Lecture Notes CS:5350 Introduction to approximation algorithms
Lecture 19: Oct 29, 2019
Scribe: Mudathir Mahgoub

1 MaxCut problem

Input An undirected graph $G = (V, E)$,

Output A partition of V into two subsets V_1, V_2 such that the number of edges between V_1 and V_2 is maximized.

Example

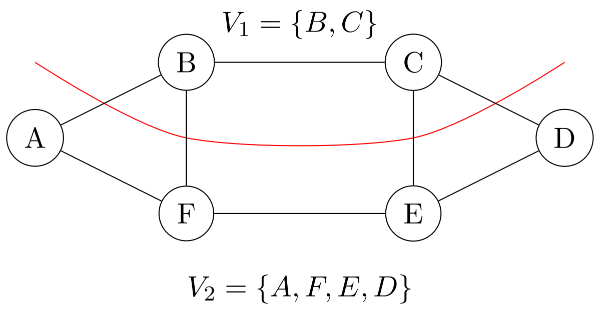


Figure 1: In this example, the partition $(V_1 = \{B, C\}, V_2 = \{A, F, E, D\})$ is a max-cut with size = 4.

1.1 Remarks

- MaxCut problem is NP-complete.
- A weighted MaxCut is a general version of the problem where edges have weights.
- We can get a $\frac{1}{2}$-approximation to MaxCut *in expectation* using a randomized algorithm.

1.2 RandomizedMaxCut algorithm

```
1: for each vertex $v \in V$ do
2:     Place $v$ in $V_1$ or $V_2$ uniformaly at random (i.e., probability = $\frac{1}{2}$)
3: return $(V_1, V_2)$
```

1.3 Analysis of the approximation

For each edge $e = \{u, v\}$, define the random variable X_e as

$$X_e = \begin{cases} 1 & \text{if } e \text{ crosses the cut } (V_1, V_2) \\ 0 & \text{otherwise.} \end{cases}$$

Then the random variable $X = \sum_{e \in E} X_e$ represents the number of edges that cross the cut. By linearity of expectation, we have

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{e \in E} X_e\right] = \sum_{e \in E} \mathbb{E}[X_e] = \sum_{e \in E} Prob[X_e = 1] = \sum_{e \in E} \frac{1}{2} = \frac{|E|}{2} = \frac{m}{2}$$

where m is the number of edges. Here $X_e = 1$ means the edge e crosses the cut (*i.e.,* one endpoint of $\{u, v\}$ is in V_1 and the other is in V_2). Since the placement of vertices are independent events, we have

$$Prob[X_e = 1] = Prob[(u \in V_1 \text{ and } v \in V_2) \text{ or } (v \in V_1 \text{ and } u \in V_2)] = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}.$$

Since the number of edges in an optimal solution for MaxCut is at most m, and the randomized algorithm above provides a solution with $\frac{m}{2}$ edges *in expectation.* we can say the randomized algorithm is a $\frac{1}{2}$-approximation to MaxCut *in expectation.*

The expectation analysis above does not give a lower bound for the probability that the cut returned by RandomizedMaxCut is a $\frac{1}{2}$-approximation. We can get a lower bound for the probability that the returned cut is a $\frac{1}{4}$-approximation by amplifying the algorithm (*i.e.,* repeating it many times) as shown below.

```
1: for  $i \leftarrow 1$ to $k$ do
2:     run RandomizedMaxCut algorithm
3: return  the largest cut
```

Theorem. *This algorithm yields a $\frac{1}{4}$-approximation with high probability (i.e., $\geq 1 - \frac{1}{n}$) in running time $O(m \log n)$.*

Proof. Let C_i for $i = 1, 2, \cdots, k$ be the random variable denoting the size of the cut produced by the i^{th} iteration. Let $Y_i = m - C_i$, which represents the random variable for the number of edges outside the cut. By linearity of expectation, we have:

$$\mathbb{E}[Y_i] = \mathbb{E}[m] - \mathbb{E}[C_i] = m - \mathbb{E}[C_i] = m - \frac{m}{2} = \frac{m}{2}.$$

Given that our goal here is to obtain a $\frac{1}{4}$-approximation, a "bad event" in iteration i is to get a cut with size $< \frac{m}{4}$ (*i.e.,* $C_i < \frac{m}{4}$). Getting this bad event in all iterations is equivalent to $\bigwedge_{i=1}^{k} \left(C_i < \frac{m}{4}\right)$.

We compute the probability of getting this bad event in all iterations as follows:

$$\begin{aligned}
Prob\left[\bigwedge_{i=1}^{k}\left(C_i < \frac{m}{4}\right)\right] &= Prob\left[\bigwedge_{i=1}^{k}\left(Y_i > \frac{3m}{4}\right)\right] \\
&= \prod_{i=1}^{k} Prob\left[Y_i > \frac{3m}{4}\right] && \text{(by independence of iterations)} \\
&< \prod_{i=1}^{k} \frac{\mathbb{E}[Y_i]}{\frac{3m}{4}} && \text{(by Markov's inequality)} \\
&= \prod_{i=1}^{k} \frac{\frac{m}{2}}{\frac{3m}{4}} = \prod_{i=1}^{k} \frac{m}{2}\frac{4}{3m} = \prod_{i=1}^{k} \frac{2}{3} = \left(\frac{2}{3}\right)^k .
\end{aligned}$$

Choosing $k = C \log n$ for large enough constant C, gives $Prob\left[\bigwedge_{i=1}^{k}\left(C_i < \frac{m}{4}\right)\right] < \left(\frac{2}{3}\right)^k \leq \frac{1}{n}$. Therefore with probability at least $1 - \frac{1}{n}$, the bad event does not happen and the algorithm produces a cut of size $\geq \frac{m}{4}$. $\square$

2 Approximation Algorithms

In the 1970s, researchers in complexity theory proved many problems to be NP-complete. Computing exact solutions to these problems is intractable (*i.e.,* extremely inefficient in practice). Therefore, many heuristics were developed to find approximate solutions efficiently.

Definition 1 (α-approximation for minimization problems)**.** *Let* Π *be a minimization problem, and* $\alpha \geq 1$ *a constant. An algorithm* A *is an* α*-approximation algorithm for* Π *if :*

1. *A runs in polynomial time.*
2. *For every input* I *of* Π

$$cost(\underbrace{A(I)}_{\substack{\text{solution produced by}\\ \text{algorithm } A \text{ on input } I}}) \leq \alpha \times cost(\underbrace{OPT(I)}_{\substack{\text{optimal solution}\\ \text{on input } I}}). \qquad (1)$$

Definition 2 (α-approximation for maximization problems)**.** *Let* Π *be a maximization problem, and* $\alpha \leq 1$ *a constant. An algorithm* A *is an* α*-approximation*[1] *algorithm for* Π *if :*

1. *A runs in polynomial time.*
2. *For every input* I *of* Π

$$cost(A(I)) \geq \alpha \times cost(OPT(I)).$$

In this class, we focus on approximation algorithms for NP-complete problems.

[1]The constant α is called the approximation ratio.

3 Minimum Vertex Cover (MVC)

Input An undirected graph $G = (V, E)$,

Output A vertex subset $S \subseteq V$ of minimum size such that for every edge $\{u, v\} \in E$, at least one of u or v is in S.

Example

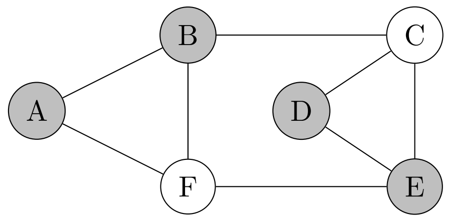


Figure 2: An example of a minimum vertex cover.

In Figure 2, the subset $S = \{A, B, D, E\} \subseteq V$ is an optimal (minimum) vertex cover that touches every edge in the graph. In this example S has minimum size because the edges in the triangle A, B, F requires 2 vertices to be covered, and similarly 2 vertices are required to cover the edges in the triangle C, D, E. So the minimum vertex cover has size 4.

3.1 Approximation algorithms for MVC

MVC problem was proved to be NP-complete [1]. We will discuss two approximation algorithms for MVC, one that uses a greedy heuristic, and one that uses a maximal matching to find a vertex cover. But first, we need a method to compare the approximated solution with the optimal one. In the inequality (1), the main bottleneck is to characterize $OPT(I)$ for an arbitrary input I. Since our goal is to find an approximation, we usually work with a lower bound (LB) on the optimal solution that is easier to characterize. This lower bound satisfies this inequality:

$$\text{cost}(LB(I)) \leq \text{cost}(OPT(I)) \tag{2}$$

Therefore instead of using inequality (1) directly, we can compare the cost of our solution to the lower bound and show:

$$\text{cost}(A(I)) \leq \alpha \times \text{cost}(LB(I)) \leq \alpha \times \text{cost}(OPT(I)) \tag{3}$$

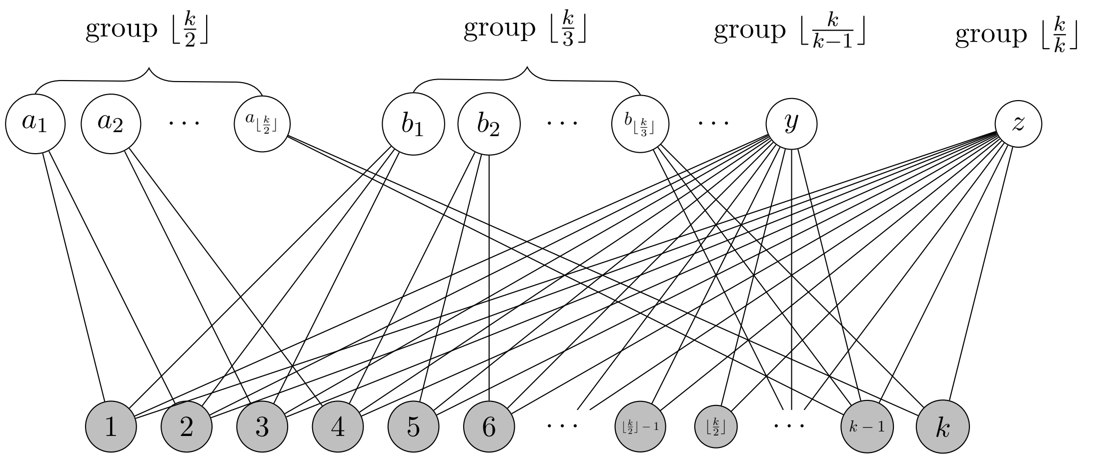
3.2 Greedy heuristic for MVC

Find a vertex cover by repeatedly picking a vertex with the highest degree in the currently active graph. This heuristic suggests the following algorithm:

```
1: $S = \phi, E' = \phi$ // $S$ maintains solution, $E'$ maintains covered edges
2: while $E \setminus E'$ is not empty do
3:     Choose a vertex $v \in V \setminus S$ with the highest degree in $G' = (V \setminus S, E \setminus E')$
4:     add $v$ to $S$
5:     add all edges incident on $v$ from $E \setminus E'$ to $E'$
6: return $S$.
```

It is clear that this algorithm runs in polynomial time $O(|V|^2)$. Actually it is not too hard to implement it in $O\left((|V| + |E|)\log|V|\right)$ if a *max heap* is used to store the vertices, where the key of each vertex is its degree. This satisfies the first condition in definition 1. So how good is the approximation provided by this algorithm? The following example in Figure 3 shows that the cost of this approximation is $\Omega(|S^*|\log n)$ where S^* is the optimal solution for MVC, and n is the number of vertices. Therefore this greedy algorithm does not satisfy the second condition in definition 1 for any constant $\alpha \geq 1$.

Example[2]



group $\lfloor\frac{k}{2}\rfloor$
group $\lfloor\frac{k}{3}\rfloor$
group $\lfloor\frac{k}{k-1}\rfloor$
group $\lfloor\frac{k}{k}\rfloor$
a_1
a_2
$a_{\lfloor\frac{k}{2}\rfloor}$
b_1
b_2
$b_{\lfloor\frac{k}{3}\rfloor}$
y
z
1
2
3
4
5
6
$\lfloor\frac{k}{2}\rfloor - 1$
$\lfloor\frac{k}{2}\rfloor$
$k-1$
k

Figure 3: The optimal solution is the set of gray vertices with cardinality k. The greedy algorithm returns the set of white vertices with cardinality $\Omega(\log|V|k)$.

In Figure 3, the graph has $k-1$ groups of white vertices at the top, and k gray vertices at the bottom. The white vertices in group $\lfloor\frac{k}{2}\rfloor$ each have degree 2, and are connected to disjoint gray vertices. Likewise the white vertices in group $\lfloor\frac{k}{3}\rfloor$ each have degree 3, and are connected to disjoint gray vertices. In general, the vertices in group $\lfloor\frac{k}{i}\rfloor, i \in \{2, 3, \cdots, k\}$ each have degree i, and are connected to disjoint gray vertices. Note that the degree of each gray vertex is at most $k-1$, because it has at most one edge to each white group.

Initially the greedy algorithm chooses vertex z which has the highest degree. After removing z and its edges, each gray vertex has degree at most $k-2$. Therefore, the algorithm chooses vertex

[2] *Introduction and the Minimum Vertex Cover Problem*, `http://www.eng.tau.ac.il/algs/ALG09/intro-vc.pdf`

y which has degree $k-1$. Continuing this process, the greedy algorithm returns the set of white vertices as a solution for MVC. The total number of white vertices is:

$$\begin{aligned}
\sum_{i=2}^{k} \lfloor \frac{k}{i} \rfloor &\geq \sum_{i=2}^{k} \frac{k}{i+1} \\
&= k \sum_{i=2}^{k} \frac{1}{i+1} = k \sum_{i=1}^{k-1} \frac{1}{i} \\
&= kH_{k-1} && (H_{k-1} \text{ is a Harmonic number}) \\
&= \Omega(k \log k)
\end{aligned}$$

The optimal solution is the set of gray vertices which has size k. Since $n = k + \Omega(k \log k)$, then $\log n = \Omega(\log k)$, which implies the greedy algorithm returns an approximation with cost $\Omega(|S^*| \log n)$.

3.3 Maximal matching approximation

Recall that a matching of a graph $G = (V, E)$ is a set of edges ($M \subseteq E$) where the edges do not touch each other. M is a maximal matching if no edge in $E \setminus M$ can be added to M without touching other edges in M. The following lemma shows that the size of any matching M is a lower bound for the size of any vertex cover S.

Lemma 1. *Let M be a matching of $G = (V, E)$. Let S be a vertex cover of G. Then $|M| \leq |S|$.*

Proof. Suppose M is a matching and S is a vertex cover. If M is empty, then it is trivial that $|M| = 0 \leq |S|$. Suppose M is not empty, and $\{u, v\}$ is an arbitrary edge in M. Since M is a matching, then for any edge $\{a, b\} \neq \{u, v\}$, we have $\{a, b\} \in M \Rightarrow \{a, b\} \cap \{u, v\} = \phi$. This means that $\{u, v\}$ is the only edge in M that contains either u or v. Since S is a vertex cover, then either u or v is in S. Therefore for each edge in M, at least one of its vertices is in S, which implies $|M| \leq |S|$. □

Example

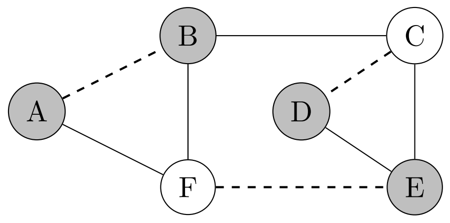


Figure 4: An example of a maximal matching and a minimum vertex cover.

In Figure 4, the set $M = \{\{A, B\}, \{C, D\}, \{E, F\}\}$ is a maximal matching for the graph G, and $S = \{A, B, D, E\}$ is a vertex cover. Note that $|M| = 3 \leq 4 = |S|$. The previous lemma suggests the following algorithm:

```
1: $S = \phi$
2: Find a maximal matching $M$.
3: **for** each edge $\{u, v\} \in M$ **do**
4:     add both $u$ and $v$ to $S$.
5: **return** $S$.
```

For the maximal matching in Figure 4, this algorithm returns $S = V$ which is a vertex cover, but not minimum.

References

[1] Jeff Erickson. *Algorithms.* 2019.