 + \sum_{i=1}^k \frac{b_i}{b_{k+1}}} \\
&\geq \frac{\min_{j=1}^{k+1} \frac{a_j}{b_j} (1 + \sum_{i=1}^k \frac{b_i}{b_{k+1}})}{1 + \sum_{i=1}^k \frac{b_i}{b_{k+1}}} \\
&= \min_{j=1}^{k+1} \frac{a_j}{b_j}.
\end{aligned}$$

Thus the proof of the claim follows by induction. □

Now we can use the previous lemma to find cut $(S, V \setminus S)$ with lower cost than the $L1$ cost.

[1] The dimension k can be reduced to $k = \mathcal{O}(\log(n))$, see [Ind01].

Claim 6.8 (Cut extraction). *There is a cut $S \subseteq V$ such that*

$$\frac{\sum_{\{u,v\}\in E} |\mathbb{1}_S(u) - \mathbb{1}_S(v)|}{\sum_{u,v\in V} |\mathbb{1}_S(u) - \mathbb{1}_S(v)|} \leq \frac{\sum_{\{u,v\}\in E} \|f(u) - f(v)\|_1}{\sum_{u,v\in V} \|f(u) - f(v)\|_1}$$

Where f is the L1 embedding defined in theorem 6.6.

Proof. We write out the L1-norm as a sum of absolute values

$$\frac{\sum_{\{u,v\}\in E} \sum_{i=1}^{k} |f_i(u) - f_i(v)|}{\sum_{u,v\in V} \sum_{i=1}^{k} |f_i(u) - f_i(v)|}$$

where $f_i : V \to \mathbb{R}$ is the i^{th} component of f. Now we use lemma 6.7 to reduce the problem to one dimension:

$$\frac{\sum_{\{u,v\}\in E} \sum_{i=1}^{k} |f_i(u) - f_i(v)|}{\sum_{u,v\in V} \sum_{i=1}^{k} |f_i(u) - f_i(v)|} \geq \min_{j=1,\ldots,k} \frac{\sum_{\{u,v\}\in E} |f_j(u) - f_j(v)|}{\sum_{u,v\in V} |f_j(u) - f_j(v)|} \tag{6.7}$$

Let j_{min} be the index that minimizes the quotient on the right hand side. Since the objective function is invariant to affine transformation to f ($\hat{f}(u) = a \cdot f(u) + b$), we may assume that

$$\max_{u\in V} f_{j_{min}}(u) = 1 \quad \text{and} \quad \min_{u\in V} f_{j_{min}}(u) = 0.$$

Now let $\tau \in [0, 1]$ be a uniformly distributed threshold and define the cut

$$S_\tau = \{v \in V : f_{j_{min}}(v) \leq \tau\} \tag{6.8}$$

Note that

$$|\mathbb{1}_{S_\tau}(u) - \mathbb{1}_{S_\tau}(v)| = \begin{cases} 0 & \text{if } f_{j_{min}}(u), f_{j_{min}}(v) \leq \tau \\ 1 & \text{if } \min\{f_{j_{min}}(u), f_{j_{min}}(v)\} \leq \tau < \max\{f_{j_{min}}(u), f_{j_{min}}(v)\} \\ 0 & \text{if } f_{j_{min}}(u), f_{j_{min}}(v) > \tau \end{cases}$$

and

$$\mathbb{E}[|\mathbb{1}_{S_\tau}(u) - \mathbb{1}_{S_\tau}(v)|] = |f_{j_{min}}(u) - f_{j_{min}}(v)|.$$

Putting everything together, we obtain

$$\begin{aligned} \frac{\sum_{\{u,v\}\in E} \|f(u) - f(v)\|_1}{\sum_{u,v\in V} \|f(u) - f(v)\|_1} &\geq \frac{\sum_{\{u,v\}\in E} |f_{j_{\min}}(u) - f_{j_{\min}}(v)|}{\sum_{u,v\in V} |f_{j_{\min}}(u) - f_{j_{\min}}(v)|} \\ &= \frac{\sum_{\{u,v\}\in E} \mathbb{E}[|\mathbb{1}_{S_\tau}(u) - \mathbb{1}_{S_\tau}(v)|]}{\sum_{u,v\in V} \mathbb{E}[|\mathbb{1}_{S_\tau}(u) - \mathbb{1}_{S_\tau}(v)|]} \\ &\geq \min_{S_\tau} \frac{\sum_{\{u,v\}\in E} |\mathbb{1}_{S_\tau}(u) - \mathbb{1}_{S_\tau}(v)|}{\sum_{u,v\in V} |\mathbb{1}_{S_\tau}(u) - \mathbb{1}_{S_\tau}(v)|}. \end{aligned}$$

The last step follows from choosing the minimal among n different cuts S_τ, and using lemma (6.7) again. □

Finally, we can show that the cut obtained from the $L1$ metric is a good approximation of the sparsest cut.

Proof of Theorem 6.4. Let d^* be an optimal pseudo-metric solution to the linear program (6.5), function $f : V \to \mathbb{R}^k$ an embedding as in Theorem 6.6, and set $S \subseteq V$ the cut extraction described in Claim 6.8. Then with high probability we have

$$\begin{aligned}\mathcal{L}(S) &\leq \frac{\sum_{\{u,v\}\in E}\|f(u)-f(v)\|_1}{\sum_{u,v\in V}\|f(u)-f(v)\|_1}\\ &\leq \mathcal{O}(\log(n))\frac{\sum_{\{u,v\}\in E} d^*(u,v)}{\sum_{u,v\in V} d^*(u,v)}\\ &\leq \mathcal{O}(\log(n))\mathcal{L}(S_{OPT}).\end{aligned}$$

Where the first inequality comes from the cut extraction, Claim 6.8, the second inequality comes from the $L1$ embedding theorem and the last inequality comes from d^* being the solution of the relaxed optimization problem. □

Note that random threshold cuts S_τ can be defined along any dimension $j \in \{1, \ldots, k\}$ of f. The theorem shows that among these at most $n \cdot k = \mathcal{O}(n \log^2(n))$ cuts S_τ, one of them is an $\mathcal{O}(\log(n))$ approximation of the sparsest cut, with high probability.

6.3 L1 Embedding

In the previous section, we saw how we can find a $\Theta(\log n)$ approximation of the sparsest cut by using in a black-box manner an embedding that maps the points to a space with $L1$ norm while stretching pairwise distances by at most an $\Theta(\log n)$ factor. In this section, we prove the existence of this embedding, i.e.,

Lemma 6.9. *Given a pseudo-metric $d : V \times V \to \mathbb{R}^+$ for an n-point space V, we construct a mapping $f : V \to \mathbb{R}^k$ for $k = \Theta(\log^2 n)$ such that for any two vertices $u, v \in V$, we have $d(u,v)/\Theta(\log n) \leq ||f(u) - f(v)||_1 \leq d(u,v)$.*

Warm up & Intuition

Here, we provide some intuitive discussions that help us to understand how we arrive at the final solution.

Approach 1: fix an arbitrary vertex $s \in V$, and define a one-dimensional function $f : V \to\in \mathbb{R}$, $f(u) := d(s, u)$. This will give us a pseudo-metric. The pseudo-metric satisfies $||f(u) - f(v)||_1 \leq d(u, v)$ by the triangle inequality. What remains is to show that $||f(u) - f(v)||_1$ is not much smaller than $d(u, v)$. This depends on the choice of node s. A natural suggestion would be to pick s at random, and that works well for some scenarios. But not always. As we shall see in the following pathological example, we do not have $d(u, v)/\Theta(\log n) \leq ||f(u) - f(v)||_1$.

Example 1: Consider a graph with n vertices such that there are two vertexs u, v of degree $n-2$ that share the same neighbours, in this situation $d(u, v) = 2$, $d(u, v_i) = d(v, v_i) = 1$ for all $v_i \neq u, v$. There are $n-2$ "in the middle of the segment uv". If we sample $s \in V$ uniformly at random, then with large probability $(n-2)/n$, we will choose a vertex "in the middle of uv". This way, we have $||f(u) - f(v)||_1 \approx 0$, so we do not have the bound $d(u, v)/\Theta(\log n) \leq ||f(u) - f(v)||_1$.

Approach 2: The failure of approach 1 indicates that a single "source vertex" s might not be enough for our purpose. To fix it, we can pick a set of vertices S as the "source vertices". More precisely, we choose a set S by including each vertex from V in S with probability $p = 1/2$.

Suppose S is chosen. Define $f : V \to \mathbb{R}$ as $f(u) = d(S, u) := \min_{s \in S} d(s, u)$. The distance to a set S is a pseudo-metric and from the triangular inequality we can deduce that $||f(u) - f(v)||_1 \leq d(u, v)$ still holds.

As for the other direction of the inequality, we can check if the approach is valid for the pathological example again. Notice that if we choose an S such that $u \in S, v \notin S$, then $f(u) = 0$ and $f(v) = 1$. This implies $||f(u) - f(v)||_1 \geq 1$, satisfying $d(u, v)/\Theta(\log n) \leq ||f(u) - f(v)||_1$. We also notice that the event $\{S : u \in S, v \notin S\}$ happens with a constant probability for sampling probability $p = 1/2$. Therefore this counter example is solved.

Example 2: We consider a graph formed by a path of length n and take $u, v \in V$ the end nodes of the path. In this situation $d(u, v) = n - 1$ but choosing S with probability $1/2$ we are very likely to sample vertices that are close to u and v, and we will have $|f(u) - f(v)| \approx \mathcal{O}(1)$. Therefore the bound $d(u, v)/\Theta(\log n) \leq ||f(u) - f(v)||_1$ does not hold.

Approach 3: The failure of approach 2 indicates that there are some graphs that require choosing a set S with a high probability, like we have seen in Example 1, and there are graphs that require a low probability, like we have seen in Example 2. This motivates the use of a multi-dimensional embedding $f : V \to \mathbb{R}^k$ where $k = \log n$. Each component of the embedding

will have a different sampling probability, solving the problems of Example 1 and Example 2. We define the component $f_i(u) := d(S_i, u)$ where S_i has been sampled with probability $p_i = 1/2^i$.

6.3.1 The algorithm and its analysis

The Algorithm With the above considerations we write the following algorithm, where we have added additional dimensions to be able to amplify the success probabilities.

Algorithm 18 L1 EMBEDDING

for $i = 1$ to $L = \log n$ **do**
  for $h = 1$ to $H = 1000 \log n$ **do** ▷ Probability amplification
    Define S_{ih} by including each $v \in V$ in it independently with prob. $1/2^i$
    Define the coordinate of f by $f_{(i-1)H+h}(u) = d(u, S_{ih})/LH$
  end for
end for
The embedding is then given as $f : V \to \mathbb{R}^{LH}$, where each f_i is defined above

Analysis: We need to prove that for every pair of points $u, v \in V$, we have $d(u, v)/\Theta(\log n) \leq ||f(u) - f(v)||_1 \leq d(u, v)$. For readability, denote the $((i-1)H + h)$-th coordinate of f as f_{ih}.

Let us start with the easy side. By construction, we have

$$\begin{aligned} ||f(u) - f(v)||_1 &= \sum_{i=1}^{L} \sum_{h=1}^{H} |f_{ih}(u) - f_{ih}(v)| \\ &= \sum_{i=1}^{L} \sum_{h=1}^{H} \frac{|d(u, S_{ih}) - d(v, S_{ih})|}{LH} \\ &\leq LH \cdot \frac{d(u, v)}{LH} = d(u, v) \end{aligned}$$

For the other direction, the following result provides the inequality with high probability.

Claim 6.10. *The L1 embedding algorithm provides an embedding $f : V \to \mathbb{R}^{LH}$, where $L = \log n$ and $H = 1000 \log n$ that satisfies the following condition with high probability:*

$$||f(u) - f(v)||_1 \geq \Theta\left(\frac{d(u, v)}{L}\right) \quad \forall u, v \in V$$

Proof. This result will be proved in two steps.

1. We fix a pair of vertices $u, v \in V$ and focus on a single component f_{ih} where $i \in \{1, \ldots, L\}$, $h \in \{1, \ldots H\}$. We prove that for a certain sequence $\hat{\rho}_i$ the lower bound

$$|f_{ih}(u) - f_{ih}(v)| \geq \frac{\hat{\rho}_i - \hat{\rho}_{i-1}}{LH}$$

holds with constant probability in the component f_{ih}.

2. After showing that this result holds for one component we will augment the probability to make it hold for components f_{ih} for all $i \in \{1, \ldots, L\}$ and a fraction of $h \in \{1, \ldots H\}$. This will allow us to provide the desired bound for the $L1$ embedding.

Let us start the first step. For $t \in \{0, 1, \ldots, \log n\}$ we define

$$\rho_t = \min_r \{|B_r(u)| \geq 2^t \text{ and } |B_r(v)| \geq 2^t\}.$$

Where $B_r(u)$ and $B_r(v)$ are closed balls of radius r. From the definition of ρ_t there exists an index j such that $\rho_j < d(u,v)/2$ and $\rho_{j+1} \geq d(u,v)/2$. We define the *truncated sequence*,

$$\hat{\rho}_i = \begin{cases} \rho_i & \text{if } i = 0, 1, \ldots, j \\ d(u,v)/2 & \text{if } i = j+1, \ldots, L \end{cases}$$

If we take an index $i > j+1$ we have $\hat{\rho}_i = \hat{\rho}_{i-1} = \frac{d(u,v)}{2}$, hence $|f_{ih}(u) - f_{ih}(v)| \geq \frac{\hat{\rho}_i - \hat{\rho}_{i-1}}{LH} = 0$ will trivially hold. Let us focus on some index $i \leq j+1$. Without loss of generality, we may assume that $\hat{\rho}_i$ is defined by u, that is, $|B^{\text{open}}_{\hat{\rho}_i}(u)| < 2^i$. We also have $|B_{\hat{\rho}_{i-1}}(v)| \geq 2^{i-1}$ by construction. Since $\hat{\rho}_{i-1} \leq \hat{\rho}_i \leq d(u,v)/2$, the balls $B^{\text{open}}_{\hat{\rho}_i}(u)$ and $B_{\hat{\rho}_{i-1}}(v)$ are disjoint. Consider the events

- $A_{ih} = \{S_{ih} \cap B^{\text{open}}_{\hat{\rho}_i}(u) = \emptyset\}$
- $B_{ih} = \{S_{ih} \cap B_{\hat{\rho}_{i-1}}(v) \neq \emptyset\}$

Where S_{ih} are the set of nodes that defines the coordinate f_{ih}. Because the two balls are disjoint, events A_{ih} and B_{ih} are independent. When the events A_{ih} and B_{ih} both happen we have $d(u, S_{ih}) \geq \hat{\rho}_i$ and $d(v, S_{ih}) \leq \hat{\rho}_{i-1}$, so

$$|f_{ih}(u) - f_{ih}(v)| = |d(S_{ih}, u) - d(S_{ih}, v)| \geq \frac{\hat{\rho}_i - \hat{\rho}_{i-1}}{LH}$$

To conclude the proof of the first step it remains to show that the event $A_{ih} \cap B_{ih}$ happens with a constant probability. The probabilities of S_{ih} not sampling a node in $B^{\text{open}}_{\hat{\rho}_i}(u)$ and S_{ih} sampling a node in $B_{\hat{\rho}_{i-1}}(v)$ can be calculated as

$$\Pr[A_{ih}] = \left(1 - \frac{1}{2^i}\right)^{|B^{\text{open}}_{\hat{\rho}_i}(u)|} \geq \left(1 - \frac{1}{2^i}\right)^{2^i} \geq 4^{-1} = \frac{1}{4}.$$

$$\Pr[B_{ih}] = 1 - \left(1 - \frac{1}{2^i}\right)^{|B_{\hat{\rho}_{i-1}}(v)|} \geq 1 - \left(1 - \frac{1}{2^i}\right)^{2^{i-1}} \geq 1 - e^{-1/2}.$$

In the preceding calculation, we use the fact that $4^{-x} \leq 1 - x \leq e^{-x}$ for $x \in [0, 1/2]$. Since A_{ih} and B_{ih} are independent events, then

$$\Pr[A_{ih} \cap B_{ih}] = \Pr[A_{ih}] \cdot \Pr[B_{ih}] \geq c$$

Where we have defined $c := (1 - e^{-1/2})/4$. And the first step of the proof is finished, the inequality $|f_{ih}(u) - f_{ih}(v)| \geq \frac{\hat{\rho}_i - \hat{\rho}_{i-1}}{LH}$ holds in a component f_{ih} with constant probability.
In the second step of the proof we amplify the success probability. For every $i \in \{1, \ldots, L\}$ let x_i be the number of indices such that the inequality holds, using the Chernoff bound we have

$$\begin{aligned}\Pr\left[x_i \leq \frac{c}{2}H\right] &\leq \Pr\left[|x - c \cdot H| \geq \frac{c}{2}H\right] \\ &\leq 2\exp\left(\frac{-c \cdot H}{12}\right) \leq \frac{2}{n^{1000c/12}} \leq \frac{2}{n^5}\end{aligned}$$

And for every $i \in \{1, \ldots, L\}$ the inequality proved in Step 1 will hold in at least $cH/2$ indices with high probability $1 - 1/n^5$. To make this result hold for all $i \in \{1, \ldots, L\}$ the probability of failure will be amplified to $\Theta(\frac{\log n}{n^5}) \leq \Theta(\frac{1}{n^4})$.
Now, since the result holds for every index i and a significant fraction of the indices h we have

$$\begin{aligned}||f(u) - f(v)||_1 &= \sum_{i=1}^{L}\sum_{h=1}^{H} |f_{ih}(u) - f_{ih}(v)| \\ &\geq \frac{c}{2}H\sum_{i=1}^{L}\frac{\hat{\rho}_i - \hat{\rho}_{i-1}}{LH} = \frac{d(u,v)}{2L}\end{aligned}$$

Where the last equality comes from a telescopic sum and the definition of $\hat{\rho}_0 = 0$ and $\hat{\rho}_L = \frac{d(u,v)}{2}$.

This proves that the embedding is valid for a pair of vertices $u, v \in V$ that we have fixed at the beginning of the proof. If we consider the probability that the embedding works for any pair of vertices the failure probability will be amplified to $\Theta(\frac{n^2}{n^4}) = \Theta(\frac{1}{n^2})$. Therefore the $L1$ embedding satisfies the distance contraction for any pair of nodes with high probability. $\square$

Chapter 7

Oblivious Routing, Cut-Preserving Tree Embedding, and Balanced Cut

In this section, we develop the notion of cut-preserving tree embeddings. We will use the problem of oblivious routing as our initial motivation. But then, we will also see that these cut-preserving trees are also useful for other approximation problems, and we discuss the balanced cut problem as one such application[1].

7.1 Oblivious Routing

Consider an undirected graph $G = (V, E)$ where every edge $e \in E$ has a given capacity c_e. Suppose we have many routing demands d_{uv} for $u, v \in V$, where d_{uv} denotes the demand to be sent from node u to node v.

For any pair (u, v) we want to define a route, or more formally a flow of size d_{uv} from vertex u to v. This is a function $r_{uv} : E \to [0, 1]$ such that:

1. $\sum_{e \in out(w)} r_{uv}(e) d_{uv} - \sum_{e \in in(w)} r_{uv}(e) d_{uv} = 0$ for all $w \neq u, v$,
2. $\sum_{e \in out(u)} r_{uv}(e) d_{uv} - \sum_{e \in in(u)} r_{uv}(e) d_{uv} = d_{uv}$
3. $-\sum_{e \in out(v)} r_{uv}(e) d_{uv} + \sum_{e \in in(v)} r_{uv}(e) d_{uv} = d_{uv}$

[1]Please read critically. This section has not been reviewed by the instructor yet. To get a quick response, please post your questions and comments about this chapter on the course moodle.

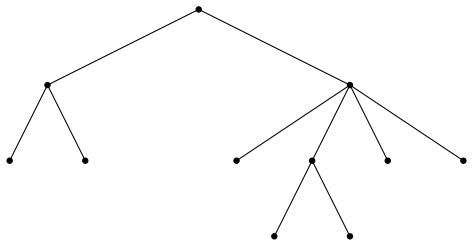
We define the *congestion* of an edge e to be the total amount of demand sent through e divided by the capacity c_e. Overall, our objective is to have a small congestion over all the edges.

Definition 7.1. ***(Oblivious Routing Problem)*** *Given an undirected graph $G = (V, E)$, capacities c_e for all $e \in E$, and demands d_{uv} for $u, v \in V$, we wish to find routes for the demands with the objective of minimizing the maximum congestion over all the edges.*
Moreover, we wish to find these routes in an oblivious manner, meaning that we should pick the route of each demand d_{uv} independently of the existence of the other demands.

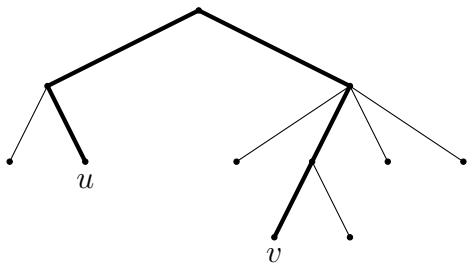
Our goal is to do this in a way that is competitive in terms of congestion with the best possible routing that we could have designed after knowing all of the demands. In this section, we discuss approximation schemes that devise these routes independently for each demand and yet are still competitive with the optimal solutions that one can obtain after knowing all of the demands.

7.2 Oblivious Routing via Trees

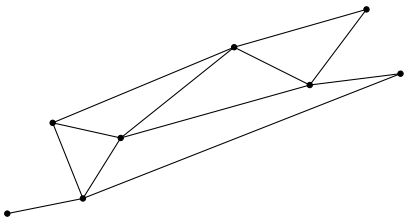
Warm-up Suppose the graph which we are given is simply one tree, e.g., the following picture:



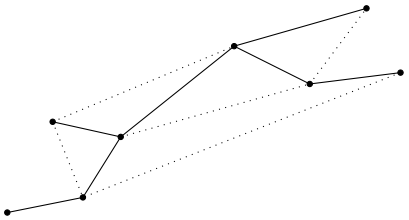
In this case, we can achieve oblivious routing since for all vertices $u, v \in V$ there is a unique shortest path between them, and all other paths from u to v must cover this path as well. The solution is therefore to send all of the demand d_{uv} along this path, as outlined below.



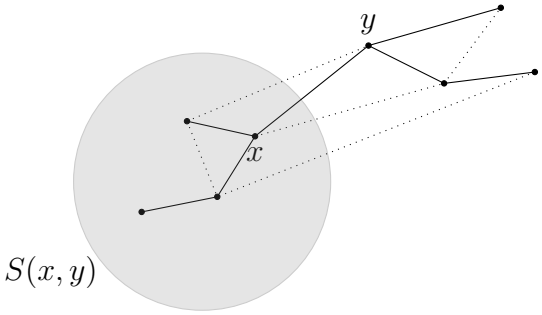
General Graphs, routing via one tree Our hope is to use a similar idea for any arbitrary graph. Consider the following graph G.



One strategy would be to pick a spanning tree of G and require that all demands are routed through this tree. For example take the following spanning tree $T \subseteq G$.



Note that if we pick an edge $\{x, y\}$ from the edge set of T then removing this edge from T disconnects the tree into two connected components. Let $S(x, y)$ denote the vertices which are in the same connected component as x, as in the following figure.



Now any demand d_{uv} that has exactly one endpoint in $S(x, y)$ will be routed through $\{x, y\}$ in our spanning tree. These are the demands that have to go through $cut(S(x, y), V \setminus S(x, y))$. Since we are routing only along the chosen tree, all these demands have to traverse through the edge $\{x, y\}$ in the tree. We therefore define

$$D(x, y) = \sum_{u \in S(x,y), v \in V \setminus S(x,y)} d_{uv},$$

as the amount that will be passed through edge $\{x, y\}$ in our scheme. Hence, in our routing, the congestion on an edge $e = \{x, y\} \in T$ is exactly $\frac{D(x,y)}{c_{xy}}$.

On the other hand, in any routing scheme on G, this demand $D(x, y)$ has to be sent through edges in $cut(S(x, y), V \setminus S(x, y))$. We can therefore lower bound the optimum congestion in any scheme by

$$OPT \geq \frac{D(x, y)}{C(x, y)},$$

where

$$C(x, y) = \sum_{e \in cut(S(x,y), V \setminus S(x,y))} c_e.$$

Thus, if we had the following edge condition, then our routing would be α-competitive.

Definition 7.2. *(**Edge Condition**) For each edge $\{u, v\} \in T$, we should have that*

$$c_{uv} \geq \frac{1}{\alpha} C(u, v) \tag{7.1}$$

The above discussion shows that this edge condition is a sufficient condition for α-competitiveness. However, we claim it is also a necessary condition for the routing scheme on the tree.

Claim 7.3. *The edge condition is necessary for the routing scheme on the tree to be α-competitive in congestion.*

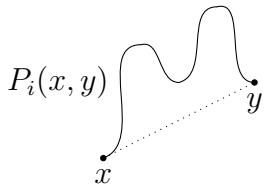
Proof. We wish to show that if our scheme is α-competitive, then the edge condition holds. Note that if our scheme is α-competitive, then it must be so for any possible demand. Consider the case in which the demands which we want to send are equal to the capacities, i.e. $d_{uv} = c_{uv}$ for all $\{u, v\} \in G$.

This problem can be routed optimally with congestion 1, by sending each demand completely along its corresponding edge. For some edge $\{u, v\} \in T$, our scheme would try to send $D(u, v)$, which in this case is $C(u, v)$, along the edge. Therefore the congestion for all edges $\{u, v\} \in T$ is $\frac{C(u,v)}{c_{uv}}$. So if our scheme is α-competitive, we must have that $\frac{C(u,v)}{c_{uv}} \leq \alpha$. □

Generalizing and Modifying the Scheme Unfortunately, routing along one subtree will not provide a competitive scheme for all graphs (examples are discussed in the exercise sessions). To remedy this, we generalise our plan of routing along a tree, in the following two ways:

1. Instead of routing along *one* tree, we route along *many* trees, namely a collection $\mathcal{T} = \{T_1, T_2, ..., T_k\}$ where each tree T_i has probability λ_i of being chosen. So $\sum_{i=1}^{k} \lambda_i = 1$ and with probability λ_i we pick T_i and route through it.

2. The trees T_i do not need to be subgraphs of our graph G. Tree T_i can be a virtual graph on vertices V where sending along an edge $\{x, y\} \in T_i$ actually means sending along some fixed path $P_i(x, y)$ in G.



Given these two generalizations, we reach a natural modified variant of the edge condition as we describe next. We will show that this condition is necessary and sufficient for α-competitiveness of the routing scheme. Let C_i and D_i be defined analogously to C and D on the tree T_i.

Definition 7.4. *(**Updated Edge Condition**) For each edge $\{u, v\} \in G$, we should have that*

$$c_{uv} \geq \frac{1}{\alpha} \sum_i \lambda_i \sum_{\substack{\{x,y\} \in T_i \\ s.t.\ \{u,v\} \in P_i(x,y)}} C_i(x, y) \tag{7.2}$$

Claim 7.5. *The update edge condition is necessary for the routing scheme to be α-competitive in congestion.*

Proof. As in the proof of claim 7.3, we consider the case in which all demands d_{uv} are equal to the capacities c_{uv}. The optimal congestion in this case is 1.

We now consider the congestion achieved by our scheme. Consider an edge $\{u, v\} \in G$. Suppose our scheme chooses tree T_i out of the collection of trees $\mathcal{T}$. Note that for every edge $\{x, y\} \in T_i$, if the path $P_i(x, y)$ goes through $\{u, v\}$ then our scheme will send $D_i(x, y)$, which in this case is $C_i(x, y)$, through $\{u, v\}$. So the demand routed through $\{u, v\}$ in T_i will be

$$\sum_{\substack{\{x,y\} \in T_i \\ s.t.\ \{u,v\} \in P_i(x,y)}} C_i(x, y).$$

But each tree T_i is chosen with probability λ_i out of our collection of trees $\mathcal{T}$. Therefore we can write the expected demand which will be routed through $\{u, v\}$ as

$$\sum_i \lambda_i \sum_{\substack{\{x,y\} \in T_i \\ s.t.\ \{u,v\} \in P_i(x,y)}} C_i(x, y).$$

Therefore if our scheme is α-competitive with the optimal congestion, this sum must be $\leq \alpha c_{uv}$ and therefore the edge condition holds. □

Next, we will show that this updated tree condition is also sufficient for achieving α-competitiveness, i.e. that having inequality 7.4 satisfied is enough to imply our routing scheme to be α-competitive in congestion.

Claim 7.6. *The updated edge condition is sufficient for the routing scheme being α-competitive in congestion.*

Proof. Assume the updated edge condition is satisfied, i.e. inequality 7.4 holds. Let's consider an arbitrary set of demands and see how they must be routed through our collection $\mathcal{T}$ of trees $T_1, \ldots, T_k$ as compared to the optimal routing in G.

It is clear that the amount routed through some tree edge $\{x, y\} \in T_i$ must in G be routed through any edge $\{u, v\}$ that lies on the fixed path corresponding to $\{x, y\}$, i.e. through any $\{u, v\} \in P_i(x, y)$.

If a tree T_i is chosen, we decide to route $D_i(x, y)$ amount of flow through $\{u, v\} \in P_i(x, y)$. With this strategy, we end up sending a total expected amount of

$$\sum_i \lambda_i \sum_{\substack{\{x,y\} \in T_i \\ s.t.\ \{u,v\} \in P_i(x,y)}} D_i(x, y)$$

through edge $\{u, v\} \in G$, where the expectation is taken over the possible choices of a tree T_i. Together with our assumption of 7.4 being satisfied, we can now upper bound the congestion on edge $\{u, v\}$ by

$$\frac{\sum_i \lambda_i \sum_{\substack{\{x,y\} \in T_i \\ s.t.\ \{u,v\} \in P_i(x,y)}} D_i(x, y)}{\frac{1}{\alpha} \sum_i \lambda_i \sum_{\substack{\{x,y\} \in T_i \\ s.t.\ \{u,v\} \in P_i(x,y)}} C_i(x, y)} \overset{(*)}{\leq} \alpha \cdot \left(\max_i \frac{D_i(x, y)}{C_i(x, y)} \right) \overset{(**)}{\leq} \alpha \cdot OPT$$

Here, $(*)$ follows from $\frac{a_1 + \cdots + a_k}{b_1 + \cdots + b_k} \leq \max_i \frac{a_i}{b_i}$ for any $a_1, \ldots, a_k, b_1, \ldots, b_k \in \mathbb{R}$ and $(**)$ is because, as seen before, $\max_i \frac{D_i(x,y)}{C_i(x,y)}$ is a lower bound on OPT. □

7.3 Existence of the Tree Collection

In this section, our goal is to show the existence of a tree collection that achieves $\mathcal{O}(\log n)$-competitiveness in oblivious routing. The proof proceeds by writing the constraint minimization of the competitiveness ratio α as an LP. We then write down the dual of this LP and show that its optimal solution is in $\mathcal{O}(\log n)$. By strong LP duality, we can then also claim the minimum competitiveness ratio achievable in the primal to be $\mathcal{O}(\log n)$.

To cast the competitiveness of the oblivious routing scheme as a linear program $\mathbf{LP}_{\textbf{Oblivious Routing}}$, we rely on the updated edge condition of the previous section, which we know is necessary and sufficient:

$\mathbf{LP}_{\textbf{Oblivious Routing}}$:

$$
\begin{aligned}
&\text{minimize} && \alpha && && \triangleleft \text{ competitiveness ratio}\\
&\text{subject to} && \alpha c_{uv} - \sum_i \lambda_i \sum_{\substack{\{x,y\}\in T_i \\ s.t.\ \{u,v\}\in P_i(x,y)}} C_i(x,y) \geq 0 && \forall \{u,v\} \in G && \triangleleft \text{ edge condition}\\
& && \sum_i \lambda_i \geq 1 && && \triangleleft \text{ valid probabilities}\\
& && \lambda_i \geq 0 && \forall i \in [k] &&
\end{aligned}
$$

Here we allow the probabilities λ_i to sum to over 1, but this is not an issue, as there always exists an optimal solution that satisfies $\sum_i \lambda_i = 1$.

Claim 7.7. *The optimal value of $\boldsymbol{LP}_{\textbf{Oblivious Routing}}$ is in $\mathcal{O}(\log n)$.*

Proof. We begin by writing down the dual of $\mathbf{LP}_{\textbf{Oblivious Routing}}$ as below.

Dual-$\mathbf{LP}_{\textbf{Oblivious Routing}}$

$$
\begin{aligned}
&\text{maximize} && z && && \triangleleft\ (1)\\
&\text{subject to} && \sum_{\{u,v\}\in G} c_{uv} l_{uv} \leq 1 && && \triangleleft\ (2)\\
& && z - \sum_{\{u,v\}\in G} \sum_{\substack{\{x,y\}\in T_i \\ s.t.\ \{u,v\}\in P_i(x,y)}} C_i(x,y)\cdot l_{uv} \leq 0 && \forall i \in [k] && \triangleleft\ (3)\\
& && l_{uv} \geq 0 && \forall \{u,v\} \in G && \triangleleft\ (4)
\end{aligned}
$$

In order to provide an upper bound on the optimal solution of **Dual-LP$_{\text{Oblivious Routing}}$**, we will think of the variable l_{uv} as denoting the length of edge $\{u, v\}$. In any valid solution, these lengths need to satisfy inequality (2) together with the edges' given capacities. We further rewrite the constraints in line (3). Instead of summing over all edges $\{u, v\}$ and restricting the inner sum to tree edges $\{x, y\}$ whose corresponding path $P_i(x, y)$ contains $\{u, v\}$, we can also do the outer sum over all tree edges and the inner over just the relevant $\{u, v\}$'s. This way, the constraint of line (3) is rewritten as:

$$z \leq \underbrace{\sum_{\{x,y\} \in T_i} C_i(x, y) \sum_{\{u,v\} \in P_i(x,y)} l_{uv}}_{=: A_i} \qquad \forall i \in [k].$$

To arrive at the tightest condition on z, and thus also the tightest lower bound on OPT, we are interested in the tree T_i that minimizes A_i under the constraint that our distances satisfy condition (2). In the following, we show that there exists such a tree T_i among our collection $\mathcal{T}$ with $A_i \leq \mathcal{O}(\log n)$.

For that, recall from Chapter 5 on probabilistic distance-preserving tree embeddings that for any graph $G = (V, E)$ one can find a distribution over a collection of trees $T_i = (V, E_i)$, each being chosen with probability λ_i, such that for any $x, y \in V$ we have

$$\begin{aligned} d_l(x, y) &\leq T_i(x, y) \quad \text{and} \\ \mathbb{E}[T_i(x, y)] = \sum_i \lambda_i T_i(x, y) &\leq \mathcal{O}(\log n) \cdot d_l(x, y) \end{aligned}$$

where $d_l(x, y)$ denotes the distance of vertices x, y in G and $T_i(x, y)$ their distance in T. Below, we will be able to apply the existence of such an embedding to our graph with distances l_{uv}.

Consider the expectation of A_i with respect to our distribution over the

trees T_i with probabilities λ_i.

$$
\begin{aligned}
\mathbb{E}_{T_i}[A_i] &= \sum_i \lambda_i A_i \\
&= \sum_i \lambda_i \sum_{\{x,y\}\in T_i} C_i(x,y) \sum_{\{u,v\}\in P_i(x,y)} l_{uv} \\
&\leq \sum_i \lambda_i \sum_{\{x,y\}\in T_i} C_i(x,y) d_l(x,y) && \text{(path length} \geq \text{distance)} \\
&\leq \sum_i \lambda_i \sum_{\{x,y\}\in T_i} C_i(x,y) T_i(x,y) && \text{(distance-preserving embedding)} \\
&= \sum_i \lambda_i \sum_{\{x,y\}\in T_i} T_i(x,y) \sum_{\{u,v\}\in Cut(S_i(x,y),V\setminus S_i(x,y))} c_{uv} && \text{(definition of } C_i\text{)} \\
&= \sum_i \lambda_i \sum_{\{u,v\}\in G} c_{uv} \cdot \sum_{\{x,y\}\in P_i(u,v)} T_i(x,y) && \text{(definition of } S_i\text{)} \\
&= \sum_i \lambda_i \sum_{\{u,v\}\in G} c_{uv} T_i(u,v) && \text{(rearranging the sum)} \\
&= \sum_{\{u,v\}\in G} c_{uv} \sum_i \lambda_i T_i(u,v) && \text{(rearranging, again)} \\
&\leq \sum_{\{u,v\}\in G} c_{uv} \cdot \mathcal{O}(\log n) \cdot d_l(u,v) && \text{(distance-preserving embedding)} \\
&\leq \sum_{\{u,v\}\in G} c_{uv} \cdot \mathcal{O}(\log n) \cdot l_{uv} && \text{(property of distance)} \\
&= \mathcal{O}(\log n) \cdot \underbrace{\sum_{uv} c_{uv} \cdot l_{uv}}_{\leq 1, \text{ by (2)}} = \mathcal{O}(\log n)
\end{aligned}
$$

From $\mathbb{E}_{T_i}[A_i] = \mathcal{O}(\log n)$ it follows directly that one of the A_i's must be in $\mathcal{O}(\log n)$. Therefore, OPT of **Dual-LP$_{\text{Oblivious Routing}}$** is in $\mathcal{O}(\log n)$ and OPT of **LP$_{\text{Oblivious Routing}}$** is in $\mathcal{O}(\log n)$.

Since any valid solution to **LP$_{\text{Oblivious Routing}}$** with objective value α corresponds to an α-congestion-competitive oblivious routing scheme, we have shown that $\mathcal{O}(\log n)$-competitiveness can be achieved.

Notice that we have only given an existence proof for a satisfactory mixture of trees, not a construction. Additionally, we have not shown yet how many trees are necessary in order to achieve $\mathcal{O}(\log n)$-competitiveness. In section 8.3, we show how to construct a polynomially large mixture of trees in polynomial time.

□

7.4 The Balanced Cut problem

We now want to tackle a new problem using the tree embedding technique to gain a good approximation for a hard problem on general graphs, the balanced cut problem. We aim to split the vertices of an undirected graph into two sets of equal size so that the weight of the edges going across the cut are minimized.

Definition 7.8 (minimum balanced cut)**.** *Given an undirected graph $G = (V, E)$ with edge costs c_e for all $e \in E$, a minimum balanced cut $(S, V \setminus S)$ is a cut consisting of exactly half the vertices which minimizes the sum of crossing edges over the cut.*

$$\min_{S \subset V, |S| = \lfloor \frac{|V|}{2} \rfloor} \sum_{e \in (S, V \setminus S)} c_e$$

Computing the balanced cut is of special interest to us as it can be used as a powerful building block for other algorithms. In "divide & conquer" approaches, we can split the graph into two sub-problems of roughly equal size in a way that the dependencies (edges) between them are small using the minimum balanced cut.

Note that the technique seen here is much more generally applicable. It allows us to go from a problem on general graphs to working on trees, which often drastically simplifies the problem. Because the tree construction in some sense preserves properties of the cut we can hope to translate the solution back to the general graph without loosing to much guarantees on the optimality of our solution.

To find the minimal balanced cut, we use the following procedure:

1. Construct a collection of trees T
2. Virtually compute the edge costs as the weight of the induced cut in T_i

$$cost_{T_i}(x, y) = C_i(x, y) = \sum_{e \in (S_i(x,y), V \setminus S_i(x,y))} c_e$$

3. Compute the optimal minimum balanced cut X_i on tree T_i using dynamic programming
4. Take the tree X_{i^*} with minimal costs on graph G as the solution

Definition 7.9 (Cut preserving tree embedding)**.** *We call a collection of trees T with $\sum_i \lambda_i = 1, \lambda_i \geq 0$ a cut preserving tree embedding if and only if*

(I) $Cut_G(S, V \setminus S) \leq Cut_{T_i}(S, V \setminus S)$

(II) $\sum_i \lambda_i Cut_{T_i}(S, V \setminus S) \leq \mathcal{O}(\log(n)) Cut_G(S, V \setminus S)$

We can interpret the λ_i's as probabilities of the different trees in the collection. Then a cut preserving tree embedding is always at least the original cut and in expectation only a $\log(n)$ factor greater.

Claim 7.10. *The minimum balanced cut of a cut preserving tree embedding yields an $\mathcal{O}(\log(n))$ approximation for the minimum balanced cut of the whole graph.*

Proof. Let $(S^*, V \setminus S^*)$ be an optimal cut for the minimum balanced cut problem of the whole graph G. Furthermore let $(X_i, V \setminus X_i)$ be the cut computed for T_i of the tree embedding and let $(X^*, V \setminus X^*)$ be the one with minimum cost on G among them.

$$\begin{aligned}
Cut_G(X^*, V \setminus X^*) &= \min_i Cut_G(X_i, V \setminus X_i) \\
&\leq \sum_i \lambda_i Cut_G(X_i, V \setminus X_i) && \text{Convexity} \\
&\leq \sum_i \lambda_i Cut_{T_i}(X_i, V \setminus X_i) && \text{Property I} \\
&\leq \sum_i \lambda_i Cut_{T_i}(S^*, V \setminus S^*) && X_i \text{ optimal on } T_i \\
&\leq \mathcal{O}(\log(n)) Cut_G(S^*, V \setminus S^*) && \text{Property II}
\end{aligned}$$

□

The collection of trees we have used for the oblivious routing problem in the previous section fulfills both properties of a cut preserving tree embedding. We will prove this in lemmas 7.11 and 7.12.

Lemma 7.11. *For every tree T_i in the collection of trees T it holds that*

$$Cut_G(S, V \setminus S) \leq Cut_{T_i}(S, V \setminus S)$$

Proof. The key insight to why the inequality holds is that each edge in the original cut (on the left) gets counted at least once on the right hand side.

Let's look at an arbitrary edge $u \in S, v \in V \setminus S$ over the cut $(S, V \setminus S)$. On the unique path from u to v in the tree T_i there exists at least one

$xy \in T_i$ for which $x \in S, y \in V \setminus S$. By definition $cost_{T_i}(x,y) = C_i(x,y) = \sum_{e \in (S_i(x,y), V \setminus S_i(x,y))} c_e$. Because the x, y were along the path from u to v $u \in S_i(x,y), v \in V \setminus S_i(x,y)$, therefore the edge uv is included in the sum of $C_i(x,y)$.

$$\begin{aligned}
Cut_G(S, V \setminus S) &= \sum_{uv \in (S, V \setminus S)} c_{uv} && \text{def. } Cut_G \\
&\leq \sum_{\substack{xy \in T_i \\ xy \in (S, V \setminus S)}} \sum_{e \in (S_i(x,y), V \setminus S_i(x,y))} c_e && \text{above discussion} \\
&= \sum_{\substack{xy \in T_i \\ xy \in (S, V \setminus S)}} cost_{T_i}(x,y) && \text{def. edge costs} \\
&= Cut_{T_i}(S, V \setminus S) && \text{def. } Cut_{T_i}
\end{aligned}$$

□

Lemma 7.12. *For every mixture of trees satisfying condition 7.4 with* $\sum_i \lambda_i = 1, \lambda_i \geq 0$ *it holds that*

$$\sum_i \lambda_i Cut_{T_i}(V, V \setminus S) \leq \mathcal{O}(\log(n)) Cut_G(S, V \setminus S)$$

Proof. Recall edge condition 7.4 for $\alpha = \mathcal{O}(\log(n))$:

$$\sum_i \lambda_i \sum_{\substack{xy \in T_i \\ uv \in P_i(x,y)}} C_i(x,y) \leq \mathcal{O}(\log(n)) c_{uv}$$

The key insight to rewrite the summation is that each $xy \in T_i$ going across the cut $(S, V \setminus S)$ must have at least one edge uv on it's path $P_i(x,y)$ which also crosses the cut. If this were not the case all vertices on the path, including x, y would belong to the same side of the cut which is not the case. This allows us to upper bound the edge weights in the tree T_i:

$$\sum_{\substack{xy \in T_i \\ xy \in (S, V \setminus S)}} C_i(x,y) \leq \sum_{uv \in (S, V \setminus S)} \sum_{\substack{xy \in T_i \\ uv \in P_i(x,y)}} C_i(x,y)$$

Now we can apply the definitions and combine them with the two attained

bounds from above to prove the claim.

$$\begin{aligned}
\sum_i \lambda_i Cut_{T_i}(V, V \setminus S) &= \sum_i \lambda_i \sum_{\substack{xy \in T_i \\ xy \in (S, V \setminus S)}} C_i(x,y) && \text{def. } Cut_{T_i} \\
&\leq \sum_i \lambda_i \sum_{uv \in (S, V \setminus S)} \sum_{\substack{xy \in T_i \\ uv \in P_i(x,y)}} C_i(x,y) && \text{bound tree weights} \\
&= \sum_{uv \in (S, V \setminus S)} \sum_i \lambda_i \sum_{\substack{xy \in T_i \\ uv \in P_i(x,y)}} C_i(x,y) \\
&\leq \sum_{uv \in (S, V \setminus S)} \mathcal{O}(\log(n)) c_{uv} && \text{edge property 7.4} \\
&= \mathcal{O}(\log(n)) \sum_{uv \in (S, V \setminus S)} c_{uv} \\
&= \mathcal{O}(\log(n)) Cut_G(S, V \setminus S) && \text{def. of } Cut_G
\end{aligned}$$

□

Chapter 8

Multiplicative Weights Update (MWU)

In this lecture, we discuss the Multiplicative Weight Updates (MWU) method. A comprehensive survey on MWU and its applications can be found in [AHK12].

8.1 Learning from Experts

Definition 8.1 (The learning from experts problem)**.** *Every day, we are to make a binary decision. At the end of the day, a binary output is revealed and we incur a mistake if our decision did not match the output. Suppose we have access to n experts $e_1, \ldots, e_n$, each of which makes a recommendation for the binary decision to take per day. How does one make use of the experts to minimize the total number of mistakes on an online binary sequence?*

Toy setting Consider a stock market with only a single stock. Every day, we decide whether to buy the stock or not. At the end of the day, the stock value will be revealed and we incur a mistake/loss of 1 if we did not buy when the stock value rose, or bought when the stock value fell. Let σ be the sequence of true outcomes. Furthermore, we denote the true outcome on day j as σ_j.

Example — Why it is non-trivial Suppose $n = 3$ and $\sigma = (1, 1, 0, 0, 1)$.

Days/σ	1	1	0	0	1
e_1	1	1	0	0	1
e_2	1	0	0	0	1
e_3	1	1	1	1	0

In hindsight, e_1 is *always* correct so we would have incurred 0 mistakes if we always followed e_1's recommendation. However, we do not know which expert is perfect (assuming a perfect expert even exists). Furthermore, it is not necessarily true that the best expert always incurs the least number of mistakes on any prefix of the sequence σ. Ignoring e_1, one can check that e_2 outperforms e_3 on the example sequence. However, at the end of day 2, e_3 incurred 0 mistakes while e_2 incurred 1 mistake. The goal is as follows: If a perfect expert exists, we hope to eventually converge to always following him/her. If not, we hope to *not* do much worse than the best expert on the entire sequence.

Warm up: Perfect expert exists As a warm up, suppose there exists a perfect expert. Then the problem would be easy to solve: Do the following on each day:

- Make a decision by taking the majority vote of the remaining experts.
- If we incur a loss, remove the experts that were wrong.

Theorem 8.2. *We incur at most* $\log_2 n$ *mistakes on any given sequence.*

Proof. Whenever we incur a mistake, at least half the experts were wrong and were removed. Hence, the total number of experts is *at least* halved whenever a mistake occurred. After at most $\log_2 n$ removals, the only expert left will be the perfect expert and we will be always correct thereafter. □

8.1.1 A deterministic MWU algorithm

Suppose that there may not be a perfect expert. The idea is similar, but we update our trust for each expert instead of completely removing an expert when he/she makes a mistake. Consider the following deterministic algorithm (DMWU):

- Initialize weights $w_i = 1$ for expert e_i, for $i \in \{1, \ldots, n\}$.
- On each day:
  - Make a decision based on the weighted majority.

– If we incur a loss, set w_i to $(1-\epsilon)\cdot w_i$ for each wrong expert, for some constant $\epsilon \in (0, \frac{1}{2})$.

Theorem 8.3. *Suppose the best expert makes m^* mistakes and* DMWU *makes m mistakes. Then,*

$$m \leq 2(1+\epsilon)m^* + \frac{2\log n}{\epsilon}$$

Proof. Observe that when DMWU makes a mistake, the weighted majority was wrong and their weight decreases by a factor of $(1-\epsilon)$. Suppose that $x = \sum_{i=1}^{n} w_i$ at the start of the day. If we make a mistake, x drops to $\leq \frac{x}{2}(1-\epsilon) + \frac{x}{2} = x(1-\frac{\epsilon}{2})$. That is, the overall weight reduces by at least a factor of $(1-\frac{\epsilon}{2})$. Since the best expert e^* makes m^* mistakes, his/her weight at the end is $(1-\epsilon)^{m^*}$, since the initial weight is 1. By the above observation, the total weight of all experts would be $\leq n(1-\frac{\epsilon}{2})^m$ at the end of the sequence, since we make m mistakes and the initial sum of weights is n. Then we can bound m in terms of m^*,

$$\begin{aligned}
& (1-\epsilon)^{m^*} \leq n(1-\frac{\epsilon}{2})^m && \text{Expert } e^*\text{'s weight is part of the overall weight} \\
\Rightarrow\quad & m^*\log(1-\epsilon) \leq \log n + m\log(1-\frac{\epsilon}{2}) && \text{Taking log on both sides} \\
\Rightarrow\quad & m^*(-\epsilon-\epsilon^2) \leq \log n + m(-\frac{\epsilon}{2}) && \text{Since } -x-x^2 \leq \log(1-x) \leq -x \text{ for } x \in (0,\frac{1}{2}) \\
\Rightarrow\quad & m \leq 2(1+\epsilon)m^* + \frac{2\log n}{\epsilon} && \text{Rearranging}
\end{aligned}$$

□

Remark 1 In the warm up toy example, $m^* = 0$.

Remark 2 For $x \in (0, \frac{1}{2})$, the inequality $-x-x^2 \leq \log(1-x) \leq -x$ is due to the Taylor expansion[1] of log. A more familiar equivalent form would be: $e^{-x-x^2} \leq (1-x) \leq e^{-x}$.

Theorem 8.4. *No deterministic algorithm $\mathcal{A}$ can be better than 2-competitive.*

Proof. Consider only two experts e_0 and e_1 where e_0 always outputs 0 and e_1 always outputs 1. Any binary sequence σ must contain at least $\frac{|\sigma|}{2}$ zeroes or $\frac{|\sigma|}{2}$ ones. Thus, $m^* \leq \frac{|\sigma|}{2}$. On the other hand, the adversary looks at $\mathcal{A}$ and produces a sequence σ which forces $\mathcal{A}$ to incur a loss every day. Thus, $m = |\sigma| \geq 2m^*$. □

[1] See https://en.wikipedia.org/wiki/Taylor_series#Natural_logarithm

8.1.2 A randomized MWU algorithm

The 2-factor in DMWU is due to the fact that DMWU deterministically takes the (weighted) majority at each step. Let us instead interpret the weights as probabilities. Consider the following randomized algorithm (RMWU):

- Initialize weights $w_i = 1$ for expert e_i, for $i \in \{1, \ldots, n\}$.
- On each day:
  - Pick a random expert with probability proportional to their weight. (i.e. Pick e_i with probability $w_i / \sum_{j=1}^{n} w_j$)
  - Follow that expert's recommendation.
  - For each wrong expert, set w_i to $(1 - \epsilon) \cdot w_i$, for some constant $\epsilon \in (0, \frac{1}{2})$.

Another way to think about the probabilities is to split all experts into two groups $A = \{\text{Experts that output 0}\}$ and $B = \{\text{Experts that output 1}\}$. Then, decide '0' with probability $\frac{w_A}{w_A + w_B}$ and '1' with probability $\frac{w_B}{w_A + w_B}$, where $w_A = \sum_{e_i \in A} w_i$ and $w_B = \sum_{e_i \in B} w_i$ are the sum of weights in each set.

Theorem 8.5. *Suppose the best expert makes m^* mistakes and* RMWU *makes m mistakes. Then,*

$$\mathbb{E}[m] \leq (1 + \epsilon)m^* + \frac{\log n}{\epsilon}$$

Proof. Fix a day $j \in \{1, \ldots, |\sigma|\}$. Let $A = \{\text{Experts that output 0 on day } j\}$ and $B = \{\text{Experts that output 1 on day } j\}$, where $w_A = \sum_{e_i \in A} w_i$ and $w_B = \sum_{e_i \in B} w_i$ are the sums of weights in each set. Let F_j be the weighted fraction of wrong experts on day j and $\bar{F}_j$ the weighted fraction of correct experts on day j. If $\sigma_j = 0$, then $F_j = \frac{w_B}{w_A + w_B}$. If $\sigma_j = 1$, then $F_j = \frac{w_A}{w_A + w_B}$. By definition of F_j, RMWU makes a mistake on day j with probability F_j. By linearity of expectation, $\mathbb{E}[m] = \sum_{j=1}^{|\sigma|} F_j$.
Since the best expert e^* makes m^* mistakes, his/her weight at the end is $(1 - \epsilon)^{m^*}$. On each day, RMWU reduces the overall weight by a factor of $(1 - \epsilon \cdot F_j)$ by penalizing wrong experts (new weights / old weights):

$$\frac{((1 - \epsilon)F_j + \bar{F}_j)(w_a + w_b)}{w_a + w_b} = ((1 - \epsilon)F_j + \bar{F}_j) = F_j + \bar{F}_j - \epsilon \cdot F_j = 1 - \epsilon \cdot F_j$$

Hence, the total weight of all experts would be $n \cdot \Pi_{j=1}^{|\sigma|}(1 - \epsilon \cdot F_j)$ at the end of the sequence, because we have n experts all initialized with weight one and reduce the weights by the second term over the whole sequence. Then,

$$
\begin{aligned}
& (1-\epsilon)^{m^*} \leq n \cdot \Pi_{j=1}^{|\sigma|}(1-\epsilon \cdot F_j) && \text{Expert } e^*\text{'s weight is part of the overall weight} \\
\Rightarrow \quad & (1-\epsilon)^{m^*} \leq n \cdot e^{\sum_{j=1}^{|\sigma|}(-\epsilon \cdot F_j)} && \text{Since } (1-x) \leq e^{-x} \\
\Rightarrow \quad & (1-\epsilon)^{m^*} \leq n \cdot e^{-\epsilon \cdot \mathbb{E}[m]} && \text{Since } \mathbb{E}[m] = \sum_{j=1}^{|\sigma|} F_j \\
\Rightarrow \quad & m^* \log(1-\epsilon) \leq \log n - \epsilon \cdot \mathbb{E}[m] && \text{Taking log on both sides} \\
\Rightarrow \quad & \mathbb{E}[m] \leq -\frac{\log(1-\epsilon)}{\epsilon} m^* + \frac{\log n}{\epsilon} && \text{Rearranging} \\
\Rightarrow \quad & \mathbb{E}[m] \leq (1+\epsilon)m^* + \frac{\log n}{\epsilon} && \text{Since } -\log(1-x) \leq -(-x-x^2) = x + x^2
\end{aligned}
$$

□

Generalization

The above results can be generalized, in a straightforward manner: Denote the loss of expert i on day t as $m_i^t \in [-\rho, \rho]$, for some constant ρ. When we incur a loss, update the weights of affected experts from w_i to $(1 - \epsilon\frac{m_i^t}{\rho})w_i$. Note that $\frac{m_i^t}{\rho}$ is essentially the normalized loss $\in [-1, 1]$.

Claim 8.6. *[Without proof] With RMWU, we have*

$$\mathbb{E}[m] \leq \min_i (\sum_t m_i^t + \epsilon \sum_t |m_i^t| + \frac{\rho \log n}{\epsilon})$$

Remark If each expert has a different ρ_i, one can modify the update rule and claim to use ρ_i instead of a uniform ρ accordingly.

8.2 Approximating Covering/Packing LPs via MWU

In this section, we see how ideas from multiplicative weights updates, as discussed above, give us a simple algorithm that computes approximate solutions for covering/packing linear programs, e.g. set cover, bin packing.
We particularly discuss the case of covering LPs; the approach for packing LPs is similar. In general, a covering LP can be formulated as follows:

Covering LP:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i & \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i \geq b_j & \forall j \in \{1, 2, \ldots, m\} \\ & x_i \geq 0 & \forall i \in \{1, 2, \ldots, n\} \end{aligned}$$

Here, we assume that all coefficients c_i, a_{ij}, and b_j are non-negative.

In the covering LP, x_i can be viewed as the number of objects of type i that is bought. The goal is to minimize the total cost of all bought objects such that all covering constraints are satisfied. For the set cover we assumed all a and b to be 1 (it is enough if any set covers an object).

First, we turn the problem into a feasibility question by turning the objective function into another constraint $\sum_{i=1}^{n} c_i x_i = K$. Using this modified feasibility variant we can solve the original covering LP via binary search (find the correct K by having a binary search over the possible K values).

Feasibility variant of a Covering LP:

Are there x_i, *for* $i \in [n]$, *such that*

$$\begin{aligned} & \sum_{i=1}^{n} c_i x_i = K & \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i \geq b_j & \forall j \in \{1, 2, \ldots, m\} \\ & x_i \geq 0 & \forall i \in \{1, 2, \ldots, n\} \end{aligned}$$

Here, we assume that all coefficients c_i, a_{ij}, and b_j are non-negative.

We will now use MWU to find an approximate solution $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, such that (A) $\sum_{i=1}^{n} c_i x_i = K$, and (B) $\sum_{i=1}^{n} a_{ij} x_i \geq b_j - \delta$, for all $j \in \{1, 2, \ldots, m\}$. Here, $\delta > 0$ is a desirably small positive parameter. By setting δ smaller and smaller, we get a better and better approximate solution. However, the runtime also grows as it depends on δ. To find such approximate solution, we appeal to the multiplicative weight updates method as follows: We think of each of the constraints $\sum_{i=1}^{n} a_{ij} x_i \geq b_j$ as one expert.

Thus, we have m experts. We start with a weight of $w_j = 1$ for each expert $j \in \{1, 2, \ldots, m\}$.

Each Iteration of MWU In any iteration t, define a coefficient p_j^t for each expert/constraint by normalizing the weights, i.e., setting $p_j^t = \frac{w_j}{\sum_j w_j}$. Instead of asking for all the m constraints to be satisfied, we ask that a linear mixture of them with these coefficients should be satisfied. That is, in iteration t, we find $\mathbf{x}^t = (x_1^t, x_2^t, \ldots, x_n^t)$ such that we have:

$$\sum_{i=1}^{n} c_i x_i^t = K, \text{ and}$$
$$\sum_{j=1}^{m} p_j^t \Big(\sum_{i=1}^{n} a_{ij} x_i^t \Big) \geq \sum_{j=1}^{m} p_j^t \cdot b_j$$

This is a much simpler problem with just two constraints. If the aforementioned feasibility problem has a YES answer—i.e. if it is feasible—then the same solution satisfies the above inequalities. We can easily find a solution $\mathbf{x}^t = (x_1^t, x_2^t, \ldots, x_n^t)$ that satisfies these two constraints (assuming the feasibility of the original problem for objective value K). We can maximize the left hand side of the inequality, subject to the equality constraint, using a greedy approach. We'll find the index i with the best value/cost ratio, i.e., the one which maximizes the ratio $\frac{\sum_j p_j^t a_{ij}}{c_i}$. Intuitively, the numerator can be seen as gain and the denominator as cost per unit. Then, we set variable $x_i^t = K/c_i$ to satisfy the equality constraint, while we set all other variables as $x_{i'}^t = 0$.

Weights update Once we have found such a solution $\mathbf{x}^t = (x_1^t, x_2^t, \ldots, x_n^t)$, we have to see how it fits the original feasibility problem. If this $\mathbf{x}^t$ already satisfies the original feasibility problem, we are done. More interestingly, suppose that this is not the case; then we need to update the weights of constraints. Intuitively, we want to increase the importance of the constraints that are violated by this solution $\mathbf{x}^t$ and decrease the importance of those that are satisfied. This should also take into account how strongly the constraint is violated/satisfied. Concretely, for the j^{th} constraint, we define $m_j^t = (\sum_{i=1}^{n} a_{ij} x_i^t) - b_j$. Notice that m_j^t is positive if we fulfill the inequality and negative if not. Then, we update the weight of this constraint as $w_j \leftarrow w_j(1 - \varepsilon \frac{m_j^t}{\rho})$. Here, ε is a small positive constant; we will discuss later how to set is so that we get the aforementioned approximation with additive

δ slack in each constraint. Also, ρ is a normalization factor and we set

$$\rho = \max\{1, \max_{\mathbf{x} \text{ s.t.} \sum_i c_i x_i = K} |(\sum_{i=1}^{n} a_{ij} x_i) - b_j|\}$$

Note that ρ does not depend on the current iteration t and can be computed using only the problem input.

Now that we have updated the weights we can start the next iteration, updating the p_j^t first and then using the same greedy approach again.

Final Output We run the above procedure for a number T of iterations. The proper value of T will be discussed. At the end, we output the average of all iterations as the final output, that is, we output

$$\bar{\mathbf{x}} = \frac{\sum_{t=1}^{T} \mathbf{x}^t}{T}$$

Theorem 8.7. *Suppose that the original feasibility problem is feasible. Then, for any given parameter $\delta > 0$, set $\varepsilon = \frac{\delta}{4\rho}$ and run the above procedure for $T = \frac{10\rho^2 \log m}{\delta^2}$ iterations. The final output $\bar{\mathbf{x}} = \frac{\sum_{t=1}^{T} \mathbf{x}^t}{T}$ satisfies each constraint up to additive δ error, besides fully satisfying the equality constraint $\sum_{i=1}^{n} c_i \bar{x}_i = K$. That is, for each constraint $j \in \{1, 2, \ldots, m\}$, we have*

$$\sum_{i=1}^{n} a_{ij} \bar{x}_i \geq b_j - \delta$$

Proof. We make use of Claim 8.6 which gives us:

$$\sum_t \mathbf{p}^t \mathbf{m}^t \leq \min_j (\sum_t m_j^t + \epsilon \sum_t |m_j^t| + \frac{\rho \log m}{\epsilon}) \tag{8.1}$$

Here, we have $\mathbf{p}^t = (p_1^t, p_2^t, \ldots, p_m^t)$ and $\mathbf{m}^t = (m_1^t, m_2^t, \ldots, m_m^t)$.
Notice that in iteration t, we chose $\mathbf{x}^t = (x_1^t, x_2^t, \ldots, x_m^t)$ such that we have $\sum_j p_j^t \big(\sum_{i=1}^{n} a_{ij} x_i^t \big) \geq \sum_j p_j^t \cdot b_j$. By rearranging this expression we get:

$$\sum_j p_j^t \left((\sum_{i=1}^{n} a_{ij} x_i^t) - b_j \right) = \mathbf{p}^t \mathbf{m}^t \geq 0$$

Hence, we conclude that the left hand side of 8.1 is non-negative, i.e., $\sum_t \mathbf{p}^t \mathbf{m}^t \geq 0$. Therefore, so is the right hand side. In particular, for any j, we have

$$0 \leq \sum_t m_j^t + \epsilon \sum_t |m_j^t| + \frac{\rho \log n}{\epsilon}$$

From this we get:

$$
\begin{aligned}
& 0 \leq \sum_t m_j^t + \epsilon \sum_t |m_j^t| + \frac{\rho \log m}{\epsilon} \\
\Rightarrow \quad & 0 \leq \sum_t m_j^t + \epsilon T \rho + \frac{\rho \log m}{\epsilon} && \rho \geq m_j^t \text{ for all } j \\
\Rightarrow \quad & 0 \leq \sum_t \left(\left(\sum_{i=1}^n a_{ij} x_i^t \right) - b_j \right) + \epsilon T \rho + \frac{\rho \log m}{\epsilon} && \text{by definition of } m_j^t \\
\Rightarrow \quad & 0 \leq \sum_t \frac{\left(\left(\sum_{i=1}^n a_{ij} x_i^t \right) - b_j \right)}{T} + \epsilon \rho + \frac{\rho \log m}{\epsilon \cdot T} && \text{divide by } T
\end{aligned}
$$

Now, since $\varepsilon = \frac{\delta}{4\rho}$ and $T = \frac{10\rho^2 \log m}{\delta^2}$, we see that:

$$
\begin{aligned}
& 0 \leq \sum_t \frac{\left(\left(\sum_{i=1}^n a_{ij} x_i^t \right) - b_j \right)}{T} + \frac{\delta}{4} + \frac{4\rho^2 \log m}{\delta \cdot T} && \text{plug in } \epsilon \\
\Rightarrow \quad & 0 \leq \sum_t \frac{\left(\left(\sum_{i=1}^n a_{ij} x_i^t \right) - b_j \right)}{T} + \frac{\delta}{4} + \frac{2\delta}{5} && \text{plug in T} \\
\Rightarrow \quad & 0 \leq \sum_t \frac{\left(\left(\sum_{i=1}^n a_{ij} x_i^t \right) - b_j \right)}{T} + \delta && \text{add and round } \delta \\
\Rightarrow \quad & \left(\sum_T \frac{b_j}{T} \right) - \delta \leq \sum_t \frac{\sum_{i=1}^n a_{ij} x_i^t}{T} && \text{split up sum and move to LHS} \\
\Rightarrow \quad & b_j - \delta \leq \sum_{i=1}^n a_{ij} \bar{x}_i && \begin{array}{r} \text{LHS: sum is independent of } t \\ \text{RHS: defintion of } \bar{x} \end{array}
\end{aligned}
$$

That is, the output $\bar{\mathbf{x}} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m)$ satisfies the j^{th} inequality up to additive error δ. This is proved for *all* constraints. □

Note that if the LP is not feasible, it will be detected during one of the iterations of the algorithm when no greedy solution will be found.

8.3 Constructive Oblivious Routing via MWU

In this section, we use the multiplicative weight updates method to provide an efficient algorithm that constructs the oblivious routing scheme that we discussed and showed to exist in the previous section.

Problem Recap Concretely, the algorithm finds a collection of trees T_i, each with a coefficient $\lambda_i \geq 0$, such that we have $\sum_i \lambda_i \geq 1$ and still, the following edge congestion condition is satisifed for each edge $\{u, v\}$ of the graph G:

$$\sum_i \lambda_i \sum_{\substack{\{x,y\} \in T_i \\ s.t.\ \{u,v\} \in P_i(x,y)}} C_i(x, y) \leq O(\log n) c_{uv}$$

Recall that in the above, for each tree T_i and each tree edge $\{x, y\} \in T_i$, we use $P_i(x, y)$ to denote the path that corresponds to the virtual edge $\{x, y\}$ and connects x and y in G. Moreover, we have

$$C_i(x, y) = \sum_{e \in cut(S(x,y), V \setminus S(x,y))} c_e$$

where $S(x, y)$ is one side of the cut that results from removing the edge $\{x, y\}$ from the tree T_i. For convenience, let us define

$$\text{load}_i(u, v) = \frac{\sum_{\substack{\{x,y\} \in T_i \\ s.t.\ \{u,v\} \in P_i(x,y)}} C_i(x, y)}{c_{uv}}$$

as the relative load that the i^{th} tree in the collection places on the edge $\{u, v\}$. Thus, our task is to find a collection so that

$$\sum_i \lambda_i \cdot \text{load}_i(u, v) \leq O(\log n)$$

Construction Plan We start with an empty collection and add trees to the collection. In iteration j, we add a new tree T_j with a coefficient $\lambda_j \in (0, 1]$. The construction ends once we find j_{end} such that $\sum_{i=1}^{j_{\text{end}}} \lambda_i \geq 1$. During the construction, we think of each constraint $\sum_i \lambda_i \cdot \text{load}_i(u, v) \leq O(\log n)$ as one of our experts. We have one constraint for each edge $\{u, v\} \in G$. To

track the performance of each of the constraints (experts) in the course of this construction, we use a potential function defined as follows:

$$\Phi_j = \sum_{\{u,v\}\in G} \exp\left(\sum_{i=1}^{j} \lambda_i \cdot \mathrm{load}_i(u,v)\right)$$

The initial potential is equal to the number of the edges of the graph. Thus, $\Phi_0 < n^2$. When we add the tree T_j with coefficient λ_j in iteration j of the construction, we want:

$$\frac{\Phi_j}{\Phi_{j-1}} = \frac{\sum_{\{u,v\}\in G} \exp\left(\sum_{i=1}^{j} \lambda_i \cdot \mathrm{load}_i(u,v)\right)}{\sum_{\{u,v\}\in G} \exp\left(\sum_{i=1}^{j-1} \lambda_i \cdot \mathrm{load}_i(u,v)\right)} \leq \exp(\lambda_j \cdot O(\log n))$$

This ensures $\Phi_{j_{\mathrm{end}}} \leq n^2 \exp(\sum_i \lambda_i \cdot O(\log n))$. Note that at this point we have $1 \leq \sum_i \lambda_i \leq 2$ and thus $\Phi_{j_{\mathrm{end}}} \leq n^2 \exp(O(\log n)) = \exp(O(\log n))$. Monotonicity of the logarithm then implies for each edge $\{u,v\} \in G$:

$$\sum_{i=1}^{j_{\mathrm{end}}} \lambda_i \cdot \mathrm{load}_i(u,v) \leq O(\log n)$$

It remains to show how we find a tree in each iteration and to bound the number of iterations.

One Iteration, Finding One Tree We now focus on iteration j and we explain how we find a tree T_j with coefficient λ_j such that the addition of this tree to the collection ensures $\frac{\Phi_j}{\Phi_{j-1}} \leq \exp(\lambda_j \cdot O(\log n))$.

Let us first examine the potential change factor $\frac{\Phi_j}{\Phi_{j-1}}$ and bound it in a more convenient manner:

$$\begin{aligned}
\frac{\Phi_j}{\Phi_{j-1}} &= \frac{\sum_{\{u,v\}\in G} \exp\left(\sum_{i=1}^{j} \lambda_i \cdot \mathrm{load}_i(u,v)\right)}{\sum_{\{u,v\}\in G} \exp\left(\sum_{i=1}^{j-1} \lambda_i \cdot \mathrm{load}_i(u,v)\right)} \\
&= 1 + \frac{\sum_{\{u,v\}\in G} \left[\exp\left(\sum_{i=1}^{j} \lambda_i \cdot \mathrm{load}_i(u,v)\right) - \exp\left(\sum_{i=1}^{j-1} \lambda_i \cdot \mathrm{load}_i(u,v)\right)\right]}{\sum_{\{u,v\}\in G} \exp\left(\sum_{i=1}^{j-1} \lambda_i \cdot \mathrm{load}_i(u,v)\right)} \\
&= 1 + \frac{\sum_{\{u,v\}\in G} \left[\exp\left(\sum_{i=1}^{j-1} \lambda_i \cdot \mathrm{load}_i(u,v)\right) \cdot \left(\exp(\lambda_j \cdot \mathrm{load}_j(u,v)) - 1\right)\right]}{\sum_{\{u,v\}\in G} \exp\left(\sum_{i=1}^{j-1} \lambda_i \cdot \mathrm{load}_i(u,v)\right)}
\end{aligned}$$

We choose λ_j small enough such that for each edge $\{u,v\} \in G$, we have $\lambda_j \cdot \mathrm{load}_j(u,v) \leq 1$. Observing this, we can make use of the inequality $e^z \leq 1 + 2z$, for all $z \leq [0,1]$. We can now upper bound

$$\frac{\Phi_j}{\Phi_{j-1}} = 1 + \frac{\sum_{\{u,v\}\in G}\left[\exp\left(\sum_{i=1}^{j-1}\lambda_i\cdot\text{load}_i(u,v)\right)\cdot\left(\exp(\lambda_j\cdot\text{load}_j(u,v))-1\right)\right]}{\sum_{\{u,v\}\in G}\exp\left(\sum_{i=1}^{j-1}\lambda_i\cdot\text{load}_i(u,v)\right)}$$
$$\leq 1 + \frac{\sum_{\{u,v\}\in G}\left[\exp\left(\sum_{i=1}^{j-1}\lambda_i\cdot\text{load}_i(u,v)\right)\cdot\left(2\lambda_i\cdot\text{load}_j(u,v)\right)\right]}{\sum_{\{u,v\}\in G}\exp\left(\sum_{i=1}^{j-1}\lambda_i\cdot\text{load}_i(u,v)\right)}$$

We next show that we can find a tree T_j such that

$$\frac{\sum_{\{u,v\}\in G}\left[\exp\left(\sum_{i=1}^{j-1}\lambda_i\cdot\text{load}_i(u,v)\right)\cdot\left(2\cdot\text{load}_j(u,v)\right)\right]}{\sum_{\{u,v\}\in G}\exp\left(\sum_{i=1}^{j-1}\lambda_i\cdot\text{load}_i(u,v)\right)} \leq O(\log n)$$

This then gives $\frac{\Phi_j}{\Phi_{j-1}} \leq \exp(\lambda_j\cdot O(\log n))$, with the use of the inequality $1+y\leq e^y$.

In order to find such a tree, we define the length of an edge $\{u,v\}$ by

$$\ell(u,v) = \frac{\exp\left(\sum_{i=1}^{j-1}\lambda_i\cdot\text{load}_i(u,v)\right)}{c_{uv}\cdot\sum_{\{u',v'\}\in G}\exp\left(\sum_{i=1}^{j-1}\lambda_i\cdot\text{load}_i(u',v')\right)}$$

After inserting the definition of the relative load of an edge, the condition on tree T_j from above translates to:

$$\sum_{\{u,v\}\in G}\left(\ell(u,v)\cdot\Big(\sum_{\substack{\{x,y\}\in T_i\\ s.t.\,\{u,v\}\in P_i(x,y)}} C_i(x,y)\Big)\right) \leq O(\log n)$$

Recall the probabilistic distance-preserving tree embedding discussed in Chapter 5. Let $\mathcal{T}$ be the corresponding collection of trees. For an edge $\{x,y\}$ of some tree of the collection $\mathcal{T}$, we define the corresponding path $P(x,y)$ as a shortest path on the graph G. Then, we can bound the right hand side of the above inequality, in expectation, as follows:

$$
\begin{aligned}
&\mathbb{E}_{T\in\mathcal{T}}\Big[\sum_{\{u,v\}\in G}\Big(\ell(u,v)\cdot\big(\sum_{\substack{\{x,y\}\in T\\ s.t.\,\{u,v\}\in P(x,y)}} C(x,y)\big)\Big)\Big]\\
&=\mathbb{E}_{T\in\mathcal{T}}\Big[\sum_{\{u,v\}\in G}\Big(\ell(u,v)\cdot\big(\sum_{\substack{\{x,y\}\in T\\ s.t.\,\{u,v\}\in P(x,y)}}\ \sum_{e\in cut(S(x,y),V\setminus S(x,y))} c_e,\big)\Big)\Big]\\
&=\mathbb{E}_{T\in\mathcal{T}}\Big[\sum_e c_e \sum_{\substack{\{x,y\}\in T\\ s.t.\,e\in cut(S(x,y),V\setminus S(x,y))}}\ \sum_{\{u,v\}\in P(x,y)} \ell(u,v)\Big]\\
&=\mathbb{E}_{T\in\mathcal{T}}\Big[\sum_e c_e\cdot \mathrm{dist}_T(e)\Big]=\sum_e c_e\cdot\mathbb{E}_{T\in\mathcal{T}}[\mathrm{dist}_T(e)]\\
&\le\sum_e c_e\cdot O(\log n)\cdot\ell(e)=O(\log n)\cdot\sum_e c_e\cdot\ell(e)\\
&=O(\log n)\cdot\sum_e c_e\cdot\frac{\exp\big(\sum_{i=1}^{j-1}\lambda_i\cdot\mathrm{load}_i(e)\big)}{c_e\cdot\sum_{\{u,v\}\in G}\exp\big(\sum_{i=1}^{j-1}\lambda_i\cdot\mathrm{load}_i(u,v)\big)}\\
&=O(\log n)\cdot\sum_e\frac{\exp\big(\sum_{i=1}^{j-1}\lambda_i\cdot\mathrm{load}_i(e)\big)}{\sum_{\{u,v\}\in G}\exp\big(\sum_{i=1}^{j-1}\lambda_i\cdot\mathrm{load}_i(u,v)\big)}\\
&=O(\log n)
\end{aligned}
$$

The equations above show that taking one probabilistic tree from the distribution satisfies the inequality in expectation. Hence, thanks to Markov's inequality and by increasing the left hand side by a constant factor, we know that the inequality is satisfied with probability at least 1/2. Therefore, if we run the algorithm for $O(\log n)$ independent repetitions, with high probability, we find one tree that satisfies the inequality. This is the tree T_j that we add to our collection. Moreover, we set λ_j such that $\max_{\{u,v\}\in G}\lambda_j\cdot\mathrm{load}_j(u,v)=1$.

Bounding the Number of Iterations We now show that the construction uses only $O(m\log n)$ iterations. In each iteration j, we set λ_j for the new tree so that for at least one edge $\{u,v\}$, we have

$$\lambda_j\cdot\mathrm{load}_j(u,v)=1$$

As argued above, at the end all edges $\{u,v\}$ satisfy:

$$\sum_{i=1}^{j_{\mathrm{end}}}\lambda_i\cdot\mathrm{load}_i(u,v)\le O(\log n)$$

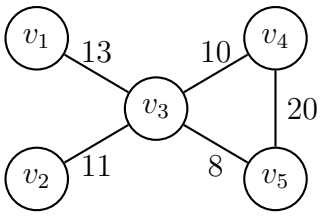
Hence, we have at most $O(m\log n)$ iterations.

8.4 Other Applications: Online routing of virtual circuits

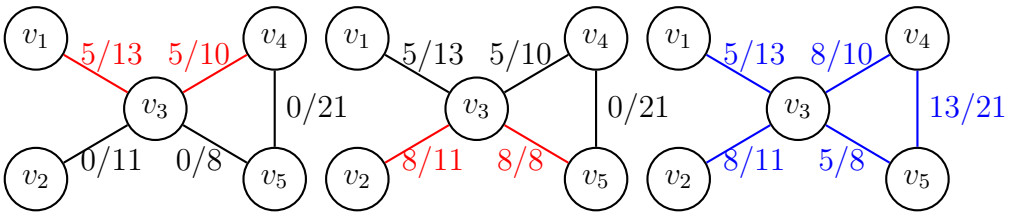
Definition 8.8 (The online routing of virtual circuits problem)**.** *Consider a graph $G = (V, E)$ where each edge $e \in E$ has a capacity u_e. A request is denoted by a triple $\langle s(i), t(i), d(i) \rangle$, where $s(i) \in V$ is the source, $t(i) \in V$ is the target, and $d(i) > 0$ is the demand for the i^{th} request respectively. Given the i^{th} request, we have to build a connection (single path P_i) from $s(i)$ to $t(i)$ with flow $d(i)$. The objective is to minimize the maximum congestion on all edges as we handle requests in an online manner. To be precise, we wish to minimize $\max_{e \in E} \frac{\sum_{i=1}^{|\sigma|} \sum_{P_i \ni e} d(i)}{u_e}$ on the input sequence σ, where $P_i \ni e$ is the set of paths that include edge e.*

Remark This is similar to the multi-commodity routing problem in chapter 4. However, in this problem, each commodity flow cannot be split into multiple paths, and the commodities appear in an online fashion.

Example Consider the following graph $G = (V, E)$ with 5 vertices and 5 edges with the edge capacities u_e annotated for each edge $e \in E$. Suppose there are 2 requests: $\sigma = (\langle v_1, v_4, 5 \rangle, \langle v_5, v_2, 8 \rangle)$.



Upon seeing $\sigma(1) = \langle v_1, v_4, 5 \rangle$, in an online algorithm (red edges) we commit to $P_1 = v_1 - v_3 - v_4$ as it minimizes the congestion to 5/10. When $\sigma(2) = \langle v_5, v_2, 8 \rangle$ appears, $P_2 = v_5 - v_3 - v_2$ minimizes the congestion given that we committed to P_1. This causes the congestion to be $8/8 = 1$. On the other hand, the optimal offline algorithm (blue edges) can attain a congestion of 8/10 via $P_1 = v_1 - v_3 - v_5 - v_4$ and $P_2 = v_5 - v_4 - v_3 - v_2$.



To facilitate further discussion, we define the following notations:

- $p_e(i) = \frac{d(i)}{u_e}$ is the demand of the i^{th} request with respect to the capacity of edge e.
- $l_e(j) = \sum_{P_i \ni e, i \leq j} p_e(i)$ is the relative load of edge e *after* request j.
- $l_e^*(j)$ is the optimal offline algorithm's relative load of edge e *after* request j.

In other words, the objective is to minimize $\max_{e \in E} l_e(|\sigma|)$ for a given sequence σ. Denoting Λ as the (unknown) optimal congestion factor, we normalize $\widetilde{p}_e(i) = \frac{p_e(i)}{\Lambda}$, $\widetilde{l}_e(j) = \frac{l_e(j)}{\Lambda}$, and $\widetilde{l}_e^*(j) = \frac{l_e^*(j)}{\Lambda}$. Let a be a constant to be determined. Consider the algorithm $\mathcal{A}$ which does the following on request $i+1$:

- Denote the cost of edge e by $c_e = a^{\widetilde{l}_e(i) + \widetilde{p}_e(i+1)} - a^{\widetilde{l}_e(i)}$.
- Return a shortest (smallest total cost) $s(i) - t(i)$ path P_i on G with edge weights c_e.

Finding the shortest path with respect to the cost function c_e tries to minimize the load impact of the new $(i+1)^{th}$ request. To analyze $\mathcal{A}$, we consider the following potential function: $\Phi(j) = \sum_{e \in E} a^{\widetilde{l}_e(j)}(\gamma - \widetilde{l}_e^*(j))$, for some constant $\gamma \geq 2$. Because of normalization, $\widetilde{l}_e^*(j) \leq 1$, so $\gamma - \widetilde{l}_e^*(j) \geq 1$. Initially, we have $\Phi(0) = \sum_{e \in E} \gamma = m\gamma$.

Lemma 8.9. *For $\gamma \geq 1$ and $0 \leq x \leq 1$, then $(1 + \frac{1}{2\gamma})^x < 1 + \frac{x}{\gamma}$.*

Proof. By Taylor series[2], $(1 + \frac{1}{2\gamma})^x = 1 + \frac{x}{2\gamma} + \mathcal{O}(\frac{x}{2\gamma}) < 1 + \frac{x}{\gamma}$. □

Lemma 8.10. *For $a = 1 + \frac{1}{2\gamma}$ and $\gamma \geq 1$, then $\Phi(j+1) - \Phi(j) \leq 0$.*

Proof. Let P_{j+1} be the path that algorithm $\mathcal{A}$ found and let P_{j+1}^* be the path that the optimal offline algorithm assigned to the $(j+1)^{th}$ request $\langle s(j+1), t(j+1), d(j+1) \rangle$. For any edge e, observe the following:

- If $e \notin P_{j+1}^*$, the load on e caused by the optimal offline algorithm remains unchanged. That is $\widetilde{l}_e^*(j+1) = \widetilde{l}_e^*(j)$. On the other hand, if $e \in P_{j+1}^*$, then $\widetilde{l}_e^*(j+1) = \widetilde{l}_e^*(j) + \widetilde{p}_e(j+1)$.
- Similarly, if $e \notin P_{j+1}$, then $\widetilde{l}_e(j+1) = \widetilde{l}_e(j)$; if $e \in P_{j+1}$, then $\widetilde{l}_e(j+1) = \widetilde{l}_e(j) + \widetilde{p}_e(j+1)$.

[2]See https://en.wikipedia.org/wiki/Taylor_series#Binomial_series

- Thus if e is neither in P_{j+1} nor in P^*_{j+1}, then $a^{\widetilde{l}_e(j+1)}(\gamma - \widetilde{l}^*_e(j+1)) = a^{\widetilde{l}_e(j)}(\gamma - \widetilde{l}^*_e(j))$. So only edges used by P_{j+1} or P^*_{j+1} affect $\Phi(j+1) - \Phi(j)$.

Using the observations above together with Lemma 8.9 and the fact that $\mathcal{A}$ computes a shortest path, one can show that $\Phi(j+1) - \Phi(j) \leq 0$. In detail:

$$
\begin{aligned}
&\Phi(j+1) - \Phi(j) \\
=&\sum_{e \in E} a^{\widetilde{l}_e(j+1)}(\gamma - \widetilde{l}^*_e(j+1)) - a^{\widetilde{l}_e(j)}(\gamma - \widetilde{l}^*_e(j)) \\
=&\sum_{e \in P_{j+1} \setminus P^*_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})(\gamma - \widetilde{l}^*_e(j)) && (1) \\
&+ \sum_{e \in P^*_{j+1}} a^{\widetilde{l}_e(j+1)}(\gamma - \widetilde{l}^*_e(j) - \widetilde{p}_e(j+1)) - a^{\widetilde{l}_e(j)}(\gamma - \widetilde{l}^*_e(j)) \\
=&\sum_{e \in P_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})(\gamma - \widetilde{l}^*_e(j)) - \sum_{e \in P^*_{j+1}} a^{\widetilde{l}_e(j+1)} \widetilde{p}_e(j+1) \\
\leq&\sum_{e \in P_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - \sum_{e \in P^*_{j+1}} a^{\widetilde{l}_e(j+1)} \widetilde{p}_e(j+1) && (2) \\
\leq&\sum_{e \in P_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - \sum_{e \in P^*_{j+1}} a^{\widetilde{l}_e(j)} \widetilde{p}_e(j+1) && (3) \\
=&\sum_{e \in P_{j+1}} (a^{\widetilde{l}_e(j) + \widetilde{p}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - \sum_{e \in P^*_{j+1}} a^{\widetilde{l}_e(j)} \widetilde{p}_e(j+1) && (4) \\
=&\sum_{e \in P_{j+1}} c_e \gamma - \sum_{e \in P^*_{j+1}} a^{\widetilde{l}_e(j)} \widetilde{p}_e(j+1) && (5) \\
\leq&\sum_{e \in P^*_{j+1}} \Big(c_e \gamma - a^{\widetilde{l}_e(j)} \widetilde{p}_e(j+1) \Big) && (6) \\
=&\sum_{e \in P^*_{j+1}} \Big((a^{\widetilde{l}_e(j) + \widetilde{p}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - a^{\widetilde{l}_e(j)} \widetilde{p}_e(j+1) \Big) \\
=&\sum_{e \in P^*_{j+1}} a^{\widetilde{l}_e(j)} \Big((a^{\widetilde{p}_e(j+1)} - 1)\gamma - \widetilde{p}_e(j+1) \Big) \\
=&\sum_{e \in P^*_{j+1}} a^{\widetilde{l}_e(j)} \Big(((1 + \frac{1}{2\gamma})^{\widetilde{p}_e(j+1)} - 1)\gamma - \widetilde{p}_e(j+1) \Big) && (7) \\
\leq&\ 0 && (8)
\end{aligned}
$$

(1) From the observations above

(2) $\widetilde{l}_e^*(j) \geq 0$

(3) $\widetilde{l}_e(j+1) \geq \widetilde{l}_e(j)$ and $a > 1$

(4) For $e \in P_{j+1}$, $\widetilde{l}_e(j+1) = \widetilde{l}_e(j) + \widetilde{p}_e(j+1)$

(5) Since $c_e = a^{\widetilde{l}_e(j)+\widetilde{p}_e(j+1)} - a^{\widetilde{l}_e(j)}$

(6) Since P_{j+1} is a shortest path (with respect to c_e)

(7) Since $a = 1 + \frac{1}{2\gamma}$

(8) Lemma 8.9 with $0 \leq \widetilde{p}_e(j+1) \leq 1$

□

Theorem 8.11. *Let $L = \max_{e \in E} \widetilde{l}_e(|\sigma|)$ be the maximum normalized load at the end of the input sequence σ. For $\gamma \geq 2, a = 1 + \frac{1}{2\gamma}$ and $L \in \mathcal{O}(\log n)$. Then $\mathcal{A}$ is $\mathcal{O}(\log n)$-competitive.*

Proof. Since $\Phi(0) = m\gamma$ and $\Phi(j+1) - \Phi(j) \leq 0$, we see that $\Phi(j) \leq m\gamma$, for all $j \in \{1, \ldots, |\sigma|\}$. Consider the edge e with the highest congestion. Since $\gamma - \widetilde{l}_e^*(j) \geq 1$, we see that

$$\left(1 + \frac{1}{2\gamma}\right)^L \leq a^L \cdot \left(\gamma - \widetilde{l}_e^*(j)\right) \leq \Phi(j) \leq m\gamma \leq n^2\gamma$$

Taking the logarithm on both sides and rearranging, we get:

$$L \leq (2\log(n) + \log(\gamma)) \cdot \frac{1}{\log(1 + \frac{1}{2\gamma})} \in \mathcal{O}(\log n)$$

□

Handling unknown Λ Since Λ is unknown but is needed for the run of $\mathcal{A}$ (to compute c_e when a request arrives), we use a dynamically estimated $\widetilde{\Lambda}$. Let β be a constant such that $\mathcal{A}$ is β-competitive according to Theorem 8.11. The following modification to $\mathcal{A}$ is a 4β-competitive: On the first request, we can explicitly compute $\widetilde{\Lambda} = \Lambda$. Whenever the actual congestion exceeds $\widetilde{\Lambda}\beta$, we reset[3] the edge loads to 0, update our estimate to $2\widetilde{\Lambda}$, and start a new phase. Thus, we have:

[3]Existing paths are preserved, just that we ignore them in the subsequent computations of c_e.

- By the updating procedure, $\widetilde{\Lambda} \leq 2\beta\Lambda$ in all phases.
- Let T be the total number of phases. In any phase $i \leq T$, the congestion at the end of phase i is at most $\frac{2\beta\Lambda}{2^{T-i}}$. Across all phases, we have $\sum_{i=1}^{T} \frac{2\beta\Lambda}{2^{T-i}} \leq 4\beta\Lambda$.

Part III

Streaming and Sketching Algorithms

Chapter 9

Basics and Warm Up with Majority Element

Thus far, we have been ensuring that our algorithms run fast. What if our system does not have sufficient memory to store all data to post-process it? For example, a router has relatively small amount of memory while tremendous amount of routing data flows through it. In a memory constrained setting, can one compute something meaningful, possible approximately, with limited amount of memory?

More formally, we now look at a slightly different class of algorithms where data elements from $[n] = \{1, \ldots, n\}$ arrive in one at a time, in a stream $S = a_1, \ldots, a_m$, where $a_i \in [n]$ arrives in the i^{th} time step. At each step, our algorithm performs some computation[1] and discards the item a_i. At the end of the stream[2], the algorithm should give us a value that approximates some value of interest.

9.1 Typical tricks

Before we begin, let us first describe two typical tricks used to amplify success probabilities of randomized algorithms. Suppose we have a randomized algorithm $\mathcal{A}$ that returns an unbiased estimate of a quantity of interest X on a problem instance I, with success probability $p > 0.5$.

Trick 1: Reduce variance Run j *independent* copies of $\mathcal{A}$ on I, and return the mean $\frac{1}{j}\sum_{i=1}^{j} A(I)$. The expected outcome $\mathbb{E}(\frac{1}{j}\sum_{i=1}^{j} \mathcal{A}(I))$ will still be X while the variance drops by a factor of j.

[1] Usually this is constant time so we ignore the runtime.

[2] In general, the length of the stream, m, may not be known.

Trick 2: Improve success Run k independent copies of $\mathcal{A}$ on I, and return the median. As each copy of $\mathcal{A}$ succeeds (independently) with probability $p > 0.5$, the probability that more than half of them fails (and hence the median fails) drops exponentially with respect to k.

Let $\epsilon > 0$ and $\delta > 0$ denote the precision factor and failure probability respectively. ROBUST combines the above-mentioned two tricks to yield a $(1 \pm \epsilon)$-approximation to X that succeeds with probability $> 1 - \delta$.

Algorithm 19 ROBUST$(\mathcal{A}, I, \epsilon, \delta)$

```
C ← ∅                                          ▷ Initialize candidate outputs
for k = O(log 1/δ) times do
    sum ← 0
    for j = O(1/ε²) times do
        sum ← sum + A(I)
    end for
    Add sum/j to candidates C                  ▷ Include new sample of mean
end for
return Median of C                             ▷ Return median
```

9.2 Majority element

Definition 9.1 ("Majority in a stream" problem)**.** *Given a stream $S = \{a_1, \ldots, a_m\}$ of items from $[n] = \{1, \ldots, n\}$, with an element $j \in [n]$ that appears strictly more than $\frac{m}{2}$ times in S, find j.*

Example Consider a stream $S = \{1, 3, 3, 7, 5, 3, 2, 3\}$. The table below shows how *guess* and *count* are updated as each element arrives.

Stream elements	1	3	3	7	5	3	2	3
Guess	1	3	3	3	5	3	2	3
Count	1	1	2	1	1	1	1	1

One can verify that MAJORITYSTREAM uses $\mathcal{O}(\log n + \log m)$ bits to store *guess* and *counter*.

Claim 9.2. MAJORITYSTREAM *correctly finds element $j \in [n]$ which appears $> \frac{m}{2}$ times in $S = \{a_1, \ldots, a_m\}$.*

Proof. (Sketch) Match each *other* element in S with a distinct instance of j. Since j appears $> \frac{m}{2}$ times, at least one j is unmatched. As each matching cancels out *count*, only j could be the final *guess*. □

Algorithm 20 MajorityStream($S = \{a_1, \ldots, a_m\}$)

```
guess ← 0
count ← 0
for a_i ∈ S do                          ▷ Items arrive in streaming fashion
    if a_i = guess then
        count ← count + 1
    else if count > 1 then
        count ← count − 1
    else
        guess ← a_i
    end if
end for
return guess
```

Remark If no element appears $> \frac{m}{2}$ times, then MajorityStream is not guaranteed to return the most frequent element. For example, for $S = \{1, 3, 4, 3, 2\}$, MajorityStream(S) returns 2 instead of 3.

Chapter 10

Estimating the moments of a stream

One class of interesting problems is computing moments of a given stream S. For items $j \in [n]$, define f_j as the number of times j appears in a stream S. Then, the k^{th} moment of a stream S is defined as $\sum_{j=1}^{n}(f_j)^k$. When $k = 1$, the first moment $\sum_{j=1}^{n} f_j = m$ is simply the number of elements in the stream S. When $k = 0$, by associating $0^0 = 0$, the zeroth moment $\sum_{j=1}^{n}(f_j)^0$ is the number of distinct elements in the stream S.

10.1 Estimating the first moment of a stream

A trivial exact solution would be to use $\mathcal{O}(\log m)$ bits to maintain a counter, incrementing for each element observed. For some upper bound M, consider the sequence $(1+\epsilon), (1+\epsilon)^2, \ldots, (1+\epsilon)^{\log_{1+\epsilon} M}$. For any stream length m, there exists $i \in \mathbb{N}$ such that $(1+\epsilon)^i \leq m \leq (1+\epsilon)^{i+1}$. Thus, to obtain a $(1+\epsilon)$-approximation, it suffices to track the exponent i to estimate the length of m. For $\epsilon \in \Theta(1)$, this can be done in $\mathcal{O}(\log \log m)$ bits.

Algorithm 21 MORRIS($S = \{a_1, \ldots, a_m\}$)

```
x ← 0
for a_i ∈ S do                          ▷ Items arrive in streaming fashion
    r ← Random probability from [0, 1]
    if r ≤ 2^{-x} then                  ▷ If not, x is unchanged.
        x ← x + 1
    end if
end for
return 2^x − 1                          ▷ Estimate m by 2^x − 1
```

The intuition behind MORRIS [Mor78] is to increase the counter (and hence double the estimate) when we expect to observe 2^x new items. For analysis, denote X_m as the value of counter x after exactly m items arrive.

Theorem 10.1. $\mathbb{E}[2^{X_m} - 1] = m$. *That is,* MORRIS *is an unbiased estimator for the length of the stream.*

Proof. Equivalently, let us prove $\mathbb{E}[2^{X_m}] = m + 1$, by induction on $m \in \mathbb{N}^+$. On the first element ($m = 1$), x increments with probability 1, so $\mathbb{E}[2^{X_1}] = 2^1 = m + 1$. Suppose it holds for some $m \in \mathbb{N}$, then

$$
\begin{aligned}
\mathbb{E}[2^{X_{m+1}}] &= \sum_{j=1}^{m} \mathbb{E}[2^{X_{m+1}} | X_m = j] \Pr[X_m = j] && \text{Condition on } X_m \\
&= \sum_{j=1}^{m} (2^{j+1} \cdot 2^{-j} + 2^j \cdot (1 - 2^{-j})) \cdot \Pr[X_m = j] && \text{Increment } x \text{ w.p. } 2^{-j} \\
&= \sum_{j=1}^{m} (2^j + 1) \cdot \Pr[X_m = j] && \text{Simplifying} \\
&= \sum_{j=1}^{m} 2^j \cdot \Pr[X_m = j] + \sum_{j=1}^{m} \Pr[X_m = j] && \text{Splitting the sum} \\
&= \mathbb{E}[2^{X_m}] + \sum_{j=1}^{m} \Pr[X_m = j] && \text{Definition of } \mathbb{E}[2^{X_m}] \\
&= \mathbb{E}[2^{X_m}] + 1 && \sum_{i=1}^{m} \Pr[X_m = j] = 1 \\
&= (m + 1) + 1 && \text{Induction hypothesis} \\
&= m + 2
\end{aligned}
$$

Note that we sum up to m because $x \in [1, m]$ after m items. □

Claim 10.2. $\mathbb{E}[2^{2X_m}] = \frac{3}{2}m^2 + \frac{3}{2}m + 1$

Proof. Exercise. □

Claim 10.3. *Var*$(2^{X_m} - 1) = \mathbb{E}[(2^{X_m} - 1 - m)^2] \leq \frac{m^2}{2}$

Proof. Exercise. Use the Claim 10.2. □

Theorem 10.4. *For* $\epsilon > 0$, $\Pr[|(2^{X_m} - 1) - m| > \epsilon m] \leq \frac{1}{2\epsilon^2}$

Proof.

$$\begin{aligned}
\Pr[|(2^{X_m}-1)-m| > \epsilon m] &\leq \frac{\mathrm{Var}(2^{X_m}-1)}{(\epsilon m)^2} && \text{Chebyshev's inequality} \\
&\leq \frac{m^2/2}{\epsilon^2 m^2} && \text{By Claim 10.3} \\
&= \frac{1}{2\epsilon^2}
\end{aligned}$$

□

Remark Using the discussion in Section 9.1, we can run MORRIS multiple times to obtain a $(1 \pm \epsilon)$-approximation of the first moment of a stream that succeeds with probability $> 1 - \delta$. For instance, repeating MORRIS $\frac{10}{\epsilon^2}$ times and reporting the mean $\widehat{m}$, $\Pr[|\widehat{m} - m| > \epsilon m] \leq \frac{1}{20}$ because the variance is reduced by $\frac{\epsilon^2}{10}$.

10.2 Estimating the zeroth moment of a stream

Trivial exact solutions could either use $\mathcal{O}(n)$ bits to track if element exists, or use $\mathcal{O}(m \log n)$ bits to remember the whole stream. Suppose there are D distinct items in the whole stream. In this section, we show that one can in fact make do with only $\mathcal{O}(\log n)$ bits to obtain an approximation of D.

10.2.1 An idealized algorithm

Consider the following algorithm sketch:

1. Take a uniformly random hash function $h : \{1, \ldots, m\} \to [0, 1]$
2. As items $a_i \in S$ arrive, track $z = \min\{h(a_i)\}$
3. In the end, output $\frac{1}{z} - 1$

Since we are randomly hashing elements into the range $[0, 1]$, we expect the minimum hash output to be $\frac{1}{D+1}$[1], so $\mathbb{E}[\frac{1}{z} - 1] = D$. Unfortunately, storing a uniformly random hash function that maps to the interval $[0, 1]$ is infeasible. As storing real numbers is memory intensive, one possible fix is to discretize the interval $[0, 1]$, using $\mathcal{O}(\log n)$ bits per hash output. However, storing this hash function would still require $\mathcal{O}(n \log n)$ space.

[1] See https://en.wikipedia.org/wiki/Order_statistic

10.2.2 An actual algorithm

Instead of a uniformly random hash function, we select a random hash from a family of pairwise independent hash functions.

Definition 10.5 (Family of pairwise independent hash functions)**.** $\mathcal{H}_{n,m}$ *is a family of pairwise independent hash functions if*

- *(Hash definition):* $\forall h \in \mathcal{H}_{n,m}$, $h : \{1, \ldots, n\} \to \{1, \ldots, m\}$
- *(Uniform hashing):* $\forall x \in \{1, \ldots, n\}$, $\Pr_{h \in \mathcal{H}_{n,m}}[h(x) = i] = \frac{1}{m}$
- *(Pairwise independent)* $\forall x, y \in \{1, \ldots, n\}$, $x \neq y$, $\Pr_{h \in \mathcal{H}_{n,m}}[h(x) = i \wedge h(y) = j] = \frac{1}{m^2}$

Remark For now, we care only about $m = n$, and write $\mathcal{H}_{n,n}$ as $\mathcal{H}_n$.

Claim 10.6. *Let n be a prime number. Then,*

$$\mathcal{H}_n = \{h_{a,b} : h(x) = ax + b \mod n, \forall a, b \in \mathbb{Z}_n\}$$

is a family of pairwise independent hash functions.

Proof. (Sketch) For any given $x \neq y$,

- There is a unique value of $h(x) \mod n$, out of n possibilities.
- The system $\{ax + b = i \mod n, ay + b = j \mod n\}$ has a unique solution for (a, b) (note that $\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix} \in \mathbb{Z}_n^{2\times 2}$ is non-singular), out of n^2 possibilities.

□

Remark If n is not a prime, we know there exists a prime p such that $n \leq p \leq 2n$, so we round n up to p. Storing a random hash from $\mathcal{H}_n$ is then storing the numbers a and b in $\mathcal{O}(\log n)$ bits.

We now present an algorithm [FM85] which estimates the zeroth moment of a stream and defer the analysis to the next lecture. In FM, ZEROS refer to the number of trailing zeroes in the binary representation of $h(a_i)$. For example, if $h(a_i) = 20 = (...10100)_2$, then $\text{ZEROS}(h(a_i)) = 2$.

Recall that the k^{th} moment of a stream S is defined as $\sum_{j=1}^{n} (f_j)^k$. Since the hash h is deterministic after picking a random hash from $\mathcal{H}_{n,n}$, $h(a_i) = h(a_j), \forall a_i = a_j \in [n]$. We first prove a useful lemma.

Algorithm 22 FM($S = \{a_1, \ldots, a_m\}$)

$h \leftarrow$ Random hash from $\mathcal{H}_{n,n}$
$Z \leftarrow 0$
for $a_i \in S$ **do** ▷ Items arrive in streaming fashion
  $Z = \max\{Z, \text{ZEROS}(h(a_i))\}$
  (ZEROS$(h(a_i))$ = # leading zeroes in binary representation of $h(a_i)$)
end for
return $2^Z \cdot \sqrt{2}$ ▷ Estimate of D

Lemma 10.7. *If $X_1, \ldots, X_n$ are pairwise independent indicator random variables and $X = \sum_{i=1}^{n} X_i$, then $Var(X) \leq \mathbb{E}[X]$.*

Proof.

$$\begin{aligned}
\text{Var}(X) &= \sum_{i=1}^{n} \text{Var}(X_i) && \text{The } X_i\text{'s are pairwise independent} \\
&= \sum_{i=1}^{n} (\mathbb{E}[X_i^2] - (\mathbb{E}[X_i])^2) && \text{Definition of variance} \\
&\leq \sum_{i=1}^{n} \mathbb{E}[X_i^2] && \text{Ignore negative part} \\
&= \sum_{i=1}^{n} \mathbb{E}[X_i] && X_i^2 = X_i \text{ since } X_i\text{'s are indicator random variables} \\
&= \mathbb{E}[\sum_{i=1}^{n} X_i] && \text{Linearity of expectation} \\
&= \mathbb{E}[X] && \text{Definition of expectation}
\end{aligned}$$

□

Theorem 10.8. *There exists a constant $C > 0$ such that*

$$\Pr[\frac{D}{3} \leq 2^Z \cdot \sqrt{2} \leq 3D] > C$$

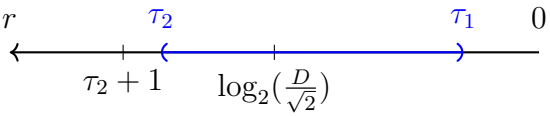
Proof. We will prove $\Pr[(\frac{D}{3} > 2^Z \cdot \sqrt{2}) \text{ or } (2^Z \cdot \sqrt{2} > 3D)] \leq 1 - C$ by separately analyzing $\Pr[\frac{D}{3} \geq 2^Z \cdot \sqrt{2}]$ and $\Pr[2^Z \cdot \sqrt{2} \geq 3D]$, then applying union bound. Define indicator variables

$$X_{i,r} = \begin{cases} 1 & \text{if ZEROS}(h(a_i)) \geq r \\ 0 & \text{otherwise} \end{cases}$$

and $X_r = \sum_{i=1}^{m} X_{i,r} = |\{a_i \in S : \text{ZEROS}(h(a_i)) \geq r\}|$. Notice that $X_n \leq X_{n-1} \leq \cdots \leq X_1$ since $\text{ZEROS}(h(a_i)) \geq r+1 \Rightarrow \text{ZEROS}(h(a_i)) \geq r$. Now,

$$\begin{aligned}
\mathbb{E}[X_r] &= \mathbb{E}[\sum_{i=1}^{m} X_{i,r}] && \text{Since } X_r = \sum_{i=1}^{m} X_{i,r} \\
&= \sum_{i=1}^{m} \mathbb{E}[X_{i,r}] && \text{By linearity of expectation} \\
&= \sum_{i=1}^{m} \Pr[X_{i,r} = 1] && \text{Since } X_{i,r} \text{ are indicator variables} \\
&= \sum_{i=1}^{m} \frac{1}{2^r} && h \text{ is a uniform hash} \\
&= \frac{D}{2^r} && \text{Since } h \text{ hashes same elements to the same value}
\end{aligned}$$

Denote τ_1 as the *smallest integer* such that $2^{\tau_1} \cdot \sqrt{2} > 3D$, and τ_2 as the *largest integer* such that $2^{\tau_2} \cdot \sqrt{2} < \frac{D}{3}$. We see that if $\tau_1 < Z < \tau_2$, then $2^Z \cdot \sqrt{2}$ is a 3-approximation of D.



- If $Z \geq \tau_1$, then $2^Z \cdot \sqrt{2} \geq 2^{\tau_1} \cdot \sqrt{2} > 3D$

- If $Z \leq \tau_2$, then $2^Z \cdot \sqrt{2} \leq 2^{\tau_2} \cdot \sqrt{2} < \frac{D}{3}$

$$\begin{aligned}
\Pr[Z \geq \tau_1] &\leq \Pr[X_{\tau_1} \geq 1] && \text{Since } Z \geq \tau_1 \Rightarrow X_{\tau_1} \geq 1 \\
&\leq \frac{\mathbb{E}[X_{\tau_1}]}{1} && \text{By Markov's inequality} \\
&= \frac{D}{2^{\tau_1}} && \text{Since } \mathbb{E}[X_r] = \frac{D}{2^r} \\
&\leq \frac{\sqrt{2}}{3} && \text{Since } 2^{\tau_1} \cdot \sqrt{2} > 3D
\end{aligned}$$

$$\begin{aligned}
\Pr[Z \leq \tau_2] &\leq \Pr[X_{\tau_2+1} = 0] && \text{Since } Z \leq \tau_2 \Rightarrow X_{\tau_2+1} = 0 \\
&\leq \Pr[\mathbb{E}[X_{\tau_2+1}] - X_{\tau_2+1} \geq \mathbb{E}[X_{\tau_2+1}]] && \text{Implied} \\
&\leq \Pr[|X_{\tau_2+1} - \mathbb{E}[X_{\tau_2+1}]| \geq \mathbb{E}[X_{\tau_2+1}]] && \text{Adding absolute sign} \\
&\leq \frac{\mathrm{Var}[X_{\tau_2+1}]}{(\mathbb{E}[X_{\tau_2+1}])^2} && \text{By Chebyshev's inequality} \\
&\leq \frac{\mathbb{E}[X_{\tau_2+1}]}{(\mathbb{E}[X_{\tau_2+1}])^2} && \text{By Lemma 10.7} \\
&\leq \frac{2^{\tau_2+1}}{D} && \text{Since } \mathbb{E}[X_r] = \frac{D}{2^r} \\
&\leq \frac{\sqrt{2}}{3} && \text{Since } 2^{\tau_2} \cdot \sqrt{2} < \frac{D}{3}
\end{aligned}$$

Putting together,

$$\begin{aligned}
&\Pr[(\frac{D}{3} > 2^Z \cdot \sqrt{2}) \text{ or } (2^Z \cdot \sqrt{2} > 3D)] \\
&\leq \Pr[\frac{D}{3} \geq 2^Z \cdot \sqrt{2}] + \Pr[2^Z \cdot \sqrt{2} \geq 3D] && \text{By union bound} \\
&\leq \frac{2\sqrt{2}}{3} && \text{From above} \\
&= 1 - C && \text{For } C = 1 - \frac{2\sqrt{2}}{3} > 0
\end{aligned}$$

□

Although the analysis tells us that there is a small success probability ($C = 1 - \frac{2\sqrt{2}}{3} \approx 0.0572$), one can use t independent hashes and output the mean $\frac{1}{k}\sum_{i=1}^{k}(2^{Z_i} \cdot \sqrt{2})$ (Recall Trick 1). With t hashes, the variance drops by a factor of $\frac{1}{t}$, improving the analysis for $\Pr[Z \leq \tau_2]$. When the success

probability $C > 0.5$ (for instance, after $t \geq 17$ repetitions), one can then call the routine k times independently and return the median (Recall Trick 2).

While Tricks 1 and 2 allows us to strength the success probability C, more work needs to be done to improve the approximation factor from 3 to $(1+\epsilon)$. To do this, we look at a slight modification of FM, due to [BYJK+02].

Algorithm 23 FM+$(S = \{a_1, \ldots, a_m\}, \epsilon)$

$N \leftarrow n^3$
$t \leftarrow \frac{c}{\epsilon^2} \in \mathcal{O}(\frac{1}{\epsilon^2})$ ▷ For some constant $c \geq 28$
$h \leftarrow$ Random hash from $\mathcal{H}_{n,N}$ ▷ Hash to a larger space
$T \leftarrow \emptyset$ ▷ Maintain t smallest $h(a_i)$'s
for $a_i \in S$ **do** ▷ Items arrive in streaming fashion
  $T \leftarrow t$ smallest values from $T \cup \{h(a_i)\}$
  (If $|T \cup \{h(a_i)\}| \leq t$, then $T = T \cup \{h(a_i)\}$)
end for
$Z = \max_{t \in T} T$
return $\frac{tN}{Z}$ ▷ Estimate of D

Remark For a cleaner analysis, we treat the *integer* interval $[N]$ as a *continuous* interval in Theorem 10.9. Note that there may be a rounding error of $\frac{1}{N}$ but this is relatively small and a suitable c can be chosen to make the analysis still work.

Theorem 10.9. *In* FM+*, for any given* $0 < \epsilon < \frac{1}{2}$, $\Pr[|\frac{tN}{Z} - D| \leq \epsilon D] > \frac{3}{4}$.

Proof. We first analyze $\Pr[\frac{tN}{Z} > (1+\epsilon)D]$ and $\Pr[\frac{tN}{Z} < (1-\epsilon)D]$ separately. Then, taking union bounds and negating yields the theorem's statement.

If $\frac{tN}{Z} > (1+\epsilon)D$, then $\frac{tN}{(1+\epsilon)D} > Z = t^{th}$ smallest hash value, implying that there are $\geq t$ hashes *smaller* than $\frac{tN}{(1+\epsilon)D}$. Since the hash uniformly distributes $[n]$ over $[N]$, for each element a_i,

$$\Pr[h(a_i) \leq \frac{tN}{(1+\epsilon)D}] = \frac{\frac{tN}{(1+\epsilon)D}}{N} = \frac{t}{(1+\epsilon)D}$$

Let $d_1, \ldots, d_D$ be the D distinct elements in the stream. Define indicator variables

$$X_i = \begin{cases} 1 & \text{if } h(d_i) \leq \frac{tN}{(1+\epsilon)D} \\ 0 & \text{otherwise} \end{cases}$$

and $X = \sum_{i=1}^{D} X_i$ is the number of hashes that are *smaller* than $\frac{tN}{(1+\epsilon)D}$. From above, $\Pr[X_i = 1] = \frac{t}{(1+\epsilon)D}$. By linearity of expectation, $\mathbb{E}[X] = \frac{t}{(1+\epsilon)}$. Then, by Lemma 10.7, $\text{Var}(X) \leq \mathbb{E}[X]$. Now,

$$
\begin{aligned}
\Pr[\frac{tN}{Z} > (1+\epsilon)D] &\leq \Pr[X \geq t] && \text{Since the former implies the latter} \\
&= \Pr[X - \mathbb{E}[X] \geq t - \mathbb{E}[X]] && \text{Subtracting } \mathbb{E}[X] \text{ from both sides} \\
&\leq \Pr[X - \mathbb{E}[X] \geq \frac{\epsilon}{2}t] && \text{Since } \mathbb{E}[X] = \frac{t}{(1+\epsilon)} \leq (1 - \frac{\epsilon}{2})t \\
&\leq \Pr[|X - \mathbb{E}[X]| \geq \frac{\epsilon}{2}t] && \text{Adding absolute sign} \\
&\leq \frac{\text{Var}(X)}{(\epsilon t/2)^2} && \text{By Chebyshev's inequality} \\
&\leq \frac{\mathbb{E}[X]}{(\epsilon t/2)^2} && \text{Since } \text{Var}(X) \leq \mathbb{E}[X] \\
&\leq \frac{4(1-\epsilon/2)t}{\epsilon^2 t^2} && \text{Since } \mathbb{E}[X] = \frac{t}{(1+\epsilon)} \leq (1 - \frac{\epsilon}{2})t \\
&\leq \frac{4}{c} && \text{Simplifying with } t = \frac{c}{\epsilon^2} \text{ and } (1 - \frac{\epsilon}{2}) < 1
\end{aligned}
$$

Similarly, if $\frac{tN}{Z} < (1-\epsilon)D$, then $\frac{tN}{(1-\epsilon)D} < Z = t^{th}$ smallest hash value, implying that there are $< t$ hashes *smaller* than $\frac{tN}{(1-\epsilon)D}$. Since the hash uniformly distributes $[n]$ over $[N]$, for each element a_i,

$$\Pr[h(a_i) \leq \frac{tN}{(1-\epsilon)D}] = \frac{\frac{tN}{(1-\epsilon)D}}{N} = \frac{t}{(1-\epsilon)D}$$

Let $d_1, \ldots, d_D$ be the D distinct elements in the stream. Define indicator variables

$$Y_i = \begin{cases} 1 & \text{if } h(d_i) \leq \frac{tN}{(1-\epsilon)D} \\ 0 & \text{otherwise} \end{cases}$$

and $Y = \sum_{i=1}^{D} Y_i$ is the number of hashes that are *smaller* than $\frac{tN}{(1-\epsilon)D}$. From above, $\Pr[Y_i = 1] = \frac{t}{(1-\epsilon)D}$. By linearity of expectation, $\mathbb{E}[Y] = \frac{t}{(1-\epsilon)}$. Then, by Lemma 10.7, $\text{Var}(Y) \leq \mathbb{E}[Y]$. Now,

$$
\begin{aligned}
&\Pr[\frac{tN}{Z} < (1-\epsilon)D] \\
&\leq \Pr[Y \leq t] && \text{Since the former implies the latter} \\
&= \Pr[Y - \mathbb{E}[Y] \leq t - \mathbb{E}[Y]] && \text{Subtracting } \mathbb{E}[Y] \text{ from both sides} \\
&\leq \Pr[Y - \mathbb{E}[Y] \leq -\epsilon t] && \text{Since } \mathbb{E}[Y] = \frac{t}{(1-\epsilon)} \geq (1+\epsilon)t \\
&\leq \Pr[-(Y - \mathbb{E}[Y]) \geq \epsilon t] && \text{Swap sides} \\
&\leq \Pr[|Y - \mathbb{E}[Y]| \geq \epsilon t] && \text{Adding absolute sign} \\
&\leq \frac{\text{Var}(Y)}{(\epsilon t)^2} && \text{By Chebyshev's inequality} \\
&\leq \frac{\mathbb{E}[Y]}{(\epsilon t)^2} && \text{Since } \text{Var}(Y) \leq \mathbb{E}[Y] \\
&\leq \frac{(1+2\epsilon)t}{\epsilon^2 t^2} && \text{Since } \mathbb{E}[Y] = \frac{t}{(1-\epsilon)} \leq (1+2\epsilon)t \\
&\leq \frac{3}{c} && \text{Simplifying with } t = \frac{c}{\epsilon^2} \text{ and } (1+2\epsilon) < 3
\end{aligned}
$$

Putting together,

$$
\begin{aligned}
\Pr[|\frac{tN}{Z} - D| > \epsilon D]] &\leq \Pr[\frac{tN}{Z} > (1+\epsilon)D]] + \Pr[\frac{tN}{Z} < (1-\epsilon)D]] && \text{By union bound} \\
&\leq 4/c + 3/c && \text{From above} \\
&\leq 7/c && \text{Simplifying} \\
&\leq 1/4 && \text{For } c \geq 28
\end{aligned}
$$

□

10.3 Estimating the k^{th} moment of a stream

In this section, we describe algorithms from [AMS96] that estimates the k^{th} moment of a stream, first for $k = 2$, then for general k. Recall that the k^{th} moment of a stream S is defined as $F_k = \sum_{i=1}^{n} (f_i)^k$, where for each element $i \in [n]$, f_i denotes the number of times value i appears in the stream.

10.3.1 $k = 2$

For each element $i \in [n]$, we associate a random variable $r_i \in_{u.a.r.} \{-1, +1\}$.

Algorithm 24 AMS-2($S = \{a_1, \ldots, a_m\}$)

```
Assign r_i ∈_{u.a.r.} {−1, +1}, ∀i ∈ [n]      ▷ For now, this takes O(n) space
Z ← 0
for a_i ∈ S do                                 ▷ Items arrive in streaming fashion
    Z ← Z + r_i                                ▷ At the end, Z = Σ_{i=1}^n r_i f_i
end for
return Z^2                                     ▷ Estimate of F_2 = Σ_{i=1}^n f_i^2
```

Lemma 10.10. *In* AMS-2*, if random variables* $\{r_i\}_{i\in[n]}$ *are pairwise independent, then* $\mathbb{E}[Z^2] = \sum_{i=1}^n f_i^2 = F_2$*. That is,* AMS-2 *is an unbiased estimator for the second moment.*

Proof.

$$\begin{aligned}
\mathbb{E}[Z^2] &= \mathbb{E}[(\sum_{i=1}^n r_i f_i)^2] && \text{Since } Z = \sum_{i=1}^n r_i f_i \text{ at the end}\\
&= \mathbb{E}[\sum_{i=1}^n r_i^2 f_i^2 + 2\sum_{1\le i<j\le n} r_i r_j f_i f_j] && \text{Expanding } (\sum_{i=1}^n r_i f_i)^2\\
&= \sum_{i=1}^n \mathbb{E}[r_i^2 f_i^2] + 2\sum_{1\le i<j\le n} \mathbb{E}[r_i r_j f_i f_j] && \text{Linearity of expectation}\\
&= \sum_{i=1}^n \mathbb{E}[r_i^2] f_i^2 + 2\sum_{1\le i<j\le n} \mathbb{E}[r_i r_j] f_i f_j && f_i\text{'s are (unknown) constants}\\
&= \sum_{i=1}^n f_i^2 + 2\sum_{1\le i<j\le n} \mathbb{E}[r_i r_j] f_i f_j && \text{Since } r_i^2 = 1, \forall i \in [n]\\
&= \sum_{i=1}^n f_i^2 + 2\sum_{1\le i<j\le n} \mathbb{E}[r_i]\mathbb{E}[r_j] f_i f_j && \text{Since } \{r_i\}_{i\in[n]} \text{ are pairwise independent}\\
&= \sum_{i=1}^n f_i^2 && \text{Since } \mathbb{E}[r_i] = 0, \forall i \in [n]\\
&= F_2 && \text{Since } F_2 = \sum_{i=1}^n f_i^2
\end{aligned}$$

□

So we have an unbiased estimator for the second moment but we are also interested in the probability of error. We want a small probability for the output Z^2 to deviate more than $(1+\epsilon)$ from the true value, i.e., $\Pr[|Z^2 - F_2| > \epsilon F_2]$ should be small.

Lemma 10.11. *In* AMS-2*, if random variables* $\{r_i\}_{i\in[n]}$ *are* 4*-wise independent*[2]*, then* $\mathrm{Var}[Z^2] \leq 2(\mathbb{E}[Z^2])^2$.

Proof. As before, $\mathbb{E}[r_i] = 0$ and $r_i^2 = 1$ for all $i \in [n]$. By 4-wise independence, the expectation of any product of at most 4 different r_i's is the product of their expectations. Thus we get $\mathbb{E}[r_i r_j r_k r_l] = \mathbb{E}[r_i]\mathbb{E}[r_j]\mathbb{E}[r_k]\mathbb{E}[r_l] = 0$, as well as $\mathbb{E}(r_i^3 r_j) = \mathbb{E}(r_i r_j) = 0$ and $\mathbb{E}(r_i^2 r_j r_k) = \mathbb{E}(r_j r_k) = 0$, where the indices i, j, k, l are pairwise different. This allows us to compute $\mathbb{E}[Z^4]$:

$$\begin{aligned}
\mathbb{E}[Z^4] &= \mathbb{E}[(\sum_{i=1}^{n} r_i f_i)^4] && \text{Since } Z = \sum_{i=1}^{n} r_i f_i \text{ at the end} \\
&= \sum_{i=1}^{n} \mathbb{E}[r_i^4] f_i^4 + 6 \sum_{1\leq i<j\leq n} \mathbb{E}[r_i^2 r_j^2] f_i^2 f_j^2 && \text{L.o.E. and 4-wise independence} \\
&= \sum_{i=1}^{n} f_i^4 + 6 \sum_{1\leq i<j\leq n} f_i^2 f_j^2 && \text{Since } r_i^4 = r_i^2 = 1, \forall i \in [n] \ .
\end{aligned}$$

Note that the coefficient of $\sum_{1\leq i<j\leq n} \mathbb{E}[r_i^2 r_j^2] f_i^2 f_j^2$ is $\binom{4}{2} = 6$ and that all other terms vanish by the computation above.

$$\begin{aligned}
\mathrm{Var}[Z^2] &= \mathbb{E}[(Z^2)^2] - (\mathbb{E}[Z^2])^2 && \text{Definition of variance} \\
&= \sum_{i=1}^{n} f_i^4 + 6 \sum_{1\leq i<j\leq n} f_i^2 f_j^2 - (\mathbb{E}[Z^2])^2 && \text{From above} \\
&= \sum_{i=1}^{n} f_i^4 + 6 \sum_{1\leq i<j\leq n} f_i^2 f_j^2 - (\sum_{i=1}^{n} f_i^2)^2 && \text{By Lemma 10.10} \\
&= 4 \sum_{1\leq i<j\leq n} f_i^2 f_j^2 && \text{Expand and simplify} \\
&\leq 2(\sum_{i=1}^{n} f_i^2)^2 && \text{Introducing } f_i^4 \text{ terms} \\
&= 2(\mathbb{E}[Z^2])^2 && \text{By Lemma 10.10}
\end{aligned}$$

[2]The random variables $\{r_i\}_{i\in[n]}$ are said to be 4-wise independent if $\Pr\left((r_{i_1}, r_{i_2}, r_{i_3}, r_{i_4}) = (\epsilon_{i_1}, \epsilon_{i_2}, \epsilon_{i_3}, \epsilon_{i_4})\right) = \prod_{j=1}^{4} \Pr(r_{i_j} = \epsilon_{i_j})$ for all $\epsilon_{i_1}, \epsilon_{i_2}, \epsilon_{i_3}, \epsilon_{i_4}$. Note that 4-wise independence implies pairwise independence.

□

Theorem 10.12. *In* AMS-2, *if* $\{r_i\}_{i\in[n]}$ *are* 4*-wise independent, then we have* $\Pr[|Z^2 - F_2| > \epsilon F_2] \leq \frac{2}{\epsilon^2}$ *for any* $\epsilon > 0$.

Proof.

$$\begin{aligned}
\Pr[|Z^2 - F_2| > \epsilon F_2] &= \Pr[|Z^2 - \mathbb{E}[Z^2]| > \epsilon \mathbb{E}[Z^2]] && \text{By Lemma 10.10} \\
&\leq \frac{\mathrm{Var}(Z^2)}{(\epsilon \mathbb{E}[Z^2])^2} && \text{By Chebyshev's inequality} \\
&\leq \frac{2(\mathbb{E}[Z^2])^2}{(\epsilon \mathbb{E}[Z^2])^2} && \text{By Lemma 10.11} \\
&= \frac{2}{\epsilon^2}
\end{aligned}$$

□

We can again apply the mean trick to decrease the variance by a factor of k and have a smaller upper bound on the probability of error.
In particular, if we pick $k = \frac{10}{\epsilon^2}$ repetitions of ASM-2 and output the mean value of the output Z^2 we have :

$$\Pr[\text{error}] \leq \frac{\mathrm{Var}[Z^2]\frac{1}{k}}{(\epsilon E[Z^2])^2} \leq \frac{1}{k} \cdot \frac{2}{\epsilon^2} = \frac{1}{5}$$

Claim 10.13. $\mathcal{O}(k \log n)$ *bits of randomness suffices to obtain a set of* k*-wise independent random variables.*

Proof. Recall the definition of hash family $\mathcal{H}_{n,m}$. In a similar fashion[3], we consider hashes from the family (for prime p):

$$\begin{aligned}
\{h_{a_{k-1},a_{k-2},\ldots,a_1,a_0} : h(x) &= \sum_{i=1}^{k-1} a_i x^i \mod p \\
&= a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \cdots + a_1 x + a_0 \mod p, \\
&\quad \forall a_{k-1}, a_{k-2}, \ldots, a_1, a_0 \in \mathbb{Z}_p\}
\end{aligned}$$

This requires k random coefficients, which can be stored with $\mathcal{O}(k \log n)$ bits. □

[3] See https://en.wikipedia.org/wiki/K-independent_hashing

Observe that the above analysis only require $\{r_i\}_{i\in[n]}$ to be 4-wise independent. Claim 10.13 implies that AMS-2 only needs $\mathcal{O}(4\log n)$ bits to represent $\{r_i\}_{i\in[n]}$.

Although the failure probability $\frac{2}{\epsilon^2}$ is large for small ϵ, one can repeat t times and output the mean (Recall Trick 1). With $t \in \mathcal{O}(\frac{1}{\epsilon^2})$ samples, the failure probability drops to $\frac{2}{t\epsilon^2} \in \mathcal{O}(1)$. When the failure probability is less than $\frac{1}{2}$, one can then call the routine k times independently, and return the median (Recall Trick 2). On the whole, for any given $\epsilon > 0$ and $\delta > 0$, $\mathcal{O}(\frac{\log(n)\log(1/\delta)}{\epsilon^2})$ space suffices to yield a $(1\pm\epsilon)$-approximation algorithm that succeeds with probability $> 1-\delta$.

10.3.2 General k

Algorithm 25 AMS-K($S = \{a_1, \ldots, a_m\}$)

$m \leftarrow |S|$ ▷ For now, assume we know $m = |S|$
$J \in_{u.a.r.} [m]$ ▷ Pick a random index
$r \leftarrow 0$
for $a_i \in S$ **do** ▷ Items arrive in streaming fashion
  if $i \geq J$ and $a_i = a_J$ **then**
    $r \leftarrow r + 1$
  end if
end for
$Z \leftarrow m(r^k - (r-1)^k)$
return Z ▷ Estimate of $F_k = \sum_{i=1}^{n}(f_i)^k$

Remark At the end of AMS-K, $r = |\{i \in [m] : i \geq J \text{ and } a_i = a_J\}|$ will be the number of occurrences of a_J in a suffix of the stream.

The assumption of known m in AMS-K can be removed via reservoir sampling[4]. The idea is as follows: Initially, initialize stream length and J as both 0. When a_i arrives, choose to replace J with i with probability $\frac{1}{i}$. If J is replaced, reset r to 0 and start counting from this stream suffix onwards. It can be shown that the choice of J is uniform over current stream length.

Lemma 10.14. *In* AMS-K, $\mathbb{E}[Z] = \sum_{i=1}^{n} f_i^k = F_k$. *That is,* AMS-K *is an unbiased estimator for the* k^{th} *moment.*

[4]See https://en.wikipedia.org/wiki/Reservoir_sampling

Proof. When $a_J = i$, there are f_i choices for J. By telescoping sums, we have

$$\begin{aligned}
&\mathbb{E}[Z \mid a_J = i] \\
&= \frac{1}{f_i}[m(f_i^k - (f_i-1)^k)] + \frac{1}{f_i}[m((f_i-1)^k - (f_i-2)^k)] + \cdots + \frac{1}{f_i}[m(1^k - 0^k)] \\
&= \frac{m}{f_i}[(f_i^k - (f_i-1)^k) + ((f_i-1)^k - (f_i-2)^k) + \cdots + (1^k - 0^k)] \\
&= \frac{m}{f_i} f_i^k \ .
\end{aligned}$$

Thus,

$$\begin{aligned}
\mathbb{E}[Z] &= \sum_{i=1}^{n} \mathbb{E}[Z \mid a_J = i] \cdot \Pr[a_J = i] && \text{Condition on the choice of } J \\
&= \sum_{i=1}^{n} \mathbb{E}[Z \mid a_J = i] \cdot \frac{f_i}{m} && \text{Since choice of } J \text{ is uniform at random} \\
&= \sum_{i=1}^{n} \frac{m}{f_i} f_i^k \cdot \frac{f_i}{m} && \text{From above} \\
&= \sum_{i=1}^{n} f_i^k && \text{Simplifying} \\
&= F_k && \text{Since } F_k = \sum_{i=1}^{n} f_i^k \ .
\end{aligned}$$

□

Lemma 10.15. *For positive reals $f_1, f_2, \ldots, f_n$ and a positive integer k, we have*

$$(\sum_{i=1}^{n} f_i)(\sum_{i=1}^{n} f_i^{2k-1}) \leq n^{1-1/k}(\sum_{i=1}^{n} f_i^k)^2 \ .$$

Proof. Let $M = \max_{i \in [n]} f_i$, then $f_i \leq M$ for any $i \in [n]$ and $M^k \leq \sum_{i=1}^{n} f_i^k$. Hence,

$$
\begin{aligned}
(\sum_{i=1}^{n} f_i)(\sum_{i=1}^{n} f_i^{2k-1}) &\le (\sum_{i=1}^{n} f_i)(M^{k-1}\sum_{i=1}^{n} f_i^k) && \text{Since } f_i^{2k-1} \le M^{k-1} f_i^k \\
&\le (\sum_{i=1}^{n} f_i)(\sum_{i=1}^{n} f_i^k)^{(k-1)/k}(\sum_{i=1}^{n} f_i^k) && \text{Since } M^k \le \sum_{i=1}^{n} f_i^k \\
&= (\sum_{i=1}^{n} f_i)(\sum_{i=1}^{n} f_i^k)^{(2k-1)/k} && \text{Merging the last two terms} \\
&\le n^{1-1/k}(\sum_{i=1}^{n} f_i^k)^{1/k}(\sum_{i=1}^{n} f_i^k)^{(2k-1)/k} && \text{Fact: } (\sum_{i=1}^{n} f_i)/n \le (\sum_{i=1}^{n} f_i^k/n)^{1/k} \\
&= n^{1-1/k}(\sum_{i=1}^{n} f_i^k)^2 && \text{Merging the last two terms .}
\end{aligned}
$$

□

Remark $f_1 = n^{\frac{1}{k}}, f_2 = \cdots = f_n = 1$ is a tight example for Lemma 10.15, up to a constant factor.

Theorem 10.16. *In* AMS-K, $\mathrm{Var}(Z) \le kn^{1-\frac{1}{k}}(\mathbb{E}[Z])^2$.

Proof. Let us first analyze $\mathbb{E}[Z^2]$.

$$
\begin{aligned}
\mathbb{E}[Z^2] = \frac{m}{m}[\ &(1^k - 0^k)^2 + (2^k - 1^k)^2 + \cdots + (f_1^k - (f_1 - 1)^k)^2 && (1)\\
+\ &(1^k - 0^k)^2 + (2^k - 1^k)^2 + \cdots + (f_2^k - (f_2 - 1)^k)^2 \\
+\ &\cdots \\
+\ &(1^k - 0^k)^2 + (2^k - 1^k)^2 + \cdots + (f_n^k - (f_n - 1)^k)^2] \\
\le m[\ &k1^{k-1}(1^k - 0^k) + k2^{k-1}(2^k - 1^k) + \cdots + kf_1^{k-1}(f_1^k - (f_1 - 1)^k) && (2)\\
+\ &k1^{k-1}(1^k - 0^k) + k2^{k-1}(2^k - 1^k) + \cdots + kf_2^{k-1}(f_2^k - (f_2 - 1)^k) \\
+\ &\cdots \\
+\ &k1^{k-1}(1^k - 0^k) + k2^{k-1}(2^k - 1^k) + \cdots + kf_n^{k-1}(f_n^k - (f_n - 1)^k)] \\
\le m[&kf_1^{2k-1} + kf_2^{2k-1} + \cdots + kf_n^{2k-1}] && (3)\\
= &kmF_{2k-1} && (4)\\
= &kF_1F_{2k-1} && (5)
\end{aligned}
$$

(1) Condition on J and expand as in the proof of Theorem 10.14

(2) For all $0 < b < a$,

$$a^k - b^k = (a-b)(a^{k-1} + a^{k-2}b + \cdots + ab^{k-2} + b^{k-1}) \leq (a-b)ka^{k-1},$$

in particular, $((a^k - (a-1)^k)^2 \leq ka^{k-1}(a^k - (a-1)^k)$.

(3) Telescope each row, then ignore remaining negative terms

(4) $F_{2k-1} = \sum_{i=1}^{n} f_i^{2k-1}$

(5) $F_1 = \sum_{i=1}^{n} f_i = m$

Then,

$$\begin{aligned}
\mathrm{Var}(Z) &= \mathbb{E}[Z^2] - (\mathbb{E}[Z])^2 && \text{Definition of variance} \\
&\leq \mathbb{E}[Z^2] && \text{Ignore negative part} \\
&\leq kF_1F_{2k-1} && \text{From above} \\
&\leq kn^{1-\frac{1}{k}}F_k^2 && \text{By Lemma 10.15} \\
&= kn^{1-\frac{1}{k}}(\mathbb{E}[Z])^2 && \text{By Theorem 10.14}
\end{aligned}$$

□

Remark Proofs for Lemma 10.15 and Theorem 10.16 were omitted in class. The above proofs are presented in a style consistent with the rest of the scribe notes. Interested readers can refer to [AMS96] for details.

Remark One can apply an analysis similar to the case when $k = 2$, then use Tricks 1 and 2.

Claim 10.17. *For $k > 2$, a lower bound of $\widetilde{\Theta}(n^{1-\frac{2}{k}})$ is known.*

Proof. Theorem 3.1 in [BYJKS04] gives the lower bound. See [IW05] for algorithm that achieves it. □

Chapter 11

Graph sketching

Definition 11.1 (Streaming connected components problem)**.** *Consider a graph of n vertices and a stream S of edge updates $\{\langle e_t, \pm\rangle\}_{t\in\mathbb{N}^+}$, where edge e_t is either added (+) or removed (-). Assume that S is "well-behaved", that is existing edges are not added and an edge is deleted only if it's already present in the graph.*

At time t, the edge set E_t of the graph $G_t = (V, E_t)$ is the set of edges present after accounting for all stream updates up to time t. How much memory do we need if we want to be able to query the connected components for G_t for any $t \in \mathbb{N}^+$?

Remark In this chapter, we mainly focus on the amount of memory needed without worrying about processing time. However, the solution that we will see can be computed in polynomial time.

Let m be the total number of distinct edges in the stream. There are two ways to represent connected components on a graph:

1. Every vertex stores a label such that vertices in the same connected component have the same label
2. Explicitly build a tree for each connected component — This yields a *maximal forest*

For now, we are interested in building a maximal forest for G_t. This can be done with memory size of $\mathcal{O}(m)$ words[1], or — in the special case of *only edge additions* — $\mathcal{O}(n)$ words[2]. However, these are unsatisfactory

[1] Toggle edge additions/deletion per update. Compute connected components on demand.

[2] We just need to store a label for each node of the graph so that nodes of a same connected component have the same label and then use the Union-Find data structure as new edges arrive. See https://en.wikipedia.org/wiki/Disjoint-set_data_structure

as $m \in \mathcal{O}(n^2)$ on a complete graph, and we may have edge deletions. We show how one can maintain a data structure with $\mathcal{O}(n \log^4 n)$ memory, with a randomized algorithm that succeeds in building the maximal forest with success probability $\geq 1 - \frac{1}{n^{10}}$.

Coordinator model For a change in perspective[3], consider the following computation model where each vertex acts independently from each other. Then, upon request of connected components, each vertex sends some information to a centralized coordinator to perform computation and outputs the maximal forest.

The coordinator model will be helpful in our analysis of the algorithm later as each vertex will send $\mathcal{O}(\log^4 n)$ amount of data (a local sketch of the graph) to the coordinator, totalling $\mathcal{O}(n \log^4 n)$ memory as required.

11.1 Finding the single cut edge

Definition 11.2 (The single cut problem)**.** *Fix an arbitrary subset $A \subseteq V$. Suppose there is exactly 1 cut edge $\{u, v\}$ between A and $V \setminus A$. How do we output the cut edge $\{u, v\}$ using $\mathcal{O}(\log n)$ bits of memory?*

Without loss of generality, assume $u \in A$ and $v \in V \setminus A$. Note that this is not a trivial problem at first glance since it already takes $\mathcal{O}(n)$ bits for any vertex to enumerate all its adjacent edges. To solve the problem, we use a *bit trick* which exploits the fact that any edge $\{a, b\} \in A$ will be considered twice by vertices in A. Since one can uniquely identify each vertex with $\mathcal{O}(\log n)$ bits, consider the following:

- Identify an edge $e = \{u, v\}$ by the concatenation of the identifiers of its endpoints: $id(e) = id(u) \circ id(v)$ if $id(u) < id(v)$
- Locally, every vertex u maintains

  $$XOR_u = \oplus\{id(e) : e \in S \wedge u \text{ is an endpoint of } e\}$$

  Thus XOR_u represents the bit-wise XOR of the identifiers of all edges that are adjacent to u.
- All vertices send the coordinator their value XOR_u and the coordinator computes

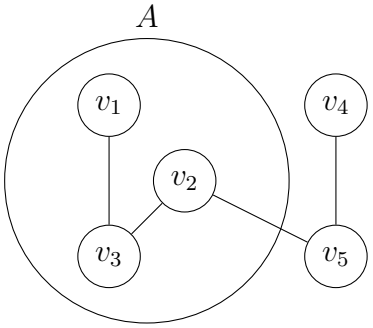
  $$XOR_A = \oplus\{XOR_u : u \in A\}$$

[3] In reality, the algorithm simulates all the vertices' actions so it is not a real multi-party computation setup.

Example Suppose $V = \{v_1, v_2, v_3, v_4, v_5\}$ where $id(v_1) = 000$, $id(v_2) = 001$, $id(v_3) = 010$, $id(v_4) = 011$, and $id(v_5) = 100$. Then, $id(\{v_1, v_3\}) = id(v_1) \circ id(v_3) = 000010$, and so on. Suppose

$$S = \{\langle\{v_1, v_2\}, +\rangle, \langle\{v_2, v_3\}, +\rangle, \langle\{v_1, v_3\}, +\rangle, \langle\{v_4, v_5\}, +\rangle, \langle\{v_2, v_5\}, +\rangle, \langle\{v_1, v_2\}, -\rangle\}$$

and we query for the cut edge $\{v_2, v_5\}$ with $A = \{v_1, v_2, v_3\}$ at $t = |S|$. The figure below shows the graph G_6 when $t = 6$:



Vertex v_1 sees $\{\langle\{v_1, v_2\}, +\rangle, \langle\{v_1, v_3\}, +\rangle$, and $\langle\{v_1, v_2\}, -\rangle\}$. So,

$$\begin{aligned}
XOR_1 &\Rightarrow 000000 && \text{Initialize}\\
&\Rightarrow 000000 \oplus id((v_1, v_2)) = 000000 \oplus 000001 = 000001 && \text{Due to } \langle\{v_1, v_2\}, +\rangle\\
&\Rightarrow 000001 \oplus id((v_1, v_3)) = 000001 \oplus 000010 = 000011 && \text{Due to } \langle\{v_1, v_3\}, +\rangle\\
&\Rightarrow 000011 \oplus id((v_1, v_2)) = 000011 \oplus 000001 = 000010 && \text{Due to } \langle\{v_1, v_2\}, -\rangle
\end{aligned}$$

Repeating the simulation for all vertices,

$$\begin{aligned}
XOR_1 = 000010 &= id(\{v_1, v_2\}) \oplus id(\{v_1, v_3\}) \oplus id(\{v_1, v_2\})\\
&= 000001 \oplus 000010 \oplus 000001\\
XOR_2 = 000110 &= id(\{v_1, v_2\}) \oplus id(\{v_2, v_3\}) \oplus id(\{v_2, v_5\}) \oplus id(\{v_1, v_2\})\\
&= 000001 \oplus 001010 \oplus 001100 \oplus 000001\\
XOR_3 = 001000 &= id(\{v_2, v_3\}) \oplus id(\{v_1, v_3\})\\
&= 001010 \oplus 000010\\
XOR_4 = 011100 &= id(\{v_4, v_5\})\\
&= 011100\\
XOR_5 = 010000 &= id(\{v_4, v_5\}) \oplus id(\{v_2, v_5\})\\
&= 011100 \oplus 001100
\end{aligned}$$

Thus, $XOR_A = XOR_1 \oplus XOR_2 \oplus XOR_3 = 000010 \oplus 000110 \oplus 001000 = 001100 = id(\{v_2, v_5\})$ as expected. Notice that after adding or deleting an edge $e = (u, v)$, updating XOR_u and XOR_v can be done by doing a bit-wise XOR of each of these values together with $id(e)$. Also, the identifier of every edge with both endpoints in A contributes two times to XOR_A.

Claim 11.3. *$XOR_A = \oplus\{XOR_u : u \in A\}$ is the identifier of the cut edge.*

Proof. For any edge $e = (a, b)$ such that $a, b \in A$, $id(e)$ contributes to both XOR_a and XOR_b. So, $XOR_a \oplus XOR_b$ will cancel out the contribution of $id(e)$ because $id(e) \oplus id(e) = 0$. Hence, the only remaining value in $XOR_A = \oplus\{XOR_u : u \in A\}$ will be the identifier of the cut edge since only one of its endpoints lies in A. □

Remark Bit tricks are often used in the random linear network coding literature (e.g. [HMK+06]).

11.2 Finding one out of $k > 1$ cut edges

Definition 11.4 (The k cut problem)**.** *Fix an arbitrary subset $A \subseteq V$. Suppose there are exactly k cut edges (u, v) between A and $V \setminus A$, and we are given an estimate $\widehat{k}$ such that $\frac{\widehat{k}}{2} \leq k \leq \widehat{k}$. How do we output a cut edge (u, v) using $\mathcal{O}(\log^2 n)$ bits of memory, with high probability?*

A straight-forward idea is to independently mark each edge, each with probability $1/\widehat{k}$. In expectation, we expect one edge to be marked. Denote the set of marked cut edges by E'.

$$
\begin{aligned}
&\Pr[|E'| = 1] \\
&= k \cdot \Pr[\text{Cut edge } \{u, v\} \text{ is marked; others are not}] \\
&= k \cdot (1/\widehat{k})(1 - (1/\widehat{k}))^{k-1} && \text{Edges marked ind. w.p. } 1/\widehat{k} \\
&\geq (\widehat{k}/2)(1/\widehat{k})(1 - (1/\widehat{k}))^{\widehat{k}} && \text{Since } \frac{\widehat{k}}{2} \leq k \leq \widehat{k} \\
&\geq \frac{1}{2} \cdot 4^{-1} && \text{Since } 1 - x \geq 4^{-x} \text{ for } x \leq \frac{1}{2} \\
&\geq \frac{1}{10}
\end{aligned}
$$

Remark The above analysis assumes that vertices can locally mark the edges in a consistent manner (i.e. both endpoints of any edge make the same decision whether to mark the edge or not). This can be achieved with a sufficiently large string of *randomness*, shared across all the nodes.

From above, we know that $\Pr[|E'| = 1] \geq 1/10$. If $|E'| = 1$, we can re-use the idea from Section 11.1. However, if $|E'| \neq 1$, then XOR_A may correspond erroneously to another edge in the graph. In the above example, $id(\{v_1, v_2\}) \oplus id(\{v_2, v_4\}) = 000001 \oplus 001011 = 001010 = id(\{v_2, v_3\})$.

To fix this, we use random bits as edge IDs instead of simply concatenating vertex IDs: randomly assign (in a consistent manner) to each edge a random ID of $k = 20 \log n$ bits. Since the XOR of random bits is random, for any edge e, $\Pr[XOR_A = id(e) \mid |E'| \neq 1] = (\frac{1}{2})^k = (\frac{1}{2})^{20 \log n}$. Hence,

$$
\begin{aligned}
&\Pr[XOR_A = id(e) \text{ for some edge } e \mid |E'| \neq 1] \\
&\leq \sum_{e \in \binom{V}{2}} \Pr[XOR_A = id(e) \mid |E'| \neq 1] && \text{Union bound over all possible edges} \\
&= \binom{n}{2} (\frac{1}{2})^{20 \log n} && \text{There are } \binom{n}{2} \text{ possible edges} \\
&= 2^{-18 \log n} && \text{Since } \binom{n}{2} \leq n^2 = 2^{2 \log n} \\
&= \frac{1}{n^{18}} && \text{Rewriting}
\end{aligned}
$$

Therefore, if we sample two or more edges from the cut, with high probability the XOR of their identifiers will be distinguishable from the identifier of another edge, allowing us to determine whether $|E'| = 1$.

The probability of sampling a single edge across the cut is $Pr[|E'| = 1] \geq \frac{1}{10}$. To amplify it we can perform $t = C \log(n) = O(\log n)$ parallel repetitions, each time sampling the edges independently and computing for every node the XOR of the sampled edges. We succeed to find an edge across the cut if at least one repetition succeeds, which happens with probability at least $1 - \left(\frac{9}{10}\right)^{C \log n} \geq 1 - \frac{1}{n^{10}}$, by setting the constant term C appropriately.

Overall, the size of the message that each node sends to the coordinator is $O(\log^2 n)$ bits.

11.3 Finding one out of arbitrarily many cut edges

For an arbitrary cut, the number of edges across the cut is $k \in [0, n^2]$. To find a single edge across the cut, we can apply the procedure described in Section 11.2 using all the powers of 2, $\widehat{k} \in \{2^0, 2^1, \ldots, 2^{\lceil \log n^2 \rceil}\}$, as an estimate of k.

Section 11.2 proves that if the estimate is within a 2-factor of the number of edges k, i.e. $\frac{\widehat{k}}{2} \leq k \leq \widehat{k}$, then we find a single edge across the cut with probability at least $1 - \frac{1}{n^{10}}$.

For the values of $\widehat{k}$ that are much bigger than k, the sampling probability is such that in expectation no edges across the cut are sampled. Conversely, if $\widehat{k}$ is much smaller than k, more than one edge across the cut is expected to be sampled, but with high probability the XOR of their identifiers will not be a valid edge ID.

In total, there are $\lceil \log n^2 \rceil + 1 = O(\log n)$ powers of 2 which are used as an estimate for k, so overall each node sends an $O(\log^3 n)$ bits message to the coordinator: for each estimate $\widehat{k}$, we perform $O(\log n)$ independent samplings of edges (to amplify success probability) and every time each node computes the XOR of the identifiers of the incident sampled edges, that has size $O(\log n)$.

11.4 Maximal forest with $\mathcal{O}(n \log^4 n)$ memory

The procedure described in section 11.3 finds an edge across a single cut with high probability. This can be used to find a maximal forest of the graph, but we are only allowed to call it for $poly(n)$ cuts: each time the procedure is called, it has a failure probability of at most $\frac{1}{n^{10}}$, and it blows up if we consider all the possible cuts, whose number is exponential in n.

We recall Borůvka's algorithm[4] for building a minimum spanning tree:

- Start with each node being one connected component
- Find the cheapest edge leaving each connected component and add it into the MST (ignoring cycles)
- Repeat until there is only one connected component

The number of connected components decreases by at least half per iteration, so it converges in $\mathcal{O}(\log n)$ iterations.

[4]For a detailed explanation, see https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Since here we do not care about edge weights, the step of finding the cheapest edge leaving each component amounts to finding one out of many cut edges, which we solved in section 11.3.

Therefore, each node sends to the coordinator a message of size $O(\log^4 n)$, obtained by repeating $\log n$ times independently the message construction outlined in section 11.3. The entire procedure is presented in ComputeSketch.

Algorithm 26 ComputeSketch($v \in V$)

```
for h = 1 to log n do                          ▷ Iterations of Borůvka
    for i ∈ {0, 1, ..., ⌈log n²⌉} do           ▷ ⌈log n²⌉ + 1 guesses of k
        for t = C log n times do               ▷ Amplify success probability
            Sample each edge w.p p = 1/2^i
            Send XOR of sampled edges incident to v
        end for
    end for
end for
```

When the coordinator receives all the messages from the nodes, it is able to compute a maximal forest by simulating the steps of Borůvka's algorithm:

- initially, every node is a single component of the graph;
- for each step $h \in \{1, \ldots, \log n\}$:
  - use part h of the messages (of size $O(\log^3 n)$) to find, for every component, a single edge connecting it to another component (if there is one) with high probability;
  - merge the newly formed components and exclude any edges forming a cycle;

This scheme can also be implemented in the streaming setting, by maintaining a message of size $O(\log^4 n)$ for every node as described above, and updating them after every edge insertion or deletion.ComputeSketches and StreamingMaximalForest outline this procedure.

More precisely, every vertex in ComputeSketches maintains $O(\log^3 n)$ copies of edge XORs using random edge IDs and marking probabilities. In order to have consistent edge IDs and marking probabilities among vertices, we use a source of shared randomness $\mathcal{R}$. Then, StreamingMaximalForest simulates Borůvka using the output of ComputeSketches. In total, this requires $O(n \log^4 n)$ memory to compute a maximal forest of the graph.

Algorithm 27 ComputeSketches($S = \{\langle e, \pm\rangle, \dots\}, \epsilon, \mathcal{R}$)

for $i = 1, \dots, n$ **do**
  $XOR_i \leftarrow 0^{(20 \log n) * \log^3 n}$ ▷ Initialize $\log^3 n$ copies
end for
for Edge update $\{\langle e = (u, v), \pm\rangle\} \in S$ **do** ▷ Streaming edge updates
  for $h = \log n$ times **do** ▷ For Borůvka simulation later
    for $i \in \{0, 1, \dots, \lceil \log n^2 \rceil\}$ **do** ▷ $\lceil \log n^2 \rceil + 1$ guesses of k
      for $t = C \log n$ times **do** ▷ Amplify success probability
        $R_{h,i,t} \leftarrow$ Randomness for this specific instance based on $\mathcal{R}$
        if Edge e is marked w.p. $1/\widehat{k} = 2^{-i}$, according to $R_{h,i,t}$ **then**
          Compute $id(e)$ using R
          $XOR_u[h, i, t] \leftarrow XOR_u[h, i, t] \oplus id(e)$
          $XOR_v[h, i, t] \leftarrow XOR_v[h, i, t] \oplus id(e)$
        end if
      end for
    end for
  end for
end for
return $XOR_1, \dots, XOR_n$

Algorithm 28 StreamingMaximalForest($S = \{\langle e, \pm \rangle, \dots\}, \epsilon$)

```
R ← Generate O(log² n) bits of shared randomness
XOR_1, ..., XOR_n ← ComputeSketches(S, ε, R)
F ← (V_F = V, E_F = ∅)                                  ▷ Initialize empty forest
for h = log n times do                                   ▷ Simulate Borůvka
    C ← ∅                                                ▷ Initialize candidate edges
    for Every connected component A in F do
        for i ∈ {1, 2, ..., ⌈log n²⌉} do                 ▷ Guess A has [2^{i-1}, 2^i] cut edges
            for t = C log n times do                     ▷ Amplify success probability
                R_{h,i,t} ← Randomness for this specific instance
                XOR_A ← ⊕{XOR_u[h, i, t] : u ∈ A}
                if XOR_A = id(e) for some edge e = (u, v) then
                    C ← C ∪ {(u, v)}                     ▷ Add cut edge (u, v) to candidates
                    Go to next connected component in F
                end if
            end for
        end for
    end for
    E_F ← E_F ∪ C, removing cycles in O(1) if necessary  ▷ Add candidates
end for
return F
```

At each step, we fail to find one cut edge leaving a connected component with probability $\leq (1 - \frac{1}{10})^t$, which can be made to be in $\mathcal{O}(\frac{1}{n^{10}})$. Applying union bound over all $\mathcal{O}(\log^3 n)$ computations of XOR_A, we see that

$$\Pr[\textit{Any } XOR_A \text{ corresponds wrongly to some edge ID}] \leq \mathcal{O}(\frac{\log^3 n}{n^{18}}) \subseteq \mathcal{O}(\frac{1}{n^{10}})$$

So, StreamingMaximalForest succeeds with high probability.

Remark One can drop the memory constraint per vertex from $\mathcal{O}(\log^4 n)$ to $\mathcal{O}(\log^3 n)$ by using a constant t instead of $t \in \mathcal{O}(\log n)$ such that the success probability is a constant larger than $1/2$. Then, simulate Borůvka for $\lceil 2 \log n \rceil$ steps. See [AGM12] (Note that they use a slightly different sketch).

Theorem 11.5. *Any randomized distributed sketching protocol for computing spanning forest with success probability $\epsilon > 0$ must have expected average sketch size $\Omega(log^3 n)$, for any constant $\epsilon > 0$.*

Proof. See [NY18]. □

Claim 11.6. *Polynomial number of bits provide sufficient independence for the procedure described above.*

Remark One can generate polynomial number of bits of randomness with $\mathcal{O}(\log^2 n)$ bits. Interested readers can check out small-bias sample spaces[5]. The construction is out of the scope of the course, but this implies that the shared randomness $\mathcal{R}$ can be obtained within our memory constraints.

[5]See https://en.wikipedia.org/wiki/Small-bias_sample_space

Part IV

Graph sparsification

Chapter 12

Preserving distances

Given a simple, unweighted, undirected graph G with n vertices and m edges, can we *sparsify* G by ignoring some edges such that certain desirable properties still hold? We will consider simple, unweighted and undirected graphs G. For any pair of vertices $u, v \in G$, denote the shortest path between them by $P_{u,v}$. Then, the distance between u and v in graph G, denoted by $d_G(u, v)$, is simply the length of shortest path $P_{u,v}$ between them.

Definition 12.1 ((α, β)-spanners)**.** *Consider a graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges. For given $\alpha \geq 1$ and $\beta \geq 0$, an (α, β)-spanner is a subgraph $G' = (V, E')$ of G, where $E' \subseteq E$, such that*

$$d_G(u, v) \leq d_{G'}(u, v) \leq \alpha \cdot d_G(u, v) + \beta$$

Remark The first inequality is because G' has less edges than G. The second inequality upper bounds how much the distances "blow up" in the sparser graph G'.

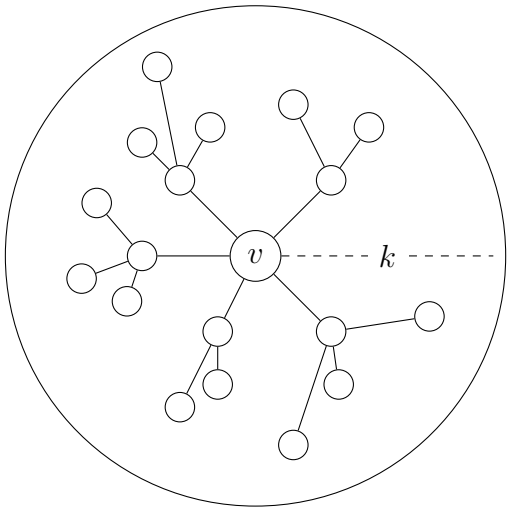
For an (α, β)-spanner, α is called the *multiplicative stretch* of the spanner and β is called the *additive stretch* of the spanner. One would then like to construct spanners with small $|E'|$ and stretch factors. An $(\alpha, 0)$-spanner is called a *α-multiplicative spanner*, and a $(1, \beta)$-spanner is called a *β-additive spanner*. We shall first look at α-multiplicative spanners, then β-additive spanners in a systematic fashion:

1. State the result (the number of edges and the stretch factor)
2. Give the construction
3. Bound the total number of edges $|E'|$
4. Prove that the stretch factor holds

Remark One way to prove the existence of an (α, β)-spanner is to use the *probabilistic method*: Instead of giving an explicit construction, one designs a random process and argues that the probability that the spanner existing is *strictly larger than 0*. However, this may be somewhat unsatisfying as such proofs do not usually yield a usable construction. On the other hand, the randomized constructions shown later are explicit and will yield a spanner *with high probability*[1].

12.1 α-multiplicative spanners

Let us first state a fact regarding the girth of a graph G. The *girth* of a graph G, denoted $g(G)$, is defined as the length of the shortest cycle in G. Suppose $g(G) > 2k$, then for any vertex v, the subgraph formed by the k-hop neighbourhood of v is a tree with distinct vertices. This is because the k-hop neighbourhood of v cannot have a cycle since $g(G) > 2k$.



Theorem 12.2. *[ADD+93] For a fixed $k \geq 1$, every graph G on n vertices has a $(2k-1)$-multiplicative spanner with $\mathcal{O}(n^{1+1/k})$ edges.*

Proof.
Construction

1. Initialize $E' = \emptyset$
2. For $e = \{u, v\} \in E$ (in arbitrary order):
   If $d_{G'}(u, v) \geq 2k$ currently, add $\{u, v\}$ into E'.
   Otherwise, ignore it.

[1] This is shown by invoking concentration bounds such as Chernoff.

Number of edges We claim that $|E'| \in \mathcal{O}(n^{1+1/k})$. Suppose, for a contradiction, that $|E'| > 2n^{1+1/k}$. Let $G'' = (V'', E'')$ be a graph obtained by iteratively removing vertices with degree $\leq n^{1/k}$ from G'. By construction, $|E''| > n^{1+1/k}$ since at most $n \cdot n^{1/k}$ edges are removed. Observe the following:

- $g(G'') \geq g(G') \geq 2k+1$, since girth does not decrease with fewer edges.
- Every vertex in G'' has degree $\geq n^{1/k} + 1$, by construction.
- Pick an arbitrary vertex $v \in V''$ and look at its k-hop neighbourhood.

$$
\begin{aligned}
n &\geq |V''| && \text{By construction} \\
&\geq |\{v\}| + \sum_{i=1}^{k} |\{u \in V'' : d_{G''}(u, v) = i\}| && \text{Look only at } k\text{-hop neighbourhood from } v \\
&\geq 1 + \sum_{i=1}^{k} (n^{1/k} + 1)(n^{1/k})^{i-1} && \text{Vertices distinct and have deg} \geq n^{1/k} + 1 \\
&= 1 + (n^{1/k} + 1)\frac{(n^{1/k})^k - 1}{n^{1/k} - 1} && \text{Sum of geometric series} \\
&> 1 + (n - 1) && \text{Since } (n^{1/k} + 1) > (n^{1/k} - 1) \\
&= n
\end{aligned}
$$

This is a contradiction since we showed $n > n$. Hence, $|E'| \leq 2n^{1+1/k} \in \mathcal{O}(n^{1+1/k})$.

Stretch factor For $e = \{u, v\} \in E$, $d_{G'}(u, v) \leq (2k-1) \cdot d_G(u, v)$ since we only leave e out of E' if the distance is at most the stretch factor at the point of considering e. For any $u, v \in V$, let $P_{u,v}$ be the shortest path between u and v in G. Say, $P_{u,v} = (u, w_1, \ldots, w_k, v)$. Then,

$$
\begin{aligned}
d_{G'}(u, v) &\leq d_{G'}(u, w_1) + \cdots + d_{G'}(w_k, v) && \text{Simulating } P_{u,v} \text{ in } G' \\
&\leq (2k-1) \cdot d_G(u, w_1) + \cdots + (2k-1) \cdot d_G(w_k, v) && \text{Apply edge stretch to each edge} \\
&= (2k-1) \cdot (d_G(u, w_1) + \cdots + d_G(w_k, v)) && \text{Rearrange} \\
&= (2k-1) \cdot d_G(u, v) && \text{Definition of } P_{u,v}
\end{aligned}
$$

□

Let us consider the family of graphs $\mathcal{G}$ on n vertices with girth $> 2k$. It can be shown by contradiction that a graph G with n vertices with girth $> 2k$ cannot have a proper $(2k-1)$-spanner[2]: Assume G' is a proper $(2k-1)$-spanner with edge $\{u, v\}$ removed. Since G' is a $(2k-1)$-spanner, $d_{G'}(u, v) \leq 2k-1$. Adding $\{u, v\}$ to G' will form a cycle of length at most $2k$, contradicting the assumption that G has girth $> 2k$.

Let $g(n, k)$ be the maximum possible number of edges in a graph from $\mathcal{G}$. By the above argument, a graph on n vertices with $g(n, k)$ edges cannot have a proper $(2k-1)$-spanner. Note that the greedy construction of Theorem 12.2 will always produce a $(2k-1)$-spanner with $\leq g(n, k)$ edges. The size of the spanner is asymptotically tight if Conjecture 12.3 holds.

Conjecture 12.3. *[Erd64] For a fixed $k \geq 1$, there exists a family of graphs on n vertices with girth at least $2k+1$ and $\Omega(n^{1+1/k})$ edges.*

Remark 1 By considering edges in increasing weight order, the greedy construction works also for weighted graphs [FS16].

Remark 2 The girth conjecture is confirmed for $k \in \{1, 2, 3, 5\}$ [Wen91, Woo06].

12.2 β-additive spanners

In this section, we will use a random process to select a subset of vertices by independently selecting vertices to join the subset. The following claim will be useful for analysis:

Claim 12.4. *If one picks vertices independently with probability p to be in $S \subseteq V$, where $|V| = n$, then*

1. $\mathbb{E}[|S|] = np$
2. *For any vertex v with degree $d(v)$ and neighbourhood $N(v) = \{u \in V : (u, v) \in E\}$,*
   - $\mathbb{E}[|N(v) \cap S|] = d(v) \cdot p$
   - $\Pr[|N(v) \cap S| = 0] \leq e^{-d(v) \cdot p}$

Proof. $\forall v \in V$, let X_v be the indicator whether $v \in S$. By construction, $\mathbb{E}[X_v] = \Pr[X_v = 1] = p$.

[2] A proper subgraph in this case refers to removing at least one edge.

1.

$$\begin{aligned}
\mathbb{E}[|S|] &= \mathbb{E}[\sum_{v\in V} X_v] && \text{By construction of } S \\
&= \sum_{v\in V} \mathbb{E}[X_v] && \text{Linearity of expectation} \\
&= \sum_{v\in V} p && \text{Since } \mathbb{E}[X_v] = \Pr[X_v = 1] = p \\
&= np && \text{Since } |V| = n
\end{aligned}$$

2.

$$\begin{aligned}
\mathbb{E}[|N(v)\cap S|] &= \mathbb{E}[\sum_{v\in N(v)} X_v] && \text{By definition of } N(v)\cap S \\
&= \sum_{v\in N(v)} \mathbb{E}[X_v] && \text{Linearity of expectation} \\
&= \sum_{v\in N(v)} p && \text{Since } \mathbb{E}[X_v] = \Pr[X_v = 1] = p \\
&= d(v)\cdot p && \text{Since } |N(v)| = d(v)
\end{aligned}$$

Probability that none of the neighbours of v is in S is

$$\Pr[|N(v)\cap S| = 0] = (1-p)^{d(v)} \leq \left(e^{-p}\right)^{d(v)} \leq e^{-p\cdot d(v)},$$

since $1 - x \leq e^{-x}$ for any x.

□

Remark $\widetilde{\mathcal{O}}$ hides logarithmic factors. For example, $\mathcal{O}(n\log^{1000} n) \subseteq \widetilde{\mathcal{O}}(n)$.

Theorem 12.5. *[ACIM99] Every graph G on n vertices has a 2-additive spanner with $\widetilde{\mathcal{O}}(n^{3/2})$ edges.*

Proof.
Construction Partition vertex set V into *light vertices* L and *heavy vertices* H, where

$$L = \{v \in V : \deg(v) \leq n^{1/2}\} \text{ and } H = \{v \in V : \deg(v) > n^{1/2}\}$$

1. Let E_1' be the set of all edges incident to some vertex in L.
2. Initialize $E_2' = \emptyset$.

- Choose $S \subseteq V$ by independently putting each vertex into S with probability $10n^{-1/2} \log n$.
- For each $s \in S$, add a Breadth-First-Search (BFS) tree rooted at s to E'_2.

Select edges in spanner to be $E' = E'_1 \cup E'_2$.
Number of edges We can bound the expected number of edges in the spanner. There are at most n light vertices, so

$$|E'_1| \leq n \cdot n^{1/2} = n^{3/2}.$$

By Claim 12.4 for $p = 10n^{-1/2} \log n$, the expected size of S is

$$\mathbb{E}[|S|] = n \cdot 10n^{-1/2} \log n = 10n^{1/2} \log n.$$

The number of edges in each BFS tree is at most $n - 1$, so

$$\mathbb{E}[|E'_2|] \leq n\mathbb{E}[|S|].$$

Therefore,

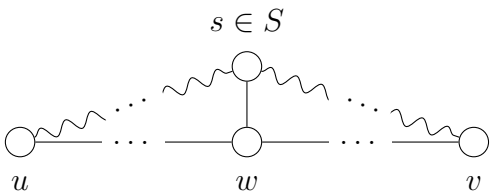
$$\begin{aligned} \mathbb{E}[|E'|] &= \mathbb{E}[|E'_1 \cup E'_2|] \leq \mathbb{E}[|E'_1| + |E'_2|] \\ &= |E'_1| + \mathbb{E}[|E'_2|] \\ &\leq n^{3/2} + n \cdot 10n^{1/2} \log n \in \widetilde{\mathcal{O}}\left(n^{3/2}\right). \end{aligned}$$

Stretch factor Consider two arbitrary vertices u and v with the shortest path $P_{u,v}$ in G. Let h be the number of heavy vertices in $P_{u,v}$. We split the analysis into two cases: (i) $h \leq 1$; (ii) $h \geq 2$. Recall that a heavy vertex has degree at least $n^{1/2}$.

Case (i) All edges in $P_{u,v}$ are adjacent to a light vertex and are thus in E'_1. Hence, $d_{G'}(u, v) = d_G(u, v)$, with additive stretch 0.

Case (ii)

Claim 12.6. *Suppose there exists a vertex $w \in P_{u,v}$ such that $(w, s) \in E$ for some $s \in S$, then $d_{G'}(u, v) \leq d_G(u, v) + 2$.*



Proof.

$$\begin{aligned} d_{G'}(u,v) &\leq d_{G'}(u,s) + d_{G'}(s,v) && (1)\\ &= d_G(u,s) + d_G(s,v) && (2)\\ &\leq d_G(u,w) + d_G(w,s) + d_G(s,w) + d_G(w,v) && (3)\\ &\leq d_G(u,w) + 1 + 1 + d_G(w,v) && (4)\\ &\leq d_G(u,v) + 2 && (5) \end{aligned}$$

(1) By triangle inequality

(2) Since we add the BFS tree rooted at s

(3) By triangle inequality

(4) Since $\{s,w\} \in E$, $d_G(w,s) = d_G(s,w) = 1$

(5) Since w lies on $P_{u,v}$

□

Let w be a heavy vertex in $P_{u,v}$ with degree $d(w) > n^{1/2}$. By Claim 12.4 with $p = 10n^{-1/2}\log n$, $\Pr[|N(w) \cap S| = 0] \leq e^{-10\log n} = n^{-10}$. Taking union bound over all possible pairs of vertices u and v,

$$\Pr\left[\exists u, v \in V, P_{u,v} \text{ has } h \geq 2 \text{ and no neighbour in } S\right] \leq \binom{n}{2} n^{-10} \leq n^{-8}$$

Then, Claim 12.6 tells us that the additive stretch factor is at most 2 with probability $\geq 1 - \frac{1}{n^8}$.

Therefore, with high probability ($\geq 1 - \frac{1}{n^8}$), the construction yields a 2-additive spanner. □

Remark A way to remove log factors from Theorem 12.5 is to sample only $n^{1/2}$ nodes into S, and then add all edges incident to nodes that don't have an adjacent node in S. The same argument then shows that this costs $\mathcal{O}(n^{3/2})$ edges in expectation.

Theorem 12.7. *[Che13] Every graph G on n vertices has a 4-additive spanner with $\widetilde{\mathcal{O}}(n^{7/5})$ edges.*

Proof.
Construction Partition vertex set V into *light vertices* L and *heavy vertices* H, where

$$L = \{v \in V : \deg(v) \leq n^{2/5}\} \text{ and } H = \{v \in V : \deg(v) > n^{2/5}\}$$

1. Let E'_1 be the set of all edges incident to some vertex in L.

2. Initialize $E'_2 = \emptyset$.

   - Choose $S \subseteq V$ by independently putting each vertex into S with probability $30n^{-3/5} \log n$.
   - For each $s \in S$, add a Breadth-First-Search (BFS) tree rooted at s to E'_2

3. Initialize $E'_3 = \emptyset$.

   - Choose $S' \subseteq V$ by independently putting each vertex into S' with probability $10n^{-2/5} \log n$.
   - For each heavy vertex $w \in H$, if there exists an edge (w, s') for some $s' \in S'$, add one such edge to E'_3.
   - $\forall s, s' \in S'$, add the shortest path among all paths from s and s' with $\leq n^{1/5}$ internal heavy vertices.
     Note: If all paths between s and s' contain $> n^{1/5}$ heavy vertices, do not add any edge to E'_3.

Select edges in the spanner to be $E' = E'_1 \cup E'_2 \cup E'_3$.

Number of edges

- Since there are at most n light vertices, $|E'_1| \leq n \cdot n^{2/5} = n^{7/5}$.

- By Claim 12.4 with $p = 30n^{-3/5} \log n$, $\mathbb{E}[|S|] = n \cdot 30n^{-3/5} \log n = 30n^{2/5} \log n$. Then, since every BFS tree has $n - 1$ edges[3], $\mathbb{E}[|E'_2|] \leq n \cdot |S| = 30n^{7/5} \log n \in \widetilde{\mathcal{O}}(n^{7/5})$.

- Since there are $\leq n$ heavy vertices, $\leq n$ edges of the form (v, s') for $v \in H, s' \in S'$ will be added to E'_3. Then, for shortest $s - s'$ paths with $\leq n^{1/5}$ heavy internal vertices, only edges adjacent to the heavy vertices need to be counted because those adjacent to light vertices are already accounted for in E'_1. By Claim 12.4 with $p = 10n^{-2/5} \log n$, $\mathbb{E}[|S'|] = n \cdot 10n^{-2/5} \log n = 10n^{3/5} \log n$. As $|S'|$ is highly concentrated around its expectation, we have $\mathbb{E}[|S'|^2] \in \widetilde{\mathcal{O}}(n^{6/5})$. So, E'_3 contributes $\leq n + \binom{|S'|}{2} \cdot n^{1/5} \in \widetilde{\mathcal{O}}(n^{7/5})$ edges to the count of $|E'|$.

[3] Though we may have repeated edges

Stretch factor Consider two arbitrary vertices u and v with the shortest path $P_{u,v}$ in G. Let h be the number of heavy vertices in $P_{u,v}$. We split the analysis into three cases: (i) $h \leq 1$; (ii) $2 \leq h \leq n^{1/5}$; (iii) $h > n^{1/5}$. Recall that a heavy vertex has degree at least $n^{2/5}$.

Case (i) All edges in $P_{u,v}$ are adjacent to a light vertex and are thus in E'_1. Hence, $d_{G'}(u,v) = d_G(u,v)$, with additive stretch 0.

Case (ii) Denote the first and last heavy vertices in $P_{u,v}$ as w and w' respectively. Recall that in Case (ii), including w and w', there are at most $n^{1/5}$ heavy vertices between w and w'. By Claim 12.4, with $p = 10n^{-2/5}\log n$,

$$\Pr[|N(w) \cap S'| = 0], \Pr[|N(w') \cap S'| = 0] \leq e^{-n^{2/5} \cdot 10n^{-2/5}\log n} = n^{-10}$$

Let $s, s' \in S'$ be vertices adjacent in G' to w and w' respectively. Observe that $s - w - w' - s'$ is a path between s and s' with at most $n^{1/5}$ internal heavy vertices. Let $P^*_{s,s'}$ be the shortest path of length l^* from s to s' with at most $n^{1/5}$ internal heavy vertices. By construction, we have added $P^*_{s,s'}$ to E'_3. Observe:

- By definition of $P^*_{s,s'}$, we have $l^* \leq d_G(s,w)+d_G(w,w')+d_G(w',s') = d_G(w,w') + 2$.
- Since there are no internal heavy vertices between $u - w$ and $w' - v$, Case (i) tells us that $d_{G'}(u,w) = d_G(u,w)$ and $d_{G'}(w',v) = d_G(w',v)$.

Thus,

$$\begin{aligned}
&d_{G'}(u,v)\\
&\leq d_{G'}(u,w) + d_{G'}(w,w') + d_{G'}(w',v) && (1)\\
&\leq d_{G'}(u,w) + d_{G'}(w,s) + d_{G'}(s,s') + d_{G'}(s',w') + d_{G'}(w',v) && (2)\\
&\leq d_{G'}(u,w) + d_{G'}(w,s) + l^* + d_{G'}(s',w') + d_{G'}(w',v) && (3)\\
&\leq d_{G'}(u,w) + d_{G'}(w,s) + d_G(w,w') + 2 + d_{G'}(s',w') + d_{G'}(w',v) && (4)\\
&= d_{G'}(u,w) + 1 + d_G(w,w') + 2 + 1 + d_{G'}(w',v) && (5)\\
&= d_G(u,w) + 1 + d_G(w,w') + 2 + 1 + d_G(w',v) && (6)\\
&\leq d_G(u,v) + 4 && (7)
\end{aligned}$$

(1) Decomposing $P_{u,v}$ in G'

(2) Triangle inequality

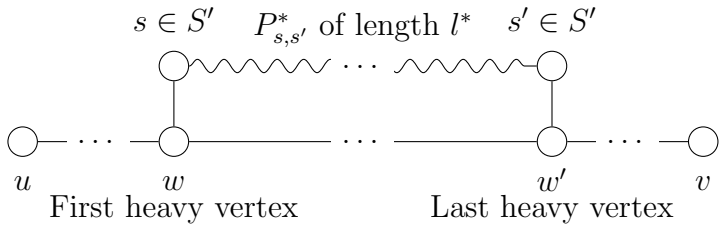
(3) $P^*_{s,s'}$ is added to E'_3

(4) Since $l^* \leq d_G(w, w') + 2$

(5) Since $(w, s) \in E'$, $(s', w') \in E'$ and $d_{G'}(w, s) = d_{G'}(s', w') = 1$

(6) Since $d_{G'}(u, w) = d_G(u, w)$ and $d_{G'}(w', v) = d_G(w', v)$

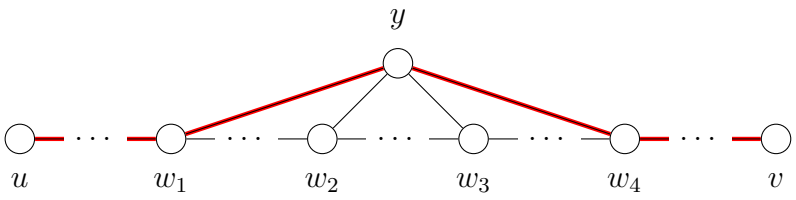
(7) By definition of $P_{u,v}$



Case (iii)

Claim 12.8. *There cannot be a vertex y that is a common neighbour to more than 3 heavy vertices in $P_{u,v}$.*

Proof. Suppose, for a contradiction, that y is adjacent to $w_1, w_2, w_3, w_4 \in P_{u,v}$ as shown in the picture. Then $u - w_1 - y - w_4 - v$ is a shorter $u - v$ path than $P_{u,v}$, contradicting the fact that $P_{u,v}$ is the shortest $u - v$ path.



Note that if y is on $P_{u,v}$ it can have at most two neighbours on $P_{u,v}$. □

Claim 12.8 tells us that $|\bigcup_{w \in H \cap P_{u,v}} N(w)| \geq \sum_{w \in H \cap P_{u,v}} |N(w)| \cdot \frac{1}{3}$. Let

$$N_{u,v} = \{x \in V : (x, w) \in E \text{ for some } w \in P_{u,v}.\}$$

Applying Claim 12.4 with $p = 30 \cdot n^{-3/5} \cdot \log n$ and Claim 12.8, we get

$$\Pr[|N_{u,v} \cap S| = 0] \leq e^{-p \cdot |N_{u,v}|} \leq e^{-p \cdot \frac{1}{3} \cdot |H \cap P_{u,v}| \cdot n^{2/5}} = e^{-10 \log n} = n^{-10}.$$

Taking union bound over all possible pairs of vertices u and v,

$$\Pr[\exists u, v \in V, P_{u,v} \text{ has } h > n^{1/5} \text{ and no neighbour in } S] \leq \binom{n}{2} n^{-10} \leq n^{-8}.$$

Then, Claim 12.6 tells us that the additive stretch factor is at most 4 with probability $\geq 1 - \frac{1}{n^8}$.

Therefore, with high probability ($\geq 1 - \frac{1}{n^8}$), the construction yields a 4-additive spanner.

□

Remark Suppose the shortest $u - v$ path $P_{u,v}$ contains a vertex from S, say s. Then, $P_{u,v}$ is contained in E' since we include the BFS tree rooted at s because it is the shortest $u - s$ path and shortest $s - v$ path by definition. In other words, the triangle inequality between u, s, v becomes tight.

Concluding remarks

	Additive β	Number of edges	Remarks
[ACIM99]	2	$\widetilde{\mathcal{O}}(n^{3/2})$	Almost [4]tight [Woo06]
[Che13]	4	$\widetilde{\mathcal{O}}(n^{7/5})$	Open: Is $\widetilde{\mathcal{O}}(n^{4/3})$ possible?
[BKMP05]	≥ 6	$\widetilde{\mathcal{O}}(n^{4/3})$	Tight [AB17]

Remark 1 A k-additive spanner is also a $(k+1)$-additive spanner.

Remark 2 The additive stretch factors appear in even numbers because current constructions "leave" the shortest path, then "re-enter" it later, introducing an even number of extra edges. Regardless, it is a folklore theorem that it suffices to only consider additive spanners with even error. Specifically, any construction of an additive $(2k+1)$-spanner on $\leq E(n)$ edges implies a construction of an additive $2k$-spanner on $\mathcal{O}(E(n))$ edges. Proof sketch: Copy the input graph G and put edges between the two copies to yield a bipartite graph H; Run the spanner construction on H; "Collapse" the parts back into one. The distance error must be even over a bipartite graph, and so the additive $(2k+1)$-spanner construction must actually give an additive $2k$-spanner by showing that the error bound is preserved over the "collapse".

[4] $\mathcal{O}(n^{4/3}/2^{\sqrt{(\log n)}})$ is still conceivable — i.e. The gap is bigger than polylog, but still subpolynomial.

Chapter 13

Preserving cuts

In the previous chapter, we looked at preserving distances via spanners. In this chapter, we look at preserving cut sizes.

Definition 13.1 (Cut and minimum cut)**.** *Consider a graph $G = (V, E)$.*

- *For $S \subseteq V, S \neq \emptyset, S \neq V$, a non-trivial cut in G is defined as the edges $C_G(S, V \setminus S) = \{(u, v) : u \in S, v \in V \setminus S\}$.*
- *The cut size is defined as $E_G(S, V \setminus S) = \sum_{e \in C_G(S, V \setminus S)} w(e)$.*
  If the graph G is unweighted, we have $w(e) = 1$ for all $e \in E$, so $E_G(S, V \setminus S) = |C_G(S, V \setminus S)|$.
- *The minimum cut size of the graph G is the minimum over all non-trivial cuts, and it is denoted by $\mu(G) = \min_{S \subseteq V, S \neq \emptyset, S \neq V} E_G(S, V \setminus S)$. If we consider an unweighted graph, the minimum cut size is the smallest number of edges whose removal disconnects the graph.*
- *A cut $C_G(S, V \setminus S)$ is said to be minimum if $E_G(S, V \setminus S) = \mu(G)$.*

Given an undirected unweighted [1] graph $G = (V, E)$, our goal in this chapter is to construct a sparse [2] weighted graph $H = (V, E')$ with $E' \subseteq E$ and weight function $w : E' \to \mathbb{R}^+$ such that

$$(1 - \epsilon) \cdot E_G(S, V \setminus S) \leq E_H(S, V \setminus S) \leq (1 + \epsilon) \cdot E_G(S, V \setminus S)$$

for every $S \subseteq V, S \neq \emptyset, S \neq V$.

[1] This can also be generalized to weighted graphs.

[2] For now, sparse means almost linear number of edges in n–we will make this concrete soon

13.1 Warm up: $G = K_n$

As a warm up, imagine that G is a complete graph. Consider the following procedure to construct H:

1. Let $p = \Omega(\frac{\log n}{\epsilon^2 n})$
2. Independently put each edge $e \in E$ into E' with probability p
3. Define $w(e) = \frac{1}{p}$ for each edge $e \in E'$

One can check that this suffices for $G = K_n$. For that, fix an arbitrary cut (any cut size is $\geq n - 1$), and analyze the probability of the above condition on the size of the cut being badly estimated in H. Then, take a union bound over all cuts. In the exercise, we discuss an even more general form of this warm up, where G is a graph with a constant edge expansion.

The rest of this section is devoted to proving a similar result for general graphs. For that, we first need to review cut counting results that you might have seen in previous courses (e.g., Algorithms, Probability, and Computing).

13.2 Prelim: Contractions and Cut Counting

Recall Karger's random contraction algorithm [Kar93][3]:

Algorithm 29 RandomContraction($G = (V, E)$)

```
while |V| > 2 do
    e ← Pick an edge uniformly at random from E
    G ← G/e                                  ▷ Contract edge e
end while
return The remaining cut                     ▷ This may be a multi-graph
```

Theorem 13.2. *For a fixed minimum cut S^* in the graph,* RandomContraction *returns it with probability $\geq 1/\binom{n}{2}$.*

Proof. Fix a minimum cut S^* in the graph and suppose $|S^*| = k$. In order for RandomContraction to successfully return S^*, *none* of the edges in S^* must be selected in the whole contraction process.

Consider the i-th step in the cycle of RandomContraction. By construction, there will be $n - i$ vertices in the graph at this point. Since $\mu(G) = k$, each vertex has degree at least k (otherwise that vertex itself

[3] Also, see https://en.wikipedia.org/wiki/Karger%27s_algorithm

gives a cut with size smaller than k), so there are at least $(n-i)k/2$ edges in the graph. Thus,

$$\begin{aligned}
\Pr[\text{Success}] &\geq \left(1-\frac{k}{nk/2}\right)\cdot\left(1-\frac{k}{(n-1)k/2}\right)\cdots\left(1-\frac{k}{3k/2}\right)\\
&= \left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{3}\right)\\
&= \left(\frac{n-2}{n}\right)\cdot\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{1}{3}\right)\\
&= \frac{2}{n(n-1)}\\
&= \frac{1}{\binom{n}{2}}
\end{aligned}$$

□

Corollary 13.3. *There are at most $\binom{n}{2}$ minimum cuts in a graph.*

Proof. Let $C_1, \cdots, C_N$ be the minimum cuts in G. According to Theorem 13.2, we know

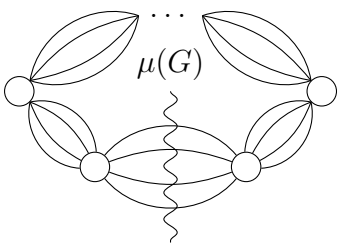
$$\Pr[C_i \text{ is found by } \textsc{RandomContraction}(G)] \geq \frac{1}{\binom{n}{2}}.$$

Now, we observe that for each two distinct indices $i, j \in [N]$, the events "C_i is found by RandomContraction(G)" and "C_j is found by RandomContraction(G)" are disjoint. Therefore, it follows that

$$\begin{aligned}
&\Pr[\text{a minimum cut is found by } \textsc{RandomContraction}(G)]\\
=&\sum_{i=1}^{N}\Pr[C_i \text{ is found by } \textsc{RandomContraction}(G)] \geq \frac{N}{\binom{n}{2}}.
\end{aligned}$$

Since $\Pr[\text{a minimum cut is found by } \textsc{RandomContraction}(G)] \leq 1$ because it is a probability, we obtain $N \leq \binom{n}{2}$. □

Remark There exist (multi-)graphs with $\binom{n}{2}$ minimum cuts: consider a cycle where there are $\frac{\mu(G)}{2}$ edges between every pair of adjacent vertices (the bound is tight when $\mu(G)$ is even).



In general, we can generalize the bound on the number of cuts that are of size at most $\alpha \cdot \mu(G)$ for $\alpha \geq 1$.

Theorem 13.4. *In an undirected graph, the number of α-minimum cuts is at most $n^{2\alpha}$.*

Proof. The proof is analogous to that of Theorem 13.2, except for taking into account that we now want the probability of any fixed α-minimum cut being output. We continue contract until there are $r = \lceil 2\alpha \rceil$ vertices remaining, and then pick one of the 2^{r-1} cuts of the resulting graph uniformly at random. Following the calculations of Theorem 13.2 and more cautious lower bounding on the success probability, it shows that the probability that we pick the fixed cut successfully is at least $1/n^{2\alpha}$, which thus means the number of α-minimum cuts is at most $n^{2\alpha}$. For more details, see Lemma 2.2 and Appendix A (in particular, Corollary A.7) of a version[4] of [Kar99]. □

13.3 Uniform edge sampling

Given a graph G with minimum cut size $\mu(G) = k$, consider the following procedure to construct H:

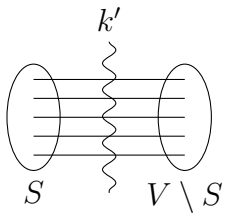
1. Set $p = \min\{1, \frac{c \log n}{\epsilon^2 k}\}$ for some constant c.
2. Independently put each edge $e \in E$ into E' with probability p.
3. Define $w(e) = \frac{1}{p}$ for each edge $e \in E'$.

Theorem 13.5. *With high probability, for every $S \subseteq V, S \neq \emptyset, S \neq V$,*

$$(1 - \epsilon) \cdot E_G(S, V \setminus S) \leq E_H(S, V \setminus S) \leq (1 + \epsilon) \cdot E_G(S, V \setminus S)$$

Proof. Fix an arbitrary cut $C_G(S, V \setminus S)$. Suppose $E_G(S, V \setminus S) = k' = \alpha \cdot k$ for some $\alpha \geq 1$.

[4] Version available at: http://people.csail.mit.edu/karger/Papers/skeleton-journal.ps



Let X_e be the indicator for the edge $e \in C_G(S, V \setminus S)$ being inserted into E'. By construction, $\mathbb{E}[X_i] = \Pr[X_i = 1] = p$. Then, by linearity of expectation, $\mathbb{E}[|C_H(S, V \setminus S)|] = \sum_{e \in C_G(S,V \setminus S)} \mathbb{E}[X_i] = k'p$. As we put $1/p$ weight on each edge in E', $\mathbb{E}[E_H(S, V \setminus S)] = k'$. Using Chernoff bound, for sufficiently large c, we get:

$$
\begin{aligned}
&\Pr[\text{Cut } C_G(S, V \setminus S) \text{ is badly estimated in } H] \\
&= \Pr[|E_H(S, V \setminus S) - \mathbb{E}[E_H(S, V \setminus S)]| > \epsilon \cdot k'] && \text{Definition of bad estimation} \\
&= \Pr[|C_H(S, V \setminus S) - \mathbb{E}[C_H(S, V \setminus S)]| > \epsilon \cdot k'p] && \text{Multiply } p \text{ on both sides} \\
&\leq 2e^{-\frac{\epsilon^2 k'p}{3}} && \text{Chernoff bound} \\
&= 2e^{-\frac{\epsilon^2 \alpha k p}{3}} && \text{Since } k' = \alpha k \\
&\leq n^{-10\alpha} && \text{For sufficiently large } c
\end{aligned}
$$

Using Theorem 13.4 and union bound over all possible cuts in G,

$$
\begin{aligned}
&\Pr[\textit{Any} \text{ cut is badly estimated in } H] \\
&\leq \int_1^\infty n^{2\alpha} \cdot \frac{1}{n^{-10\alpha}} d\alpha && \text{From Theorem 13.4 and above} \\
&\leq n^{-5} && \text{Loose upper bound}
\end{aligned}
$$

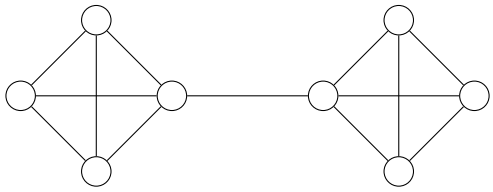
Therefore, all cuts in G are well estimated in H with high probability. □

Theorem 13.6. *[Kar99] Given a graph G, consider sampling every edge $e \in E$ into E' with independent random weights in the interval $[0, 1]$. Let $H = (V, E')$ be the sampled graph and suppose that the expected weight of every cut in H is $\geq \frac{c \log n}{\epsilon^2}$, for some constant c. Then, with high probability every cut in H has weighted size within $(1 \pm \epsilon)$ of its expectation.*

Theorem 13.6 can be proved by using a variant of the earlier proof. Interested readers can see Theorem 2.1 of [Kar99].

13.4 Non-uniform edge sampling

Unfortunately, uniform sampling does not work well on graphs with small minimum cut. Consider the following example of a graph composed of two cliques of size n with only one edge connecting them:



Running the uniform edge sampling will not sparsify the above dumbbell graph as $\mu(G) = 1$ leads to large sampling probability p.

Before we describe a non-uniform edge sampling process [BK96], we first introduce the definition of *k-strong components*.

Definition 13.7 (k-connected)**.** *A graph is k-connected if the value of each cut of G is at least k.*

Definition 13.8 (k-strong component)**.** *A k-strong component is a maximal k-connected vertex-induced subgraph.*

Definition 13.9 (edge strength)**.** *Given an edge e, its strength (or strong connectivity) k_e is the maximum k such that e is in a k-strong component. We say an edge is k-strong if $k_e \geq k$.*

Remark The (standard) connectivity of an edge $e = (u, v)$ is the minimum cut size over all cuts that separate its endpoints u and v. In particular, an edge's strong connectivity is no more than the edge's (standard) connectivity since a cut size of k between u and v implies there is no $(k + 1)$-connected component containing both u and v.

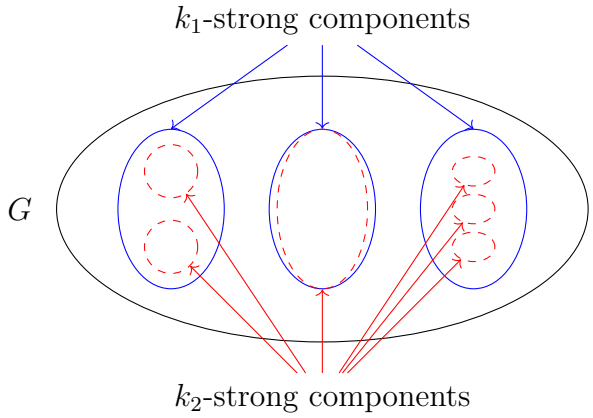
Lemma 13.10. *The following holds for k-strong components:*

1. *k_e is uniquely defined for every edge e.*
2. *For any k, the k-strong components are disjoint.*
3. *For any 2 values k_1, k_2 ($k_1 < k_2$), k_2-strong components are a* refinement *of k_1-strong components.*

4. $\sum_{e \in E} \frac{1}{k_e} \leq n - 1$.

   Intuition: If the graph is a tree, each k_e equal to one. Because there are $n-1$ edges in a tree, the sum is equal to $n-1$. If there are a lot of edges (the graph is not a tree), then many of them have high strength and the sum is therefore less than $n-1$.

Proof.



1. k_e is a maximum and there is only one maximum therefore k_e is unique.

2. Suppose, for a contradiction, there are two different intersecting k-strong components. Since their union is also k-strong, this contradicts the fact that they were maximal.

3. For $k_1 < k_2$, a k_2-strong component is also k_1-strong, so it is a subset of some k_1-strong component.

4. Consider a minimum cut $C_G(S, V \setminus S)$. Since $k_e \geq \mu(G)$ for all edges $e \in C_G(S, V \setminus S)$, these edges contribute at most $\mu(G) \cdot \frac{1}{k_e} \leq \mu(G) \cdot \frac{1}{\mu(G)} = 1$ to the summation. Remove these edges from G and repeat the argument on the remaining connected components (excluding isolated vertices). Since each cut removal contributes at most 1 to the summation and the process stops when we reach n components, then $\sum_{e \in E} \frac{1}{k_e} \leq n - 1$.

□

For a graph G with minimum cut size $\mu(G) = k$, consider the following procedure to construct H:

1. Set $q = \frac{c \log n}{\epsilon^2}$ for some constant c.

2. Independently put each edge $e \in E$ into E' with probability $p_e = \min\{1, \frac{q}{k_e}\}$.

3. Define $w(e) = \frac{1}{p_e}$ for each edge $e \in E'$.

Lemma 13.11. $\mathbb{E}[|E'|] \leq \mathcal{O}(\frac{n \log n}{\epsilon^2})$

Proof. Let X_e be the indicator random variable whether edge $e \in E$ was selected into E'. By construction, $\mathbb{E}[X_e] = \Pr[X_e = 1] = p_e$. Then,

$$\begin{aligned}
\mathbb{E}[|E'|] &= \mathbb{E}[\sum_{e \in E} X_e] && \text{By definition} \\
&= \sum_{e \in E} \mathbb{E}[X_e] && \text{Linearity of expectation} \\
&= \sum_{e \in E} p_e && \text{Since } \mathbb{E}[X_e] = \Pr[X_e = 1] = p_e \\
&= \sum_{e \in E} \frac{q}{k_e} && \text{Since } p_e = \frac{q}{k_e} \\
&\leq q(n-1) && \text{Since } \sum_{e \in E} \frac{1}{k_e} \leq n - 1 \\
&\in \mathcal{O}\left(\frac{n \log n}{\epsilon^2}\right) && \text{Since } q = \frac{c \log n}{\epsilon^2} \text{ for some constant } c
\end{aligned}$$

□

Remark If we apply Chernoff bound we see that $|E'|$ is highly concentrated around its expectation:

$$\Pr\left[\left||E'| - \mathbb{E}[|E'|]\right| \geq \epsilon \cdot \mathbb{E}[|E'|]\right] \leq 2 \exp\left(\frac{-c \log n}{3}\right)$$

We clearly see that the probability that $|E'|$ is not close to its expectation is really low.

Theorem 13.12. *With high probability, for every $S \subseteq V, S \neq \emptyset, S \neq V$,*

$$(1 - \epsilon) \cdot E_G(S, V \setminus S) \leq E_H(S, V \setminus S) \leq (1 + \epsilon) \cdot E_G(S, V \setminus S)$$

Proof. Let $k_1 < k_2 < \cdots < k_s$ be all possible strength values in the graph. Consider G' as a weighted graph with edge weights $\frac{k_e}{q}$ for each edge $e \in E$, and a family of unweighted graphs $F_1, \ldots, F_s$, where $F_i = (V, E_i)$ is the graph with edges $E_i = \{e \in E : k_e \geq k_i\}$ belonging to the k_i-strong components of G. Observe that:

- $s \leq |E|$ since each edge has only 1 strength value
- By construction of F_i's, if an edge e has strength k_i in F_i, $k_e = k_i$ in G
- $F_1 = G$
- For each $i \leq s-1$, F_{i+1} is a subgraph of F_i
- By defining $k_0 = 0$, one can write $G' = \sum_{i=1}^{s} \frac{k_i - k_{i-1}}{q} F_i$. This is because an edge with strength k_i will appear in $F_i, F_{i-1}, \ldots, F_1$ and the terms will telescope to yield a weight of $\frac{k_i}{q}$.

The sampling process in G' directly translates to a sampling process in each graph in $\{F_i\}_{i\in[s]}$ — when we add an edge e into E', we also add it to the edge sets of $F_{k_e}, \ldots, F_1$, i.e., we use the same decision of whether to keep edge e or not in G' for all F_i-s that contain it.

First, consider the sampling on the graph $F_1 = G$. We know that each edge $e \in E$ is sampled with probability $p_e = q/k_e$, where $k_e \geq k_1$ by construction of F_1. In this graph, consider any non-trivial cut $C_{F_1}(S, V \setminus S)$ and let e be any edge of this cut. We can observe that $k_e \leq E_{F_1}(S, V \setminus S)$, otherwise contradicting its strength k_e. Then, by using the indicator random variables X_e whether the edge $e \in E_1$ has been sampled, the expected size of this cut in F_1 after the sampling is

$$\begin{aligned}
\mathbb{E}[E_{F_1}(S, V \setminus S)] &= \mathbb{E}\left[\sum_{e \in C_{F_1}(S, V\setminus S)} X_e\right] \\
&= \sum_{e \in C_{F_1}(S, V\setminus S)} \mathbb{E}[X_e] && \text{Linearity of expectation} \\
&= \sum_{e \in C_{F_1}(S, V\setminus S)} \frac{q}{k_e} && \text{Since } \mathbb{E}[X_e] = \Pr[X_e = 1] = \frac{q}{k_e} \\
&\geq \sum_{e \in C_{F_1}(S, V\setminus S)} \frac{q}{E_{F_1}(S, V \setminus S)} && \text{Since } k_e \leq E_{F_1}(S, V \setminus S) \\
&= \frac{c \log n}{\epsilon^2} && \text{Since } q = \frac{c \log n}{\epsilon^2}
\end{aligned}$$

Since this holds for any cut in F_1, we can apply Theorem 13.6 to conclude that, with high probability, all cuts in F_1 have size within $(1 \pm \epsilon)$ of their expectation. Note that the same holds after scaling the edge weights in $\frac{k_1 - k_0}{q} F_1 = \frac{k_1}{q} \cdot F_1$.

In a similar way, consider any other subgraph $F_i \subseteq G$ as previously defined. Since an F_i contains the edges from the k_i-strong components of

G, any edge $e \in E_i$ belongs only to one of them. Let D be the k_i-strong component such that $e \in D$. By observing that e necessarily belongs to a k_e-connected subgraph of G by definition, and that $k_e \geq k_i$, then such a k_e-connected subgraph is entirely contained in D. Hence, the strength of e with respect to the graph D is equal to k_e. By a similar argument as done for F_1, we can show that the expected size of a cut $C_D(S, V \setminus S)$ in D after the sampling of the edges is

$$\begin{aligned}
\mathbb{E}[E_D(S, V \setminus S)] &= \sum_{e \in C_D(S, V \setminus S)} \frac{q}{k_e} && \text{Since } \mathbb{E}[X_e] = \Pr[X_e = 1] = \frac{q}{k_e} \\
&\geq \sum_{e \in C_D(S, V \setminus S)} \frac{q}{E_D(S, V \setminus S)} && \text{Since } k_e \leq E_D(S, V \setminus S) \\
&= \frac{c \log n}{\epsilon^2} && \text{Since } q = \frac{c \log n}{\epsilon^2}
\end{aligned}$$

Therefore, we can once again apply Theorem 13.6 to the subgraph D, which states that, with high probability, all cuts in D are within $(1 \pm \epsilon)$ of their expected value. We arrive at the conclusion that this also holds for F_i by applying the same argument to all the k_i-strong components of F_i.

To sum up, for each $i \in [s]$ Theorem 13.6 tells us that every cut in F_i is well-estimated with high probability. Then, a union bound over $\{F_i\}_{i \in [s]}$ provides a lower bound of the probability that all F_i's have all cuts within $(1 \pm \epsilon)$ of their expected values, and we can see that this also happens with high probability. This tells us that any cut in G is well-estimated with high probability, also because all multiplicative factors $k_i - k_{i-1}$ in the calculation $G' = \sum_{i=1}^{s} \frac{k_i - k_{i-1}}{q} F_i$ are positive. □

Part V

Online Algorithms and Competitive Analysis

Chapter 14

Warm up: Ski rental

We now study the class of online problems where one has to commit to provably good decisions as data arrive in an online fashion. To measure the effectiveness of online algorithms, we compare the quality of the produced solution against the solution from an optimal offline algorithm that knows the whole sequence of information a priori. The tool we will use for doing such a comparison is *competitive analysis*.

Remark We do not assume that the optimal offline algorithm has to be computationally efficient. Under the competitive analysis framework, only the *quality of the best possible solution* matters.

Definition 14.1 (α-competitive online algorithm)**.** *Let σ be an input sequence, c be a cost function, $\mathcal{A}$ be the online algorithm and OPT be the optimal offline algorithm. Then, denote $c_{\mathcal{A}}(\sigma)$ as the cost incurred by $\mathcal{A}$ on σ and $c_{OPT}(\sigma)$ as the cost incurred by OPT on the same sequence. We say that an online algorithm is α-competitive if for any input sequence σ, $c_{\mathcal{A}}(\sigma) \leq \alpha \cdot c_{OPT}(\sigma)$.*

Definition 14.2 (Ski rental problem)**.** *Suppose we wish to ski every day but we do not have any skiing equipment initially. On each day, we can choose between:*

- *Rent the equipment for a day at CHF* 1
- *Buying the equipment (once and for all) for CHF B*

In the toy setting where we may break our leg on each day (and cannot ski thereafter), let d be the (unknown) total number of days we ski. What is the best online strategy for renting/buying?

Claim 14.3. $\mathcal{A}$ = *"Rent for B days, then buy on day B+1" is a 2-competitive algorithm.*

Proof. If $d \leq B$, the optimal offline strategy is to rent everyday, incurring a cost of $c_{OPT}(d) = d$. $\mathcal{A}$ will rent for d days and also incur a loss of $c_A(d) = d = c_{OPT}(d)$. If $d > B$, the optimal offline strategy is to buy the equipment immediately, incurring a loss of $c_{OPT}(d) = B$. $\mathcal{A}$ will rent for B days and then buy the equipment for CHF B, incurring a cost of $c_A(d) = 2B \leq 2c_{OPT}(d)$. Thus, for any d, $c_{\mathcal{A}}(d) \leq 2 \cdot c_{OPT}(d)$. □

Chapter 15

Linear search

Definition 15.1 (Linear search problem)**.** *We have a stack of n papers on the desk. Given a query, we do a linear search from the top of the stack. Suppose the i-th paper in the stack is queried. Since we have to go through i papers to reach the queried paper, we incur a cost of i doing so. We have the option to perform two types of swaps in order to change the stack:*

Free swap *Move the queried paper from position i to the top of the stack for 0 cost.*

Paid swap *For any consecutive pair of items (a, b) before i, swap their relative order to (b, a) for 1 cost.*

What is the best online strategy for manipulating the stack to minimize total cost on a sequence of queries?

Remark One can reason that the free swap costs 0 because we already incurred a cost of i to reach the queried paper.

15.1 Amortized analysis

Amortized analysis[1] is a way to analyze the complexity of an algorithm on a sequence of operations. Instead of looking the worst case performance on a single operation, it measures the total cost for a *batch* of operations.

The dynamic resizing process of hash tables is a classical example of amortized analysis. An insertion or deletion operation will typically cost

[1] See https://en.wikipedia.org/wiki/Amortized_analysis

$\mathcal{O}(1)$ unless the hash table is almost full or almost empty, in which case we double or halve the hash table of size m, incurring a runtime of $\mathcal{O}(m)$.

Worst case analysis tells us that dynamic resizing will incur $\mathcal{O}(m)$ run time per operation. However, resizing only occurs after $\mathcal{O}(m)$ insertion/deletion operations, each costing $\mathcal{O}(1)$. Amortized analysis allows us to conclude that this dynamic resizing runs in amortized $\mathcal{O}(1)$ time. There are two equivalent ways to see it:

- Split the $\mathcal{O}(m)$ resizing overhead and "charge" $\mathcal{O}(1)$ to each of the earlier $\mathcal{O}(m)$ operations.
- The *total* run time for every sequential chunk of m operations is $\mathcal{O}(m)$. Hence, each step takes $\mathcal{O}(m)/m = \mathcal{O}(1)$ amortized run time.

15.2 Move-to-Front

Move-to-Front (MTF) [ST85] is an online algorithm for the linear search problem where we move the queried item to the top of the stack (and do no other swaps). We will show that MTF is a 2-competitive algorithm for linear search. Before we analyze MTF, let us first define a potential function Φ and look at examples to gain some intuition.

Let Φ_t be the number of pairs of papers (i, j) that are ordered differently in MTF's stack and OPT's stack at time step t. By definition, $\Phi_t \geq 0$ for any t. We also know that $\Phi_0 = 0$ since MTF and OPT operate on the same initial stack sequence.

Example One way to interpret Φ is to count the number of inversions between MTF's stack and OPT's stack. Suppose we have the following stacks (visualized horizontally) with $n = 6$:

	1	2	3	4	5	6
MTF's stack	a	b	c	d	e	f
OPT's stack	a	b	e	d	c	f

We have the inversions (c, d), (c, e) and (d, e), so $\Phi = 3$.

Scenario 1 We swap (b, e) in OPT's stack — A new inversion (b, e) was created due to the swap.

	1	2	3	4	5	6
MTF's stack	a	b	c	d	e	f
OPT's stack	a	e	b	d	c	f

Now, we have the inversions (b, e), (c, d), (c, e) and (d, e), so $\Phi = 4$.

Scenario 2 We swap (e, d) in OPT's stack — The inversion (d, e) was destroyed due to the swap.

	1	2	3	4	5	6
MTF's stack	a	b	c	d	e	f
OPT's stack	a	b	d	e	c	f

Now, we have the inversions (c, d) and (c, e), so $\Phi = 2$.

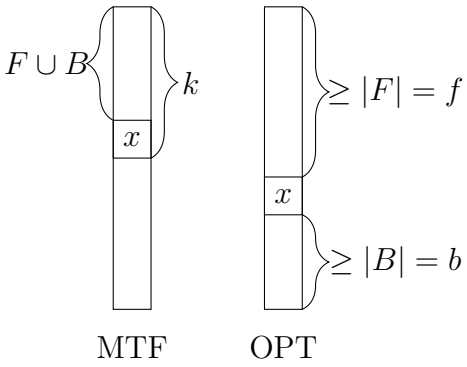
In either case, we see that any paid swap results in ± 1 inversions, which changes Φ by ± 1.

Claim 15.2. *MTF is 2-competitive.*

Proof. We will consider the potential function Φ as before and perform amortized analysis on any given input sequence σ. Let $a_t = c_{MTF}(t) + (\Phi_t - \Phi_{t-1})$ be the amortized cost of MTF at time step t, where $c_{MTF}(t)$ is the cost MTF incurs at time t. Suppose the queried item x at time step t is at position k in MTF's stack. Denote:

$F = \{\text{Items } \textit{on top} \text{ of } x \text{ in MTF's stack and } \textit{on top} \text{ of } x \text{ in OPT's stack}\}$
$B = \{\text{Items } \textit{on top} \text{ of } x \text{ in MTF's stack and } \textit{underneath } x \text{ in OPT's stack}\}$

Let $|F| = f$ and $|B| = b$. There are $k-1$ items in front x, so $f+b = k-1$.



Since x is the k-th item, MTF will incur $c_{MTF}(t) = k = f + b + 1$ to reach item x, then move it to the top. On the other hand, OPT needs to

spend at least $f + 1$ to reach x. Suppose OPT does p paid swaps, then $c_{OPT}(t) \geq f + 1 + p$.

To measure the change in potential, we first look at the swaps done by MTF and how OPT's swaps can affect them. Let $\Delta_{MTF}(\Phi_t)$ be the change in Φ due to MTF and $\Delta_{OPT}(\Phi_t)$ be the change in Φ_t due to OPT. Thus, $\Delta(\Phi_t) = \Delta_{MTF}(\Phi_t) + \Delta_{OPT}(\Phi_t)$. In MTF, moving x to the top destroys b inversions and creates f inversions, so the change in Φ due to MTF is $\Delta_{MTF}(\Phi_t) = f - b$. If OPT chooses to do a free swap, Φ does not increase as both stacks now have x before any element in F. For every paid swap that OPT performs, Φ changes by one since inversions only locally affect the swapped pair and thus, $\Delta_{OPT}(\Phi_t) \leq p$.

Therefore, the effect on Φ from both processes is: $\Delta(\Phi_t) = \Delta_{MTF}(\Phi_t) + \Delta_{OPT}(\Phi_t) \leq (f - b) + p$. Putting together, we have $c_{OPT}(t) \geq f + 1 + p$ and $a_t = c_{MTF}(t) + (\Phi_t - \Phi_{t-1}) = k + \Delta(\Phi_t) \leq 2f + 1 + p \leq 2 \cdot c_{OPT}(t)$. Summing up over all queries in the sequence yields:

$$2 \cdot c_{OPT}(\sigma) = \sum_{t=1}^{|\sigma|} 2 \cdot c_{OPT}(t) \geq \sum_{t=1}^{|\sigma|} a_t$$

With $a_t = c_{MTF}(t) + (\Phi_t - \Phi_{t-1})$ and using the fact that the sum over the change in potential is telescoping, we get:

$$\begin{aligned} \sum_{t=1}^{|\sigma|} a_t &= \sum_{t=1}^{|\sigma|} c_{MTF}(t) + (\Phi_t - \Phi_{t-1}) \\ &= \sum_{t=1}^{|\sigma|} c_{MTF}(t) + (\Phi_{|\sigma|} - \Phi_0) \end{aligned}$$

Since $\Phi_t \geq 0 = \Phi_0$ and $c_{MTF}(\sigma) = \sum_{t=1}^{|\sigma|} c_{MTF}(t)$:

$$\sum_{t=1}^{|\sigma|} c_{MTF}(t) + (\Phi_{|\sigma|} - \Phi_0) \geq \sum_{t=1}^{|\sigma|} c_{MTF}(t) = c_{MTF}(\sigma)$$

We have shown that $c_{MTF}(\sigma) \leq 2 \cdot c_{OPT}(\sigma)$ which completes the proof. □

Chapter 16

Paging

Definition 16.1 (Paging problem [ST85])**.** *Suppose we have a fast memory (cache) that can fit k pages and an unbounded sized slow memory. Accessing items in the cache costs 0 units of time while accessing items in the slow memory costs 1 unit of time. After accessing an item in the slow memory, we can bring it into the cache by evicting an incumbent item if the cache was full. What is the best online strategy for maintaining items in the cache to minimize the total access cost on a sequence of queries?*

Denote *cache miss* as accessing an item that is not in the cache. Any sensible strategy should aim to reduce the number of cache misses. For example, if $k = 3$ and $\sigma = \{1, 2, 3, 4, \ldots, 2, 3, 4\}$, keeping item 1 in the cache will incur several cache misses. Instead, the strategy should aim to keep items $\{2, 3, 4\}$ in the cache. We formalize this notion in the following definition of *conservative strategy*.

Definition 16.2 (Conservative strategy)**.** *A strategy is conservative if on any consecutive subsequence that includes only k distinct pages, there are at most k cache misses.*

Remark Some natural paging strategies such as "Least Recently Used (LRU)" and "First In First Out (FIFO)" are conservative.

Claim 16.3. *If $\mathcal{A}$ is a deterministic online algorithm that is α-competitive, then $\alpha \geq k$.*

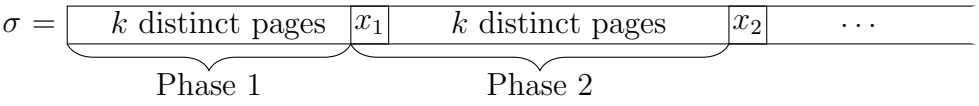
Proof. Consider the following input sequence σ on $k + 1$ pages: since the cache has size k, at least one item is not in the cache at any point in time. Iteratively pick $\sigma(t + 1)$ as the item not in the cache after time step t.

Since $\mathcal{A}$ is deterministic, the adversary can simulate $\mathcal{A}$ for $|\sigma|$ steps and build σ accordingly. By construction, $c_{\mathcal{A}}(\sigma) = |\sigma|$.

On the other hand, since OPT can see the entire sequence σ, OPT can choose to evict the page i that is requested furthest in the future. The next request for page i has to be at least k requests ahead in the future, since by definition of i all other pages $j \neq i \in \{1, ..., k+1\}$ have to be requested before i. Thus, in every k steps, OPT has ≤ 1 cache miss. Therefore, $c_{OPT} \leq \frac{|\sigma|}{k}$ which implies: $k \cdot c_{OPT} \leq |\sigma| = c_{\mathcal{A}}(\sigma)$. □

Claim 16.4. *Any conservative online algorithm $\mathcal{A}$ is k-competitive.*

Proof. For any given input sequence σ, partition σ into m *maximal phases* — $P_1, P_2, \ldots, P_m$ — where each phase has k distinct pages, and a new phase is created only if the next element is different from the ones in the current phase. Let x_i be the first item that does not belong in Phase i.



By construction, OPT has to pay ≥ 1 to handle the elements in $P_i \cup \{x_i\}$, for any i; so $c_{OPT} \geq m$. On the other hand, since $\mathcal{A}$ is conservative, $\mathcal{A}$ has $\leq k$ cache misses per phase. Hence, $c_{\mathcal{A}}(\sigma) \leq k \cdot m \leq k \cdot c_{OPT}(\sigma)$. □

Remark A randomized algorithm can achieve $\mathcal{O}(\log k)$-competitiveness. This will be covered in the next lecture.

16.1 Types of adversaries

Since online algorithms are analyzed on all possible input sequences, it helps to consider adversarial inputs that may induce the worst case performance for a given online algorithm $\mathcal{A}$. To this end, one may wish to classify the classes of adversaries designing the input sequences (in increasing power):

Oblivious The adversary designs the input sequence σ at the beginning. It does not know any randomness used by algorithm $\mathcal{A}$.

Adaptive At each time step t, the adversary knows all randomness used by algorithm $\mathcal{A}$ thus far. In particular, it knows the exact state of the algorithm. With these in mind, it then picks the $(t+1)$-th element in the input sequence.

Fully adaptive The adversary knows all possible randomness that will be used by the algorithm $\mathcal{A}$ when running on the full input sequence σ. For

instance, assume the adversary has access to the same pseudorandom number generator used by $\mathcal{A}$ and can invoke it arbitrarily many times while designing the adversarial input sequence σ.

Remark If $\mathcal{A}$ is deterministic, then all three classes of adversaries have the same power.

16.2 Random Marking Algorithm (RMA)

Consider the Random Marking Algorithm (RMA), a $\mathcal{O}(\log k)$-competitive algorithm for paging against oblivious adversaries:

- Initialize all pages as marked
- Upon request of a page p
  - If p is not in cache,
    - If all pages in cache are marked, unmark all
    - Evict a random unmarked page
  - Mark page p

Example Suppose $k = 3$, $\sigma = (2, 5, 2, 1, 3)$.

Suppose the cache is initially:

Cache	1	3	4
Marked?	✓	✓	✓

When $\sigma(1) = 2$ arrives, all pages were unmarked. Suppose the random eviction chose page '3'. The newly added page '2' is then marked.

Cache	1	2	4
Marked?	✗	✓	✗

When $\sigma(2) = 5$ arrives, suppose random eviction chose page '4' (between pages '1' and '4'). The newly added page '5' is then marked.

Cache	1	2	5
Marked?	✗	✓	✓

When $\sigma(3) = 2$ arrives, page '2' in the cache is marked (no change).

Cache	1	2	5
Marked?	✗	✓	✓

When $\sigma(4) = 1$ arrives, page '1' in the cache is marked. At this point, any page request that is not from $\{1, 2, 5\}$ will cause a full unmarking of all pages in the cache.

Cache	1	2	5
Marked?	✓	✓	✓

When $\sigma(5) = 3$ arrives, all pages were unmarked. Suppose the random eviction chose page '5'. The newly added page '3' is then marked.

Cache	1	2	3
Marked?	✗	✗	✓

We denote a phase as the time period between 2 consecutive full unmarking steps. That is, each phase is a maximal run where we access k distinct pages. In the above example, $\{2, 5, 2, 1\}$ is such a phase for $k = 3$.

Observation As pages are only unmarked at the beginning of a new phase, the number of unmarked pages is monotonically decreasing within a phase.

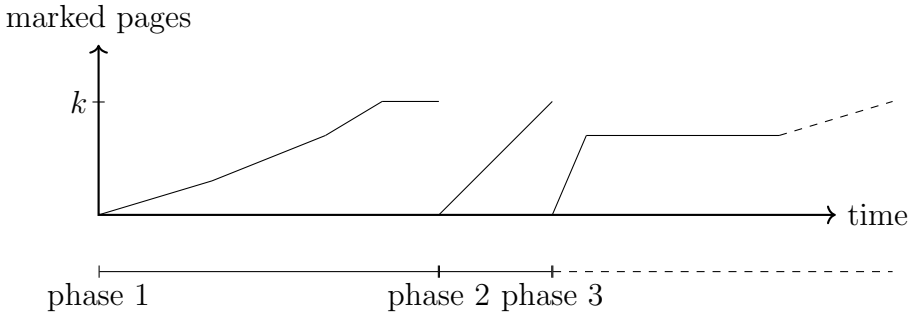


Figure 16.1: The number of marked pages within a phase is monotonically increasing.

Theorem 16.5. *RMA is $\mathcal{O}(\log k)$-competitive against any oblivious adversary.*

Proof. Let P_i be the set of pages at the *start* of phase i. Since requesting a marked page does not incur any cost, it suffices to analyze the *first time* any request occurs within the phase.

Let m_i be the number of *unique new requests* (pages that are not in P_i) and o_i as the number of *unique old requests* (pages that are in P_i). By definition, $o_i \leq k$ and $m_i + o_i = k$.

We have $c_{RMA}(\text{Phase } i) = (\text{Cost due to new requests}) + (\text{Cost due to old requests})$. We first focus on the extra cost incurred from the old requests,

that is when an old page is requested that has already been kicked out upon the arrival of a new request.

Order the old requests in the order which they appear in the phase and let x_j be the j^{th} old request, for $j \in \{1, \ldots, o_i\}$. Define l_j as the number of *distinct new* requests before x_j.

For $j \in \{1, \ldots, o_i\}$, consider the first time the j^{th} old request x_j occurs. Since the adversary is oblivious, x_j is equally likely to be in any position in the cache at the start of the phase. After seeing $(j-1)$ old requests and marking their cache positions, there are $k-(j-1)$ initial positions in the cache that x_j could be in. Since we have only seen l_j new requests and $(j-1)$ old requests, there are at least[1] $k - l_j - (j-1)$ old pages remaining in the cache. So, the probability that x_j is in the cache when requested is at least $\frac{k-l_j-(j-1)}{k-(j-1)}$. Then,

$$
\begin{aligned}
\text{Cost due to old requests} &= \sum_{j=1}^{o_i} \Pr[x_j \text{ is } \textit{not} \text{ in cache when requested}] && \text{Sum over old requests} \\
&\leq \sum_{j=1}^{o_i} \frac{l_j}{k-(j-1)} && \text{From above} \\
&\leq \sum_{j=1}^{o_i} \frac{m_i}{k-(j-1)} && \text{Since } l_j \leq m_i = |\mathcal{N}| \\
&\leq m_i \cdot \sum_{j=1}^{k} \frac{1}{k-(j-1)} && \text{Since } o_i \leq k \\
&= m_i \cdot \sum_{j=1}^{k} \frac{1}{j} && \text{Rewriting} \\
&= m_i \cdot H_k && \text{Since } \sum_{i=1}^{n} \frac{1}{i} = H_n
\end{aligned}
$$

Since every new request incurs a unit cost, the cost due to these requests is m_i.

Together for new and old requests, we get $c_{RMA}(\text{Phase } i) \leq m_i + m_i \cdot H_k$.

We now analyze OPT's performance. By definition of phases, among all requests between two consecutive phases (say, $i-1$ and i), a total of $k + m_i$ distinct pages are requested. So, OPT has to incur at least $\geq m_i$ to bring in

[1] We get an equality if *all* these requests kicked out an old page.

these new pages. To avoid double counting, we lower bound $c_{OPT}(\sigma)$ for both odd and even i: $c_{OPT}(\sigma) \geq \sum_{\text{odd } i} m_i$ and $c_{OPT}(\sigma) \geq \sum_{\text{even } i} m_i$. Together,

$$2 \cdot c_{OPT}(\sigma) \geq \sum_{\text{odd } i} m_i + \sum_{\text{even } i} m_i \geq \sum_i m_i$$

Therefore, we have:

$$c_{RMA}(\sigma) \leq \sum_i (m_i + m_i \cdot H_k) = \mathcal{O}(\log k) \sum_i m_i \leq \mathcal{O}(\log k) \cdot c_{OPT}(\sigma)$$

□

Remark In the above example, $k = 3$, phase $1 = (2, 5, 2, 1)$, $P_1 = \{1, 3, 4\}$, new requests $= \{2, 5\}$, old requests $= \{1\}$. Although '2' appeared twice, we only care about analyzing the first time it appeared.

16.3 Lower Bound for Paging via Yao's Principle

Yao's Principle Often, it is considerably easier to obtain (distributional) lower bounds against deterministic algorithms, than to (directly) obtain deterministic lower bound instances against randomized algorithms. We use Yao's principle to bridge this gap. Informally, this principle tells us that if *no* deterministic algorithm can do well on a given distribution of random inputs (D), then for any randomized algorithm, there is a deterministic bad input so that the cost of the randomized algorithm on this particular input will be high (C). We next state and prove Yao's principle.

Before getting to the principle, let us observe that given the sequence of random bits used, a randomized algorithm behaves deterministically. Hence, one may view a randomized algorithm as a random choice from a distribution of deterministic algorithms.

Let $\mathcal{X}$ be the space of problem inputs and $\mathcal{A}$ be the space of all possible deterministic algorithms. Denote probability distributions over $\mathcal{A}$ and $\mathcal{X}$ by $p_a = \Pr[A = a]$ and $q_x = \Pr[X = x]$, where X and A are random variables for input and deterministic algorithm, respectively. Define $c(a, x)$ as the cost of algorithm $a \in A$ on input $x \in X$.

Theorem 16.6 ([Yao77])**.**

$$C = \max_{x \in X} \mathbb{E}_p[c(a, x)] \geq \min_{a \in A} \mathbb{E}_q[c(a, x)] = D$$

Proof.

$$
\begin{aligned}
C &= \sum_x q_x \cdot C && \text{Sum over all possible inputs } x \\
&\geq \sum_x q_x \mathbb{E}_p[c(A,x)] && \text{Since } C = \max_{x \in X} \mathbb{E}_p[c(A,x)] \\
&= \sum_x q_x \sum_p p_a c(a,x) && \text{Definition of } \mathbb{E}_p[c(A,x)] \\
&= \sum_a p_a \sum_q q_x c(a,x) && \text{Swap summations} \\
&= \sum_a p_a \mathbb{E}_q[c(a,X)] && \text{Definition of } \mathbb{E}_q[c(a,X)] \\
&\geq \sum_a p_a \cdot D && \text{Since } D = \min_{a \in A} \mathbb{E}_q[c(a,X)] \\
&= D && \text{Sum over all possible algorithms } a
\end{aligned}
$$

□

Application to the paging problem

Theorem 16.7. *Any (randomized) algorithm has competitive ratio* $\Omega(\log k)$ *against an oblivious adversary.*

Proof. Fix an arbitrary deterministic algorithm $\mathcal{A}$. Let $|\sigma| = m$. Consider the following random input sequence σ where the i-th page is drawn from $\{1, \ldots, k+1\}$ uniformly at random.

By construction of σ, the probability of having a cache miss is $\frac{1}{k+1}$ for $\mathcal{A}$, regardless of what $\mathcal{A}$ does. Hence, $\mathbb{E}[c_{\mathcal{A}}(\sigma)] = \frac{m}{k+1}$.

On the other hand, an optimal offline algorithm may choose to evict the page that is requested furthest in the future. As before, we denote a phase as a maximal run where there are k distinct page requests. This means that $\mathbb{E}[c_{OPT}(\sigma)] = \text{Expected number of phases} = \frac{m}{\text{Expected phase length}}$.

To analyze the expected length of a phase, suppose there are i distinct pages so far, for $0 \leq i \leq k$. The probability of the next request being new is $\frac{k+1-i}{k+1}$, and one expects to get $\frac{k+1}{k+1-i}$ requests before having $i+1$ distinct pages. Thus, the expected length of a phase is $\sum_{i=0}^{k} \frac{k+1}{k+1-i} = (k+1) \cdot H_{k+1}$. Therefore, $\mathbb{E}[c_{OPT}(\sigma)] = \frac{m}{(k+1) \cdot H_{k+1}}$.

So far we have obtained that $D = \mathbb{E}[c_{\mathcal{A}}(\sigma)] = \frac{m}{k+1}$; from Yao's Minimax Principle we know that $C \geq D$, hence we can also compare the competitive ratios $\frac{C}{\mathbb{E}[c_{OPT}(\sigma)]} \geq \frac{D}{\mathbb{E}[c_{OPT}(\sigma)]} = H_{k+1} = \Theta(\log k)$. □

Remark The length of a phase is essentially the coupon collector problem with $n = k + 1$ coupons.

Chapter 17

The k-server problem

Definition 17.1 (k-server problem [MMS90])**.** *Consider a metric space (V, d) where V is a set of n points and $d : V \times V \to \mathbb{R}$ is a distance metric between any two points. Suppose there are k servers placed on V and we are given an input sequence $\sigma = (v_1, v_2, \dots)$. Upon request of $v_i \in V$, we have to move one server to point v_i to satisfy that request. What is the best online strategy to minimize the total distance travelled by servers to satisfy the sequence of requests?*

Remark We do not fix the starting positions of the k servers, but we compare the performance of OPT on σ with same initial starting positions.

The paging problem is a special case of the k-server problem where the points are all possible pages, the distance metric is unit cost between any two different points, and the servers represent the pages in cache of size k.

Progress It is conjectured that a deterministic k-competitive algorithm exists and a randomized poly$(\log k)$-competitive algorithm exists. A long-standing conjecture of a $O(\log k)$-competitive randomized algorithm existing was disproved in 2023 [BCR23]. The table below shows the current progress on this problem.

	Competitive ratio	Type
[MMS90]	k-competitive, for $k = 2$ and $k = n - 1$	Deterministic
[FRR90]	$2^{\mathcal{O}(k \log k)}$-competitive	Deterministic
[Gro91]	$2^{\mathcal{O}(k)}$-competitive	Deterministic
[KP95]	$(2k - 1)$-competitive	Deterministic
[BBMN11]	poly$(\log n, \log k)$-competitive	Randomized

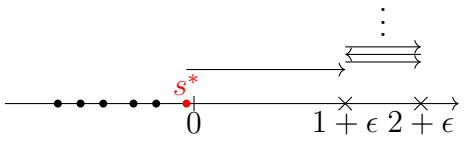
Remark [BBMN11] uses a probabilistic tree embedding, a concept we have seen in earlier lectures.

17.1 Special case: Points on a line

Consider the metric space where V are points on a line and $d(u, v)$ is the distance between points $u, v \in V$. One can think of all points lying on the 1-dimensional number line $\mathbb{R}$.

17.1.1 Greedy is a bad idea

A natural greedy idea would be to pick the closest server to serve any given request. However, this can be arbitrarily bad. Consider the following:

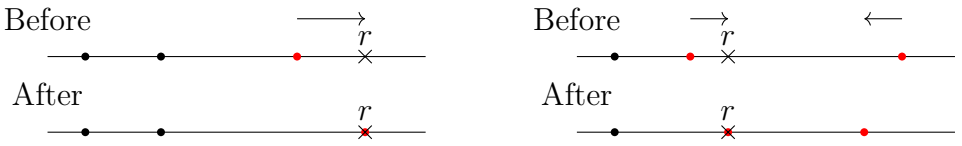


Without loss of generality, suppose all servers currently lie on the left of "0". For $\epsilon > 0$, consider the sequence $\sigma = (1+\epsilon, 2+\epsilon, 1+\epsilon, 2+\epsilon, \dots)$. The first request will move a single server s^* to "$1+\epsilon$". By the greedy algorithm, subsequent requests then repeatedly use s^* to satisfy requests from both "$1+\epsilon$" and "$2+\epsilon$" since s^* is the closest server. This incurs a total cost of $\geq |\sigma|$ while OPT could station 2 servers on "$1+\epsilon$" and "$2+\epsilon$" and incur a constant total cost on input sequence σ.

17.1.2 Double coverage

The *double coverage* algorithm does the following:

- If request r is on one side of all servers, move the closest server to cover it
- If request r lies between two servers, move both towards it at *constant speed* until r is covered



Theorem 17.2. *Double coverage (DC) is k-competitive on a line.*

Proof. Without loss of generality,

- Suppose location of DC's servers on the line are: $x_1 \leq x_2 \leq \cdots \leq x_k$
- Suppose location of OPT's servers on the line are: $y_1 \leq y_2 \leq \cdots \leq y_k$

Define potential function $\Phi = \Phi_1 + \Phi_2 = k \cdot \sum_{i=1}^{k} |x_i - y_i| + \sum_{i<j}(x_j - x_i)$, where Φ_1 is k times the "paired distances" between x_i and y_i and Φ_2 is the pairwise distance between any two servers in DC.

We denote the potential function at time step t by $\Phi_t = \Phi_{t,1} + \Phi_{t,2}$. For a given request r at time step t, we will first analyze OPT's action then DC's action. We analyze the change in potential $\Delta(\Phi)$ by looking at $\Delta(\Phi_1)$ and $\Delta(\Phi_2)$ separately, and further distinguish the effects of DC and OPT on $\Delta(\Phi)$ via $\Delta_{DC}(\Phi)$ and $\Delta_{OPT}(\Phi)$ respectively.

Suppose OPT moves server s^* by a distance of $x = d(s^*, r)$ to reach the point r. Then, $c_{OPT}(t) \geq x$. Since s^* moved by x, $\Delta(\Phi_{t,1}) \leq kx$. Since OPT does not move DC's servers, $\Delta(\Phi_{t,2}) = 0$. Hence, $\Delta_{OPT}(\Phi_t) \leq kx$.

There are three cases for DC, depending on where r appears.

1. r appears exactly on a current server position
   DC does nothing. So, $c_{DC}(t) = 0$ and $\Delta_{DC}(\Phi_t) = 0$. Hence,

$$\begin{aligned} c_{DC}(t) + \Delta(\Phi_t) &= c_{DC}(t) + \Delta_{DC}(\Phi_t) + \Delta_{OPT}(\Phi_t) \\ &\leq 0 + kx + 0 = kx \\ &\leq k \cdot c_{OPT}(t) \end{aligned}$$

2. r appears on one side of all servers $x_1, \ldots, x_k$ (say $r > x_k$ without loss of generality)
   DC will move server x_k by a distance $y = d(x_k, r)$ to reach point r. That is, $c_{DC}(t) = y$. Since OPT has a server at r, $y_k \geq r$. So, $\Delta_{DC}(\Phi_{t,1}) = -ky$. Since only x_k moved, $\Delta_{DC}(\Phi_{t,2}) = (k-1)y$. Hence,

$$\begin{aligned} c_{DC}(t) + \Delta(\Phi_t) &= c_{DC}(t) + \Delta_{DC}(\Phi_t) + \Delta_{OPT}(\Phi_t) \\ &\leq y - ky + (k-1)y + kx \\ &= kx \\ &\leq k \cdot c_{OPT}(t) \end{aligned}$$

3. r appears between two servers $x_i < r < x_{i+1}$
   Without loss of generality, say r is closer to x_i and denote $z = d(x_i, r)$. DC will move server x_i by a distance of z to reach point r, and server x_{i+1} by a distance of z to reach $x_{i+1} - z$. That is, $c_{DC}(t) = 2z$.

Claim 17.3. *At least one of x_i or x_{i+1} is moving closer to its partner (y_i or y_{i+1} respectively).*

Proof. Suppose, for a contradiction, that both x_i and x_{i+1} are moving *away* from their partners. That means $y_i \leq x_i < r < x_{i+1} \leq y_{i+1}$ at the end of OPT's action (before DC moved x_i and x_{i+1}). This is a contradiction since OPT must have a server at r but there is no server between y_i and y_{i+1} by definition. □

Since at least one of x_i or x_{i+1} is moving closer to its partner, $\Delta_{DC}(\Phi_{t,1}) \leq z - z = 0$.

Meanwhile, since x_i and x_{i+1} are moved a distance of z towards each other, $(x_{i+1} - x_i) = -2z$ while the total change against other pairwise distances cancel out, so $\Delta_{DC}(\Phi_{t,2}) = -2z$.

Hence,

$$c_{DC}(t)+\Delta(\Phi_t) = c_{DC}(t)+\Delta_{DC}(\Phi_t)+\Delta_{OPT}(\Phi_t) \leq 2z-2z+kx = kx \leq k \cdot c_{OPT}(t)$$

In all cases, we see that $c_{DC}(t) + \Delta(\Phi_t) \leq k \cdot c_{OPT}(t)$. Hence,

$$\sum_{t=1}^{|\sigma|}(c_{DC}(t) + \Delta(\Phi_t)) \leq \sum_{t=1}^{|\sigma|} k \cdot c_{OPT}(t) \qquad \text{Summing over } \sigma$$

$$\Rightarrow \quad \sum_{t=1}^{|\sigma|} c_{DC}(t) + (\Phi_{|\sigma|} - \Phi_0) \leq k \cdot c_{OPT}(\sigma) \qquad \text{Telescoping}$$

$$\Rightarrow \quad \sum_{t=1}^{|\sigma|} c_{DC}(t) - \Phi_0 \leq k \cdot c_{OPT}(\sigma) \qquad \text{Since } \Phi_t \geq 0$$

$$\Rightarrow \quad c_{DC}(\sigma) \leq k \cdot c_{OPT}(\sigma) + \Phi_0 \quad \text{Since } c_{DC}(\sigma) = \sum_{t=1}^{|\sigma|} c_{DC}(t)$$

Since Φ_0 is a constant that captures the initial state, *DC* is k-competitive. □

Remark One can generalize the approach of double coverage to points on a tree. The idea is as follows: For a given request point r, consider the set of servers S such that for $s \in S$, there is no other server s' between s and r. Move all servers in S towards r "at the same speed" until one

of them reaches r. This generalization gives us a k-competitiveness on a tree; building on this we can use the Probabilistic Tree Embedding approach (stretching distances by only $\mathcal{O}(\log n)$ in expectation) getting immediately an $\mathcal{O}(k \log n)$-competitiveness in expectation on a graph.

Bibliography

[AB17] Amir Abboud and Greg Bodwin. The $\frac{4}{3}$ additive spanner exponent is tight. *Journal of the ACM (JACM)*, 64(4):28, 2017.

[ACIM99] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

[ADD+93] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 459–467. SIAM, 2012.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 184–193. IEEE, 1996.

[BBMN11] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server

problem. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 267–276. IEEE, 2011.

[BCR23] Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 581–594. ACM, 2023.

[BK96] András A Benczúr and David R Karger. Approximating st minimum cuts in $\widetilde{O}(n^2)$ time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 47–55. ACM, 1996.

[BKMP05] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (α, β)-spanners and purely additive spanners. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 672–681. Society for Industrial and Applied Mathematics, 2005.

[BYJK+02] Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[BYJKS04] Ziv Bar-Yossef, Thathachar S Jayram, Ravi Kumar, and D Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004.

[Che13] Shiri Chechik. New additive spanners. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 498–512. Society for Industrial and Applied Mathematics, 2013.

[DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 624–633. ACM, 2014.

[Erd64] P. Erdös. Extremal problems in graph theory. In *"Theory of graphs and its applications," Proc. Symposium Smolenice*, pages 29–36, 1964.

[Fei98] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[FM85] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[FRR90] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 454–463. IEEE, 1990.

[FRT03] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455. ACM, 2003.

[FS16] Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 9–17. ACM, 2016.

[Gra66] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[Gro91] Edward F Grove. The harmonic online k-server algorithm is competitive. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 260–266. ACM, 1991.

[HMK+06] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[Ind01] Piotr Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 10–33. IEEE, 2001.

[IW05] Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208. ACM, 2005.

[Joh74] David S Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

[Kar93] David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *SODA*, volume 93, pages 21–30, 1993.

[Kar99] David R Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

[Kar01] David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Rev.*, 43(3):499–522, March 2001.

[KP95] Elias Koutsoupias and Christos H Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995.

[LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

[MMS90] Mark S Manasse, Lyle A McGeoch, and Daniel D Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

[NY18] Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. *arXiv preprint arXiv:1807.05135*, 2018.

[RT87] Prabhakar Raghavan and Clark D Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[ST85] Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[Vaz13] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[Wen91] Rephael Wenger. Extremal graphs with no c4's, c6's, or c10's. *Journal of Combinatorial Theory, Series B*, 52(1):113–116, 1991.

[Woo06] David P Woodruff. Lower bounds for additive spanners, emulators, and more. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 389–398. IEEE, 2006.

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 222–227. IEEE, 1977.