6.896 Sublinear Time Algorithms	Oct 19, 2001

Lecture 11: Estimating Max-Cut

Lecturer: Ronitt Rubinfeld — *Scribe: Kevin Matulef*

1 Finishing the Max-Cut Approximation Algorithm

Recall from the last lecture that we let (H_1, H_2) be an arbitrary partition, and we defined the successive hybrid partitions $(H_1^0, H_2^0)...(H_1^\ell, H_2^\ell)$, where the partition (H_1^i, H_2^i) places the nodes of $V^1...V^i$ according to the algorithm, and the rest of the nodes via (H_1, H_2). We wish to prove that with high probability, each successive hybrid partition does not decrease the cut size too much. More formally:

Claim 1 (Main Claim) *Fix $U^1, U^2, ...U^{i-1}$. Then*

$$\Pr_{U^i}[\exists \text{ partition } (U_1^i, U_2^i) \text{ of } U^i \text{ s.t. } \mu(H_1^i, H_2^i) \geq \mu(H_1^{i-1} H_2^{i-1}) - \frac{3\epsilon}{4\ell}] \geq 1 - \frac{\delta}{2\ell}$$

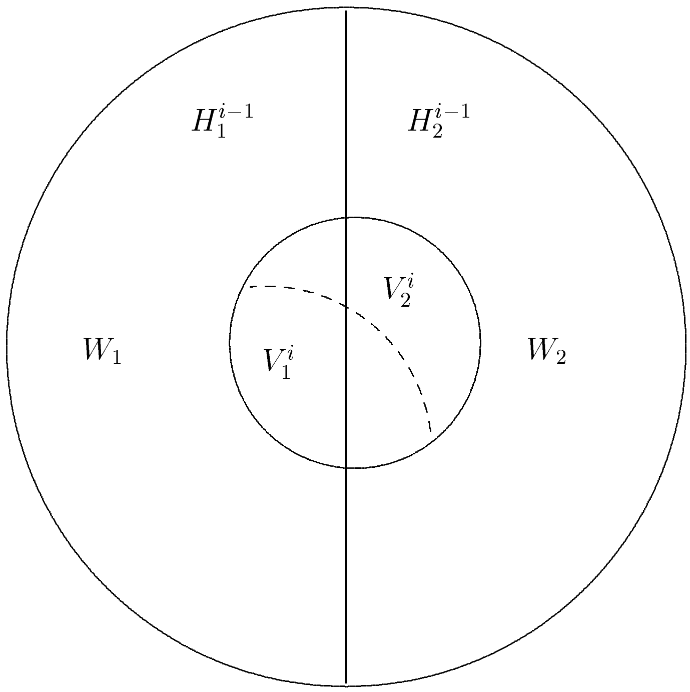
If we can prove this claim, then we can use the union bound to show that the total decrease in the cut after we have partitioned the whole graph according to the algorithm is at most $\frac{3}{4} \cdot \epsilon$, with probability at least $1 - \frac{\delta}{2}$. Thus the correctness of the algorithm will be implied.

Proof

For convenience we define the following sets:

- $W_j = V \backslash V^i \cap H_j^{i-1}$
- $U_j^i =$ the part of the sample that fell in H_j^{i-1}.

Here a picture lends some intuition:



Definition 2 *U^i is "good" if for at least $(1-\epsilon/8)|V^i|$ nodes in the induced partition V^i*

$$(*)\ \forall j \in \{1,2\}\ \left|\frac{\Gamma(v,U_j^i)}{t} - \frac{\Gamma(v,W_j)}{n}\right| \leq \frac{\epsilon}{32}$$

Intuitively, U^i is good, if for most nodes $v \in V^i$, the sample U^i is representative of the number of neighbors v has in W_1 and W_2.

Claim 3 $\Pr[U^i \text{ is good}] \geq 1 - \frac{\delta}{2\ell}$

Proof This proof is left as an exercise on the problem set. ■

Definition 4 *A node $v \in V^i$ is "unbalanced" if $|\Gamma(v,W_1) - \Gamma(v,W_2)| \geq \frac{\epsilon}{8}n$.*

To prove the main claim, we want to argue that by relocating the vertices in V^i, the cut size does not decrease too much. To do this, we must consider the changes due to:
Edges from V^i to W_1, or from V^i to W_2

- For vertices in V^i that are unbalanced and satisfy $*$, there is no decrease in the cut size. This is because if U^i is a fairly representative sample of the number of neighbors v has in W_1 vs W_2, and if v has a lot more neighbors in either W_1 or W_2, then the algorithm will place v across from the majority of its neighbors in $W_1 \bigcup W_2$, which is at least as good as what H_1, H_2 would have done with respect to $W_1 \bigcup W_2$.
- For vertices in V^i that are unbalanced and do not satisfy $*$, the decrease in the cut size may be anything up to n. However, if U^i is good, then there are at most $\frac{\epsilon}{8} \cdot |V^i|$ such vertices (that do not satisfy $*$). Therefore, the total decrease in cut size due to the relocation of these vertices is $\leq n \cdot \frac{\epsilon}{8} \cdot |V^i| = \frac{\epsilon n^2}{8\ell}$.
- For vertices in V_i that are balanced, it does not matter much what side of the cut they are placed on, since they have almost the same number of neighbors on each. For each of these vertices, the decrease due to relocation is at most $2 \cdot \frac{\epsilon n}{8}$ (from the definition of what it means to be balanced). Therefore the total decrease due the relocation of these vertices is at most $|V^i| \cdot 2 \cdot \frac{\epsilon n}{8} \leq \frac{\epsilon n^2}{4\ell}$.

Edges from V^i to V^i

- The algorithm pays no attention to the edges between vertices in V^i. Therefore we can say nothing about the decrease in cut size due to the changes in these edges, except that it is at most $|V^i|^2 \leq \frac{n^2}{\ell^2} \leq \frac{\epsilon n^2}{32\ell}$. Note that this is still less than $\approx n^2/2$, which is what we would get if we didn't divide this partitioning problem into subproblems by partitioning each V^i separately.

Summing the contribution of all of the above cases, we see that the total decrease is $\leq \frac{3\epsilon n^2}{4\ell}$, which is what we wanted to show.
■

Note that so far, the max-cut approximation algorithm that we have given involves partitioning the entire graph, and is not sublinear. It is an exercise on the HW to use these ideas in a sublinear approximation algorithm.

2 Approximating the Edit Distance Efficiently

Definition 5 *Let Σ be an alphabet (e.g. $\Sigma = \{0, 1\}$). Then for two strings $A, B \in \Sigma^n$, we define the edit distance, $D(A, B)$, to be the minimum number of insert, delete, and change operations needed to convert A to B.*

For example, let $A = 101010$ and $B = 100101$. Then $D(A, B) = 2$, since A can be changed to B with one insert and one change operation.

Finding the edit distance between two strings is a non-trivial problem. The best known deterministic algorithms run in time $O(\frac{n^2}{\log n})$ using dynamic programming. On the other hand, no lower bounds are known (besides the obvious $O(n)$).

Definition 6 *For two strings $A, B \in \Sigma^n$, we define the hamming distance, $H(A, B)$, to be the minimum number of change operations needed to convert A to B.*

For example, with A and B in the example above, $H(A, B) = 4$, since A and B differ in 4 places.

The Hamming distance is trivial to find in $O(n)$ time deterministically. The following is a fast approximation algorithm for finding the Hamming distance:

1. Pick i uniformly at random. Check if $A[i] \neq B[i]$. The probability that they differ is exactly $\frac{H(A,B)}{n}$.
2. Repeat $d \cdot \log n$ times. If p is the number of times a difference is observed, then by Chernoff bounds, with high probability $|\frac{p}{d \cdot \log n} - \frac{H(A,B)}{n}| \leq \epsilon$

Obviously we can't use this same algorithm to approximate the edit distance. We need a more clever algorithm, which we will see next time.