CS 270: Combinatorial Algorithms and Data Structures Spring 2023

Lecture 20 — March 23, 2023

Prof. Jelani Nelson *Scribes: Lance Mathias, Ajit Kadaveru*

1 Overview

Today:

1. Approximations using SDP
2. Approximate solutions to the MaxCut problem
3. Approximations in another model (streaming)

After spring break:

1. LP solving algorithms
2. Algorithmic fairness (guest lecture)
3. Spectral graph theory
4. Hardness within P (guest lecture)
5. Lower bounds

2 Problem: MaxCut (unweighted)

Suppose we have undirected graph $G = (V, E)$

Goal: Find the cut $(S, V \setminus S)$ with the maximum number of edges crossing the cut

Remark. This problem is NP-hard.

2.1 Simple approximation algorithms for MaxCut

1. Pick a random partition:
   For each $v \in V$, put it in S with probability $\frac{1}{2}$, else put it in $V \setminus S$. Thus, each edge has a $\frac{1}{2}$ probability of being cut.
   $$\mathbb{E}\left[\# \text{ of edges in cut}\right] = \mathbb{E}\left[\sum_{e \in E} \mathbb{1}\{\text{e cut}\}\right] = \sum_{e \in E} \mathbb{P}(\text{e cut}) = \frac{m}{2}$$
   Since $\mathsf{OPT} \leq m$, it follows that random partition is a $\frac{1}{2}$-approximation.
2. Greedy:
   Start with $S = \{1\}, V \setminus S = \{2, \ldots, n\}$
   While there exists some vertex v s.t. moving v to the other partition increases the value of the cut, do so.
   It can be shown that Greedy will also produce a cut containing at least $\frac{m}{2}$ edges and thus is also a $\frac{1}{2}$-approximation, although we won't show it here.

These were the best-known approximations until the SDP approximation was discovered.

3 Semidefinite Programming (SDP)

SDP is a generalization of linear programming in which we optimize over positive semidefinite matrices, formulated as follows:

$$\begin{aligned} &\min_X tr(C^T X) \\ \text{subject to} &\begin{cases} tr(A_i^T X) = b_i & \forall i \\ X \succeq 0 \end{cases} \end{aligned} \tag{1}$$

Recall:

1. The trace of a square matrix is the sum of diagonal elements: $tr(M) = \sum_{i=1}^n M_{ii}$
2. $tr(A^T B) = \sum_{i=1}^n \sum_{j=1}^n A_{ij} B_{ij}$
3. Recall the three equivalent definitions of positive semidefinite:
    (a) $\forall y, y^T A y \geq 0$
    (b) all eigenvalues of A are nonnegative
    (c) $\exists B$ such that $B^T B = A$ (this implies that $A_{ij} = \langle b_i, b_j \rangle$, where b_i, b_j are columns of B)

Remark. Since SDPs generalize linear programs,

4 MaxCut Algorithms

4.1 QP Solution to MaxCut

Consider the following quadratic program:

$$\begin{aligned} &\max_x \sum_{(u,v)\in E} \frac{1 - x_u x_v}{2} \\ &\text{subject to } x_v^2 = 1 \quad \forall v \in V \end{aligned} \tag{2}$$

$$\text{where } x_u = \begin{cases} 1 \text{ if } u \in S \\ -1 \text{ if } u \in V \setminus S \end{cases}$$

Thus, $\frac{1-x_u x_v}{2} = 1$ if u and v are on opposite sides of the cut, and is 0 otherwise. It follows that the QP yields an exact solution to MaxCut.
Since it is known that MaxCut is NP-hard, this means that solving this QP is also NP-hard.

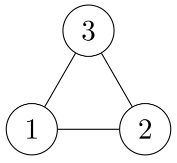

Figure 1: A simple graph G

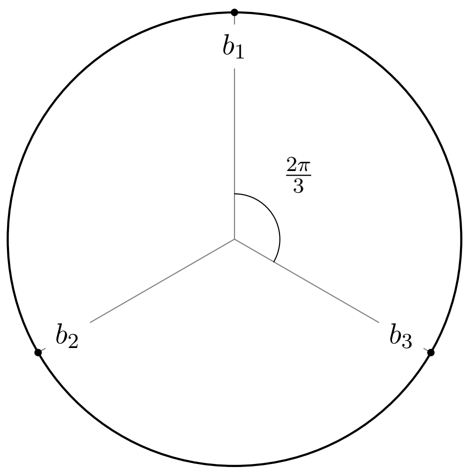

Figure 2: Optimal solutions to the vector program (3)

4.2 SDP Relaxation of MaxCut

It turns out that SDP is equivalent to something called ”vector programming”. Consider the following vector programming relaxation devised by Goemans and Williamson [1]:

$$\begin{aligned} \max \sum_{(u,v)\in E} \frac{1-\langle b_u, b_v\rangle}{2} \\ \text{subject to } \|b_v\|_2^2 = 1 \quad \forall v \in V \end{aligned} \tag{3}$$

Suppose that each b_v is a column of matrix B. Then, we can construct a PSD matrix X where $X = B^T B$ as shown above.
By the properties of PSD matrices, we can formulate an SDP which is equivalent to the above vector program since $\langle b_u, b_v\rangle = X_{uv}$ and $\|x_u\|_2^2 = X_{uu}$.

4.3 Integrality Gap of SDP Relaxation

Consider the graph G shown in Figure 1. By inspection, we see that the max cut is 2. It turns out that the optimal solution to the vector program relaxation is to set vectors b_1, b_2, b_3 to be evenly spaced around the unit circle, as shown in Figure 2. Note that in this example, for every pair of vectors $b_u, b_v, \langle b_u, b_v\rangle = -\frac{1}{2}$. Thus, $\frac{1-\langle b_u,b_v\rangle}{2} = \frac{3}{4}$ and so the objective value of the relatxation is $\frac{9}{4} \geq 2$, and so the integrality gap is present.

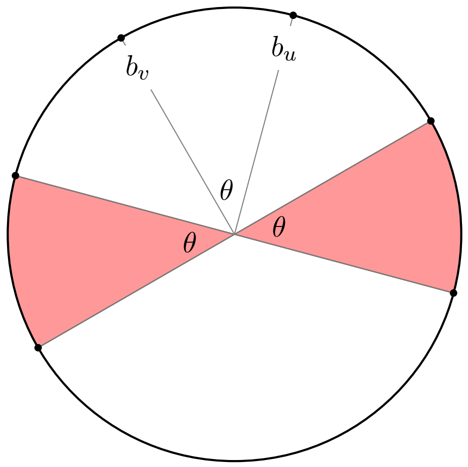

Figure 3: region in which $\text{sign}(\langle b_u, g\rangle) \neq \text{sign}(\langle b_v, g\rangle)$

4.4 GW Algorithm (hyperplane rounding)

Given the set of vectors $\{b_v\}$ obtained by solving the vector program (Equation 3), we can now construct the cut itself. First, construct a random vector $g \sim \mathcal{N}(0, I)$. Now, we can partition each vertex b_u based on $\text{sign}(\langle b_u, g\rangle)$. We can interpret this as picking a random hyperplane parameterized by g, and partitioning vertices based on which side of the hyperplane the corresponding vector b_u lies on. We calculate the approximation ratio of this algorithm as follows:

$$\frac{\mathbb{E}[\text{size of cut}]}{\mathsf{OPT(QP)}} \geq \frac{\mathbb{E}[\text{size of cut}]}{\mathsf{OPT(SDP)}} = \frac{\sum_{e\in E}\mathbb{P}(e\text{cut})}{\sum_{(u,v)\in E}\frac{1-\langle b_u, b_v\rangle}{2}}$$

Which follows from the fact that $\mathsf{OPT(QP)}$ is the exact optimum solution to MaxCut, and $\mathsf{OPT(SDP)} \geq \mathsf{OPT(QP)}$ since SDP is a relaxation of MaxCut.

Furthermore, we can compute

$$\mathbb{P}(e \text{ cut }) = \frac{2\theta}{2\pi} = \frac{\theta}{\pi}$$

Since (u, v) will be cut if $\langle b_u, g\rangle$ and $\langle b_v, g\rangle$ have opposite signs, which happens within a range of 2θ (the shaded region in Figure 3) out of 2π total angle. So our approximation ratio is

$$\frac{\sum_{(u,v)\in E}\frac{\angle(u,v)}{\pi}}{\sum_{(u,v)\in E}\frac{1-\cos(\angle(u,v))}{2}} = \frac{2}{\pi}\cdot\frac{\sum_{(u,v)\in E}\angle(u,v)}{\sum_{(u,v)\in E}1-\cos(\angle(u,v))}$$

Corollary 4.1.

$$\frac{\alpha_i}{\beta_i} \geq z \quad \forall i \implies \frac{\sum_i \alpha_i}{\sum_i \beta_i} \geq z$$

Now, we define

$$\gamma_{GW} \coloneqq \inf_{\theta\in[0,2\pi]} \frac{2}{\pi}\frac{\theta}{1-\cos(\theta)}$$

By Corollary 4.1, it follows that our approximation ratio is at least γ_{GW}, which is ≈ 0.87856.

It is known that the integrality gap of the SDP relaxation can be made arbitrarily close to γ_{GW}, though it is not shown here.

5 Streaming Algorithms

Traditional data structures assume we can store all of the data we have, and are concerned with how to organize the data. By contrast, streaming algorithms use sublinear memory (we can't store everything that we've seen), and also have to worry about *what* data to store. We will now consider some examples of streaming algorithms.

Example 5.1. Counter: keep track of the number of times we increment the counter.

Naive solution: store the count as an integer, which takes $\log n$ bits.

However, there exists an approximation algorithm which is within $(1 \pm \epsilon)$ of the actual value with probability $\geq 1 - \delta$ using only $O(\log \log n + \log \frac{1}{\epsilon} + \log \log \frac{1}{\delta})$ bits.

Example 5.2. Count the number of distinct integers in a set. For simplicity, assume we don't delete elements from the set.

A trivial solution to this problem is to maintain a bitvector which keeps track of whether or not a certain integer is in the set. This requires $O(n)$ bits, which is too much memory!

We will now discuss an approximation algorithm which requires only $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} + \log n)$ bits. To start, we do the following:

1. Pick a random hash function $h : [n] \to [0, 1]$ (we can't use a truly random hash function for reasons discussed earlier in class, so we'll use a pseudorandom hash function, which is usually good enough).
2. Initialize variable $Z \leftarrow 1$

If arbitrary element i is inserted into the set, we update our approximation using the following procedure:

```
procedure INSERT(i)
    $Z \leftarrow \min(z, h(i))$
```

To query our unique item counter, we define the following:

```
procedure QUERY
    return $\frac{1}{Z} - 1$
```

Lemma 5.3. $\mathbb{E}[Z] = \frac{1}{t+1}$ *(order statistics)*

Proof.

$$\mathbb{E}[Z] = \int_0^1 \mathbb{P}(Z > x)dx = \int_0^1 (\mathbb{P}(h(1) > x))^t \, dx = \int_0^1 (1 - x)^t dx = \frac{1}{t+1}$$

□

So, our query procedure will return the correct value in expectation. However, suppose we want a stronger guarantee on our correctness in practice, e.g. we want to show that our approximation is within ε of the correct answer with probability at least $1 - \delta$. To show this, we will compute the variance of Z and apply Chebyshev's inequality.

$$Var[Z] = \mathbb{E}[(Z - \mathbb{E}[Z])^2] = \mathbb{E}[Z^2] - \mathbb{E}[Z]^2 = \frac{2}{(t+1)(t+2)} - \frac{1}{(t+1)^2} = \Theta(\frac{1}{(t+1)^2})$$

Making the simplifying assumption that $t \approx t+1$, we get

$$Var[Z] \approx \Theta\left(\frac{1}{t^2}\right)$$

$$\mu = \mathbb{E}[Z] = \frac{1}{t+1} \approx \frac{1}{t}$$

By Chebyshev,

$$\mathbb{P}(|Z-\mu| > \epsilon\mu) < \frac{Var[Z]}{\epsilon^2\mu^2} \approx \Theta\left(\frac{1}{\varepsilon^2}\right)$$

But since ϵ is small, say 0.1, then we might get a nonsense bound like $P(|Z-\mu| > \epsilon\mu) < 100$, which is totally useless to us.

Idea 1: We can improve this bound to something useful by storing k independent hash functions and k separate variables $Z_1, \ldots Z_k$. Each time we call UPDATE, we will update each Z_i independently using the ith hash function for all $i \in [k]$.

To query, we'll return $\frac{1}{\tilde{Z}} - 1$, where

$$\tilde{Z} := \frac{1}{k}\sum_{i=1}^{k} Z_i$$

is the mean of our k running variables. To compute our new success probability, we will first compute

$$\tilde{\mu} = \mathbb{E}[\tilde{Z}] = \sum_{i=1}^{k} \mathbb{E}[Z_i] = \mathbb{E}[Z]$$

$$Var[\tilde{Z}] = \frac{1}{k^2}\sum_{i=1}^{k} Var[Z_i] = \frac{1}{kt^2}$$

and apply Chebyshev's, which gives a new upper bound on our failure probability of

$$\mathbb{P}(|\tilde{Z}-\tilde{\mu}| > \epsilon\tilde{\mu}) < \frac{Var[\tilde{Z}]}{\epsilon^2\mu^2} \approx \Theta\left(\frac{1}{k\tilde{\varepsilon}^2}\right)$$

If we set $k = \frac{C}{\epsilon^2} \cdot \frac{1}{\delta}$ for appropriate constant C, then our failure probability will be upper bounded by δ, as desired, but this may require a very large value of k which may take up more space than we would like.

Idea 2: To further improve our algorithm so that we actually attain the desired space complexity, consider a "median of means" approach: Set $k = \Theta\left(\frac{1}{\varepsilon^2}\right)$, such that $\tilde{Z}_i$ has failure probability $\frac{1}{3}$. Then, create t independent copies of $\tilde{Z}$, $\tilde{Z}_1, \ldots, \tilde{Z}_t$. To update, we independently update each $\tilde{Z}_i$ as defined above, and to query, we return the median of the mean estimates $\tilde{Z}$, $\tilde{Z}_1, \ldots, \tilde{Z}_t$.

If we set $t := C \log \frac{1}{\delta}$, we attain the desired space complexity, and it can be shown that our failure probability is bounded by δ as desired by applying the Chernoff bound.

Now that we've seen a randomized, approximate solution to the unique items problem, a natural follow-up is: Is a deterministic, exact solution using $o(n)$ space possible?

Claim 5.4. If there exists a space-S algorithm $\mathcal{A}$ that does so, then there exists an injection $f : \{0,1\} \to \{0,1\}^S$

Proof. We define f as follows. Given $x \in \{0,1\}^n$, we let $S = \{i : x_i = 1\}$. We then feed each element of S to $\mathcal{A}$ in a streaming fashion, then at the end define $f(x)$ to be the memory contents $\mathrm{mem}(\mathcal{A})$ of $\mathcal{A}$. To show that this is an injection, we show how any z in the range of f can be uniquely inverted. The inversion procedure works as follows. We first set $t = \mathcal{A}.\texttt{query}()$, so that $t = |S|$. Then for $i = 1, \ldots, n$ in order, we perform $\mathcal{A}.\texttt{insert}(i)$ then query $\mathcal{A}$ again to determine whether the number of distinct elements stayed the same or increased. If it stayed the same, then $i \in S$. If it increased, then $i \notin S$. In this way we are able to recover the value of each coordinate of x, and thus invert f. □

The above lower bound proof is called an *encoding argument*, because it shows that if a low-memory algorithm existed for some problem, it would imply an encoding of a set into a number of bits that falls below the information-theoretic minimum (in this case, simply put we would have an injection from one set into a strictly smaller set). There exist similar encoding arguments that prove that both deterministic+approximate and randomized+exact algorithms are also both impossible by the encoding argument.

References

[1] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, nov 1995.