15-854: Approximations Algorithms **Lecturer:** Anupam Gupta
Topic: Max-cut, Hardness of Approximations **Date:** 9/14
Scribe: Haowen Chan

2.1 The Max-Cut Problem

Given an undirected graph $G = (V, E)$, a **cut** in G is a subset $S \subseteq V$. Let $\overline{S} = V \backslash S$, and let $E(S, \overline{S})$ denote the set of edges with one vertex in S and one vertex in $\overline{S}$. We will use the term "cut" to refer variously to the partition $(S, \overline{S})$ of the vertices, and also the set of edges $E(S, \overline{S})$ crossing between S and $\overline{S}$; typically, we will use this term in a way that is unambiguous, and will disambiguate where necessary.

The Max-Cut problem is to find the cut S that maximizes $|E(S, \overline{S})|$. In the weighted version of Max-Cut, we are also given a edge weight function $w : E \to \mathbb{R}$, and the problem is to find a cut with maximum weight. The Max-Cut problem is NP-hard, and is interesting to contrast with the Min-Cut problem, which is solvable in polynomial time (e.g., by reducing to the s-t min-cut problem, which is dual to the Max-Flow problem, and is solvable by algorithms such as one by Edmonds and Karp [1].

2.1.1 Approx Max-Cut using Local Search

THe general idea in local search algorithms is the same: Start with a solution, find an improving step to make a better solution, and repeat until stuck. In more detail:

1. Start with some arbitrary $S_0 \subseteq V$.
2. In step i, let our current cut be denoted by S_i.
3. If $\exists v \in S_i$ s.t. moving v to $\overline{S_i}$ increases the resulting cut, then: Let $S_{i+1} \leftarrow S_i \backslash \{v\}$, go to step 2.
4. Analogous statement for $\overline{S_i}$, i.e. If $\exists v \in \overline{S_i}$ s.t. moving v to S_i increases the resulting cut, then: Let $S_{i+1} \leftarrow S_i \cup \{v\}$, go to step 2.
5. If no vertices can be moved to improve S_i, terminate and output $(S_i, \overline{S_i})$.

Theorem 2.1.1 *The local search algorithm for* Max-Cut *terminates.*

Proof: Each step must increase $|E(S_i, \overline{S_i})|$ by at least 1. Since $|E(S_i, \overline{S_i})| \leq |E| = m$, the algorithm terminates in at most m steps. ■

Lemma 2.1.2 *At a local optimum, the cut has at least* $\frac{m}{2}$ *edges.*

Proof: Let the cut at the local optimum be S. Consider an arbitrary vertex v with degree d_v. Suppose $v \in S$; an identical argument applies if $v \in \overline{S}$. Some of the edges adjacent to v have their other endpoint in S (and hence don't cross the cut $(S, \overline{S})$), while the other edges have their other endpoint in $\overline{S}$ (i.e., they cross the cut). At a local optimum, the number of edges crossing the cut must be greater than the number of edges not crossing the cut, otherwise we could move v into $\overline{S}$ to get a better cut. Hence, number of edges adjacent to v that are crossing the cut $\geq \frac{d_v}{2}$.

$$
\begin{aligned}
\text{Total size of cut} &= \tfrac{1}{2}\textstyle\sum_{v\in V} v\text{'s edges crossing the cut} \\
&\geq \tfrac{1}{2}\textstyle\sum_{v\in V} \frac{d_v}{2} \\
&= \tfrac{2m}{4} \quad \text{since} \textstyle\sum_{v\in V} d_v = 2m \\
&= \tfrac{m}{2}
\end{aligned}
$$

■

Recall that for minimization problems, we had defined the approximation ratio to be

$$\rho_A = \max_{\text{instances } I} \frac{c(A(I))}{\mathsf{opt}(I)}$$

For maximization problems, we can define the approximation ratio similarly:

$$\rho_A = \max_{\text{instances } I} \frac{\mathsf{opt}(I)}{c(A(I))}$$

Note that in both cases, a lower value of ρ_A implies A is a better algorithm.

A general note: usually, it is difficult to estimate the value of the optimal solution (since the problems are NP-hard), and hence to work out the worst case performance of A compared to the optimum. Hence, for maximization problems, we look for upper bounds on $\mathsf{opt}(I)$, which we denote by $\mathsf{UBD}(I)$. Note that this implies that

$$A(I) \leq \text{opt}(I) \leq UBD(I). \tag{2.1.1}$$

Now, if we can show that $\frac{UBD(I)}{c(A(I))} \leq \alpha$ for all instances I, then we have $\rho_A \leq \alpha$.

Theorem 2.1.3 *The local search algorithm described above is a 2-approximation algorithm.*

Proof: It is easy to see that the number of edges m is an upper bound on MAX-CUT (G), since the maximum cut can't have more edges than exist in the graph. Since the cut produced by the local search algorithm always has at least $\frac{m}{2}$ edges, we have shown that for all instances I,

$$\frac{\mathsf{opt}(I)}{c(A(I))} \leq \frac{\mathsf{UBD}(I)}{c(A(I))} \leq \frac{m}{m/2} = 2. \tag{2.1.2}$$

■

Note that if we had a weighted instance of MAX-CUT, then the local-search algorithm could potentially run for $\Omega(\sum_e w_e)$ steps. (Can you construct such an example?) Since the *size* of the instance is measured in the number of bits used to represent the instance, which is $O(\sum_e \log w_e)$, this would be an exponential time algorithm. We will now present two other algorithms that can be modified to handle arbitrary weights, and run in polynomial time in the *size* of the instance.

2.1.2 Max-Cut using Greedy Algorithm

Fix an ordering on the vertices $v_1, v_2, \ldots, v_n$. Start with two empty bins $S, \overline{S}$. Pick the first vertex v_1, put it in S. For each subsequent vertex, put it in the bin such that $|E(S, \overline{S})|$ is maximized.

Theorem 2.1.4 *The greedy algorithm is a 2-approximation algorithm for* MAX-CUT.

Proof: Consider any edge in the graph. One vertex must come first in the ordering. Call the later vertex *responsible* for the edge.

Let r_i be the number of edges that v_i is responsible for. Since every edge has exactly one responsible vertex, $\sum_{i=1}^{n} r_i = m$.

Claim: When v_i is added, at least $\frac{r_i}{2}$ edges are added to the cut. Because: each of the other vertices adjacent to the r_i edges that v_i is responsible for have already been assigned to either S or $\overline{S}$. The set which contains the most endpoint vertices of these r_i edges must contain at least $\frac{r_i}{2}$ endpoint vertices (i.e. it cannot be that both sets contain less than the average number of vertices per set). By the greedy algorithm, v_i will the added to the other set, thus adding at least $\frac{r_i}{2}$ edges to the cut.

$$\begin{aligned}
\text{Number of edges in final cut} &= \textstyle\sum_{i=1}^{n} \text{number of edges added by each responsible vertex} \\
&\geq \textstyle\sum_{i=1}^{n} \frac{r_i}{2} \\
&\geq \frac{m}{2}
\end{aligned}$$

Since $\mathsf{opt}(I) \leq m$, the algorithm is a 2-approximation. ■

2.1.3 Using a Randomized Algorithm

Here's an algorithm that seems to work by sheer luck: we assign each vertex to either S or $\overline{S}$ uniformly at random (i.e., there is a probability $\frac{1}{2}$ of each happening).

Lemma 2.1.5 $\mathbf{E}[$*num. edges in cut*$] = \frac{m}{2}$

Proof: Let us number the edges 1 to m. Define an indicator variable X_i for each edge i s.t. $X_i = 1$ if the edge crosses the cut and $X_i = 0$ if the edge doesn't cross the cut. Since we assigned the vertices independently randomly, probability that both endpoints of i are in the same set = probability that the endpoints are in different sets $= \frac{1}{2}$. Hence, $\mathbf{E}[X_i] = \frac{1}{2}$. Expected total number of edges crossing the cut $= \sum_{i=1}^{m} \mathbf{E}[X_i] = \frac{m}{2}$ by linearity of expectation. ■ Hence the random algorithm is a 2-approx algorithm on expectation.

2.1.4 Improving the Algorithm

2.1.4.1 How good was the upper bound UBD$(I) = m$?

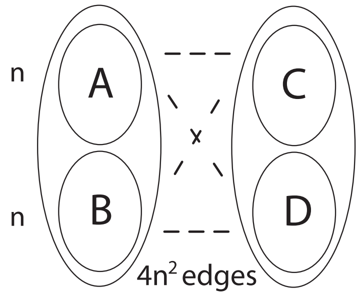
It is usually a good idea to think about how good the lower bounds are, when trying to look for better approximation ratios. In this case, we can show instances where $\mathsf{opt}(I) \approx \frac{m}{2} = \frac{1}{2}\mathsf{UBD}(I)$.

Consider K_{2n}, the complete graph on $2n$ nodes: this has $m = n(2n-1)$ edges. For any cut with $|S| = i$, $|\overline{S}| = 2n - i$, each of the i vertices in S has $2n - i$ edges going across the cut to each of the vertices in $\overline{S}$. Hence the total number of edges in the cut is $i(2n-i)$. This is maximized when $i = n$, and hence the largest cut in this graph has n^2 edges.

Using $\mathsf{UBD} = m$, we have $\frac{UBD}{OPT} = 2 - \frac{1}{n}$, which gets arbitrarily close to 2 as n gets large. And hence, with this upper bound, even the optimal algorithm could never be considered better than a 2-approximation. Hence no approximation algorithm can be better than 2-approx with an upper bound of m, which indicates that we may want to look for better lower bounds.

2.1.4.2 How good are these algorithms?

Of course, there may be instances where the algorithm performs poorly as well, and one needs to find these examples. For the MAX-CUT problem, we will consider $K_{2n,2n}$, the complete bipartite graph with $2n$ nodes on each side. Clearly, the largest cut has $4n^2$ edges crossing it. Let us divide each side into equal subsets of size n as below.



The Local Search Algorithm. Consider a cut $(A \cup C, B \cup D)$. Each vertex has exactly n edges crossing the cut (and n edges not crossing the cut). Hence, no benefit is gained by moving any vertex from one side of the cut to another, and this will be a local optimum. Since the total edges in this locally optimal cut $= 2n^2$, we know that the local search algorithm cannot be better than 2-approximate, even with an improved upper-bound on the optimum.

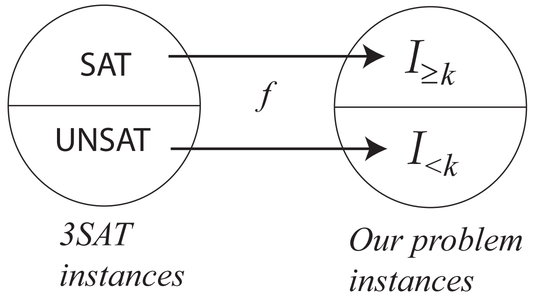
The Greedy Algorithm. Consider processing the vertices in round-robin order from the sets A,B,C,D. i.e. Let $A = a_1, \ldots, a_n$, $B = b_1, \ldots, b_n$, $C = c_1, \ldots, c_n$, $D = d_1, \ldots, d_n$. Process the vertices in the order a_1,b_1,c_1,d_1,a_2,b_2,c_2,d_2,$\ldots$, d_n.

Then it is possible to achieve a cut $(A \cup C, B \cup D)$. We keep alternating the set in which we place each vertex that is being processed; at any point in time each vertex that is being processed is indifferent between being placed in S or $\overline{S}$. Hence the greedy algorithm also achieves a cut of $2n^2$ edges in the worst case for this instance.

The Best-known Approximation Algorithm for Max-Cut In 1994, Goemans and Williamson [2] used a Semidefinite programming relaxation of the MAX-CUT problme to obtain an performance guarantee of $\rho_{GW} = \frac{1}{0.87856}$. It is conjectured that ρ_{GW} may be the best possible approximation assuming $P \neq NP$.

2.2 Proving Inapproximability

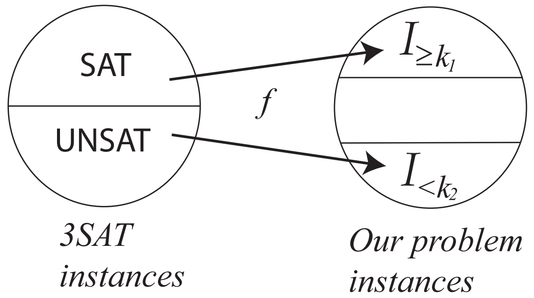
Recall that to prove some (decision) problem is NP-hard, we reduce some NP-hard problem to the problem at hand. For example, consider the decision Max-Cut problem: For a given graph G and given parameter k, does there exist a cut of size $\geq k$ in G? To prove that this problem is NP-hard, we could perform a polynomial time reduction f from some NP-hard problem (e.g. 3SAT).



In the above figure, $I_{\geq k}$ denotes graph instances with a solution of value k or more, while $I_{<k}$ denotes graph instances with a solution of value less than k. I.e., the set $I_{\geq k}$ would contain all graphs with a max-cut of k edges or more, while $I_{<k}$ would contain all the graphs with a max-cut of less than k edges.

The reduction f maps an instance of 3SAT into a graph G such that the 3SAT expression is satisfiable iff $G \in I_{\geq k}$. The existence of this mapping shows that deciding membership in $I_{\geq k}$ is at least as hard as deciding satisfiability in 3SAT, hence it is NP-hard.

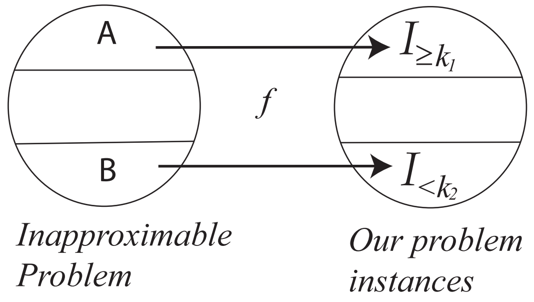
So far this approach can show that a problem is NP-hard. To show that the problem is inapproximable to some factor, we must now show that our mapping can map to problem instances where the max-cut satisfies a certain "gap-property": we must exhibit a reduction f that maps an instance of 3SAT into a graph G such that $G \in I_{\geq k_1}$ iff the 3SAT instance is satisfiable and $G \in I_{<k_2}$ iff the 3SAT instance is unsatisfiable. (Here $k_1 > k_2$, and the "gap" refers to the gap between the values k_1 and k_2.)



Note that a $\frac{k_1}{k_2}$ approximation algorithm for Max-Cut can determine membership in either $I_{\geq k_1}$ or $I_{<k_2}$. This is because given a graph in $I_{\geq k_1}$, the algorithm can always produce a cut with at least $k_1/\frac{k_1}{k_2} = k_2$ edges, while graphs in $I_{<k_2}$ always yield cuts of less than k_2 edges since the

algorithm cannot do better than the optimum. Hence, the existence of such a reduction f proves that approximating the problem (in this case, MAX-CUT) to within $\frac{k_1}{k_2}$ is NP-hard.

In other cases, we can exhibit a reduction of a known inapproximable problem to our problem to show that our problem is also inapproximable.



A canonical example of an inapproximable problem is the MAX-3SAT problem. Recall that a 3SAT formula φ is the conjunction (AND) of many *clauses*, and each clause is the disjunction (OR) of three *literals*. The MAX-3SAT problem seeks to find a truth assignment that maximizes the number of satisfied clauses in φ. It is clear that the MAX-3SAT problem is NP-hard, since it can distinguish between satisfiable and unsatisfiable formulae ϕ.

Note that MAX-3SAT is $\frac{8}{7}$ approximable by a simple randomized algorithm: set the truth value of each variable to be true or false uniformly at random. Since each clause requires just one literal to be true for the clause to be satisfied, and each literal has a probability of $\frac{1}{2}$ to be true, the probability that at least one out of the three literals in a clause is satisfied is $\frac{7}{8}$. By linearity of expectations, the expected number of clauses that are satisfied is $\frac{7}{8}$ of all the clauses in the expression. The following theorem of Håstad shows that this may be the best possible approximation algorithm for the problem.

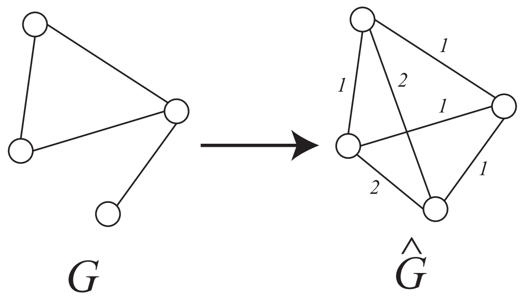
Theorem 2.2.1 (Håstad [3, 4]) *MAX-3SAT approximable to less than $\frac{8}{7}$ implies $P = NP$.*

2.2.1 An Inapproximability Result

Let us now prove an inapproximability result using the general approach we have sketched in the preceeding discussion. We will prove this result for the K**-center** problem: Given a complete graph $G = (V, \binom{V}{2})$ with a length d_e on each edge e satisfying the triangle inequality (i.e., satisfying $d_{uv} + d_{vw} \geq d_{uw}$), pick set of vertices S with $|S| = K$, such that S minimizes $\max_{v \in V} d(v, S)$. (Here, $d(v, S)$ is defined as $\min_{x \in S} d_{vx}$.) Intuitively, can we pick K vertices in G such that every vertex in G is "close" to one of the K vertices picked?

Theorem 2.2.2 *For any $\epsilon > 0$, the K-center problem is not $(2 - \epsilon)$-approximable unless $P = NP$.*

Proof: We will prove this by a reduction from the dominating set problem. Recall that the NP-hard DOMINATING SET problem is the following: Given a graph $G = (V, E)$ and integer K, is there a set D of at most K nodes such that each vertex not in the set D is adjacent to some node in D? We will exhibit a reduction of instances G of the dominating set problem to instances $\hat{G}$ of the K-center problem, such that "Yes" instances for Dominating set are mapped to K-center instances with solution value 1, and "No" instances are mapped to instances with solution value 2.



Given an instance $(G = (V, E), K)$ of the dominating set problem, define a complete graph $\hat{G}$ on the vertex set V. Define a length function on the edges: if (i, j) is an edge in E, define the length d_{ij} in $\hat{G}$ to be 1; if (i, j) is not an edge in E, let the length of (i, j) in $\hat{G}$ be $d_{ij} = 2$. Note that since all distances in $\hat{G}$ are either 1 or 2, the solution value of any K center instance must be one of these two numbers.

Note that if we have a "Yes" instance of dominating set, then there is a dominating set D of size K: now placing a center at each of the vertices in D would ensure that each vertex in $V \setminus D$ would be at unit distance to D in $\hat{G}$, and hence the solution value of the K-center instance would be 1. Conversely, if there is a solution to the K-center instance with value 1, then there is a subset $S \subseteq V$ such that all other vertices are at distance 1 from it. By construction, each of these vertices must have an edge to some vertex in S, and hence S is a dominating set of size K.

Let us recall that any polynomial time algorithm that achieves $(2 - \epsilon)$-approximation can disambiguate whether $\hat{G}$ has a K-center with distance 1 or distance 2. Indeed, suppose such a $(2 - \epsilon)$-approximation algorithm exists where $\epsilon > 0$. Then such an algorithm, when applied to an instance of $\hat{G}$ where the optimal solution is 1, will always return a solution with a value of $1(2 - \epsilon) < 2$. On the other hand, it can never do better than optimal on any instance where the optimal value is 2, so in such a case it will always return a solution of at most 2. Hence the algorithm returns a solution with value less than 2 iff the problem instance $\hat{G}$ had a K-center of distance 1. This allows us to solve the decision dominating set problem in polynomial time since G has a dominating set of K nodes iff $\hat{G}$ has a K-center with distance 1. Hence, for any $\epsilon > 0$, K-center cannot be approximated to $2 - \epsilon$ if $P \neq NP$. ■

References

[1] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network problems. *J. Assoc. Computing Machinery*, 19:248–264, 1922.

[2] M. Goemans and D. Williamson. .879 approximation algorithms for max cut and max 2sat. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 422–431, 1994.

[3] Håstad J. Some optimal inapproximability results. In *Proceedings of the 37th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1997.

[4] Håstad J. Some optimal inapproximability results. *Journal of the ACM*, 48:768–859, 2001.