Introduction to Algorithms	**Approximation Algorithm for Max-Cut**
CS 4820 Spring 2023	**Handout**

The Max-Cut problem is defined as follows: Given an undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$, find a cut[1] (A, B) that maximizes the cut-size[2]. We assume that G has no self-loops.

As we have seen in HW 7, the decision version of this problem is NP-complete, and a natural question is to design approximation algorithms for the Max-Cut. In this handout, we will design a 2-approximation algorithm for Max-Cut. We briefly mention that there are better approximation algorithms for Max-Cut, but the techniques used are beyond the scope of this class.

In Section 1, we present a randomized 2-approximation algorithm for Max-Cut, and in Section 2, we show how to use efficient constructions of universal hash functions to "derandomize" this algorithm to obtain a (deterministic) 2-approximation algorithm for Max-Cut.

1 A Randomized 2-Approximation Algorithm

We present a simple (randomized) algorithm that achieves approximation ratio of 2. Informally, the algorithm just outputs a *random partition* of the set of nodes. The more formal description is below. We first note some notation that we will use for throughout: we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$, and associate the set of vertices V with $[n]$.

$\mathcal{A}^R(G = (V, E))$:

1. Let $f : [n] \to \{0, 1\}$ be a random function. i.e., for every $i \in [n]$, $f(i)$ is independently and uniformly chosen (from $\{0, 1\}$).
2. Let $A = \{i \in [n] : f(i) = 1\}$, $B = [n] \setminus A$.
3. If $|C(A, B)| \geq m/2$, output (A, B), else repeat (go to (1)).

We note that if the above algorithm terminates, then indeed the output is a 2-approximation (since any cut has size at most m). Thus, we focus on proving that the algorithm terminates. Towards this goal, define X to be the r.v. denoting the cut size (i.e., $|C(A, B)|$), and let $Y = m - X$ be the r.v. denoting the number of edges that are not in the cut. We use the usual trick of writing X as a sum of indicator random variables to estimate the mean.

For every $e \in E$, define $I_e = 1$ if $e \in C(A, B)$ and 0 otherwise. It is clear that $X = \sum_{e \in E} I_e$, and thus by linearity of expectation $\mathbb{E}[X] = \sum_{e \in E} \mathbb{E}[I_e]$. We now estimate $\mathbb{E}[I_e]$, which is exactly the probability that $e \in C(A, B)$. We note that $e \in C(A, B)$ only if $f(u) \neq f(v)$. We claim $\Pr[f(u) \neq f(v)] = 1/2$. This can be seen as follows:

Since f is a random function, $f(u)$ and $f(v)$ are independent, uniform random variables on $\{0, 1\}$. Thus, $\Pr[f(u) \neq f(v)] = \Pr[f(u) = 0, f(v) = 1] + \Pr[f(u) = 1, f(v) = 0] = 1/4 + 1/4 = 1/2$.

It thus follows that $\mathbb{E}[X] = m/2$, and thus $\mathbb{E}[Y] = m/2$. In Step 3 of the algorithm above, it repeats if $Y > m/2$. Let $\mathcal{E}_{bad}$ be the event that $Y > m/2$, and $\mathcal{E}_{good} = \overline{\mathcal{E}_{bad}}$. We now bound the probability of $\mathcal{E}_{bad}$ using the Markov's inequality.

We have, $\Pr[\mathcal{E}_{bad}] = \Pr[Y > m/2] =$ [3] $\Pr\left[Y \geq \frac{m+1}{2}\right] = \Pr\left[Y \geq \mathbb{E}[Y](1 + \frac{1}{m})\right] \leq 1/(1 + \frac{1}{m})$, where the last equality follows by the fact that $\mathbb{E}[Y] = m/2$, and the last inequality is by Markov's bound.

It thus follows $\Pr[\mathcal{E}_{bad}] \leq m/(m + 1)$, and $\Pr[\mathcal{E}_{good}] \geq 1/(m + 1)$, where recall $\mathcal{E}_{bad}$ denotes the probability of repeating (in Step 3), and $\mathcal{E}_{good}$ denotes the probability that the algorithm terminates.

[1] recall a cut is simply a partition of the vertices of G into 2 disjoint subsets

[2] defined as the cardinality of the set $C(A, B) = \{e = (u, v) \in E : u \in A, v \in B\}$

[3] using the fact that Y takes integer values

Let Z denote the number of iterations made by the algorithm. It follows that Z is a geometric random variable with $p \geq 1/(m+1)$. It follows that $\mathbb{E}[Z] = 1/p \leq m+1$.

We note that each iteration takes time $O(m+n)$, and hence the expected running time of the algorithm is $O(m(m+n))$.

2 Derandomization via Hash Functions

2.1 A Universal Hash Function Family

We now devise a (deterministic) 2-approximation algorithm for Max-Cut by using universal hash functions. Let $\mathcal{H} = \{h : [n] \to \{0,1\}\}$ be a universal hash function family. As discussed in class and the textbook, one way of constructing $\mathcal{H}$ is the following: let $w = \lceil \log n \rceil$. Note that each $x \in [n]$ can be uniquely identified by an element of $\{0,1\}^w$ (bitstrings of length w), just using the bit representation of x. For the description of the hash functions, we think of $x \in [n]$ as an element in $\{0,1\}^w$.

For every $a \in \{0,1\}^w$, we define a hash function $h_a \mathcal{H}$ as $h_a(x) = \sum_{j=1}^{w} a_j x_j \pmod 2$. It was proved in class (also see the K&T book) that this is a universal hash function family[4]. Thus, for any $x \neq y$, we have the property that for a randomly chosen H from $\mathcal{H}$, $\Pr[H(x) = H(y)] \leq 1/2$.

Finally, we note that the size of the family $\mathcal{H}$ is $\leq 2n$, since each hash function is indexed by a string in $\{0,1\}^w$ and $w = \lceil \log n \rceil$.

2.2 Another randomized approximation algorithm: a step towards derandomization

Consider the following algorithm that is different from $\mathcal{R}$ in the following way: instead of picking a random f, the new algorithm picks a random hash function.

$\mathcal{A}^H(G = (V, E))$:

1. Let $\mathcal{H}$ be the hash family constructed in the previous section.
2. Let $h : [n] \to \{0,1\}$ be a randomly chosen hash function from $\mathcal{H}$.
3. Let $A_h = \{i \in [n] : h(i) = 1\}$, $B_h = [n] \setminus A$.
4. If $|C(A_h, B_h)| \geq m/2$, output (A_h, B_h), else repeat (go to (2)).

As before, it is clear that if the algorithm terminates, the output is indeed a 2-approximation. To analyze the performance of this new algorithm, we import some definitions from the analysis of $\mathcal{A}^R$: Define X to be the r.v. denoting the cut size (i.e., $|C(A,B)|$), and let $Y = m - X$ be the r.v. denoting the number of edges that are not in the cut. For every $e \in E$, define $I_e = 1$ if $e \in C(A,B)$ and 0 otherwise. It is clear that $X = \sum_{e \in E} I_e$, and thus $\mathbb{E}[X] = \sum_{e \in E} \mathbb{E}[I_e]$. We now estimate $\mathbb{E}[I_e]$, which is the probability that $e \in C(A,B)$. This requires a different argument from before.

We note that $e \in C(A,B)$ only if $h(u) \neq h(v)$. We claim $\Pr[h(u) \neq h(v)] \geq 1/2$. This is immediate just by using the fact that $\mathcal{H}$ is universal, and hence as noted in the previous section $\Pr[H(u) = H(v)] \leq 1/2$. Thus, it follows that $\mathbb{E}[X] \geq m/2$, and thus $\mathbb{E}[Y] \leq m/2$.

The expected running time of $\mathcal{A}^H$ can now be bounded by $O((m+n)m)$ exactly the same way as $\mathcal{A}^R$.

We conclude by noting the following corollary that is immediate from the bound that $\mathbb{E}[X] \geq m/2$.

Corollary 2.1. *There exists $h \in \mathcal{H}$ such that $|C(A_h, B_h)| \geq m/2$.*

[4] in fact, a more general construction was presented

3 A deterministic 2-approximation algorithm for Max-Cut

The idea is to simply brute-force over all hash functions $\mathcal{H}$ to find a *good* hash function that achieves cut-size $m/2$ (whose existence we proved in the previous section), rather than randomly choosing hash functions from $\mathcal{H}$ as done in $\mathcal{A}^H$. Formally, the algorithm is presented below.

$\mathcal{A}^D(G = (V, E))$:

1. Let $\mathcal{H}$ be the universal hash family constructed from Section 2.1.
2. for all $h \in \mathcal{H}$, compute $|C(A_h, B_h)|$, where $A_h = \{i \in [n] : h(i) = 1\}$, $B_h = [n] \setminus A$.
3. Output (A_h, B_h) that has largest cut-size among all $h \in \mathcal{H}$.

We note that $|\mathcal{H}|$ is $\leq 2n$ (see Section 2.1). Further, computing $C(A_h, B_h)$ can be done in time $O(m + n)$. Thus, the algorithm runs in time $O(n(m + n))$. Further, correctness is direct from the Corollary 2.1.