A Lecture on Derandomization[1]

- Consider the following problem **MAXCUT**. The input is an undirected graph $G = (V, E)$. The output is a subset $S \subseteq V$. The objective is to maximize $|\partial(S)|$, where

$$\partial(S) = \{e \in E : \ e \text{ has one endpoint in } S \text{ and one in } V \setminus S\} .$$

  We will use the following notation for the rest of lecture:

  - opt_G will be the size of a maximum cut on G, i.e. $\text{opt}_G = \max_{S \subseteq V} |\partial(S)|$.
  - $m = |E|$, the number of edges of G.[2]
  - $n = |V|$, the number of vertices of G, and we let the vertices be $1, ..., n$.

  Finding opt_G is known to be NP-hard, so we don't expect a polynomial time algorithm for MAXCUT. But we could hope for an efficient α**-approximation** algorithm, i.e. one which yields a cut S with $|\partial(S)| \geq \alpha \cdot \text{opt}_G$.

  We will begin with a randomized $1/2$-approximation algorithm for MAXCUT, i.e. a randomized algorithm producing an S which satisfies $E[|\partial(S)|] \geq \alpha \cdot \text{opt}_G$. We will then "derandomize" this algorithm using two ideas — the **method of conditional expectations** and the **method of small spaces** — to obtain two different deterministic $1/2$-approximation algorithms for MAXCUT.

- **Randomized** $1/2$**-approximation for MAXCUT.** The algorithm is simple:

  Algorithm A (Randomized $1/2$-approximation).

  - Get a vector $\vec{r} = (r_1, r_2, ..., r_n) \in \{0,1\}^n$ of n independent random bits.
  - Include vertex i in S exactly if $r_i = 1$.

  To prove this satisfies $\mathbf{Exp}[|\partial S|] \geq \frac{1}{2} \cdot \text{opt}_G$, it suffices, by the note above, to show $\mathbf{Exp}[|\partial S|] \geq \frac{1}{2} \cdot m$. In fact,

  Claim 1. $\mathbf{Exp}[|\partial S|] = \frac{1}{2} \cdot m$.

  Proof. $\mathbf{Exp}_{\vec{r} \in \{0,1\}^n}[|\partial S(\vec{r})] = \sum_{e \in E} \Pr[e \in \partial S] = \sum_{(i,j) \in E} \Pr[r_i \neq r_j] = m \cdot \frac{1}{2}$. □

  Thus we really have a randomized $1/2$-approximation algorithm.

  Observe that since $\mathbf{Exp}_{\vec{r} \in \{0,1\}^n}[|\partial S(\vec{r})]$ is a weighted average of $|\partial S(\vec{r})|$ over the choices of $\vec{r}$, we must have some $\vec{r}$ with $|\partial S(\vec{r})| \geq \frac{m}{2}$. One way to "derandomize" would be to systematically go through all 2^n possibilities for $\vec{r}$ until we find it. This is a deterministic algorithm for $1/2$-approximation, but exponential time... Let's do better.

[1] *Lecture notes by Matthew Ellison, with minor modifications by Deeparnab Chakrabarty. Last modified : 29th May, 2023 These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!*

[2] Note that $\text{opt}_G \leq m$, with equality exactly when G is bipartite.

- **Method of Conditional Expectations**[3] We will first derandomize Algorithm A using the *method of conditional expectations*. Let A be a random variable denoting the size of $|\partial(S)|$ resulting from Algorithm A. We proved $\mathbf{Exp}[A] = \frac{m}{2}$ above, and consider expanding this identity as follows:

$$\begin{aligned} \frac{m}{2} &= \mathbf{Exp}[A] \\ &= \Pr[r_1 = 0] \cdot \mathbf{Exp}[A|r_1 = 0] + \Pr[r_1 = 1] \cdot \mathbf{Exp}[A|r_1 = 1]. \end{aligned}$$

Therefore $m/2$ is a weighted average of $\mathbf{Exp}[A|r_1 = 0]$ and $\mathbf{Exp}[A|r_1 = 1]$, and so one must be $\geq m/2$. Suppose we had $\mathbf{Exp}[A|r_1 = 1] \geq m/2$. This means there is a good approximate cut with $r_1 = 1$, and we could expand again!

$$\begin{aligned} m/2 &\leq \mathbf{Exp}[A|r_1 = 1] \\ &= \Pr[r_2 = 0] \cdot \mathbf{Exp}[A|r_1 = 1, r_2 = 0] + \Pr[r_2 = 1] \cdot \mathbf{Exp}[A|r_1 = 1, r_2 = 1] \end{aligned}$$

We'd then know one of $\mathbf{Exp}[A|r_1 = 1, r_2 = 0]$ and $\mathbf{Exp}[A|r_1 = 1, r_2 = 1]$ was $\geq m/2$, say $\mathbf{Exp}[A|r_1 = 1, r_2 = 0]$. This would mean there is a good approximate cut with $r_1 = 1, r_2 = 0$!

There's nothing stopping us from continuing this process until we find something like

$$\mathbf{Exp}[A|r_1 = 1, r_2 = 0, ..., r_n = 1] \geq m/2.$$

At this point, though, all the randomness is gone, and we'd conclude that choosing S according to these $r_i's$ gives a cut with size $m/2$.

In summary, if we were able to quickly compute these conditional expectations to find a good choice of r_i at each step, we would have a deterministic algorithm to find a $1/2$-approximate max cut — like cutting a beam through the search tree of the 2^n-time algorithm mentioned above.

- How can we compute these conditional expectations? Suppose we're considering $\mathbf{Exp}[A|r_1 = 1, r_2 = 0, ..., r_i = 1]$. We can split the vertices of G into three groups: the one's already assigned to the cut (call that set R), the one's already assigned *not* to the cut ($\neg R$), and the rest (X). Certainly we're going to get all the edges between R and $\neg R$, let $E(R : \neg R)$ be that count. The other edges we might include are those between R and X, between $\neg R$ and X, and within X[4]. One can check that that the chance of any one of these other edges ending up in the cut is $1/2$, and so by linearity of expectation we have

$$\mathbf{Exp}[A|...] = E(R : \neg R) + \frac{1}{2} \cdot \left[E(R : X) + E(\neg R : X) + E(X : X)\right].$$

Using this, we arrive at the following derandomized 1/2-approximation algorithm for MAXCUT.

Algorithm B ($1/2$-approximation, via Conditional Expectations).

  - $R \leftarrow \emptyset, \neg R \leftarrow \emptyset$
  - for $i = 1$ to n:
    - if (#edges i to R)$\leq$ (# edges i to $\neg R$)

[3] Sometimes called the Method of Conditional Probabilities.

[4] Ignoring loops!

    - $\cdot$ $R \leftarrow R + \{i\}$
  - $*$ else
    - $\cdot$ $\neg R \leftarrow \neg R + \{i\}$
- return R

The cut produced by Algorithm B will be guaranteed to have $\geq m/2$ edges by our work above, and Algorithm B may be made to run in $O(m)$ time since it only needs to touch every edge once.

- **Method of Small Probablity Spaces.** Next we will look at a different way to derandomize Algorithm A using the *method of small spaces*. The idea is that we will deterministically search through possibilities for $\vec{r}$ in Algorithm 1, but only a small subset H of them.[5] In particular, we'd like an $H \in \{0,1\}^n$ satisfying

  a. $|H| \ll 2^n$ (we really want polynomial in n)

  b. $\mathbf{Exp}_{\vec{r} \in_R H}[|\partial S(\vec{r})|] \geq m/2$.

  If we had such an H we could deterministically test all $\vec{r} \in H$ and (by 1) it wouldn't take too long and (by 2) we would be guaranteed to find a cut of size $\geq m/2$.

  How can we find such a set H? The key observation is that we would like to restrict attention to $\vec{r}$ such that pairs of vertices (i,j) joined by an edge are unlikely to have $r_i = r_j$, in turn making it likely edge (i,j) is in the cut. This sounds like a job for hashing.

  Let $\tilde{H}$ be a UHF of $h : [n] \rightarrow \{0,1\}$. We will use the Carter-Wegman family from before as an example:

  - Pick p prime in $[n, 2n]$.
  - $\tilde{H} = \{h_{a,b} :\ a \in \{1, ..., p-1\}, b \in \{0, ..., p-1\}\}$, where

$$h_{a,b}(x) = ((ax+b) \bmod p) \bmod 2.$$

  Note that $|\tilde{H}| = p(p-1) \leq p^2 \leq 4n^2$, and that, by the properties of a UHF, $\Pr_{h \in \tilde{H}}[h(i) = h(j)] \leq 1/2$.

  But this means that if we define our H as

$$H = \{(h(1), h(2), ..., h(n)) : h \in \tilde{H}\},$$

  we have $|H| = O(n^2)$, checking property 1, and also $\Pr_{\vec{r} \in H}[r_i \neq r_j] \geq 1/2$, checking property 2 by linearity of expectation. Thus we obtain our second derandomized version of Algorithm A:

  Algorithm C ($1/2$-approximation, via Small Spaces).

  - Define H as above, using the Carter-Wegman family (and n the number of vertices in the input graph).
  - Run through $\vec{r} \in H$ until we find one yielding an S with $|\partial S| \geq m/2$, which we return.

[5] The letter H is to foreshadow hashing.

- **Final Notes**
  - Algorithm B is fast, $O(m)$, time but not (obviously at least) parallelizable. Algorithm C is parallelizable, but not so fast. Is to possible to find a deterministic algorithm which is both fast and parallelizable?
  - BPP denotes the class of problems with "bounded-error probabilistic polynomial time" algorithms (see `https://en.wikipedia.org/wiki/BPP_(complexity)` for more details). It is conjectured that P=BPP, which means that any problem with a "bounded-error probabilistic polynomial time" solution also has a polynomial time deterministic solution. For MAXCUT, for example, Algorithm A shows 1/2-approximation of MAXCUT is in BPP and Algorithms B and C show it's in P. Many problems in BPP are still waiting on polynomial deterministic algorithms, such as "polynomial identity testing," the problem of determining whether a polynomial is identically equal to the zero polynomial.