**6.842 Randomness and Computation** February 25, 2020

# Lecture 6

*Lecturer: Ronitt Rubinfeld* *Scribe: Alexander Dimitrakakis*

## Overview

- A simple randomized algorithm for max cut
- Pairwise independent sample spaces
- Derandomization

## 1 Max Cut

Let us first define the maximum cut problem that we will work with throughout this lecture.

**Definition 1 (Max Cut)** *Given a graph $G = (V, E)$, the max cut of the graph $G$ is a partition of the vertices of $G$ into two sets, say $S$ and $T$, that maximizes the following quantity $|\{(u, v)|u \in S, v \in T\}|$*

This is a particularly hard problem, as it is NP-Complete. In the example in the very simple graph shown in Fig. 2, the max cut would be to assign $S = \{1\}$ and $T = \{0, 2\}$. Then both edges would cross the cut and we would necessarily have a maximum cut 2-partition of the graph.

Let us now detail the construction of a simple randomized algorithm for finding a cut that is within a factor of 2 of the max cut. The algorithm is detailed below:

**Figure 1**: Randomized Max Cut

**Algorithm 1**
1: $S = \emptyset$
2: $T = \emptyset$
3: **for** each vertex i = 1, ..., n **do**
4: Flip a coin $r_i \in \{0, 1\}$
5: If $r_i = 1$, place vertex $i$ into subset $S$, or into subset $T$ if $r_i = 0$
6: Return $S, T$

As is evident from the algorithm, we are just placing the vertices randomly into either sets $S$ or $T$. The expected value of the cut, i.e. the number of edges crossing the cut, is calculated below. Before we calculate it, however, we must define the random variable $\mathbb{1}_{(u,v)}$ as follows:

$$\mathbb{1}_{(u,v)} = \begin{cases} 1 & \text{if } r_u \neq r_v \\ 0 & \text{otherwise.} \end{cases}$$

Now we are ready to calculate the expected value of the cut:

$$E[\text{cut}] = E[\sum_{(u,v)\ in E} \mathbb{1}_{(u,v)}] = \sum_{(u,v)\ in E} E[\mathbb{1}_{(u,v)}] = \sum_{(u,v)\ in E} Pr[\mathbb{1}_{(u,v)} = 1] = \frac{|E|}{2} \quad (1)$$

The main assumption that we have made is in calculating the probability that a particular edge crosses the cut ($Pr[\mathbb{1}_{(u,v)} = 1]$). We assumed that $r_u$ and $r_v$ are independent cointosses from Algorithm 1

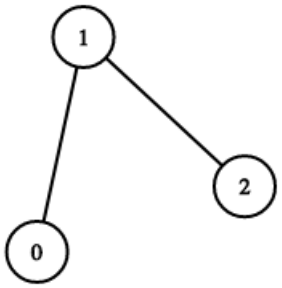


**Figure 2**: A simple 3 node graph.

and, thus, the edge $(u, v)$ crosses the cut if $r_u \neq r_v$. Assuming independence of these two variables, we get that $Pr[r_u \neq r_v] = 1/2$, given that each is equally likely to be zero or one. What we have assumed here is something weaker than 'total' independence, it is pairwise independence of the coin tosses.

Another interesting observation is that because the expected value of the cut is $\frac{|E|}{2}$, there must be at least one partition of the vertices into two sets that creates a cut of value greater or equal to $\frac{|E|}{2}$, since the average of all the random cuts is $\frac{|E|}{2}$. Also, for any graph, the cut size cannot exceed the number of edges present in the graph ($|E|$), so we get the following inequality:

$$\frac{|E|}{2} \leq \text{Max Cut} \leq |E|$$

.
The only source of randomness in the algorithm comes from the unbiased cointosses. We flip $n = |V|$ coins to determine where to place each vertex. If we looked at all the $2^n$ possibilities of every combination of cointosses occuring, we would surely get the max cut of the graph. However, this requires an exponential number of operations and is not useful. In the remainder of the lecture we will try to reduce the number of random bits required so that we definitevely get a good result, such as a max cut of size at least $\frac{|E|}{2}$.

## 2 Pairwise Independence

We start this section by defining a few notions of independence. Let there be $n$ random variables $X_1, \cdots, X_n$, where each belongs to the domain $T$ with $|T| = t$ ($X_i \in T$).


**Figure 3**: The Randomness Generator.

**Definition 2 (Independent)** *The random variables $X_1, \cdots X_n$ are independent, if for every $b_1 \cdots b_n \in T^n$, the following holds: $Pr[X_1 \cdots X_n = b_1 \cdots b_n] = (\frac{1}{t})^n$.*

**Definition 3 (Pairwise Independent)** *The random variables $X_1, \cdots X_n$ are pairwise independent, if for every $i \neq j$ and $b_i, b_j \in T$, the following holds: $Pr[X_i, X_j = b_i, b_j] = (\frac{1}{t})^2$.*

**Definition 4 ($k$-wise Independent)** *The random variables $X_1, \cdots, X_n$ are $k$-wise independent, if for every distinct $i_1, \cdots, i_k$ and $b_{i_1}, \cdots, b_{i_k} \in T^k$, the following holds: $Pr[X_{i_1}, \cdots, X_{i_k} = b_{i_1}, \cdots, b_{i_k}] = (\frac{1}{t})^k$.*

The strongest form of independence is Independence used in Definition 2 and the weakest is pairwise independence. However, in the proof of the expected value of the cut of the randomized max cut algorithm presented (Algorithm 1), we only needed to use pairwise independence.

# 3 Derandomization of the Max Cut Algorithm

## 3.1 Partial Enumeration

We now attempt to derandomize the max cut algorithm (Algorithm 1) and present an algorithm that deterministically finds a cut within a factor of 2 of the optimal cut. One possibility would be to enumerate over the $2^n$ possibilities of the outcomes of the random cointosses. Then we would get the true maximum cut and not an approximation of it. However, this would take exponential time, so we need to test fewer random bits somehow.

Our approach is to do something called partial enumeration. More specifically, we will only look at a subset of the $2^n$ possibilities of the random bits and require that the bits/cointosses satisfy pairwise independence, but not complete independence. If the cointosses satisfy pairwise independence, then Equation 1 we saw earlier still holds and $E[\text{cut}] = \frac{|E|}{2}$. If this holds, enumerating over all the possible cointosses in our set of pairwise independent bits will result in at least one cut that has value greater or equal to $\frac{|E|}{2}$.

As can be seen in Fig. 3, the 'Randomness Generator' is provided $m$ totally independent bits according to Definition 2 and returns $n$ pairwise independent bits according to Definition 3. Once we have these

pairwise independent bits we can pass them all into our original Randomized Max Cut Algorithm (Algorithm 1) and output the cut that produces the maximum cut value among all the ones we have seen. This cut is guaranteed to be within a factor of 2 of the optimal max cut according to the reasoning presented above. The runtime for this algorithm is $O(2^m \times (\text{runtime for Algorithm 1}))$, because we run Algorithm 1 $2^m$ times for every possibility of the totally independent bits $b_1...b_m$.

## 3.2 Randomness Generator

Now we focus on the details of the 'Randomness Generator'. Once we have it we should be done. There are multiple different constructions that work for this generator. One example that is covered as an exercise in Problem Set 2 is that we are given $m$ totally independent bits $b_1...b_m$ and we consider the XOR, in other words the parity, of every possible subset of these bits excluding the empty subset. For this we only need $m = \log n + 1$ to output $n$ pairwise independent bits. The proof is exercise 2 of Problem Set 2.

An Alternative construction would be to focus on the set of integers $\mathbb{Z}_q = [0, 1, 2, ..., q-1]$ where $q$ is prime and use a family of hash functions. We will use $2 \log q$ totally independent random bits to generate $q$ pairwise independent bits.

Consider a family of hash functions $\mathcal{H} = \{h_1, h_2, ...\}$ with every $h_i : N \to M$. We say that this family is a family of pairwise independent hash functions if when $h \in_u \mathcal{H}$:

- for every $x \in N$, then $h(x) \in_u M$
- for every $x_1 \neq x_2 \in N$, then $h(x_1)$ and $h(x_2)$ are independent

An equivalent way to define a pairwise independent hash function family is the following: For every $x_1 \neq x_2 \in N$ and for every $y_1, y_2 \in M$, $Pr_{h \in \mathcal{H}}[h(x_1) = y_1 \wedge h(x_2) = y_2] = \frac{1}{|M|^2}$.

We consider the following hash family. Given $a, b \in \mathbb{Z}_q$, $h_{a,b}(i) = ai + b \mod q$. This function set defines the family $\mathcal{H}_0$ that has $q^2$ elements, one for every pair of values $(a, b)$. Our randomness generator then works as follows:

- Pick $a, b \in \mathbb{Z}_q$
- Compute and save $r_i = ai + b \mod q$
- Output $r_1 r_2 ... r_q$

Notice that if $q = 2$, every $r_i$ is a bit that takes the value 0 or 1. However, our construction is more general. Now all we have to do is to prove that our hash family $\mathcal{H}_0$ is a pairwise independent hash function family. Given two values $i_1, i_2 \in \mathbb{Z}_q$ that are distinct, the hash function family is pairwise independent if:

$$Pr_{a,b \in \mathbb{Z}_q}[h_{a,b}(i_1) = c \wedge h_{a,b}(i_2) = d] = \frac{1}{q^2} \iff Pr_{a,b \in \mathbb{Z}_q}[ai_1 + b \mod q = c \wedge ai_2 + b \mod q = d] = \frac{1}{q^2}$$

.
We now use some linear algebra. The above equation implies that since the probability is equal to $1/q^2$, there is exactly one pair of values $(a, b)$ that make both $ai_1 + b \mod q = c$ and $ai_2 + b \mod q = d$.

Hence, the question is whether there is a unique solution to the following equation when $a$ and $b$ are the variables:

$$\begin{bmatrix} i_1 & 1 \\ i_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} c \\ d \end{bmatrix}$$

This is equivalent to asking whether the first matrix $\begin{bmatrix} i_1 & 1 \\ i_2 & 1 \end{bmatrix}$ is invertible. Its determinant is equal to $i_1 - i_2 \neq 0$, because $i_1, i_2 \in \mathbb{Z}_q$ and $i_1 \neq i_2$, so the determinant is non-singular. Therefore there is a unique solution for $(a, b)$ and we have proven that the family of hash functions $\mathcal{H}_0$ is a pairwise independent hash function family.