Lecture 7

Lecturer: Madhu Sudan *Scribe: Prahladh Harsha*

In today's lecture, we shall wrap up our discussion on MULTI-CUT, briefly summarize the different types of approximation algorithms we have seen so far, discuss approximation preserving reductions and finally introduce Gap problems.

1 Multi-Cut

To recall, the MULTI-CUT was defined as follows:
Input: A graph $G = (V, E)$ and k pairs of vertices $(s_1, t_1), \ldots (s_k, t_k)$
Goal: Find a subset of the edges $F \subseteq E$ such that the graph $G' = (V, E \backslash F)$ has no paths from s_i to t_i for each pair (s_i, t_i).
Objective: Minimize $|F|$

In the last lecture, we obtained a LP-relaxation for the above problem by formulating the following Metric Embedding problem.

Metric Embedding

Variables:	$\{d_{i,j}\}_{i,j \in V}$
Constraints:	$d_{i,j} = d_{j,i}, \quad \forall i, j \in V$
	$d_{i,i} = 0, \quad \forall i \in V$
	$d_{i,j} + d_{j,k} \geq d_{i,k}, \quad \forall i, j, k \in V$
	$d_{s_i,t_i} \geq 1, \quad \forall$ pairs (s_i, t_i)
Objective:	$\min \sum_{(i,j) \in E} d_{i,j}$

Let the optimal solution of this LP problem be denoted by LP_{opt} and let $d_{i,j}$ denote the value of the variables at the optimal. In order to obtain an integer-approximation to the original MULTI-CUT problem based on this LP solution, we define the following. For every vertex $v \in V$ and $r \geq 0$, let

$$B(v, r) = \{u \in V : d_{v,u} \leq r\} \tag{1}$$

$$Vol(v, r) = \sum_{u,w \in B(v,r), (u,w) \in E} d_{u,w} + \tag{2}$$

$$\sum_{(u,w) \in E, u \in B(u,r), w \notin B(v,r)} (r - d_{v,u}) + \tag{3}$$

$$\left(\frac{LP_{opt}}{2k}\right) \times (\text{ Number of source}(s_i) \text{ and sinks } (t_i) \text{ in } B(v, r)) \tag{4}$$

The ball $B(v, r)$ thus contains all vertices which are within a distance r from v. $Vol(v, r)$ contains 3

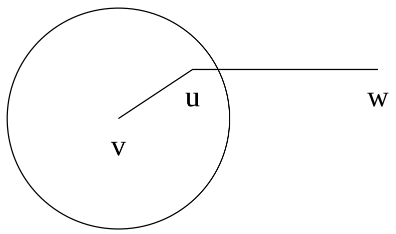

Figure 1: Case 2 of definition of $Vol(v, r)$

terms, (2) corresponds to all edges both of whose endpoints are in $B(v, r)$, (3) corresponds to all edges partially in $B(v,r)$ (ie., one vertex in $(B(v,r)$ and the other outside) and (4) corresponds to the weight of the number of source, sinks in $B(v,r)$. It is to be noted that in the summation in line (3), we do not add the distance $d_{u,w}$, but the distance $(r - d_{v,u})$. It can easily be seen that $r - d_{v,u} \leq d_{u,w}$ [We have $d_{v,u} + d_{u,w} \geq d_{v,w} > r$ (see Figure 1)]

Let α be the rounding parameter, which we shall fix later suitably. The algorithm for obtaining an $\frac{2}{\alpha}$-approximation is as follows.

INTEGER-APPROXIMATE-MULTI-CUT

1. Start at some source (say s_1).

2. Find smallest r_1 such that $Vol(s_1, r_1) \geq \alpha Cut(B(s_1,r_1), \overline{B(s_1,r_1)})$ (*)

3. Remove the edges in the cut mentioned above.

4. Repeat steps 1-4 on $\overline{B(s_1,r_1)}$ until all the source-sink pairs are disconnected.

The following 2 claims prove that the resultant graph at the end of the algorithm is indeed a $\frac{2}{\alpha}$-approximation for the MULTI-CUT problem on the original graph G.

Claim 1 *For a suitable choice of α, there exists an r_1 such that $0 < r_1 < \frac{1}{2}$ and (*) holds.*

Since $r_1 < \frac{1}{2}$ and $d_{s_1,t_1} \geq 1$, we have that $t_1 \notin B(s_1, r_1)$ (see step 2 of the above algorithm). Thus the solution is feasible as it indeed does separate all the source-sink pairs.

Claim 2 *The above algorithm gives a $\frac{2}{\alpha}$-approximation for* MULTI-CUT

We shall first prove claim 2 assuming claim 1.

Proof of Claim 2: The number of edges removed in a particular iteration of step 3 is the number of edges in the cut considered, say $Cut(B(s_i,r_i), \overline{B(s_i,r_i)})$ and this by the choice of r_i in step 2 is at most $\frac{1}{\alpha} Vol(s_i,r_i)$. Thus,

$$\begin{aligned}
\text{Total Number of edges removed} &= \sum Cut(B(s_i,r_i), \overline{B(s_i,r_i)}) \\
&\leq \frac{1}{\alpha} \sum Vol(s_i, r_i) \\
&\leq \frac{1}{\alpha}\left(LP_{opt} + 2k\left(\frac{LP_{opt}}{2k}\right)\right) \\
&= \frac{2}{\alpha} LP_{opt}
\end{aligned}$$

The first two inequality follow from the fact that all as all the cuts considered are disjoint. The third inequality follows from the fact that $\sum Vol(s_i,r_i)$ is at most the sum of all the edge distances in the LP optimal solution and the weighted sum of the source and sinks. We clearly have that the optimal solution OPT to MULTI-CUT satisfies $LP_{opt} \leq OPT$. This and the last inequality imply that the solution obtained by the above mentioned algorithm is at most $\frac{2}{\alpha} OPT$ thus leading to a $\frac{2}{\alpha}$-approximation algorithm. ■

Proof of Claim 1: We can make the following observations regarding the growth of the function $Vol(s_1, r_1)$ with r_1.

- $Vol(s_1, 0) \geq \frac{LP_{opt}}{2k}$ as the ball contains at least the source s_1.
- $Vol(s_1, \infty) \leq 2LP_{opt}$ since all the edges and the weighted source and sinks are included in this ball.

- $Vol(s_1, r_1)$ is a function increasing with r_1. Though it is a discontinuous function, its derivative is defined at the points at which it is continuous. We in fact have,

$$\begin{aligned}\frac{d(Vol(s_1,r_1))}{dr_1} &\geq \sum_{(u,w)\in E, u\in B(s_1,r_1), w\notin B(s_1,r_1)} 1 \quad \text{[By definition of } Vol(s_1,r)\text{]} \\ &= Cut(B(s_1,r_1), \overline{B(s_1,r_1)}) \quad \text{[Definition of Cut]}\end{aligned}$$

Suppose the claim were false. That is for all $r_1 < \frac{1}{2}$, we have that $Vol(s_1, r_1) < \alpha Cut(B(s_1, r_1), \overline{B(s_1, r_1)})$. We then have that $\forall r_1 \in (0, \frac{1}{2})$

$$\begin{aligned}\frac{d(Vol(s_1,r_1))}{dr_1} &\geq Cut(B(s_1,r_1), \overline{B(s_1,r_1)}) \\ &> \frac{1}{\alpha} Vol(s_1,r_1) \\ \text{This implies that } \int_0^{\frac{1}{2}} \frac{d(Vol(s_1,r_1))}{Vol(s_1,r_1)} &> \int_0^{\frac{1}{2}} \frac{1}{\alpha} dr_1 \\ \ln\left(\frac{Vol(s_1,\frac{1}{2})}{Vol(s_1,0)}\right) &> \frac{1}{2\alpha}\end{aligned}$$

From the fact that $Vol(s_1, \frac{1}{2}) \leq Vol(s_1, \infty)$ and $Vol(s_1, \infty) \leq 2LP_{opt}$, we have that

$$\ln 4k \geq \ln\left(\frac{Vol(s_1,\frac{1}{2})}{Vol(s_1,0)}\right) > \frac{1}{2\alpha}$$

Choosing $\alpha = \frac{1}{2 \ln 4k}$, we get a contradiction. Hence Claim 1 is true. ■

2 Classes of Approximation Problems

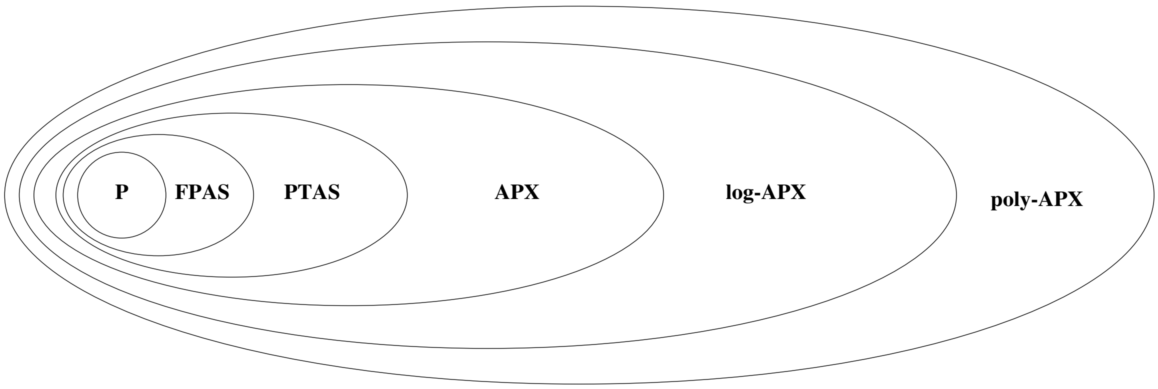

Figure 2: Classes of Approximation Problems

We have so far seen various types of approximation algorithms. To start with, we had problems like KNAPSACK which could be approximated to any $(1+\epsilon)$ (for all $\epsilon > 0$) by an algorithm running in time polynomial in the input length and $\frac{1}{\epsilon}$ (Such problems fall in the class **FPAS**). We then discussed problems like MAKESPAN in the class **PTAS** which had $(1+\epsilon)$-approximation algorithms for any $\epsilon > 0$ and running in time polynomial in the input length and arbitrarily dependent on $\frac{1}{\epsilon}$. Further higher, we have the class **APX**, which consists of problems which can be approximated to some constant (eg: Vertex-Cover. We then have problems like Set-Cover and Multi-Cut which could be approximated to within logarithmic factors. (**log-APX**). We could proceed in this manner and define classes like **polylog-APX**, **poly-APX** and so on. Beyond these we could further have classes like **nopoly-APX**, a class of problems for which no polynomial approximation can be obtained in polynomial time.

2.1 An approximation for Independent Set

The Max-Clique (or its complementary the Max-Independent-Set) is known to lie in **poly-APX**. We shall later show in this course that this is the best possible. ie., there exists an $\epsilon > 0$ such that approximating the maximal independent set size within a factor of n^ϵ is **NP**-hard. We shall now give a polynomial time approximation algorithm (due to Boppana and Halldorsson [BH 90]) for the Max-Independent-Set under the assumption that the given graph has fairly large independent set using a clever use of the Ramsey theory on graphs.

Theorem 3 *Given a graph with an independent set of size at least $(\frac{n}{k} + \epsilon n)$, for some constant $k \geq 2$ and $\epsilon > 0$, then one can find in polynomial time an independent set of size at least $O(n^{\frac{1}{k-1}})$.*

To start with, let us look at the case $k = 2$. If the graph has an independent set of size greater than $\frac{n}{2} + \epsilon n$, then the min-vertex cover is of size lesser than $\frac{n}{2} - \epsilon n$. Using the 2-approximation algorithm for Vertex-Cover, we obtain a vertex cover of size at most $2(\frac{n}{2} - \epsilon n) = n - 2\epsilon n$, thus giving us an independent set of size at least $2\epsilon n$.

Looking at the case $k = 2$ more closely, we note that for any 2-clique (ie., an edge), at most one of its vertices can belong to an independent set. Thus throwing away the 2 vertices of the 2-clique and adding one of them to the independent set, we can reduce our problem size. Thus starting with a graph on n vertices with an independent set of size at least $\frac{n}{2} + \epsilon n$, we now have a graph on $(n-2)$ vertices with an independent set of size at least $(\frac{n-2}{2} + \epsilon n)$. Proceeding in this manner, we finally have a graph with no 2-cliques and hence it is an independent set. We could have done no more than r reductions where $(n - 2r) + r \geq \frac{n}{2} + \epsilon n$ (ie. $r \leq \frac{n}{2} - \epsilon n$). We thus have produced an independent set of size at least $n - 2(\frac{n}{2} - \epsilon n) = 2\epsilon n$

Emulating a similar argument for the case $k > 2$, we note that for any k-clique, at most one of its vertices can belong to an independent set. Thus throwing away the k vertices of the k-clique and adding one of them to the independent set, we can reduce our problem size. Thus starting with a graph on n vertices with an independent set of size at least $\frac{n}{k} + \epsilon n$, we now have a graph on $(n-k)$ vertices with an independent set of size at least $(\frac{n-k}{k} + \epsilon n)$. Proceeding in this manner, we finally have a graph G' with no k-cliques. We can easily check that we could have done no more reductions than $(\frac{n}{k} - \frac{\epsilon n}{k-1})$. Thus the size of the final graph G' is at least $n - k(\frac{n}{k} - \frac{\epsilon n}{k-1}) = \frac{\epsilon k n}{k-1}$. Let this number be m. Unlike earlier, we cannot directly obtain an independent set form this graph G' on m vertices as in the case $k = 2$. Here, we use the Ramsey theory to our rescue.

The Ramsey Number $R(s,t)$ is defined to be the minimum n such that any graph on n vertices either has a clique of size s or an independent set of size t. It can easily be seen from the simple fact $\binom{s+t-2}{t-1} = \binom{s+t-3}{s-2} + \binom{s+t-3}{t-2}$ that $R(s,t) \leq \binom{s+t-2}{t-1}$. Following the lines of the proof of this statement, we can easily see that the following algorithm, given a graph G with $|G| \geq \binom{s+t-2}{t-1}$, produces either a clique of G of size s or an independent set of size t.

Ramsey-Clique-Independent(G, s, t)

Input : A graph $G = (V, E)$ such that $|G| \geq \binom{s+t-2}{t-1}$.
(Notation: $|G|$ = number of vertices in G, $N(v) = \{u \in V | (u, v) \in E\}$)

1. Let v be an arbitrary vertex in V.

2. If $|N(v)| \geq \binom{s+t-3}{s-2}$

then **2a.** $(V', b) \leftarrow$ Ramsey-Clique-Independent$(G \backslash \{v\}, s-1, t)$.

2b. If $b =$ "clique" output ($V' \cup \{v\}$, clique) else output (V', Independent).

else **2c.** $(V', b) \leftarrow$ Ramsey-Clique-Independent$(G \backslash \{v\}, s, t-1)$.

2d. If $b =$ "clique" output (V', clique) else output ($V' \cup \{v\}$, Independent).

One can easily check that $m \geq \binom{s+t-2}{s-1}$ where $s = k$ and $t = m^{1/(k-1)}$. Hence $m \geq R(k, m^{1/(k-1)})$. So if we ran Ramsey-Clique-Independent on $(G', k, m^{1/(k-1)})$ we know that it outputs either a k-clique or an independent set of size $m^{1/(k-1)}$. But, by construction G' has no k-cliques. Hence the algorithm outputs an independent set of size at least $m^{1/(k-1)}$ which is $O(n^{1/(k-1)})$. [Since $m = \frac{\epsilon k n}{k-1}$]. It can easily be checked that the entire algorithm runs in polynomial time.

3 Approximation Preserving Reductions

Most of the optimization problems we have encountered come in 3 different flavors.

- Search Problem: Here, the goal is to find both the value of the optimal solution as well as the optimal solution itself.
- Computational Problem: Here, one has to find just the value of the optimal solution.
- Decision problem: In this version, the goal is to decide whether the optimal solution is above or below a particular value given as part of the input.

Most optimization problems are such that all the above mentioned 3 problems are equally hard. Thus in this case it is sufficient to prove hardness, by showing that the decision version of these problems are hard. However, in the case of approximation problems, as the decision version does not seem to be well defined, we need reductions between either the computational version or the search version of the problems involved. And moreover as we are dealing with approximate solutions, it is not sufficient if the reductions between 2 approximation problems map optimal solutions to optimal solutions. They also have to map approximate solutions to approximate solutions. In other words, the reductions should be *approximation preserving*.

Approximation Preserving Reductions: A reduction f between a problem A and B is said to be approximation preserving if

- The reduction f maps (polynomial time mapping) every instance $(\Pi_1, \text{feasible}_1?, \text{Objective}_1)$ of problem A into an instance $(\Pi_2, \text{feasible}_2?, \text{Objective}_2)$ of the problem B.
- Given an $(1+\epsilon)$-approximate solution S_2 to $(\Pi_2, \text{feasible}_2?, \text{Objective}_2)$, it should be possible to invert (in polynomial time) the solution and obtain a $(1+c\epsilon)$-approximate solution S_1 to $(\Pi_1, \text{feasible}_1?, \text{Objective}_1)$ where c is a fixed constant.

As an example of an approximation preserving reduction, we shall study a reduction between Max-3sat and Max-2sat. This reduction is due to Garey, Johnson and Stockmeyer. (However it was not realized to be an approximate preserving reduction then). Given any boolean formula $\Phi = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_k$, an instance of Max-3sat, the reduction f maps it to formula Φ', an instance of Max-2sat, clause by clause based on the following rules

1. If ϕ_i has at most 2 variables (ie., it is of the form $(x_{i,1})$ or $(x_{i,1} \vee x_{i,2})$), then $\phi'_i = \phi_i$.
2. If ϕ_i has 3 variables (ie., it is of the form $(x_{i,1} \vee x_{i,2} \vee x_{i,3})$), then $\phi'_i = (x_{i,1}) \wedge (x_{i,2}) \wedge (x_{i,3}) \wedge (y_i) \wedge (\overline{x_{i,1}} \vee \overline{x_{i,2}}) \wedge (\overline{x_{i,2}} \vee \overline{x_{i,3}}) \wedge (\overline{x_{i,3}} \vee \overline{x_{i,1}}) \wedge (x_{i,1} \vee y_i) \wedge (x_{i,2} \vee y_i) \wedge (x_{i,3} \vee y_i)$.

It is clear that if a clause ϕ is of the first form, then $f(\phi)$ (ie., ϕ') is satisfiable iff ϕ is satisfiable. However if the clause ϕ is of the second form (ie.,$(x_1 \vee x_2 \vee x_3)$), then it can easily be checked that if ϕ is satisfied by a particular assignment to the variables x_1, x_2 and x_3, then there exists a particular assignment to the variable y such that exactly 7 of the clauses in ϕ' are satisfied. Also if a particular assignment to the variables x_1, x_2 and x_3 does not satisfy ϕ, then there exists no assignment to the variable y such that more than 6 clauses of ϕ' are satisfied (retaining the same assignment to x_1, x_2 and x_3). Moreover, every assignment to the variables x_1, x_2, x_3 and y satisfies at most 7 clauses and for every assignment to x_1, x_2 and x_3, there exists an assignment to y such that at least 6 clauses of ϕ' are satisfied. From this it is clear that if Φ (an instance of MAX-3SAT) has an optimal solution of x clauses being satisfied, then $\Phi' = f(\Phi)$ (an instance of MAX-2SAT) has an optimal solution of $(6m + x)$ clauses being satisfied where m is the number of clauses with 3 variables in Φ.

Now, for the inversion procedure. Given an $(1 + \epsilon)$-approximate solution to $\Phi' = f(\Phi)$, we have to obtain a $(1 + c\epsilon)$-approximate solution to Φ for some constant c. ie., given an assignment that satisfies $\frac{1}{1+\epsilon}(6m + x)$ clauses of Φ', we have to obtain an assignment that satisfies at least $\frac{1}{1+c\epsilon}x$ clauses of Φ. Mere projection of solutions does not work as if the assignment to Φ' satisfies $(6m+x')(= \frac{1}{1+\epsilon}(6m+x))$ clauses, then, by projection, we have a solution that satisfies x' clauses in Φ and this x' need not be greater than $\frac{1}{1+c\epsilon}x$, for very large ϵ. In such cases, we use the $\frac{4}{3}$-approximation algorithm to MAX-3SAT. Thus the inversion algorithm is as follows

Inversion Algorithm INV

Input: An assignment A' to Φ' (an instance of MAX-2SAT obtained through the mapping $f(\Phi) = \Phi'$) satisfying $\frac{1}{1+\epsilon}(6m + x)$ clauses where $(6m + x)$ is the optimal number of clauses satisfiable in Φ'

1. Project assignment A' of Φ' onto an assignment A_1 of Φ in the obvious manner.

2. Obtain a $\frac{4}{3}$-approximate solution A_2 to Φ.

3. Among the assignments A_1 and A_2, output the better assignment A (ie., the one that satisfies more number of clauses).

One can easily check that if A' is a $(1 + \epsilon)$-approximate solution to Φ', then for small ϵ ($\epsilon \leq \frac{1}{9}(\frac{4x}{3m} - 1)$), A_1 is a $(1+9\epsilon)$-approximate solution to Φ and for large ϵ ($\epsilon > \frac{1}{9}(\frac{4x}{3m}-1)$), then A_2 is $(1+9\epsilon)$-approximate solution to Φ. Thus f is indeed an approximation preserving reduction.

4 Gap Problems

To exhibit hardness (**NP**-hardness) of a particular approximation problem (say MAX-3SAT), it is necessary that we reduce some **NP**-complete problem to the approximation problem. For example, to show that MAX-3SAT is hard to approximate, it would suffice if we could build a (polynomial time) reduction f from HAMILTONIAN to MAX-3SAT such that if G is Hamiltonian, then $\Phi = f(G)$ has a large optimum and if G is not Hamiltonian, then $\Phi = f(G)$ has a small optimum. If there exists a polynomial time approximation algorithm for MAX-3SAT that is able to distinguish between these two case (large optimum and small optimum), then coupled with this reduction this algorithm shows that HAMILTONIAN $\in$ **P**. Thus we are only concerned if there exists an algorithm that distinguishes the instances of MAX-3SAT which have a large optimum from those that have a small optimum. We are not bothered about how this algorithm behaves with instances that have intermediate optimum values. This leads to the notion of *Gap Problems*. Gap problems are a special type of decision problems in which the union of the YES and NO instances does not equal the entire space of instances as in a regular decision problem. In other words, we have a gap between the YES and NO instances. For example, in the case of MAX-3SAT, the corresponding gap problem GAP-3SAT$_g$ is defined as follows:

GAP-3SAT$_g$

Instance: (Φ, k) where Φ is a 3CNF formula and k is an integer.

YES: (Φ, k) such that MAX-3SAT$(\Phi) \geq k$

NO: (Φ, k) such that MAX-3SAT$(\Phi) < \frac{k}{g}$

We shall study the relationship between gap problems and approximation problems in detail in the next class.

References

[BH 90] Ravi B. Boppana and Magnus M. Halldorsson. Approximating maximum independent sets by excluding subgraphs. In John R. Gilbert and Rolf G. Karlsson, editors, *SWAT 90, 2nd Scandinavian Workshop on Algorithm Theory*, volume 447 of Lecture Notes in Computer Science, pages 13-25, Bergen, Norway, 11-14 July 1990. Springer.