Advanced Algorithms

Computer Science, ETH Zürich

Mohsen Ghaffari

These notes will be updated, throughout the semester. Feedback and comments would be greatly appreciated and should be emailed to ghaffari@inf.ethz.ch.

Last update: October 8, 2019

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 r successes in n draws *without replacement* when there are K successful items in total

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

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

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

Theorem (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 (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,*

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

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

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 $\Pr[|X - \mathbb{E}(X)| \leq \epsilon \cdot \mathbb{E}(X)] \geq 1 - \delta$.

Part I

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

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,

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

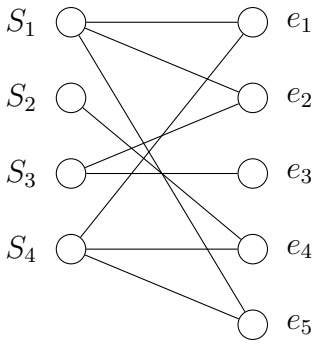
Definition 1.2 (Minimum set cover problem)**.** *Given a universe of elements $\mathcal{U}$, a collection of subsets $\mathcal{S}$, and a non-negative cost function $c : \mathcal{S} \to \mathbb{R}^+$, find a subset $S^* \subseteq \mathcal{S}$ such that:*

(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$(\mathcal{U}, \mathcal{S}, c)$

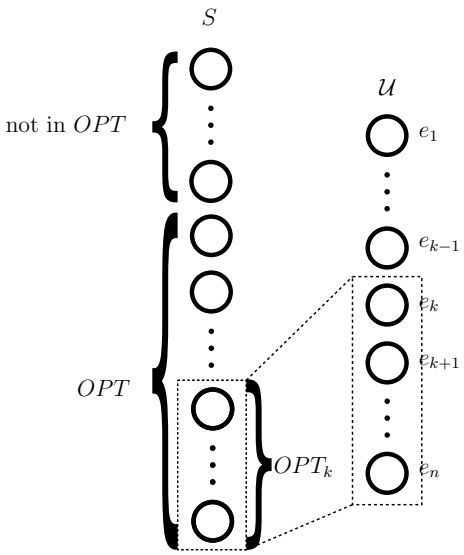
$T \leftarrow \emptyset$ ▷ Selected subset of $\mathcal{S}$
$C \leftarrow \emptyset$ ▷ Covered vertices
while $C \neq \mathcal{U}$ **do**
  $S_i \leftarrow \arg\min_{S_i \in S \setminus T} \frac{c(S_i)}{|S_i \setminus C|}$ ▷ Pick set with lowest price-per-item
  $T \leftarrow T \cup \{S_i\}$ ▷ Add S_i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

[2] $H_n = \sum_{i=1}^{n} \frac{1}{n} = \ln(n) + \gamma \leq \ln(n) + 0.6 \in \mathcal{O}(\log(n))$, 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. 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 $T_k \subset T$. 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 $e_k \ldots e_n$. Suppose $OPT_k = \{O_1, \ldots, O_p\}$ where $O_i \in S \; \forall i \in [p]$. We make the following observations:

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

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 $\mathcal{S} \setminus T$ 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

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

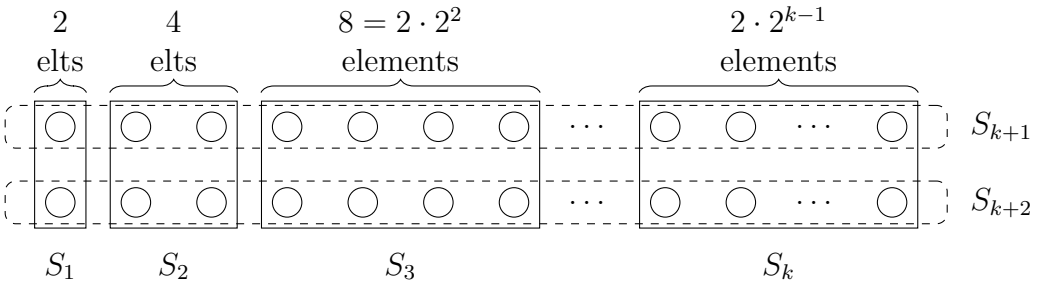
Rearranging, $\texttt{price}(e_k) \leq \frac{c(OPT)}{n-k+1}$. Summing over all elements, we have:

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

□

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}, $\ldots$, 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 $f = \max_{i \in \{1,\ldots,m\}} \texttt{degree}(e_i)$ 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

When $f = 2$ and $c(S_i) = 1, \forall S_i \in \mathcal{S}$, we can reduce minimum vertex cover to 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 set cover as follows:

- Each edge $e_i \in E$ in G becomes an element e_i' in $\mathcal{U}^*$

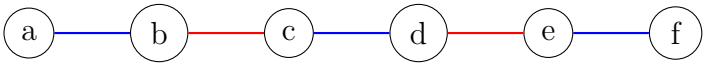
- Each vertex $v_j \in V$ in G becomes an element S_j in $\mathcal{S}^*$ and $e'_i \in S_j \iff e_i$ is adjacent to $v_j \in I$

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 $f = 2$. 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.

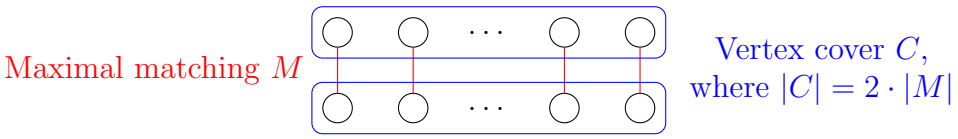
Algorithm 2 GreedyMaximalMatching(V, E)

```
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
```

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.

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 any arbitrary given parameter $\epsilon > 0$, 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. The algorithm should be able to provide a $1+\varepsilon$ approximation — as specified in the item above — for *any* given fixed ϵ. If the algorithm runs in time that is polynomial in the input size and in $1/\epsilon$, we call the algorithm *fully polynomial time approximation schemes* (FPTAS):

Definition 2.2 (Fully polynomial time approximation scheme (FPTAS))**.** *For a given cost metric c, an optimal algorithm OPT and any arbitrary*

given parameter $\epsilon > 0$, 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 us denote the item with the highest profit by $p_{max} = \max_{i \in \{1,\ldots,n\}} \texttt{profit}(i)$. Now let's consider items i with $\texttt{size}(i) > B$. Clearly these items can not be chosen due to the size constraint. Therefore we can remove them and relabel the remaining items, in $\mathcal{O}(n)$ time. Thus without loss of generality we assume $\texttt{size}(i) \leq B, \forall i \in \{1, \ldots, n\}$.

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 $\texttt{size}$ of a subset as the sum of the $\texttt{size}$ 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 $\texttt{size}$ of a subset chosen from $\{1, \ldots, i\}$ such that the total $\texttt{profit}$ equals p. Trivially, set $M[1, \texttt{profit}(1)] = \texttt{size}(1)$ and $M[1, p] = \infty$ for $p \neq \texttt{profit}(1)$. To handle boundaries, define $M[i, j] = \infty$ for $j \leq 0$. Then, for $M[i+1, p]$,

- If $\texttt{profit}(i+1) > p$, then we cannot pick item i.
  So, $M[i+1, p] = M[i, p]$.
- If $\texttt{profit}(i+1) \leq p$, then we pick item i.
  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 by back-tracing from $M[n, np_{max}]$.

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 call this DP algorithm 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 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,

$$\sum_{i=1}^{n} \texttt{loss}(i) \leq nk \qquad \texttt{loss}(i) \leq k \text{ for any item } i$$

$$\leq \epsilon \cdot p_{max} \qquad \text{Since } k = \lfloor \frac{\epsilon}{n} p_{max} \rfloor$$

$$\leq \epsilon \cdot \texttt{profit}(OPT(I)) \qquad \text{Since } p_{max} \leq \texttt{profit}(OPT(I))$$

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.

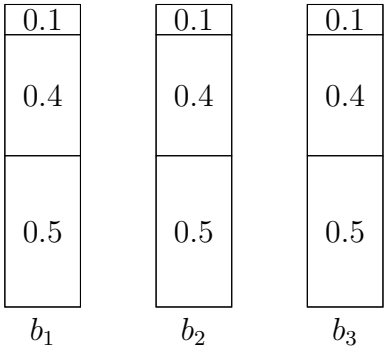
Example Recall the earlier example where budget $B = 10$ and $\mathcal{S} = \{1 : \langle 10, 130\rangle, 2 : \langle 7, 103\rangle, 3 : \langle 6, 91\rangle, 4 : \langle 4, 40\rangle, 5 : \langle 3, 38\rangle\}$. For $\epsilon = \frac{1}{2}$, one would set $k = \max\{1, \lfloor \frac{\epsilon p_{max}}{n} \rfloor\} = \max\{1, \lfloor \frac{130/2}{5} \rfloor\} = 13$. 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.*

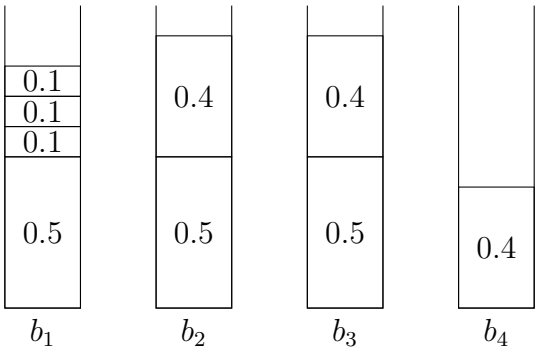
Proof. Suppose FirstFit terminates with $|B| = m$ bins. By Lemma 2.6, $\sum_{i=1}^{n} \texttt{size}(i) > \frac{m-1}{2}$, as $m-1$ bins are at least half-full. Since $\sum_{i=1}^{n} \texttt{size}(i) \leq$

$|OPT(I)|$, 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. □

In the following sections, we work towards a PTAS for bin packing whose runtime will be exponential in $\frac{1}{\epsilon}$. To do this, we first consider two simplifying assumptions and design algorithms for them. Then, we adapt the algorithm to a PTAS by removing the assumptions one at a time.

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).

Let $M = \lceil \frac{1}{\epsilon} \rceil$ and x_i be the number of items of the i^{th} possible size. Let R be the number of weight configurations, or possible item configurations (multiset of item weights) in a bin. By assumption 1, each bin can only contain $\leq M$ items. By assumption 2, $R \leq \binom{M+k}{M}$. Further, the total number of bin configurations is at most $\binom{n+R}{R}$. Since k is a constant, 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 since R is a constant (with respect to constants ϵ and k).

Remark 1 Number of configurations are computed by solving combinatorics problems of the following form: How many non-negative integer solutions are there to $x_1 + \cdots + x_n \leq k$?[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]See slides 22 and 23 of http://www.cs.ucr.edu/~neal/2006/cs260/piyush.pdf for illustration of $\binom{M+k}{M}$ and $\binom{n+R}{R}$.

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

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 group of Q items), let us use separate bins (hence the additive Q term). □

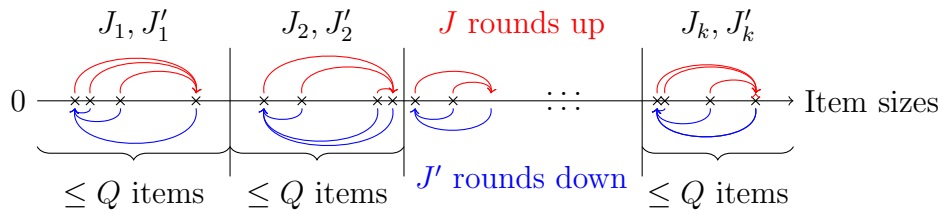


Figure 2.1: Partition items into k groups, each with $\leq Q$ items; Label groups in ascending sizes; J rounds up item sizes, J' rounds down item sizes.

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

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: PTAS

We now consider the general case where we do not make any assumptions on the problem instance I. First, we put aside all items with size smaller than $\min\{\frac{1}{2}, \frac{\epsilon}{2}\}$ and apply PTAS-BinPacking. Then, we add back the small items in a greedy manner with FirstFit to complete the packing.

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'

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

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 $I = \{p_1, \ldots, p_n\}$, where job i takes p_i units of time to complete, find an assignment of jobs to m identical machines such that the completion time (i.e. makespan) is minimized.*

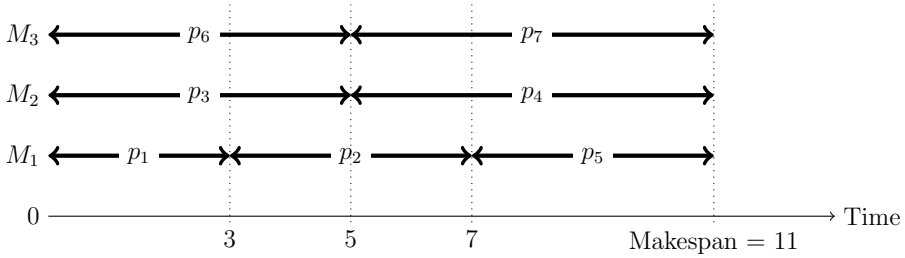
For any problem instance I, let $OPT(I)$ be an optimal job assignment and $|OPT(I)|$ be the corresponding makespan. One can see that:

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

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 $I = \{p_1 = 3,\ p_2 = 4,\ p_3 = 5,\ p_4 = 6,\ p_5 = 4,\ p_6 = 5,\ p_7 = 6\}$ and $m = 3$ machines. Then, the lower bound on the makespan is $L(I) = \max\{6, 11\} = 11$. This is achieveable 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$(I = \{p_1, \ldots, p_n\}, m)$

$M_1, \ldots, M_m \leftarrow \emptyset$ ▷ All machines are initially free
for $i \in \{1, \ldots, n\}$ **do**
  $j \leftarrow \text{argmin}_{j \in \{1,\ldots,m\}} \sum_{p \in M_j} p$ ▷ Pick the least loaded machine
  $M_j \leftarrow M_j \cup \{p_i\}$ ▷ Add job i to this machine
end for
return $M_1, \ldots, M_m$

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, so $t \cdot m \leq \sum_{i=1}^{n} p_i \leq m \cdot |OPT(I)|$. Since $p_{\text{last}} \leq p_{max} \leq |OPT(I)|$, we have $|\text{Graham}(I)| = t + p_{\text{last}} \leq 2 \cdot |OPT(I)|$. □

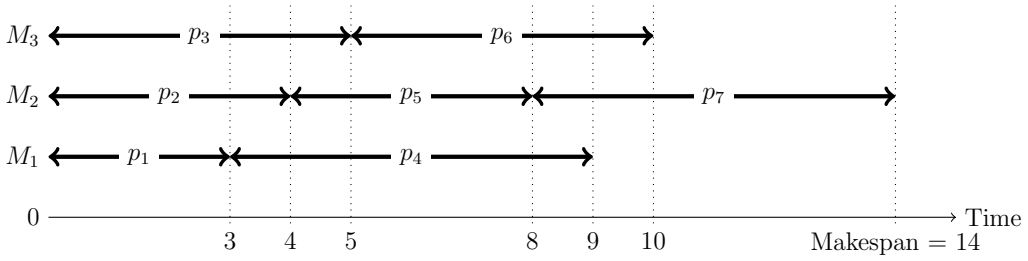
Corollary 2.16. $|OPT(I)| \leq 2 \cdot L(I)$, *where* $L(I) = \max\{p_{max}, \frac{1}{m} \sum_{i=1}^{n} p_i\}$.

Proof. From 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* $|\text{MODIFIEDGRAHAM}(I)| = |OPT(I)|$.

Proof. For $m \geq n$, $|\text{MODIFIEDGRAHAM}(I)| = |OPT(I)|$ 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 $|\text{MODIFIEDGRAHAM}(I)| > |OPT(I)|$. 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 MODIFIEDGRAHAM(I) to have a makespan larger than $OPT(I)$[4]. That is, $|\text{MODIFIEDGRAHAM}(I \setminus \{p_n\})| \leq |OPT(I)|$ and $p_{\text{last}} = p_n$. Let $\mathcal{C}$ be the configuration of machines after MODIFIEDGRAHAM assigned $\{p_1, \ldots, p_{n-1}\}$.

[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 $|\text{MODIFIEDGRAHAM}(\{p_1, \ldots, p_j\})| > |OPT(I)|$, 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.

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.

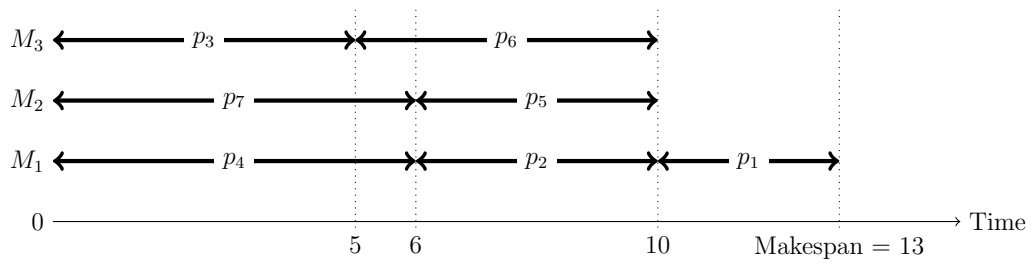
By assumption on p_n, we know that assigning p_n to any machine (in particular, the heavy machines) in $\mathcal{C}$ causes the makespan to exceed $|OPT(I)|$. Since p_n is the smallest job, no other job can be assigned to the heavy machines otherwise $|OPT(I)|$ cannot attained by $OPT(I)$.

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. □

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, $13 = |\textsc{ModifiedGraham}(I)| \leq \frac{4}{3} \cdot |OPT(I)| = 14.666\ldots$



2.3.2 PTAS for minimum makespan scheduling

Recall that any makespan scheduling instance (I, m) has a lower bound $L(I) = \max\{p_{max}, \frac{1}{m}\sum_{i=1}^{n} p_i\}$. 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 times with 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 $\text{Bin}(I, t) \leq m$.

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

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 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 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 additional $(1 + \epsilon)$ buffer. 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 non-small items) 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 $\mathcal{O}(\frac{1}{\epsilon})$ values of t to try. Filtering small jobs and rounding remaining jobs take $\mathcal{O}(n)$. From 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* $\text{poly}(|I|, \frac{1}{\epsilon})$

Remark The fraction $\frac{3}{4}$ in the definition of FPRAS is arbitrary. In fact, any fraction $\frac{1}{2} + \alpha$ for $\alpha > 0$ suffices. For any $\delta > 0$, one can invoke $\mathcal{O}(\frac{1}{\delta})$ independent copies of $\mathcal{A}(I)$ then return the median. Then, the median is a correct estimation with probability greater than $\geq 1 - \delta$. This is also sometimes known as *probability amplification*.

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 DNF if*

- *$F = C_1 \vee \cdots \vee C_m$ is a disjunction of clauses*
- *$\forall i \in [m]$, a clause $C_i = l_{i,1} \wedge \cdots \wedge l_{i,|C_i|}$ is a conjunction 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 on 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, 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.

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 $f(F) = |\bigcup_{i=1}^{m} S_i|$. 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, $\frac{k}{2^n}$ can be exponentially small. This means that we 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

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

Let $|M|$ denote the total number of 1's in M. Since $|S_i| = 2^{n-|C_i|}$, $|M| = \sum_{i=1}^{m} |S_i| = \sum_{i=1}^{m} 2^{n-|C_i|}$. As every column represents a satisfying assignment, there are exactly $f(F)$ "topmost" 1's.

	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: Red 1's indicate the ("topmost") smallest index clause C_i satisfied for each assignment α_j

DNF-Count samples 1's in M by picking clauses according to their length (shorter clauses are more likely), then uniformly selecting a satisfying assignment for a picked clause by flipping coins for variables not in it. DNF-Count then returns the fraction of "topmost" 1's as an estimate of $f(F)$.

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 $|M| = \sum_{i=1}^{m} |S_i| = \sum_{i=1}^{m} 2^{n-|C_i|}$.

Algorithm 10 DNF-Count(F, ϵ)

$X \leftarrow 0$ ▷ Empirical number of "topmost" 1's sampled
for $k = \frac{9m}{\epsilon^2}$ times **do**
  $C_i \leftarrow$ Sample one of m clauses, where $\Pr[C_i \text{ chosen}] = \frac{2^{n-|C_i|}}{|M|}$
  $\alpha_j \leftarrow$ Sample one of $2^{n-|C_i|}$ satisfying assignments of C_i
  IsTopmost $\leftarrow$ True
  for $l \in \{1, \ldots, i-1\}$ **do** ▷ Check if α_j is "topmost"
    if $C_l[\alpha] = 1$ **then** ▷ Checkable in $\mathcal{O}(n)$ time
      IsTopmost $\leftarrow$ False
    end if
  end for
  if IsTopmost **then**
    $X \leftarrow X + 1$
  end if
end for
return $\frac{|M| \cdot X}{k}$

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

□

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

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 $k = \frac{9m}{\epsilon^2}$,

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

Negating, we get:

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

□

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. □

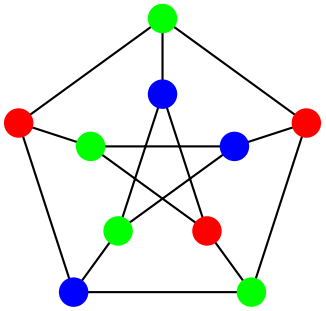
Theorem 3.6. DNF-Count *is a FPRAS for DNF counting.*

Proof. By Lemmas 3.4 and 3.5. □

3.2 Counting graph colorings

Definition 3.7 (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)



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.

3.2.1 Sampling a coloring uniformly

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

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

```
Greedily color the graph
for k = O(n log n/ε) 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 3.8. *For $q \geq 2\Delta + 1$, the distribution of colorings returned by* SampleColor *is ϵ-close to a uniform distribution on all valid colorings.*

Proof. Beyond the scope of this course. □

3.2.2 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 $r_i = \frac{|\Omega_i|}{|\Omega_{i-1}|}$.

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 $\Omega_{i-1} \backslash \Omega_i = \{c : c(u) = c(v)\}$. 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,

$$|\Omega_i| \geq (\Delta + 1)|\Omega_{i-1} \setminus \Omega_i| \geq (\Delta + 1)(|\Omega_{i-1}| - |\Omega_i|)$$

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

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 $q^n \Pi_{i=1}^m \widehat{r_i}$

Lemma 3.9. *For all* $i \in \{1, \ldots, m\}$, $\Pr[|\widehat{r_i} - r_i| \leq \frac{\epsilon}{2m} \cdot r_i] \geq \frac{3}{4m}$.

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 $k = \frac{128m^3}{\epsilon^2}$,

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

Dividing by k and negating, we have:

$$\Pr[|\widehat{r_i} - r_i| \leq \frac{\epsilon}{2m} \cdot r_i] = 1 - \Pr[|X - kp| \geq \frac{\epsilon}{2m} kp] \geq 1 - \frac{1}{4m} = \frac{3}{4m}$$

□

Lemma 3.10. 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 3.11. 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 3.10, 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[1] 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 3.9, $\Pr[|\widehat{r_i} - r_i| \leq \frac{\epsilon}{2m} \cdot r_i] \geq 1 - \frac{1}{4m}$ for any estimate r_i. Therefore,

$$\begin{aligned}
\Pr[|q^n \Pi_{i=1}^m \widehat{r_i} - f(G)| \leq \epsilon f(G)] &= \Pr[|q^n \Pi_{i=1}^m \widehat{r_i} - f(G)| \leq \epsilon f(G)] \\
&\geq (1 - \frac{1}{4m})^m \\
&\geq \frac{3}{4}
\end{aligned}$$

□

Remark Recall from Claim 3.8 that SampleColor actually gives an approximate uniform coloring. A more careful analysis can absorb the approximation of SampleColor under Color-Count's ϵ factor.

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

Chapter 4

Rounding ILPs

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.

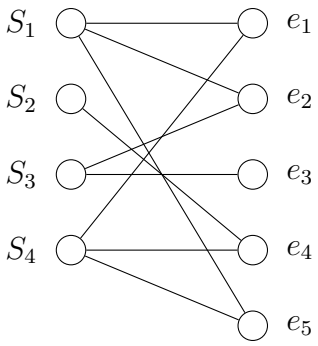
Remark We can define LPs and ILPs for maximization problems similarly. One can also solve maximization problems with a minimization LPs using the same constraints but negated objective function. The optimal value from the solved LP will then be the negation of the maximized optimal value.

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 $\Delta = \max_{i \in \{1,\ldots,m\}} \text{degree}(S_i)$ and $f = \max_{i \in \{1,\ldots,n\}} \text{degree}(x_i)$ 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:

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

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

Upon solving $\text{ILP}_{\text{Set cover}}$, the set $\{S_i : i \in \{1, \ldots, n\} \wedge y_i^* = 1\}$ 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:

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

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

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:

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

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, $y_i \leq f \cdot x_i^*, \forall i \in \{1, \ldots, m\}$. 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 $j \in \{1, \ldots, n\}$, item e_j is not covered w.p. $\leq e^{-1}$.*

Proof. For any $j \in \{1, \ldots, n\}$,

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

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 ∈ {1, ..., 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 $j \in \{1, \ldots, n\}$,

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

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 covers all n items 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 $i \in [k]$,

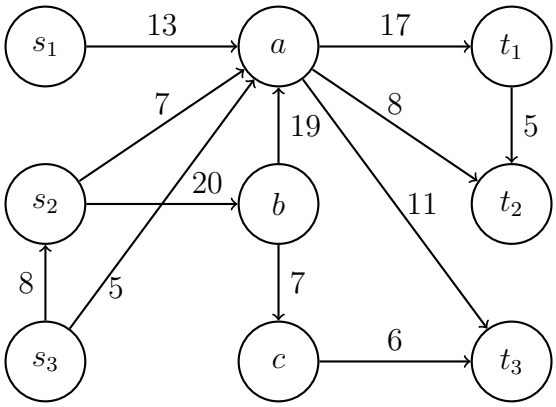
$$\sum_{e \in out(v)} f(e, i) - \sum_{e \in in(v)} f(e, i) = 0, \quad \forall e \in E, \forall v \in V \setminus \{s_i \cup t_i\}$$

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

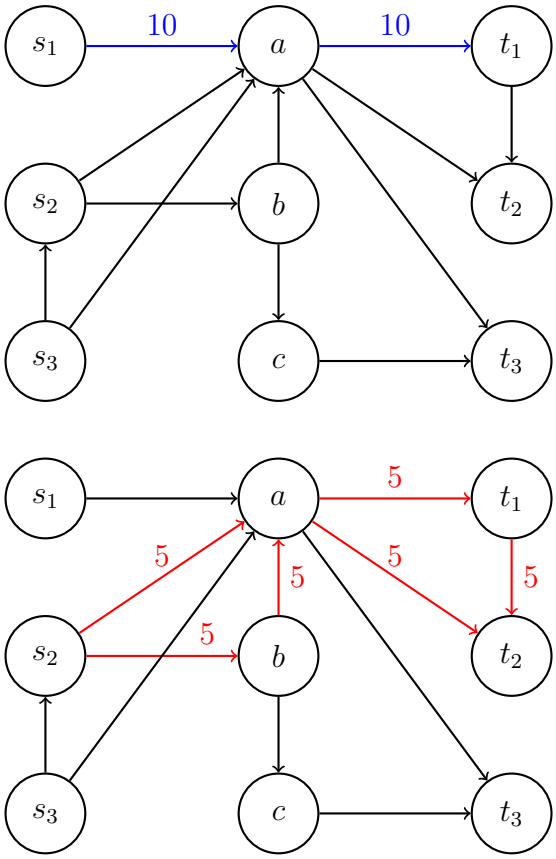
(v) (Congestion factor): $\forall e \in E, \sum_{i=1}^{k} d_i \mathbb{1}_{e \in p_i} \leq \lambda \cdot c(e)$*, where indicator* $\mathbb{1}_{e \in p_i} = 1 \iff e \in p_i$.

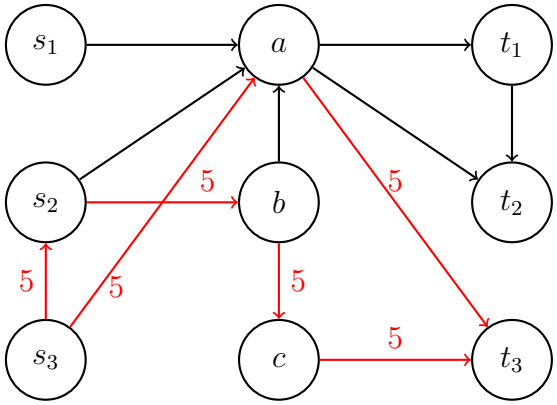
(vi) (Minimum congestion): λ is minimized.

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

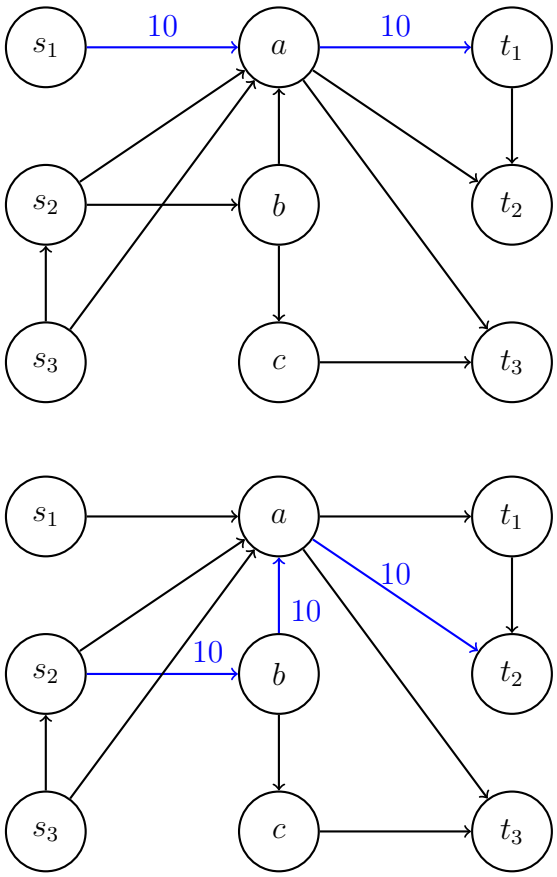


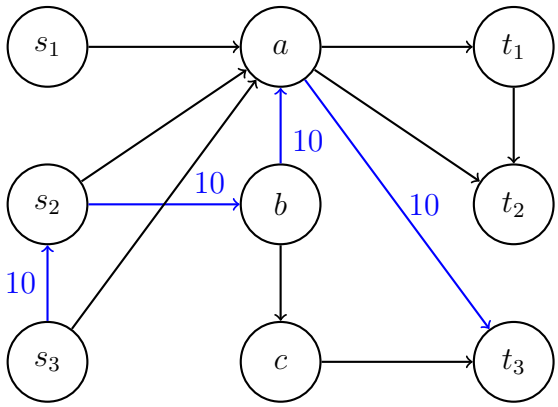
For demands $d_1 = d_2 = d_3 = 10$, there exists a flow assignment such that the total demands flowing on each edge is below its capacity:





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:

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

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

◁ **(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$_{\text{MCR-Given-Network}}$

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

◁ **(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 $\mathbb{E}(obj(\boldsymbol{y})) \leq obj(\boldsymbol{x}^*) \leq obj(\boldsymbol{y}^*)$.

Chapter 5

Probabilistic tree embedding

Trees are a special kind of graphs without cycles and some $\mathcal{NP}$-hard problems are known to admit exact polynomial time solutions on trees. Motivated by existence of efficient algorithms on trees, 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 consider a *probabilistic tree embedding* of G into a collection of trees $\mathcal{T}$ such 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): $\forall u, v \in V, \mathbb{E}_{T \in \mathcal{T}}[d_T(u, v)] \leq c \cdot d_G(u, v)$
- ($\mathcal{T}$ is a probability space): $\sum_{T \in \mathcal{T}} \Pr[T] = 1$

Bartal [Bar96] gave a construction[2] for probabilistic tree embedding with poly-logarithmic stretch factor c, and proved[3] that a stretch factor $c \in$

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

[2] Theorem 8 in [Bar96]

[3] Theorem 9 in [Bar96]

$\Omega(\log n)$ is required for general graphs. A construction that yields $c \in \mathcal{O}(\log n)$, in expectation, was subsequently found by [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. Denote $B(v, r) := \{u \in V : d_G(u, v) \leq r\}$ as the ball of distance r around vertex v, including v.

5.1.1 Idea: Ball carving

Before we present the actual construction, we argue that the following *ball carving* approach will yield a probabilistic tree embedding.

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

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 5.1 illustrates the process of building a tree T from a given graph G.

Lemma 5.2. *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 $D = diam(V)$.*

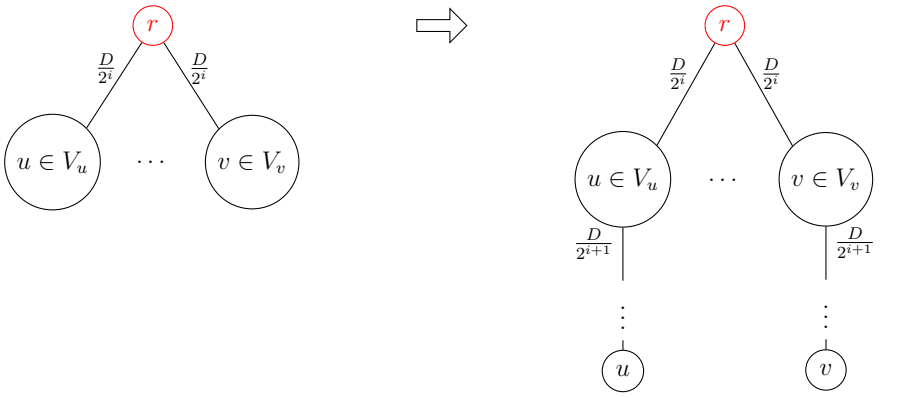
Proof. If T splits u and v at level i, then the path from u to v in T has to include two edges of length $\frac{D}{2^i}$, hence $d_T(u, v) \geq \frac{2D}{2^i}$. To be precise,

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

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

Algorithm 14 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
  $V_1, \ldots, V_l \leftarrow$ 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 $i \in \{1, \ldots, l\}$ **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



□

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 5.3. 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 5.2, $d_T(u,v) \geq \frac{2D}{2^i} = \frac{D}{2^{i-1}} \geq d_G(u,v)$. □

Claim 5.4. ConstructT$(G, C = V, D = diam(V))$ *returns a tree* T *such that*

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

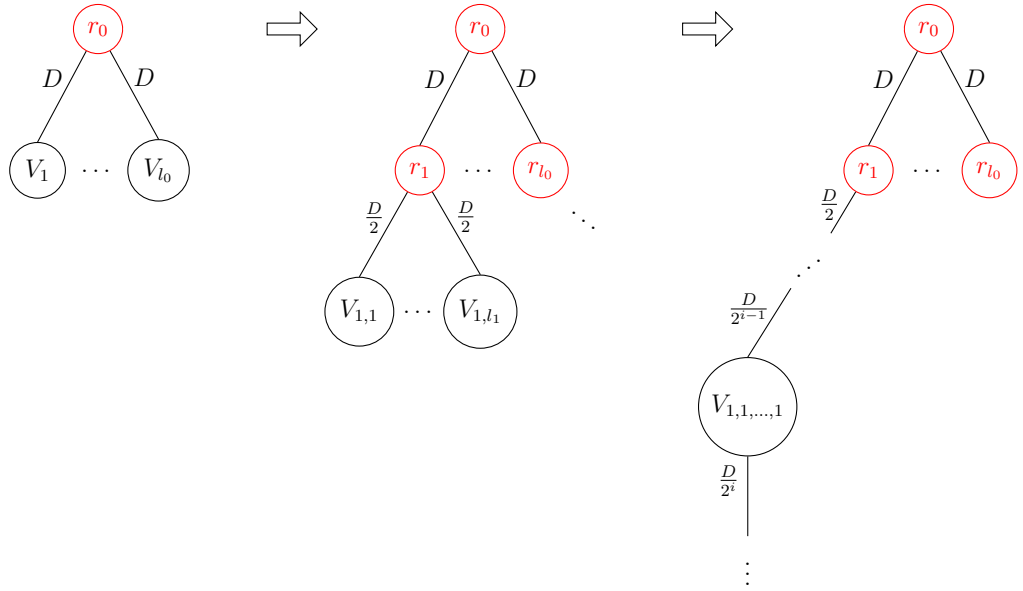


Figure 5.1: Recursive ball carving with $\lceil \log_2(D) \rceil$ levels. Red vertices are auxiliary nodes that are not in the original graph G. Denoting the root as the 0^{th} level, edges from level i to level $i+1$ have weight $\frac{D}{2^i}$.

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,

$$
\begin{aligned}
\mathbb{E}[d_T(u,v)] &= \sum_{i=0}^{\log(D)-1} \Pr[\mathcal{E}_i] \cdot [d_T(u,v), \text{ given } \mathcal{E}_i] && \text{Definition of expectation} \\
&\leq \sum_{i=0}^{\log(D)-1} \Pr[\mathcal{E}_i] \cdot \frac{4D}{2^i} && \text{By Lemma 5.2} \\
&\leq \sum_{i=0}^{\log(D)-1} (\alpha \cdot \frac{d_G(u,v)}{D/2^i}) \cdot \frac{4D}{2^i} && \text{Property (B) of ball carving} \\
&= 4\alpha \log(D) \cdot d_G(u,v) && \text{Simplifying}
\end{aligned}
$$

□

5.1.2 Ball carving construction

We now give a concrete construction of ball carving that satisfies properties (A) and (B) as defined.

Algorithm 15 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 $i \in \{1, \ldots, n\}$ **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, $v = \pi_{\pi(v)}$.

Claim 5.5. 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 $\leq \frac{D}{2}$. □

Definition 5.6 (Ball cut)**.** *A ball* $B(u, r)$ *is cut if* BallCarving *puts the vertices in* $B(u, r)$ *in different partitions of* $V_1, \ldots, V_l$*. We say* V_i *cuts* $B(u, r)$ *if there exists* $w, y \in B(u, r)$ *such that* $w \in V_i$ *and* $y \notin V_i$*.*

Lemma 5.7. *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 in BallCarving. Consider an ordering of vertices in increasing distance from u: $v_1, v_2, \ldots, v_n$, such that $d_G(u, v_1) \leq d_G(u, v_2) \leq \cdots \leq d_G(u, v_n)$.

Observe that for V_i to be the *first* partition that cuts $B(u, r)$, a *necessary condition* is for v_i to appear before any v_j (i.e. $\pi(v_i) < \pi(v_j), \forall 1 \leq j < i$). Consider the largest $1 \leq j < i$ such that $\pi(v_j) < \pi(v_i)$:

- If $B(u,r) \cap B(v_j,r) = \emptyset$, then $B(u,r) \cap B(v_i,r) = \emptyset$ since $d_G(u,v_j) \leq d_G(u,v_i)$.
- If $B(u,r) \subseteq B(v_j,r)$, then vertices in $B(u,r)$ would have been removed before v_i is considered.
- If both $B(u,r) \cap B(v_j,r) = \emptyset$, $B(u,r) \cap B(v_i,r) = \emptyset$ and $B(u,r) \not\subseteq B(v_j,r)$, 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 $\pi(v_j) < \pi(v_i)$, then V_i does not cut $B(u,r)$. So,

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

In the last inequality: For V_i to cut $B(u,r)$, we need $\theta \in (d_G(u,v_i) - r, d_G(u,v_i) + r)$, hence the numerator of $\leq 2r$; The denominator $\frac{D}{8}$ is because the range of values that θ is sampled from is $\frac{D}{4} - \frac{D}{8} = \frac{D}{8}$. □

Claim 5.8. 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 $B(u,r)$.

$$
\begin{aligned}
&\Pr[u \text{ and } v \text{ not in same partition}] \\
&\leq \Pr[B(u,r) \text{ is cut in BallCarving}] \\
&\leq \mathcal{O}(\log n) \cdot \frac{r}{D} && \text{By Lemma 5.7} \\
&= \mathcal{O}(\log n) \cdot \frac{d_G(u,v)}{D} && \text{Since } r = d_G(u,v)
\end{aligned}
$$

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

If we apply Claim 5.8 with Claim 5.4, 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. For details, see Theorem 5.16 in Section 5.3.

5.1.3 Contraction of T

Notice in Figure 5.1 that we introduce auxiliary vertices in our tree construction and wonder if we can build a T without additional vertices (i.e. $V(T) = V(G)$. In this section, we look at Contract which performs *tree contractions* to remove the auxiliary vertices. It remains to show that the produced tree that 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 5.9. 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 5.9 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 5.10 (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.

Let us denote $I = (G, f, \{s_i, t_i, d_i\}_{i=1}^k)$ as the given instance. Let $OPT_G(I)$ be the optimal solution on G and $A_T(I)$ be the solution produced by NetworkDesign. Denote the costs as $|OPT_G(I)|$ and $|A_T(I)|$ respectively. We now compare the solutions $OPT_G(I)$ and $A_T(I)$ by comparing edge costs $(u,v) \in E$ in G and tree embedding T.

Claim 5.11. $|A_T(I)|$ *using edges in* $G \leq |A_T(I)|$ *using edges in* T.

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

$c_e = 0, \forall e \in E$ ▷ Initialize capacities
$T \leftarrow$ CONSTRUCTT(G) ▷ Build probabilistic tree embedding T of G
$T \leftarrow$ CONTRACT(T) ▷ $V(T) = V(G)$ after contraction
for $i \in \{1, \ldots, k\}$ **do** ▷ Solve problem on T
  $P^T_{s_i,t_i} \leftarrow$ 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**
    $P^G_{u,v} \leftarrow$ Find shortest $u - v$ path in G
    $c_e \leftarrow c_e + d_i$, for each edge in $e \in P^G_{u,v}$
  end for
end for
return $\{e \in E : c_e\}$

Proof. (Sketch) For any pair of vertices $u, v \in V$, $d_G(u, v) \leq d_T(u, v)$. □

Claim 5.12. *$|A_T(I)|$ using edges in $T \leq |OPT_G(I)|$ using edges in T.*

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

Claim 5.13. *$\mathbb{E}[|OPT_G(I)|$ using edges in $T] \leq \mathcal{O}(\log n) \cdot |OPT_G(I)|$*

Proof. (Sketch) T stretches edges by at most a factor of $\mathcal{O}(\log n)$. □

By the three claims above, NETWORKDESIGN gives a $\mathcal{O}(\log n)$-approximation to the buy-at-bulk network design problem, in expectation. For details, refer to Section 8.6 in [WS11].

5.3 Extra: Ball carving with $\mathcal{O}(\log n)$ stretch factor

If we apply Claim 5.8 with Claim 5.4, 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.

5.3.1 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 5.14. *For $i \in \mathbb{N}$, if $r > \frac{D}{16}$, then* $\Pr[B(u,r)$ *is cut*$] \leq \frac{16r}{D}$

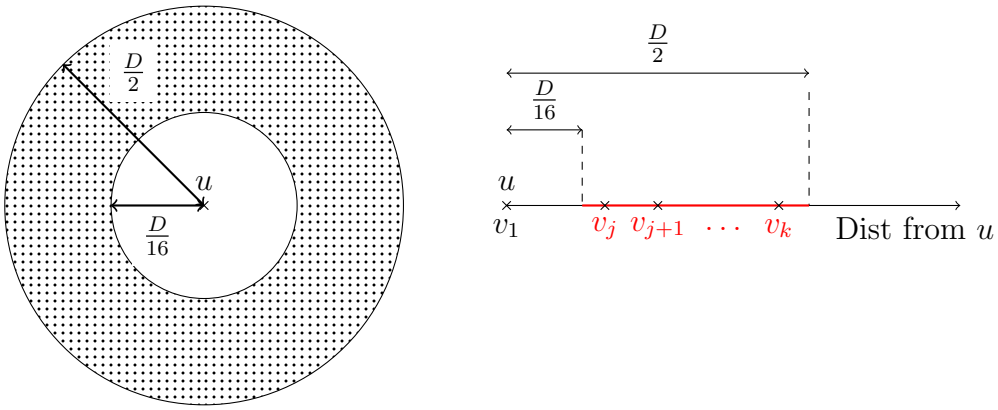
Proof. If $r > \frac{D}{16}$, then $\frac{16r}{D} > 1$. As $\Pr[B(u,r)$ is cut at level $i]$ is a probability ≤ 1, the claim holds. □

Remark Although lemma 5.14 is not a very useful inequality per se (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 5.15. *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. Since $B(v_i,\theta)$ 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,

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

□

5.3.2 Plugging into ConstructT

Recall that ConstructT is a recursive algorithm which handles graphs of diameter $\leq \frac{D}{2^i}$ at each level. 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 5.15 applies for levels $i \in [0, i^* - 5]$ and lemma 5.14 applies for levels $i \in [i^* - 4, \log(D) - 1]$.

Theorem 5.16. $\mathbb{E}[d_T(u,v)] \in \mathcal{O}(\log n) \cdot d_G(u,v)$

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 $\Pr[\mathcal{E}_i] \leq \Pr[B(u,r) \text{ is cut at level } i]$.

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

(1) Definition of expectation

(2) By Lemma 5.2

(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 5.15

(5) By Lemma 5.14 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 $|B(u, \infty)| \leq n$

□

Part II

Streaming and sketching algorithms

Chapter 6

Warm up

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.

6.1 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.

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

[1] Usually this is constant time so we ignore the runtime.

[2] In general, the length of the stream, m, may not be known.

probability $p > 0.5$, the probability that more than half of them fails (and hence the median fails) drops exponential 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 18 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
```

6.2 Majority element

Definition 6.1 ("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.*

Algorithm 19 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
```

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 6.2. 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*. □

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 7

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.

7.1 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 20 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 7.1. $\mathbb{E}[2^{X_m} - 1] = m$. *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 7.2. $\mathbb{E}[2^{2X_m}] = \frac{3}{2}m^2 + \frac{3}{2}m + 1$

Proof. Exercise. □

Claim 7.3. $\mathbb{E}[(2^{X_m} - 1 - m)^2] \leq \frac{m^2}{2}$

Proof. Exercise. Use the Claim 7.2. □

Theorem 7.4. *For* $\epsilon > 0$, $\Pr[|(2^{X_m} - 1) - m| > \epsilon m] \leq \frac{1}{2\epsilon^2}$

Proof.

$$\begin{aligned}
\Pr[|(2^{X_m}-1)-m| > \epsilon m] &= \Pr[((2^{X_m}-1)-m)^2 > (\epsilon m)^2] && \text{Square both sides} \\
&\leq \frac{\mathbb{E}[((2^{X_m}-1)-m)^2]}{(\epsilon m)^2} && \text{Markov's inequality} \\
&\leq \frac{m^2/2}{\epsilon^2 m^2} && \text{By Claim 7.3} \\
&= \frac{1}{2\epsilon^2}
\end{aligned}$$

□

Remark Using the discussion in Section 6.1, 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 $\widehat{m}$, $\Pr[|\widehat{m} - m| > \epsilon m] \leq \frac{1}{20}$.

7.2 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.

7.2.1 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

7.2.2 An actual algorithm

Instead of a uniformly random hash function, we select a random hash from a family of pairwise independent hash functions.

Definition 7.5 (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 7.6. *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 a, b,

- 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 (x, y), 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.

Lemma 7.7. *If $X_1, \ldots, X_n$ are pairwise independent indicator random variables and $X = \sum_{i=1}^{n} X_i$, then $Var(X) \leq \mathbb{E}[X]$.*

Algorithm 21 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$(h(a_i))$ = # trailing zeroes in binary representation of $h(a_i)$)
end for
return $2^Z \cdot \sqrt{2}$ ▷ Estimate of D

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 7.8. *There exists a constant $C > 0$ such that*

$$\Pr[\frac{D}{3} \leq 2^Z \cdot \sqrt{2} \leq 3D] > C$$

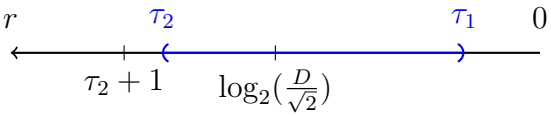
Proof. We will prove $\Pr[(\frac{D}{3} > 2^Z \cdot \sqrt{2}) \text{ or } (2^Z \cdot \sqrt{2} > 3D)] \leq 1 - C$ by separately analyzing $\Pr[\frac{D}{3} \geq 2^Z \cdot \sqrt{2}]$ and $\Pr[2^Z \cdot \sqrt{2} \geq 3D]$, then applying union bound. Define indicator variables

$$X_{i,r} = \begin{cases} 1 & \text{if ZEROS}(h(a_i)) \geq r \\ 0 & \text{otherwise} \end{cases}$$

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 ZEROS$(h(a_i)) \geq r + 1 \Rightarrow$ 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 $2^Z \cdot \sqrt{2} \leq 2^{\tau_2} \cdot \sqrt{2} < \frac{D}{3}$

$$
\begin{aligned}
\Pr[Z \geq \tau_1] &\leq \Pr[X_{\tau_1} \geq 1] && \text{Since } Z \geq \tau_1 \Rightarrow X_{\tau_1} \geq 1 \\
&\leq \frac{\mathbb{E}[X_{\tau_1}]}{1} && \text{By Markov's inequality} \\
&= \frac{D}{2^{\tau_1}} && \text{Since } \mathbb{E}[X_r] = \frac{D}{2^r} \\
&\leq \frac{\sqrt{2}}{3} && \text{Since } 2^{\tau_1} \cdot \sqrt{2} > 3D
\end{aligned}
$$

$$
\begin{aligned}
\Pr[Z \leq \tau_2] &\leq \Pr[X_{\tau_2+1} = 0] && \text{Since } Z \leq \tau_2 \Rightarrow X_{\tau_2+1} = 0 \\
&\leq \Pr[\mathbb{E}[X_{\tau_2+1}] - X_{\tau_2+1} \geq \mathbb{E}[X_{\tau_2+1}]] && \text{Implied} \\
&\leq \Pr[|X_{\tau_2+1} - \mathbb{E}[X_{\tau_2+1}]| \geq \mathbb{E}[X_{\tau_2+1}]] && \text{Adding absolute sign} \\
&\leq \frac{\mathrm{Var}[X_{\tau_2+1}]}{(\mathbb{E}[X_{\tau_2+1}])^2} && \text{By Chebyshev's inequality} \\
&\leq \frac{\mathbb{E}[X_{\tau_2+1}]}{(\mathbb{E}[X_{\tau_2+1}])^2} && \text{By Lemma 7.7} \\
&\leq \frac{2^{\tau_2+1}}{D} && \text{Since } \mathbb{E}[X_r] = \frac{D}{2^r} \\
&\leq \frac{\sqrt{2}}{3} && \text{Since } 2^{\tau_2} \cdot \sqrt{2} < \frac{D}{3}
\end{aligned}
$$

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 $C > 0.5$, 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 22 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 7.9. 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 7.9. *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 7.7, $\mathrm{Var}(X) \leq \mathbb{E}[X]$. Now,

$$
\begin{aligned}
\Pr[\frac{tN}{Z} > (1+\epsilon)D] &\leq \Pr[X \geq t] && \text{Since the former implies the latter} \\
&= \Pr[X - \mathbb{E}[X] \geq t - \mathbb{E}[X]] && \text{Subtracting } \mathbb{E}[X] \text{ from both sides} \\
&\leq \Pr[X - \mathbb{E}[X] \geq \frac{\epsilon}{2}t] && \text{Since } \mathbb{E}[X] = \frac{t}{(1+\epsilon)} \leq (1 - \frac{\epsilon}{2})t \\
&\leq \Pr[|X - \mathbb{E}[X]| \geq \frac{\epsilon}{2}t] && \text{Adding absolute sign} \\
&\leq \frac{\mathrm{Var}(X)}{(\epsilon t/2)^2} && \text{By Chebyshev's inequality} \\
&\leq \frac{\mathbb{E}[X]}{(\epsilon t/2)^2} && \text{Since } \mathrm{Var}(X) \leq \mathbb{E}[X] \\
&\leq \frac{4(1-\epsilon/2)t}{\epsilon^2 t^2} && \text{Since } \mathbb{E}[X] = \frac{t}{(1+\epsilon)} \leq (1 - \frac{\epsilon}{2})t \\
&\leq \frac{4}{c} && \text{Simplifying with } t = \frac{c}{\epsilon^2} \text{ and } (1 - \frac{\epsilon}{2}) < 1
\end{aligned}
$$

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 7.7, $\mathrm{Var}(Y) \leq \mathbb{E}[Y]$. 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,

$$
\begin{aligned}
\Pr[|\frac{tN}{Z} - D| > \epsilon D]] &\leq \Pr[\frac{tN}{Z} > (1+\epsilon)D]] + \Pr[\frac{tN}{Z} < (1-\epsilon)D]] && \text{By union bound} \\
&\leq 4/c + 3/c && \text{From above} \\
&\leq 7/c && \text{Simplifying} \\
&\leq 1/4 && \text{For } c \geq 28
\end{aligned}
$$

□

7.3 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 $F_k = \sum_{j=1}^{n} (f_j)^k$.

7.3.1 $k = 2$

For each element $i \in [n]$, we associate a random variable $r_i \in_{u.a.r.} \{-1, +1\}$.

Lemma 7.10. *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* 2^{nd} *moment.*

Algorithm 23 AMS-2($S = \{a_1, \ldots, a_m\}$)

For each $i \in [n]$, assign $r_i \in_{u.a.r.} \{-1, +1\}$ ▷ For now, this takes $\mathcal{O}(n)$ space
$Z \leftarrow 0$
for $a_i \in S$ **do** ▷ Items arrive in streaming fashion
  $Z \leftarrow Z + r_i$ ▷ At the end, $Z = \sum_{i=1}^{n} r_i f_i$
end for
return Z^2 ▷ Estimate of $F_2 = \sum_{i=1}^{n} (f_i)^2$

Proof.

$$\begin{aligned}
\mathbb{E}[Z^2] &= \mathbb{E}[(\sum_{i=1}^{n} r_i f_i)^2] && \text{Since } Z = \sum_{i=1}^{n} r_i f_i \text{ at the end} \\
&= \mathbb{E}[\sum_{i=1}^{n} r_i^2 f_i^2 + 2 \sum_{1 \le i < j \le n} r_i r_j f_i f_j] && \text{Expanding } (\sum_{i=1}^{n} r_i f_i)^2 \\
&= \sum_{i=1}^{n} \mathbb{E}[r_i^2 f_i^2] + 2 \sum_{1 \le i < j \le n} \mathbb{E}[r_i r_j f_i f_j] && \text{Linearity of expectation} \\
&= \sum_{i=1}^{n} \mathbb{E}[r_i^2] f_i^2 + 2 \sum_{1 \le i < j \le n} \mathbb{E}[r_i r_j] f_i f_j && f_i\text{'s are (unknown) constants} \\
&= \sum_{i=1}^{n} f_i^2 + 2 \sum_{1 \le i < j \le n} \mathbb{E}[r_i r_j] f_i f_j && \text{Since } (r_i)^2 = 1, \forall i \in [n] \\
&= \sum_{i=1}^{n} f_i^2 + 2 \sum_{1 \le i < j \le n} \mathbb{E}[r_i] \mathbb{E}[r_j] f_i f_j && \text{Since } \{r_i\}_{i \in [n]} \text{ are pairwise independent} \\
&= \sum_{i=1}^{n} f_i^2 + 2 \sum_{1 \le i < j \le n} 0 \cdot f_i f_j && \text{Since } \mathbb{E}[r_i] = 0, \forall i \in [n] \\
&= \sum_{i=1}^{n} f_i^2 && \text{Simplifying} \\
&= F_2 && \text{Since } F_2 = \sum_{i=1}^{n} (f_i)^2
\end{aligned}$$

□

Lemma 7.11. *In* AMS-2*, if random variables* $\{r_i\}_{i \in [n]}$ *are 4-wise independent, then* $Var[Z^2] \le 2(\mathbb{E}[Z^2])^2$.

Proof. As before, $\mathbb{E}[r_i] = 0$ and $\mathbb{E}[r_i^2] = 1$ for all $i \in [n]$. By 4-wise independence, the expectation of any product of ≤ 4 different r_i's is the product of their expectation, which is zero. For instance, $\mathbb{E}[r_i r_j r_k r_l] = \mathbb{E}[r_i]\mathbb{E}[r_j]\mathbb{E}[r_k]\mathbb{E}[r_l] = 0$. Note that 4-wise independence implies pairwise independence, $r_i^2 = r_i^4 = 1$ and $r_i = r_i^3$.

$$\begin{aligned}
\mathbb{E}[Z^4] &= \mathbb{E}[(\sum_{i=1}^{n} r_i f_i)^4] && \text{Since } Z = \sum_{i=1}^{n} r_i f_i \text{ at the end} \\
&= \sum_{i=1}^{n} \mathbb{E}[r_i^4] f_i^4 + 6 \sum_{1 \leq i < j \leq n} \mathbb{E}[r_i^2 r_j^2] f_i^2 f_j^2 && \text{L.o.E. and 4-wise independence} \\
&= \sum_{i=1}^{n} f_i^4 + 6 \sum_{1 \leq i < j \leq n} f_i^2 f_j^2 && \text{Since } \mathbb{E}[r_i^4] = \mathbb{E}[r_i^2] = 1, \forall i \in [n]
\end{aligned}$$

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 $\binom{4}{2}\binom{2}{2} = 6$. All other terms besides $\sum_{i=1}^{n} \mathbb{E}[r_i^4] f_i^4$ and $6 \sum_{1 \leq i < j \leq n} \mathbb{E}[r_i^2 r_j^2] f_i^2 f_j^2$ evaluate to 0 because of 4-wise independence.

$$\begin{aligned}
\mathrm{Var}[Z^2] &= \mathbb{E}[(Z^2)^2] - (\mathbb{E}[Z^2])^2 && \text{Definition of variance} \\
&= \sum_{i=1}^{n} f_i^4 + 6 \sum_{1 \leq i < j \leq n} f_i^2 f_j^2 - (\mathbb{E}[Z^2])^2 && \text{From above} \\
&= \sum_{i=1}^{n} f_i^4 + 6 \sum_{1 \leq i < j \leq n} f_i^2 f_j^2 - (\sum_{i=1}^{n} f_i^2)^2 && \text{By Lemma 7.10} \\
&= 4 \sum_{1 \leq i < j \leq n} f_i^2 f_j^2 && \text{Expand and simplify} \\
&\leq 2(\sum_{i=1}^{n} f_i^2)^2 && \text{Upper bound} \\
&= 2(\mathbb{E}[Z^2])^2 && \text{By Lemma 7.10}
\end{aligned}$$

□

Theorem 7.12. *In* AMS-2, *if* $\{r_i\}_{i \in [n]}$ *are 4-wise independent,* $\Pr[|Z^2 - F_2| > \epsilon F_2] \leq \frac{2}{\epsilon^2}$ *for any* $\epsilon > 0$.

Proof.

$$\begin{aligned}
\Pr[|Z^2 - F_2| > \epsilon F_2] &= \Pr[|Z^2 - \mathbb{E}[Z^2]| > \epsilon \mathbb{E}[Z^2]] && \text{By Lemma 7.10} \\
&\leq \frac{\text{Var}(Z^2)}{(\epsilon \mathbb{E}[Z^2])^2} && \text{By Chebyshev's inequality} \\
&\leq \frac{2(\mathbb{E}[Z^2])^2}{(\epsilon \mathbb{E}[Z^2])^2} && \text{By Lemma 7.11} \\
&= && \frac{2}{\epsilon^2}
\end{aligned}$$

□

Claim 7.13. *$\mathcal{O}(k \log n)$ 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[2], 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. □

Observe that the above analysis only require $\{r_i\}_{i\in[n]}$ to be 4-wise independent. Claim 7.13 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 < 0.5, 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 $> 1 - \delta$.

7.3.2 General 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 suffix of the stream.

[2]See https://en.wikipedia.org/wiki/K-independent_hashing

Algorithm 24 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 $F_k = \sum_{i=1}^{n} (f_i)^k$

The assumption of known m in AMS-K can be removed via reservoir sampling[3]. 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 7.14. *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.*

Proof. When $a_J = i$, there are f_i choices for J. By telescoping sums, we have:

$$
\begin{aligned}
&\mathbb{E}[Z \mid a_J = i] \\
&= \frac{1}{f_i}[m(f_i^k - (f_i - 1)^k)] + \frac{1}{f_i}[m((f_i - 1)^k - (f_i - 2)^k)] + \cdots + \frac{1}{f_i}[m(1^k - 0^k)] \\
&= \frac{m}{f_i}[(f_i^k - (f_i - 1)^k) + ((f_i - 1)^k - (f_i - 2)^k) + \cdots + (1^k - 0^k)] \\
&= \frac{m}{f_i} f_i^k
\end{aligned}
$$

[3] See https://en.wikipedia.org/wiki/Reservoir_sampling

$$\begin{aligned}
\mathbb{E}[Z] &= \sum_{i=1}^{n} \mathbb{E}[Z \mid a_J = i] \cdot \Pr[a_J = i] && \text{Condition on the choice of } J \\
&= \sum_{i=1}^{n} \mathbb{E}[Z \mid a_J = i] \cdot \frac{f_i}{m} && \text{Since choice of } J \text{ is uniform at random} \\
&= \sum_{i=1}^{n} \frac{m}{f_i} f_i^k \cdot \frac{f_i}{m} && \text{From above} \\
&= \sum_{i=1}^{n} f_i^k && \text{Simplifying} \\
&= F_k && \text{Since } F_k = \sum_{i=1}^{n} f_i^k
\end{aligned}$$

□

Lemma 7.15. *For every n positive reals $f_1, f_2, \ldots, f_n$,*

$$(\sum_{i=1}^{n} f_i)(\sum_{i=1}^{n} f_i^{2k-1}) \leq n^{1-1/k}(\sum_{i=1}^{k} f_i^k)^2$$

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,

$$\begin{aligned}
(\sum_{i=1}^{n} f_i)(\sum_{i=1}^{n} f_i^{2k-1}) &\leq (\sum_{i=1}^{n} f_i)(M^{k-1} \sum_{i=1}^{n} f_i^k) && \text{Pulling out a } M^{k-1} \text{ factor} \\
&\leq (\sum_{i=1}^{n} f_i)(\sum_{i=1}^{n} f_i^k)^{(k-1)/k}(\sum_{i=1}^{n} f_i^k) && \text{Since } M^k \leq \sum_{i=1}^{n} f_i^k \\
&= (\sum_{i=1}^{n} f_i)(\sum_{i=1}^{n} f_i^k)^{(2k-1)/k} && \text{Merging the last two terms} \\
&\leq n^{1-1/k}(\sum_{i=1}^{n} f_i^k)^{1/k}(\sum_{i=1}^{n} f_i^k)^{(2k-1)/k} && \text{Fact: } (\sum_{i=1}^{n} f_i)/n \leq (\sum_{i=1}^{n} f_i^k/n)^{1/k} \\
&= n^{1-1/k}(\sum_{i=1}^{n} f_i)^2 && \text{Merging the last two terms}
\end{aligned}$$

□

Remark $f_1 = n^{1/k}, f_2 = \cdots = f_n = 1$ is a tight example for Lemma 7.15, up to a constant factor.

Theorem 7.16. *In* AMS-K, $Var(Z) \leq kn^{1-\frac{1}{k}}(\mathbb{E}[Z])^2$

Proof. Let us first analyze $\mathbb{E}[Z^2]$.

$$
\begin{aligned}
&\mathbb{E}[Z^2] \\
&= \frac{m}{m}[\ (1^k - 0^k)^2 + (2^k - 1^k)^2 + \cdots + (f_1^k - (f_1 - 1)^k)^2 && (1)\\
&+\ (1^k - 0^k)^2 + (2^k - 1^k)^2 + \cdots + (f_2^k - (f_2 - 1)^k)^2 \\
&+\ \ldots \\
&+\ (1^k - 0^k)^2 + (2^k - 1^k)^2 + \cdots + (f_n^k - (f_n - 1)^k)^2] \\
&\leq m[\ k \cdot 1^{k-1}(1^k - 0^k) + k \cdot 2^{k-1} \cdot (2^k - 1^k) + \cdots + k \cdot f_1^{k-1} \cdot (f_1^k - (f_1 - 1)^k) && (2)\\
&+\ k \cdot 1^{k-1}(1^k - 0^k) + k \cdot 2^{k-1} \cdot (2^k - 1^k) + \cdots + k \cdot f_2^{k-1} \cdot (f_2^k - (f_2 - 1)^k) \\
&+\ \ldots \\
&+\ k \cdot 1^{k-1}(1^k - 0^k) + k \cdot 2^{k-1} \cdot (2^k - 1^k) + \cdots + k \cdot f_n^{k-1} \cdot (f_n^k - (f_n - 1)^k)] \\
&\leq m[k \cdot f_1^{2k-1} + k \cdot f_2^{2k-1} + \cdots + k \cdot f_n^{2k-1}] && (3)\\
&= k \cdot m \cdot F_{2k-1} && (4)\\
&= k \cdot F_1 \cdot F_{2k-1} && (5)
\end{aligned}
$$

(1) By definition of $\mathbb{E}[Z^2]$ (condition on J and expand in the same style as the proof of Theorem 7.14).

(2) For all $0 < b < a$ and $a = b + 1$,

$$a^k - b^k = (a - b)(a^{k-1} + a^{k-2}b + \cdots + ab^{k-2} + b^{k-1}) \leq (a - b)ka^{k-1}$$

(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,

$$
\begin{aligned}
\mathrm{Var}(Z) &= \mathbb{E}[Z^2] - (\mathbb{E}[Z])^2 && \text{Definition of variance}\\
&\leq \mathbb{E}[Z^2] && \text{Ignore negative part}\\
&\leq k \cdot F_1 \cdot F_{2k-1} && \text{From above}\\
&\leq kn^{1-1/k}F_k^2 && \text{By Lemma 7.15}\\
&= kn^{1-1/k}(\mathbb{E}[Z])^2 && \text{By Theorem 7.14}
\end{aligned}
$$

□

Remark Proofs for Lemma 7.15 and Theorem 7.16 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 7.17. *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 8

Graph sketching

Definition 8.1 (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" where existing edges are not added and edge deletions can only occur after additions.*

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 $t \in \mathbb{N}^+$?

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 where vertices in the same connected component has 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 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 $\geq 1 - \frac{1}{n^{10}}$.

[1] Toggle edge additions/deletion per update. Compute connected components on demand.

[2] Use the Union-Find data structure. See https://en.wikipedia.org/wiki/Disjoint-set_data_structure

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.

8.1 Warm up: Finding the single cut

Definition 8.2 (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 on first glance since it already takes $\mathcal{O}(n)$ bits for any vertex to enumerate all 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:

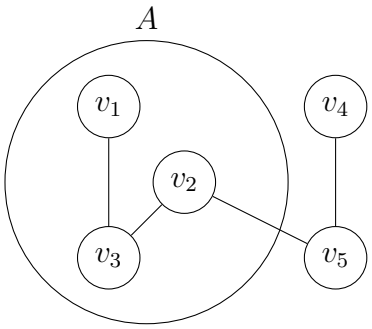
- Identify an edge by the concatenation the identifiers of its endpoints (say, $u \circ v$ if $id(u) < id(v)$)
- Locally, every vertex u maintains $XOR_u = \oplus\{id(e_i) : e_i \in S \wedge e_i \text{ has a endpoint } u\}$
- Vertices send the coordinator their sum and the coordinator computes $XOR_A \oplus \{XOR_u : u \in A\}$

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 $t = 6$:

[3] In reality, the algorithm simulates all the vertices' actions so it is not a real multi-party computation setup.



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 adding and deleting edges both add the edge ID to each vertex's XOR sum, and every edge in A contributes an even number of times to the coordinator's XOR sum.

Claim 8.3. *$XOR_A = \oplus\{XOR_u : u \in A\}$ is the identifier of the cut edge.*

Proof. For any edge (a, b) such that $a, b \in A$, $id((a, b))$ is in both XOR_a and XOR_b. So, $XOR_a \oplus XOR_b$ will cancel out the contribution of $id((a, b))$. Hence, the only remaining value in $XOR_A = \oplus\{XOR_u : u \in A\}$ will be the cut edge since only one endpoint lies in A. □

Remark Bit tricks are often used in the random linear network coding literature (e.g. [HMK+06]).

8.2 Warm up 2: Finding one out of $k > 1$ cut edges

Definition 8.4 (The k cut problem)**.** *Fix an arbitrary subset $A \subseteq V$. Suppose there is exactly k cut edge (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 $\mathcal{O}(\log n)$ 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'.

$$
\begin{aligned}
&\Pr[|E'| = 1] \\
&= k \cdot \Pr[\text{Cut edge } \{u, v\} \text{ is marked; others are not}] \\
&= k \cdot (1/\widehat{k})(1 - (1/\widehat{k}))^{k-1} && \text{Edges marked ind. w.p. } 1/\widehat{k} \\
&\geq (\widehat{k}/2)(1/\widehat{k})(1 - (1/\widehat{k}))^{\widehat{k}} && \text{Since } \frac{\widehat{k}}{2} \leq k \leq \widehat{k} \\
&\geq \frac{1}{2} \cdot 4^{-1} && \text{Since } 1 - x \geq 4^{-x} \text{ for } x \leq 1/2 \\
&\geq \frac{1}{10}
\end{aligned}
$$

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 done with a sufficiently large string of *shared randomness*. We discuss this in Section 8.3.

From above, we know that $\Pr[|E'| = 1] \geq 1/10$. If $|E'| = 1$, we can re-use the idea from Section 8.1. 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) each edge with 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,

$$
\begin{aligned}
&\Pr[XOR_A = id(e) \text{ for some edge } e \mid |E'| \neq 1] \\
&\leq \sum_{e \in \binom{V}{2}} \Pr[XOR_A = id(e) \mid |E'| \neq 1] && \text{Union bound over all possible edges} \\
&= \binom{n}{2} (\frac{1}{2})^{20 \log n} && \text{There are } \binom{n}{2} \text{ possible edges} \\
&= 2^{-18 \log n} && \text{Since } \binom{n}{2} \leq n^2 = 2^{2 \log n} \\
&= \frac{1}{n^{18}} && \text{Rewriting}
\end{aligned}
$$

Now, we can correctly distinguish $|E'| = 1$ from $|E'| \neq 1$ and $\Pr[|E'| = 1] \geq \frac{1}{10}$. For any given $\epsilon > 0$, there exists a constant $C(\epsilon)$ such that if we repeat $t = C(\epsilon) \log n$ times, the probability that *all* t tries fail to extract a single cut is $(1 - \frac{1}{10})^t \leq \frac{1}{n^{1+\epsilon}}$.

8.3 Maximal forest with $\mathcal{O}(n \log^4 n)$ memory

Recall that Borůvka's algorithm[4] builds a minimum spanning tree by iteratively finding the cheapest edge leaving connected components and adding them into the MST. The number of connected components decreases by at least half per iteration, so it converges in $\mathcal{O}(\log n)$ iterations.

For any arbitrary cut, the number of edge cuts is $k \in [0, n]$. Guessing through $\widehat{k} = 2^0, 2^1, \ldots, 2^{\lceil \log n \rceil}$, one can use Section 8.2 to find a cut edge:

- If $\widehat{k} \ll k$, the marking probability will select nothing (in expectation).
- If $\widehat{k} \gg k$, more than one edge will get marked, which we will then detect (and ignore) since XOR_A will likely not be a valid edge ID.

Using a source of randomness $\mathcal{R}$, every vertex in ComputeSketches maintains $\mathcal{O}(\log^3 n)$ copies of edge XORs using random (but consistent) edge IDs and marking probabilities:

- $\lceil \log n \rceil$ times for Borůvka simulation later
- $\lceil \log n \rceil$ times for guesses of cut size k
- $C(\epsilon) \cdot \log n$ times to amplify success probability of Section 8.2

[4] See https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Algorithm 25 ComputeSketches($S = \{\langle e, \pm\rangle, \dots\}, \epsilon, \mathcal{R}$)

```
for i = 1, ..., n do
    XOR_i ← 0^{(20 log n)*log^3 n}                ▷ Initialize log^3 n copies
end for
for Edge update {⟨e = (u, v), ±⟩} ∈ S do          ▷ Streaming edge updates
    for b = log n times do                          ▷ Simulate Borůvka
        for i ∈ {1, 2, ..., log n} do               ▷ log n guesses of k̂
            for t = C(ε) log n times do             ▷ Amplify success probability
                R_{b,i,t} ← Randomness for this specific instance based on R
                if Edge e is marked w.p. k̂ = 2^i, according to R_{b,i,t} then
                    Compute id(e) using R
                    XOR_u[b, i, t] ← XOR_u[b, i, t] ⊕ id(e)
                    XOR_v[b, i, t] ← XOR_v[b, i, t] ⊕ id(e)
                end if
            end for
        end for
    end for
end for
return XOR_1, ..., XOR_n
```

Then, StreamingMaximalForest simulates Borůvka using the output of ComputeSketches:

- Find an out-going edge from each connected component via Section 8.2
- Join connected components by adding edges to graph

Since each edge ID uses $\mathcal{O}(\log n)$ memory and $\mathcal{O}(\log^3 n)$ copies were maintained per vertex, a total of $\mathcal{O}(n \log^4 n)$ memory suffices. 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 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

$$\Pr[\textit{Any } XOR_A \text{ corresponds wrongly some edge ID}] \leq \mathcal{O}(\frac{\log^3 n}{n^{18}}) \subseteq \mathcal{O}(\frac{1}{n^{10}})$$

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).

Algorithm 26 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 b = log n times do                                   ▷ Simulate Borůvka
    C ← ∅                                                ▷ Initialize candidate edges
    for Every connected component A in F do
        for i ∈ {1, 2, ..., ⌈log n⌉} do                  ▷ Guess A has [2^{i-1}, 2^i] cut edges
            for t = C(ε) log n times do                  ▷ Amplify success probability
                R_{b,i,t} ← Randomness for this specific instance
                XOR_A ← ⊕{XOR_u[b, 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
```

Theorem 8.5. *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 8.6. *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 III

Graph sparsification

Chapter 9

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 9.1 ((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'.

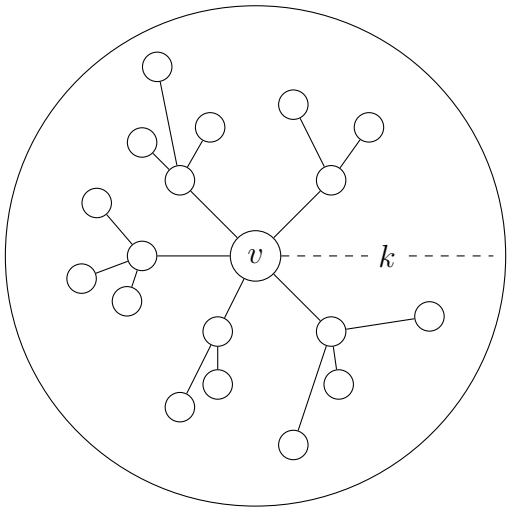
For an (α, β)-spanner, α is called the *multiplicative stretch* of the spanner and β is called the *additive stretch* of the spanner. One would then like to construct spanners with small $|E'|$ and stretch factors. An $(\alpha, 0)$-spanner is called a *α-multiplicative spanner*, and a $(1, \beta)$-spanner 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].

9.1 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 9.2. ?? *[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 **??** will always produce a $(2k-1)$-spanner with $\leq g(n, k)$ edges. The size of the spanner is asymptotically tight if Conjecture **??** holds.

Conjecture 9.3. *[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 is also optimal for weighted graphs [FS16].

Remark 2 The girth conjecture is confirmed for $k \in \{1, 2, 3, 5\}$ [Wen91, Woo06].

9.2 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 9.4. *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 $N(v) = \{u \in V : (u, v) \in E\}$,*
   - $\mathbb{E}[|N(v) \cap S|] = d(v) \cdot p$
   - $\Pr[|N(v) \cap S| = 0] \leq e^{-\frac{d(v) \cdot p}{2}}$

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.

$$\begin{aligned}
\mathbb{E}[|N(v)\cap S|] &= \mathbb{E}[\sum_{v\in N(v)} X_v] && \text{By definition of } N(v)\cap S \\
&= \sum_{v\in N(v)} \mathbb{E}[X_v] && \text{Linearity of expectation} \\
&= \sum_{v\in N(v)} p && \text{Since } \mathbb{E}[X_v] = \Pr[X_v = 1] = p \\
&= d(v)\cdot p && \text{Since } |N(v)| = d(v)
\end{aligned}$$

By one-sided Chernoff bound,

$$\begin{aligned}
\Pr[|N(v)\cap S| = 0] &= \Pr[|N(v)\cap S| \le (1-1)\cdot \mathbb{E}[|N(v)\cap S|]] \\
&\le e^{-\frac{\mathbb{E}[|N(v)\cap S|]}{2}} \\
&= e^{-\frac{d(v)\cdot p}{2}}
\end{aligned}$$

□

Remark $\widetilde{\mathcal{O}}$ hides logarithmic factors. For example, $\mathcal{O}(n\log^{1000} n) \subseteq \widetilde{\mathcal{O}}(n)$.

Theorem 9.5. *[ACIM99] For a fixed $k \ge 1$, 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

$$L = \{v \in V : \deg(v) \le n^{1/2}\} \text{ and } H = \{v \in V : \deg(v) > n^{1/2}\}$$

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

1. Since there are at most n light vertices, $|E'_1| \leq n \cdot n^{1/2} = n^{3/2}$.
2. By Claim 9.4 with $p = 10n^{-1/2} \log n$, $\mathbb{E}[|S|] = n \cdot 10n^{-1/2} \log n = 10n^{1/2} \log n$. Then, since every BFS tree has $n-1$ edges[3], $|E'_2| \leq n \cdot |S|$, thus

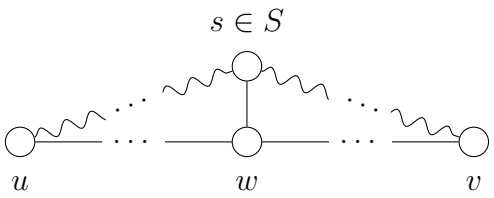
$$\mathbb{E}[|E'|] = \mathbb{E}[|E'_1 \cup E'_2|] \leq \mathbb{E}[|E'_1| + |E'_2|] = \mathbb{E}[|E'_1|] + \mathbb{E}[|E'_2|] \leq n^{3/2} + n \cdot 10n^{1/2} \log n \in \widetilde{\mathcal{O}}(n^{3/2})$$

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 9.6. *Suppose there exists a vertex $w \in P_{u,v}$ such that $(w, s) \in E$ for some $s \in S$, then $d_{G'}(u, v) \leq d_G(u, v) + 2$.*



[3]Though we may have repeated edges

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 u, w, v lie on $P_{u,v}$

□

Let w be a heavy vertex in $P_{u,v}$ with degree $d(w) > n^{1/2}$. By Claim 9.4 with $p = 10n^{-1/2}\log n$, $\Pr[|N(w) \cap S| = 0] \leq e^{-\frac{10\log n}{2}} = n^{-5}$. Taking union bound over all possible pairs of vertices u and v,

$$\Pr[\exists u, v \in V, P_{u,v} \text{ has no neighbour in } S] \leq \binom{n}{2} n^{-5} \leq n^{-3}$$

Then, Claim 9.6 tells us that the additive stretch factor is at most 2 with probability $\geq 1 - \frac{1}{n^3}$.

Therefore, with high probability ($\geq 1 - \frac{1}{n^3}$), the construction yields a 2-additive spanner. □

Remark A way to remove log factors from Theorem 9.5 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 9.7. *[Che13] For a fixed $k \geq 1$, 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 edge (w, s') for some $s' \in S'$, add (w, s') to E'_3.
   - $\forall s, s' \in S'$, add the shortest path between s and s' with $\leq n^{1/5}$ internal heavy vertices to E'_3.
     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 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 9.4 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[4], $|E'_2| \leq n \cdot |S| = 30n^{7/5} \log n \in \widetilde{\mathcal{O}}(n^{7/5})$.
- 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 9.4 with $p = 10n^{-2/5} \log n$, $\mathbb{E}[|S'|] = n \cdot 10n^{-2/5} \log n = 10n^{3/5} \log n$. So, E'_3 contributes $\leq n + \binom{|S'|}{2} \cdot n^{1/5} \leq n + (10n^{3/5} \log n)^2 \cdot n^{1/5} \in \widetilde{\mathcal{O}}(n^{7/5})$ edges to the count of $|E'|$.

[4]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 9.4, with $p = 10n^{-2/5}\log n$,

$$\Pr[|N(w) \cap S'| = 0] = \Pr[|N(w') \cap S'| = 0] \leq e^{-\frac{n^{2/5} \cdot 10n^{-2/5}\log n}{2}} = n^{-5}$$

Let $s, s' \in S'$ be adjacent vertices 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 $P^*_{s,s'}$, $l^* \leq d_G(s,w) + d_G(w,w') + d_G(w',s') = d_G(w,w') + 2$.
- 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,

$$\begin{aligned}
&d_{G'}(u,v) \\
&= d_{G'}(u,w) + d_{G'}(w,w') + d_{G'}(w',v) && (1)\\
&\leq d_{G'}(u,w) + d_{G'}(w,s) + d_{G'}(s,s') + d_{G'}(s',w') + d_{G'}(w',v) && (2)\\
&= d_{G'}(u,w) + d_{G'}(w,s) + l^* + d_{G'}(s',w') + d_{G'}(w',v) && (3)\\
&\leq d_{G'}(u,w) + d_{G'}(w,s) + d_G(w,w') + 2 + d_{G'}(s',w') + d_{G'}(w',v) && (4)\\
&= d_{G'}(u,w) + 1 + d_G(w,w') + 2 + 1 + d_{G'}(w',v) && (5)\\
&= d_G(u,w) + 1 + d_G(w,w') + 2 + 1 + d_G(w',v) && (6)\\
&\leq d_G(u,v) + 4 && (7)
\end{aligned}$$

(1) Decomposing $P_{u,v}$ in G'

(2) Triangle inequality

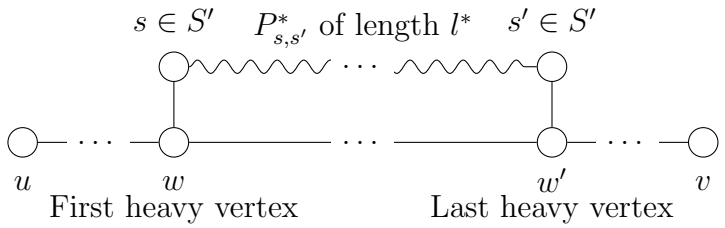
(3) $P^*_{s,s'}$ is added to E'_3

(4) Since $l^* \leq d_G(w, w') + 2$

(5) Since $(w, s) \in E'$ and $(s', w') \in E'$ and $d_{G'}(w, s) = d_{G'}(s', w') = 1$

(6) Since $d_{G'}(u, w) = d_G(u, w)$ and $d_{G'}(w', v) = d_G(w', v)$

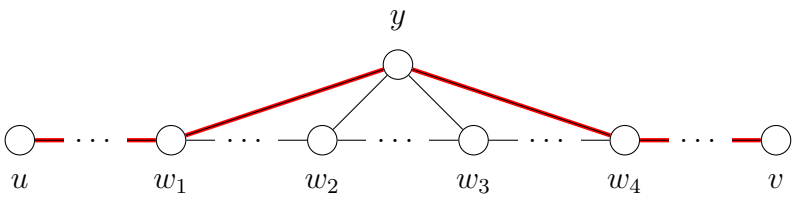
(7) By definition of $P_{u,v}$



Case (iii)

Claim 9.8. *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 immediately contradicts that $P_{u,v}$ was the shortest path involving all of $\{y, w_1, w_2, w_3, w_4\}$. □

Claim 9.8 tells us that $|\bigcup_{w \in \text{Heavy}} N(w)| \geq \sum_{w \in \text{Heavy}} |N(w)| \cdot \frac{1}{3}$. Let

$$N_{u,v} = \{x \in V : (x, w) \in P_{u,v} \text{ for some } w \in P_{u,v}\}$$

Applying Claim 9.4 with $p = 30 \cdot n^{-3/5} \cdot \log n$ and Claim 9.8, we get

$$\mathbb{E}[|N_{u,v} \cap S|] \geq n^{1/5} \cdot n^{2/5} \cdot \frac{1}{3} \cdot 30 \cdot n^{-3/5} \cdot \log n = 10 \log n$$

and

$$\Pr[|N(v) \cap S| = 0] \leq e^{-\frac{10 \log n}{2}} = n^{-5}$$

Taking union bound over all possible pairs of vertices u and v,

$$\Pr[\exists u, v \in V, P_{u,v} \text{ has no neighbour in } S] \leq \binom{n}{2} n^{-5} \leq n^{-3}$$

Then, Claim 9.6 tells us that the additive stretch factor is at most 4 with probability $\geq 1 - \frac{1}{n^3}$.

Therefore, with high probability ($\geq 1 - \frac{1}{n^3}$), 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[5] 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".

Chapter 10

Preserving cuts

In the previous chapter, we looked at preserving distances via spanners. In this chapter, we look at preserving cut sizes.

Definition 10.1 (Cut and minimum cut)**.** *Consider a graph $G = (V, E)$.*

- *For $S \subseteq V, S \neq \emptyset, S \neq V$, $C_G(S, V \setminus S) = \{(u, v) : u \in S, v \in V \setminus S\}$ is a non-trivial cut in G*
- *Define cut size $E_G(S, V \setminus S) = \sum_{e \in C_G(S, V \setminus S)} w(e)$*
  For unweighted G, $w(e) = 1$ for all $e \in E$, so $E_G(S, V \setminus S) = |C_G(S, V \setminus S)|$
- *Minimum cut size of the graph G is denoted by $\mu(G) = \min_{S \subseteq V, S \neq \emptyset, S \neq V} E_G(S, V \setminus S)$*
- *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 graph $G = (V, E)$, our goal in this lecture is to construct a 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 $S \subseteq V, S \neq 0, S \neq V$. Recall Karger's random contraction algorithm [Kar93][1]:

Theorem 10.2. *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. Suppose $|S^*| = k$. To successfully return S^*, *none* of the edges in S^* must be selected in the whole contraction process.

[1] Also, see https://en.wikipedia.org/wiki/Karger%27s_algorithm

Algorithm 27 RandomContraction($G = (V, E)$)

while $|V| > 2$ **do**
  $e \leftarrow$ Pick an edge uniformly at random from E
  $G \leftarrow G/e$ ▷ Contract edge e
end while
return The remaining cut ▷ This may be a multi-graph

By construction, there will be $n-i$ vertices in the graph at step i of RandomContraction. Since $\mu(G) = k$, each vertex has degree $\geq k$ (otherwise that vertex itself gives a cut smaller than k), so there are $\geq (n-i)k/2$ edges in the graph. Thus,

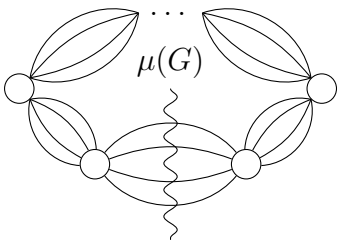
$$\begin{aligned}
\Pr[\text{Success}] &\geq (1 - \frac{k}{nk/2}) \cdot (1 - \frac{k}{(n-1)k/2}) \dots (1 - \frac{k}{3k/2}) \\
&= (1 - \frac{2}{n}) \cdot (1 - \frac{2}{n-1}) \dots (1 - \frac{2}{3}) \\
&= (\frac{n-2}{n}) \cdot (\frac{n-3}{n-1}) \dots (\frac{1}{3}) \\
&= \frac{2}{n(n-1)} \\
&= 1/\binom{n}{2}
\end{aligned}$$

□

Corollary 10.3. *There are $\leq \binom{n}{2}$ minimum cuts in a graph.*

Proof. Since RandomContraction successfully produces *any given minimum cut* with probability at least $1/\binom{n}{2}$, there can be at most $\leq \binom{n}{2}$ many minimum cuts. □

Remark There exists (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.



In general, we can bound the number of cuts that are of size at most $\alpha \cdot \mu(G)$ for $\alpha \geq 1$.

Theorem 10.4. *In an undirected graph, the number of α-minimum cuts is less than $n^{2\alpha}$.*

Proof. See Lemma 2.2 and Appendix A (in particular, Corollary A.7) of a version[2] of [Kar99]. □

10.1 Warm up: $G = K_n$

Consider the following procedure to construct H:

1. Let $p = \Omega(\frac{\log n}{n})$
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[3] that this suffices for $G = K_n$.

10.2 Uniform edge sampling

For a graph G with minimum cut size $\mu(G) = k$, consider the following procedure to construct H:

1. Set $p = \frac{c \log n}{\epsilon^2 k}$ 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 $e \in E'$

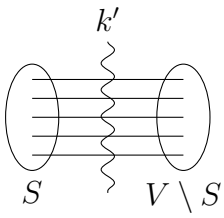
Theorem 10.5. *With high probability, for every $S \subseteq V, S \neq \emptyset, S \neq V$,*

$$(1 - \epsilon) \cdot E_G(S, V \setminus S) \leq E_H(S, V \setminus S) \leq (1 + \epsilon) \cdot E_G(S, V \setminus S)$$

Proof. Fix an arbitrary cut $(S, V \setminus S)$. Suppose $E_G(S, V \setminus S) = k' = \alpha \cdot k$ for some $\alpha \geq 1$.

[2] Version available at: http://people.csail.mit.edu/karger/Papers/skeleton-journal.ps

[3] Fix a cut, analyze, then take union bound.



Let X_e be the indicator for the edge $e \in C_G(S, V \setminus S)$ being selected 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:

$$
\begin{aligned}
&\Pr[\text{Cut } (S, V \setminus S) \text{ is badly estimated in } H] \\
&= \Pr[|E_H(S, V \setminus S) - \mathbb{E}[E_H(S, V \setminus S)]| > \epsilon \cdot k'] && \text{What it means to be badly estimated} \\
&\leq 2e^{-\frac{\epsilon^2 k' p}{3}} && \text{Chernoff bound} \\
&= 2e^{-\frac{\epsilon^2 \alpha k p}{3}} && \text{Since } k' = \alpha k \\
&\leq n^{-10\alpha} && \text{For sufficiently large } c
\end{aligned}
$$

Using Theorem 10.4 and union bound over all possible cuts in G,

$$
\begin{aligned}
&\Pr[\textit{Any} \text{ cut is badly estimated in } H] \\
&\leq \int_1^\infty n^{2\alpha} \cdot \frac{1}{n^{-10\alpha}} d\alpha && \text{From Theorem 10.4 and above} \\
&\leq n^{-5} && \text{Loose upper bound}
\end{aligned}
$$

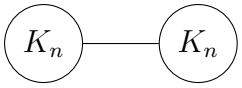
□

Theorem 10.6. *[Kar94] For a graph G, consider sampling every edge independently with probability p_e into E', and assign weights $1/p_e$ to each edge $e \in E'$. Let $H = (V, E')$ be the sampled graph and suppose $\mu(H) \geq \frac{c \log n}{\epsilon^2}$, for some constant c. Then, with high probability, every weighted cut size in H is (well-estimated) within $(1 \pm \epsilon)$ of the original cut size in G.*

Theorem 10.6 can be proved by using a variant of the earlier proof. Interested readers can see Theorem 2.1 of [Kar94].

10.3 Non-uniform edge sampling

Unfortunately, uniform sampling does not work well on graphs with small minimum cut.



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 define *k-strong components*.

Definition 10.7 (k-connected)**.** *A graph is k-connected if the value of each cut of G is at least k.*

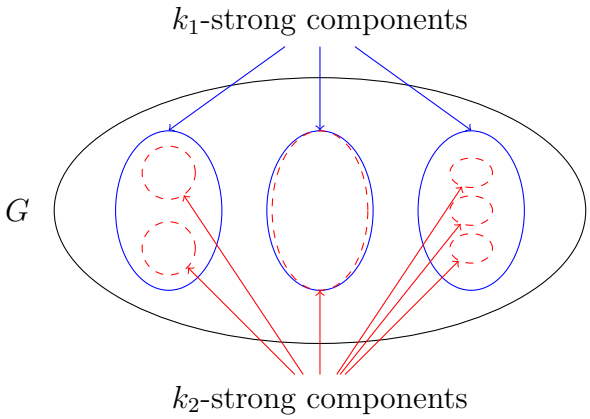
Definition 10.8 (k-strong component)**.** *A k-strong component is a maximal k-connected vertex-induced subgraph. For an edge e, define its strong connectivity / strength k_e as the maximum k such that e is in a k-strong component. We say an edge is k-strong if $k_e \geq k$.*

Remark The (standard) connectivity of an edge $e = (u, v)$ is the minimum cut size that separates 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 10.9. *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 there are a lot of edges, then many of them have high strength.

Proof.



1. By definition of maximum

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 $\leq \mu(G) \cdot \frac{1}{k_e} \leq \mu(G) \cdot \frac{1}{\mu(G)} = 1$ to the summation. Remove these edges from G and repeat the argument on any remaining connected components. Since each cut removal contributes at most 1 to the summation and the process stops when we reach n components, $\sum_{e \in E} \frac{1}{k_e} \leq n - 1$.

□

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 $p_e = \frac{q}{k_e}$

3. Define $w(e) = \frac{1}{p_e} = \frac{k_e}{q}$ for each edge $e \in E'$

Lemma 10.10. $\mathbb{E}[|E'|] \leq \mathcal{O}(\frac{n \log n}{\epsilon^2})$

Proof. Let X_e be the indicator whether edge e was selected into E'. By construction, $\mathbb{E}[X_e] = \Pr[X_e = 1] = p_e$. Then,

$$\begin{aligned}
\mathbb{E}[|E'|] &= \mathbb{E}[\sum_{e \in E} X_e] && \text{By definition} \\
&= \sum_{e \in E} \mathbb{E}[X_e] && \text{Linearity of expectation} \\
&= \sum_{e \in E} p_e && \text{Since } \mathbb{E}[X_e] = \Pr[X_e = 1] = p_e \\
&= \sum_{e \in E} \frac{q}{k_e} && \text{Since } p_e = \frac{q}{k_e} \\
&= q(n-1) && \text{Since } \sum_{e \in E} \frac{1}{k_e} \leq n - 1 \\
&\in \mathcal{O}(\frac{n \log n}{\epsilon^2}) && \text{Since } q = \frac{c \log n}{\epsilon^2} \text{ for some constant } c
\end{aligned}$$

□

Remark One can apply Chernoff bounds to argue that $|E'|$ is highly concentrated around its expectation.

Theorem 10.11. *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 $F_i = (V, E_i)$ where $E_i = \{e \in E : k_e \geq k_i\}$. 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 i in F_i, $k_e = i$ in G
- $F_1 = G$
- For each i, 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 $F_{k_e}, \ldots, F_s$. For each $i \in [s]$, Theorem 10.6 tells us that every cut in F_i is well-estimated with high probability. Then, a union bound over $\{F_i\}_{i\in[s]}$ will tell us that any cut in G is well-estimated with high probability. □

Part IV

Online algorithms and competitive analysis

Chapter 11

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 11.1 (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 11.2 (Ski rental problem)**.** *Suppose we wish to ski every day but we do not have any skiing equipment initially. On each day, we could:*

- *Rent the equipment for a day at CHF* 1
- *Buy the equipment (once and for all) at 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 11.3. *$\mathcal{A}$ = "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 d. $\mathcal{A}$ will rent for d days and also incur d. If $d > B$, the optimal offline strategy is to buy the equipment immediately, incurring B. $\mathcal{A}$ will rent for B days then buy, incurring $2B$. Thus, for any d, $c_{\mathcal{A}}(d) \leq 2 \cdot c_{OPT}(d)$. □

Chapter 12

Linear search

Definition 12.1 (Linear search problem)**.** *Suppose 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 query is the i-th paper in the stack and it costs i units of work to reach it. There are two types of swaps we can make to 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 it.

12.1 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 $\mathcal{O}(1)$ unless the hash table is almost full or almost empty, in which case we

[1] See https://en.wikipedia.org/wiki/Amortized_analysis

double or halve the hash table, incurring a runtime of $\mathcal{O}(m)$ time for doubling or halving a hash table of size 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.

12.2 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 $\Phi = 2$.

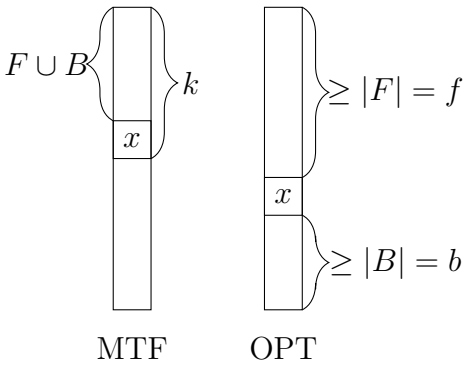
In either case, we see that any paid swap results in ± 1 inversions, which changes Φ by ± 1.

Claim 12.2. *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 by MTF at time t. Suppose the queried item at time step t is at position k in MTF. Denote:

$F = \{\text{Items in front of } x \text{ in MTF's stack and } \textit{in front} \text{ of } x \text{ in OPT's stack}\}$
$B = \{\text{Items in front of } x \text{ in MTF's stack and } \textit{behind } x \text{ in OPT's stack}\}$

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 $c_{MTF}(t) = k$ 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. In MTF, moving x to the top destroys b inversions and creates f inversions, so the change in Φ due to MTF is $f - b$. If OPT chooses to do a free swap, Φ decreases 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. That is, the change in Φ due to MTF is $\leq p$.

Thus, the effect on Φ from both processes is: $\Delta(\Phi_t) \leq (f-b)+p$. Putting together, we have $c_{OPT}(t) \geq f + 1 + p$ and $a_t = c_{MTF}(t) + (\Phi_t - \Phi_{t-1}) = k + \Delta(\Phi_t) \leq 2f + 1 + p$. Hence,

$$a_t \leq 2 \cdot c_{OPT}(t) \tag{1}$$

$$\Rightarrow \quad \sum_{t=1}^{|\sigma|} a_t \leq \sum_{t=1}^{|\sigma|} 2 \cdot c_{OPT}(t) \tag{2}$$

$$\Rightarrow \quad \sum_{t=1}^{|\sigma|} c_{MTF}(t) + (\Phi_t - \Phi_{t-1}) \leq 2 \cdot c_{OPT}(\sigma) \tag{3}$$

$$\Rightarrow \quad \sum_{t=1}^{|\sigma|} c_{MTF}(t) + (\Phi_{|\sigma|} - \Phi_0) \leq 2 \cdot c_{OPT}(\sigma) \tag{4}$$

$$\Rightarrow \quad \sum_{t=1}^{|\sigma|} c_{MTF}(t) \leq 2 \cdot c_{OPT}(\sigma) \tag{5}$$

$$\Rightarrow \quad c_{MTF}(\sigma) \leq 2 \cdot c_{OPT}(\sigma) \tag{6}$$

(1) Since $a_t = 2f + 1 + p$ and $c_{OPT}(t) \geq f + 1 + p$

(2) Summing over all inputs in the sequence σ

(3) Since $a_t = c_{MTF}(t) + (\Phi_t - \Phi_{t-1})$

(4) Telescoping

(5) Since $\Phi_t \geq 0 = \Phi_0$

(6) Since $c_{MTF}(\sigma) = \sum_{t=1}^{|\sigma|} c_{MTF}(t)$

□

Chapter 13

Paging

Definition 13.1 (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 13.2 (Conservative strategy)**.** *A strategy is conservative if on any consecutive subsequence that has 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 13.3. *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.

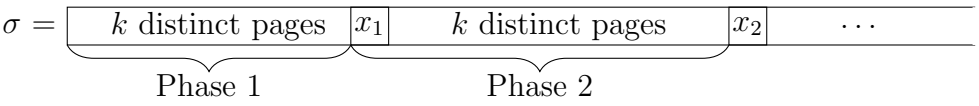
Since $\mathcal{A}$ is deterministic, the adversary can simulate $\mathcal{A}$ for $|\sigma|$ steps and build σ accordingly. By construction, $c_{\mathcal{A}}(\sigma) = |\sigma|$.

On the other hand, since OPT can see the entire sequence σ, OPT can choose to evict the page that is requested furthest in the future. Then, in every k steps, OPT has ≤ 1 cache miss. Thus, $c_{OPT} \leq \frac{|\sigma|}{k}$.

Hence, $k \cdot c_{OPT} \leq c_{\mathcal{A}}(\sigma)$. □

Claim 13.4. *Any conservative online algorithm $\mathcal{A}$ is k-competitive.*

Proof. For any given input sequence σ, partition σ into *maximal phases* — $P_1, P_2, \ldots$ — 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. On the other hand, since $\mathcal{A}$ is conservative, $\mathcal{A}$ has $\leq k$ cache misses per phase.

Hence, $c_{\mathcal{A}}(\sigma) \leq k \cdot$ Number of phases $\leq k \cdot c_{OPT}(\sigma)$. □

Remark A randomized algorithm can achieve $\mathcal{O}(\log k)$-competitiveness. This will be covered in the next lecture.

13.1 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.

13.2 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?	✗	✗	✓

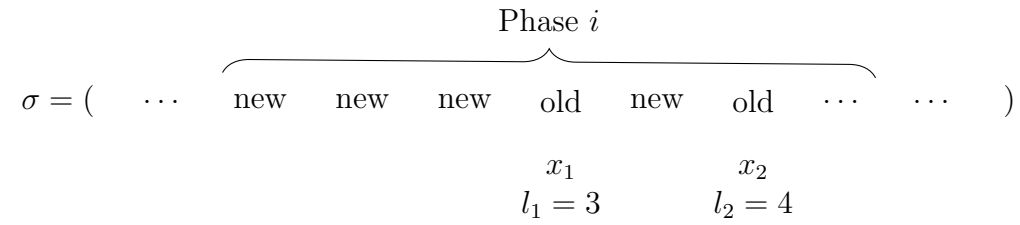
We denote a phase as the time period between 2 consecutive full unmarking steps. That is, each phase is a maximal run where we access k distinct pages. In the above example, $\{2, 5, 2, 1\}$ is such a phase for $k = 3$.

Observation As pages are only unmarked at the beginning of a new phase, the number of unmarked pages is monotonically decreasing within a phase.

Theorem 13.5. *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.

Denote $\mathcal{N}$ as the set of *new requests* (pages that are not in P_i) and $\mathcal{O}$ as the set of *old requests* (pages that are in P_i). By definition, $|\mathcal{O}| \leq k$ and $|\mathcal{N}| + |\mathcal{O}| = k$. Order the old requests in $\mathcal{O}$ in the order which they appear in the phase and let x_j be the j^{th} old request, for $j \in \{1, \ldots, |\mathcal{O}|\}$. Define $m_i = |\mathcal{N}|$, and l_j as the number of *distinct new* requests before x_j.



For $j \in \{1, \ldots, |\mathcal{O}|\}$, 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,

[1] We get an equality if *all* these requests kicked out an old page.

$$
\begin{aligned}
\text{Cost due to } \mathcal{O} &= \sum_{j=1}^{|\mathcal{O}|} \Pr[x_j \text{ is } \textit{not} \text{ in cache when requested}] && \text{Sum over } \mathcal{O} \\
&\leq \sum_{j=1}^{|\mathcal{O}|} \frac{l_j}{k-(j-1)} && \text{From above} \\
&\leq \sum_{j=1}^{|\mathcal{O}|} \frac{m_i}{k-(j-1)} && \text{Since } l_j \leq m_i = |\mathcal{N}| \\
&\leq m_i \cdot \sum_{j=1}^{k} \frac{1}{k-(j-1)} && \text{Since } |\mathcal{O}| \leq k \\
&= m_i \cdot \sum_{j=1}^{k} \frac{1}{j} && \text{Rewriting} \\
&= m_i \cdot H_k && \text{Since } \sum_{i=1}^{n} \frac{1}{i} = H_n
\end{aligned}
$$

Since every new request incurs a unit cost, the cost due to $\mathcal{N}$ is m_i. Hence, $c_{RMA}(\text{Phase } i) = (\text{Cost due to } \mathcal{N}) + (\text{Cost due to } \mathcal{O}) \leq m_i + m_i \cdot H_k$.

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 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 $1 = (2, 5, 2, 1)$, $P_1 = \{1, 3, 4\}$, $\mathcal{N} = \{5\}$, $\mathcal{O} = \{2, 1\}$. Although '2' appeared twice, we only care about analyzing the first time it appeared.

Chapter 14

Yao's Minimax Principle

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 14.1 ([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.

$$\begin{aligned}
C &= \sum_x q_x \cdot C && \text{Sum over all possible inputs } x \\
&\geq \sum_x q_x \mathbb{E}_p[c(A, x)] && \text{Since } C = \max_{x \in X} \mathbb{E}_p[c(A, x)] \\
&= \sum_x q_x \sum_a p_a c(a, x) && \text{Definition of } \mathbb{E}_p[c(A, x)] \\
&= \sum_a p_a \sum_x q_x c(a, x) && \text{Swap summations} \\
&= \sum_a p_a \mathbb{E}_q[c(a, X)] && \text{Definition of } \mathbb{E}_q[c(a, X)] \\
&\geq \sum_a p_a \cdot D && \text{Since } D = \min_{a \in A} \mathbb{E}_q[c(a, X)] \\
&= D && \text{Sum over all possible algorithms } a
\end{aligned}$$

□

Implication If one can argue that *no* deterministic algorithm can do well on a given distribution of random inputs, then no randomized algorithm can do well on *all* inputs.

14.1 Application to the paging problem

Theorem 14.2. *Any (randomized) algorithm has competitive ratio* $\Omega(\log k)$ *against an oblivious adversary.*

Proof. Fix an arbitrary deterministic algorithm $\mathcal{A}$. Let $n = k + 1$ and $|\sigma| = m$. 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, $\mathbb{E}[c_{OPT}(\sigma)] = \frac{m}{(k+1) \cdot H_{k+1}}$.

Putting together, we have $\frac{\mathbb{E}[c_{\mathcal{A}}(\sigma)]}{\mathbb{E}[c_{OPT}(\sigma)]} = H_{k+1} = \Theta(\log k)$. □

Remark The length of a phase is essentially the coupon collector problem with $n = k + 1$ coupons.

Chapter 15

The k-server problem

Definition 15.1 (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 $(\log k)$-competitive algorithm exists. The table below shows the current progress on this problem.

	Competitive ratio	Type
[MMS90]	k-competitive, for $k = \{2, n-1\}$	Deterministic
[FRR90]	$2^{\mathcal{O}(k \log k)}$-competitive	Deterministic
[Gro91]	$2^{\mathcal{O}(k)}$-competitive	Deterministic
[KP95]	$(2k-1)$-competitive	Deterministic
[BBMN11]	$\text{poly}(\log n, \log k)$-competitive	Randomized
[Lee18]	$\mathcal{O}(\log^6 k)$-competitive	Randomized

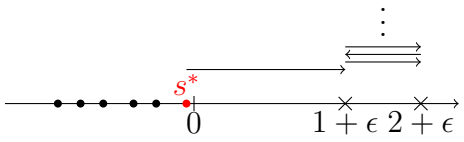
Remark [BBMN11] uses a probabilistic tree embedding, a concept we have seen in earlier lectures.

15.1 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 $\mathbb{R}$.

15.1.1 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:

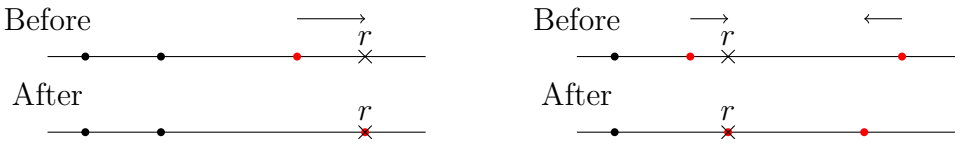


Without loss of generality, suppose all servers currently lie on the left of "0". For $\epsilon > 0$, consider the sequence $\sigma = (1+\epsilon, 2+\epsilon, 1+\epsilon, 2+\epsilon, \dots)$. The first request will move a single server s^* to "$1+\epsilon$". By the greedy algorithm, subsequent requests 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.

15.1.2 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 15.2. *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 15.3. *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.

Chapter 16

Multiplicative Weights Update (MWU)

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

Definition 16.1 (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.

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

Days	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

is expert e_1 (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.

16.1 Warm up: Perfect expert exists

Suppose there exists a perfect expert. 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 16.2. *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. □

16.2 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 by 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 16.3. *Suppose the best expert makes* m^* *mistakes and* DMWU *makes* m *mistakes. Then,*

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

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 $\sum_{i=1}^{n} w_i = x$ 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 $(1-\epsilon)^{m^*}$. 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. Then,

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

□

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

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

Theorem 16.4. *No deterministic algorithm $\mathcal{A}$ can do 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, $m = |\sigma| \geq 2m^*$. □

16.3 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 $i \in \{1, \ldots, n\}$.

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

- On each day:
  - Pick a random expert with probability proportional to their weight. (i.e. Pick e_i with probability $w_i / \sum_{i=1}^{n} w_i$)
  - 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 16.5. *Suppose the best expert makes m^* mistakes and* RMWU *makes m mistakes. Then,*

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

Proof. Fix an arbitrary day $j \in \{1, \ldots, |\sigma|\}$. Denote $A = \{\text{Experts that output 0 on day } j\}$ 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 sum of weights in each set. Let F_j be the weighted fraction of wrong 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. 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. Then,

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

□

16.4 Generalization

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

Claim 16.6 (Without proof)**.** *With RMWU, we have*

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

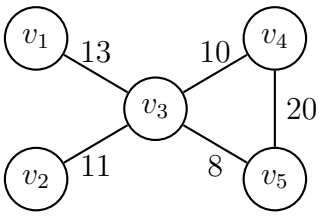
Remark If each expert has a different ρ_i, one can modify the update rule and claim to use ρ_i instead of a uniform ρ accordingly.

16.5 Application: Online routing of virtual circuits

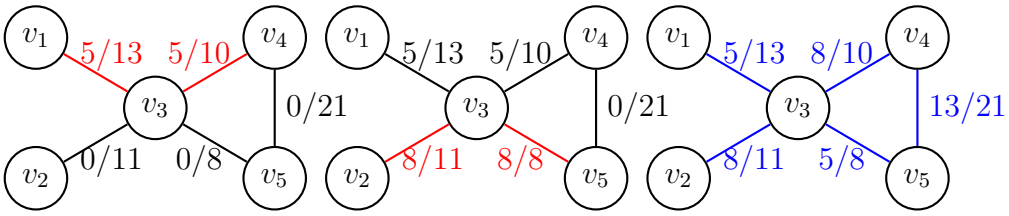
Definition 16.7 (The online routing of virtual circuits problem)**.** *Consider a graph $G = (V, E)$ where each edge $e \in E$ has a capacity u_e. A request is denoted by a triple $\langle s(i), t(i), d(i)\rangle$, where $s(i) \in V$ is the source, $t(i) \in V$ is the target, and $d(i) > 0$ is the demand for the i^{th} request respectively. Given the i^{th} request, we are 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 lecture 5. 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 $\sigma(1) = \langle v_1, v_4, 5\rangle$, we (red edges) 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 relative demand i with respect to the capacity of edge e.
- $l_e(j) = \sum_{P_i \ni e, i \le j} p_e(i)$ as the relative load of edge e *after* request j
- $l_e^*(j)$ as 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 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 the shortest (smallest total c_e cost) $s(i) - t(i)$ path P_i on G with edge weights c_e

Finding the shortest path via 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 $\gamma \ge 2$. Because of normalization, $\widetilde{l}_e^*(j) \le 1$, so $\gamma - \widetilde{l}_e^*(j) \ge 1$. Initially, when $j = 0$, $\Phi(0) = \sum_{e \in E} \gamma = m\gamma$.

Lemma 16.8. *For $\gamma \ge 1$ and $0 \le x \le 1$, $(1 + \frac{1}{2\gamma})^x < 1 + \frac{x}{\gamma}$.*

Proof. By Taylor series[2], $(1 + \frac{1}{2\gamma})^x = 1 + \frac{x}{2\gamma} + \mathcal{O}(\frac{x}{2\gamma}) < 1 + \frac{x}{\gamma}$. □

[2]See https://en.wikipedia.org/wiki/Taylor_series#Binomial_series

Lemma 16.9. *For* $a = 1 + \frac{1}{2\gamma}$, $\Phi(j+1) - \Phi(j) \leq 0$.

Proof. Let P_{j+1} be the path that $\mathcal{A}$ found and 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 due to 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, (i) If $e \notin P_{j+1}$, then $\widetilde{l}_e(j+1) = \widetilde{l}_e(j)$; (ii) If $e \in P_{j+1}$, then $\widetilde{l}_e(j+1) = \widetilde{l}_e(j) + \widetilde{p}_e(j+1)$.

- If e is neither in P_{j+1} nor in P^*_{j+1}, then $a^{\widetilde{l}_e(j+1)}(\gamma - \widetilde{l}^*_e(j+1)) = a^{\widetilde{l}_e(j)}(\gamma - \widetilde{l}^*_e(j))$.
  That is, only edges used by P_{j+1} or P^*_{j+1} affect $\Phi(j+1) - \Phi(j)$.

Using the observations above together with Lemma 16.8 and the fact that $\mathcal{A}$ computes a shortest path, one can show that $\Phi(j+1) - \Phi(j) \leq 0$. In detail,

$$
\begin{aligned}
&\Phi(j+1) - \Phi(j) \\
=&\sum_{e\in E} a^{\widetilde{l}_e(j+1)}(\gamma - \widetilde{l}^*_e(j+1)) - a^{\widetilde{l}_e(j)}(\gamma - \widetilde{l}^*_e(j)) \\
=&\sum_{e\in P_{j+1}\setminus P^*_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})(\gamma - \widetilde{l}^*_e(j)) && (1)\\
&+\sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j+1)}(\gamma - \widetilde{l}^*_e(j) - \widetilde{p}_e(j+1)) - a^{\widetilde{l}_e(j)}(\gamma - \widetilde{l}^*_e(j)) \\
=&\sum_{e\in P_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})(\gamma - \widetilde{l}^*_e(j)) - \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j+1)}\widetilde{p}_e(j+1) \\
\le&\sum_{e\in P_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j+1)}\widetilde{p}_e(j+1) && (2)\\
\le&\sum_{e\in P_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j)}\widetilde{p}_e(j+1) && (3)\\
=&\sum_{e\in P_{j+1}} (a^{\widetilde{l}_e(j)+\widetilde{p}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j)}\widetilde{p}_e(j+1) && (4)\\
\le&\sum_{e\in P^*_{j+1}} \Big((a^{\widetilde{l}_e(j)+\widetilde{p}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - a^{\widetilde{l}_e(j)}\widetilde{p}_e(j+1)\Big) && (5)\\
=&\sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j)}\Big((a^{\widetilde{p}_e(j+1)} - 1)\gamma - \widetilde{p}_e(j+1)\Big) \\
=&\sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j)}\Big(((1+\frac{1}{2\gamma})^{\widetilde{p}_e(j+1)} - 1)\gamma - \widetilde{p}_e(j+1)\Big) && (6)\\
\le&\ 0 && (7)
\end{aligned}
$$

(1) From observations above

(2) $\widetilde{l}^*_e(j) \ge 0$

(3) $\widetilde{l}_e(j+1) \ge \widetilde{l}_e(j)$

(4) For $e \in P_{j+1}$, $\widetilde{l}_e(j+1) = \widetilde{l}_e(j) + \widetilde{p}_e(j+1)$

(5) Since P_{j+1} is the shortest path

(6) Since $a = 1 + \frac{1}{2\gamma}$

(7) Lemma 16.8 with $0 \le \widetilde{p}_e(j+1) \le 1$

□

Theorem 16.10. *Let $L = \max_{e \in E} \widetilde{l}_e(|\sigma|)$ be the maximum normalized load at the end of the input sequence σ. For $a = 1 + \frac{1}{2\gamma}$ and $\gamma \geq 2$, $L \in \mathcal{O}(\log n)$. That is, $\mathcal{A}$ 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

$$(1 + \frac{1}{2\gamma})^L \leq a^L \cdot (\gamma - \widetilde{l}_e^*(j)) \leq \Phi(j) \leq m\gamma \leq n^2\gamma$$

Taking log 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 16.10. 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.

- 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 $\sum_{i=1}^{T} \frac{2\beta\Lambda}{2^{T-i}} \leq 4\beta\Lambda$.

[3]Existing paths are preserved, just that we ignore them in the subsequent computations of c_e.

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.

[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.

[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.

[Kar94] David R Karger. Random sampling in cut, flow, and network design problems. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 648–657, 1994.

[Kar99] David R Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

[KP95] Elias Koutsoupias and Christos H Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995.

[Lee18] James R Lee. Fusible hsts and the randomized k-server conjecture. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 438–449. IEEE, 2018.

[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.