4. Derandomizing MAX-CUT

4.1 Introduction

In this lecture we look at a derandomization technique that works in certain situations. As an application we look at the randomized MAX-CUT algorithm, and see how it can be derandomized.

4.2 Recalling MAX-CUT algorithm

Let's recall the randomized MAX-CUT algorithm that on expectation gives a cut of size at least $m/2$. We randomly partition the vertices of the graph into parts V_1 and V_2, where each vertex is placed in a part with probability $1/2$. Now each edge is a cut-edge with probability $1/2$, and therefore the random partition gives a cut of size $m/2$ in expectation. Now, we show how to sequentially assign vertices to each part so that we end up with a cut of size at least $m/2$.

4.3 Method of conditional expectation

Consider the run of a randomized algorithm $\mathcal{A}$ such that $E(\mathcal{A}(I))$ is some value M. Suppose that it uses t random bits $R_1, R_2, \ldots, R_t$ during its run. We can write the expectation as follows.

$$E[\mathcal{A}(I)] = \sum_{\alpha \in \{0,1\}^n} \Pr[\bigwedge_i R_i = \alpha_i] \mathcal{A}(I, \alpha_1, \alpha_2, \ldots, \alpha_t).$$

We can rewrite this equation as follows.

$$\begin{aligned} E[\mathcal{A}(I)] = &\Pr[R_1 = 0] \sum_{\alpha^{(1)} \in \{0,1\}^{n-1}} \Pr[\bigwedge_{i \geq 2} R_i \mid R_1 = 0] \mathcal{A}(I, 0, \alpha_2, \alpha_3, \ldots, \alpha_n) \\ &+ \Pr[R_1 = 1] \sum_{\alpha^{(1)} \in \{0,1\}^{n-1}} \Pr[\bigwedge_{i \geq 2} R_i \mid R_1 = 1] \mathcal{A}(I, 1, \alpha_2, \alpha_3, \ldots, \alpha_n). \\ &= \Pr[R_1 = 0] E[\mathcal{A}(I) \mid R_1 = 0] + \Pr[R_1 = 1] E[\mathcal{A}(I) \mid R_1 = 1] \end{aligned}$$

Notice that $E[\mathcal{A}(I)]$ is a weighted average of the conditional expectations $E[\mathcal{A}(I) \mid R_1 = 0]$ and $E[\mathcal{A}(I) \mid R_1 = 1]$. Therefore, one of the two must be at least as large as $E[\mathcal{A}(I)]$. If we have a method to compute these conditional expectations, then we have that $E[\mathcal{A}(I)] \leq E[\mathcal{A}(I) \mid R_1 = \alpha_1]$. If we were to continue this process sequentially, at the end of the process we would have set values to eachof the random bits while maintaining the

inequality. This will guarantee that we will find values $\alpha_1, \alpha_2, \ldots, \alpha_t$ such that $E[\mathcal{A}(I)] \leq E[\mathcal{A}(I) \mid R_1 = \alpha_1, R_2 = \alpha_2, \ldots, R_t = \alpha_t]$. But if we fix all the random bits of $\mathcal{A}$ what we have is a deterministic algorithm whose output is at least as large as the expectation.

This is the generic algorithm for derandomizing using the method of conditional expectations. It is only applicable if we have an efficient method to calculate the conditional expectations. Next we see a concrete application of this technique in obtaining a deterministic algorithm that finds a cut of size at least $m/2$.

4.4 Derandomizing MAX-CUT

Now let's see how to apply the generic technique in the case of MAX-CUT. Let $G(V, E)$ be the graph with n vertices and m edges. Suppose we have a partial cut, i.e. sets $S, T \subset V$ such that $S \cap T = \emptyset$ and $|S \cup T| = k$. Let us now look at the $(k+1)^{st}$ vertex v. The randomized algorithm places it in S or T with equal probability. So what is the conditional expectation of the cut-size when we place v in S. This is easily computed as $|E(S,T)| + |E(\{v\},T)| + \frac{1}{2}|\{(u,w) \mid$ either u or w is outside $S \cup T \cup \{v\}\}|$. If we place v in T, then we have the expected cut size as $|E(S,T)| + |E(\{v\},S)| + \frac{1}{2}|\{(u,w) \mid$ either u or w is outside $S \cup T \cup \{v\}\}|$. It is clear that we can compute these values efficiently. So all that is left for us to do is to choose the one that is greater. If you notice the expression the only term that we are checking for is $|E(\{v\},S)|$ and $|E(\{v\},T)|$ since all the others are independent of where we place v. This shows that the derandomized algorithm is a greedy algorithm that sequentially places vertices trying to maximize the cut size at that point!

4.5 Pairwise independence

Recall that in the randomized MAX-CUT algorithm that we saw, we want the random variables corresponding to the vertices to satisfy the condition that $\Pr[R_v \neq R_w] = 1/2$. This guarantees the correctness of that algorithm. This condition is satisfied even the random variables are not completely independent. With this in mind we come to the notion of pairwise independence.

A set of random variables $R_1, R_2, \ldots, R_t$ is pairwise independent if each R_i is unbiased, and for any two R_i, R_j, R_i and R_j are independent. Notice that if the random variables are pairwise independent, then $\Pr[R_u = 1 \wedge R_v = 0] = \Pr[R_u = 1]\Pr[R_v = 0 \mid R_v = 1] = \Pr[R_u = 1]\Pr[R_v = 0]$. Therefore it is sufficient for us to use pairwise independent random variables in the MAX-CUT algorithm. The interesting thing is that to construct n pairwise independent bits, it is sufficient to have $O(\log n)$ independent bits. We will see the construction now.

Let $b_1, b_2, \ldots, b_k$ be k binary unbiased random bits. For each $S \subset [k]$ where S is non-empty, let $R_S = \oplus_{i \in S} b_i$. Notice that R_S is unbiased (why?). Also, for $S \neq T$, $R_S = R_{S \cap T} \oplus R_{S \setminus T}$ and $R_T = R_{S \cap T} \oplus R_{T \setminus S}$. Now $R_{S \cap T}$ is common in both R_S and R_T and is unbiased. Similary, at least one of $R_{S \setminus T}$ or $R_{T \setminus S}$ is non-empty and unbiased. Therefore, R_S and R_T are independent.

So, to get the n bits for the MAX-CUT algorithm, it is sufficient to choose $\log(n+1)$

independent random bits, and then use the construction above. For each assignment to the $\log(n+1)$ bits, we get a set of n bits, and use it to calculate the cut size. Finally we choose the cut that has the maximum size among everything. What is the running time of this algorithm?