Seminar on Sublinear Time Algorithms

Lecture 3

April 7, 2010

Lecturer: Robert Krauthgamer Scribe by: Tamar Zondiner Updated: April 29, 2010

1 Maximum Matching

During the previous lecture, and in the homework, we proved that a Maximal Matching in a graph is of at least half the size of a Maximum Matching. We now construct an approximation algorithm with $\pm\epsilon n$ error for the problem of Maximal Matching, thus resulting in a $(2, \epsilon)$-approximation for a Maximum Matching.

Suppose we are given a graph G, with max-degree$=D$. For each vertex we keep a vector of D neighbors (some coordinates may be empty). We assume that no vertex is isolated.

Our algorithm assigns each edge e, a priority p_e, creating a random permutation of the edges of the graph. Let M be the greedy matching defined by p. An edge e is in the matching if $\forall e_i$, adjacent edges to e $(i = 1, 2, .., 2(D-1))$ s.t. $p_{e_i} > p_e$, it holds that $e_i \notin M$. This is checked recursively. The base case for this recursion is finding an edge e s.t. $\forall e_i$ adjacent edges to e, it holds that $p_{e_i} < p_e$.

Algorithm for approximating Maximal Matching:

1. Choose a random permutation of the edges, by assigning each edge, e, a random priority p_e.
2. Choose $s = O(\frac{D}{\epsilon^2})$ edges, denoted as $e_1, ..., e_s$ at random from the $n \cdot D$ possible pairs $(v, i) \in V \times [D]$.
3. For each edge e_i, evaluate X_i, which is an indicator for the edge e_i being in M, by exploring the neighborhood of e_i.
4. Report $(\frac{\sum X_i}{s}) \cdot Dn$ =the probability of a single edge $\times$ number of edges, as an approximation of the fraction of edges in the matching.

Analysis:

Claim 1 *The Algorithm returns, w.h.p., an approximation within additive ϵn of the size of the Maximal Matching, M.*

Proof

$$E[X_1] \;=\; P[X_1 = 1] \;=\; \frac{|M|}{Dn}.$$

$$E[ALG] \;=\; \frac{Dn}{s} E[\sum X_i] \;=\; \frac{Dn}{s} \cdot s \cdot \frac{|M|}{Dn} \;=\; |M|.$$

$$Var[X_1] \;=\; E[X_1^2] - E[X_1]^2 \;\leq\; E[X_1^2] \;=\; E[X_1] \;=\; \frac{|M|}{Dn}.$$
$$Var[ALG] \;=\; \frac{D^2n^2}{s^2} \cdot Var[\sum X_i] \;=\; \frac{D^2n^2}{s} \cdot \frac{|M|}{Dn} \;\leq\; \frac{Dn|M|}{s}.$$

Using Chebyshev's inequality, we get:

$$P[|ALG - |M|| \geq \epsilon n] \;\leq\; \frac{Var(ALG)}{\epsilon^2 \cdot n^2} \;\leq\; \frac{Dn|M|}{s\epsilon^2 n^2} \;=\; \frac{|M|}{10n} \;\leq\; \frac{1}{10}$$

for $s = 10\frac{D}{\epsilon^2}$. So with high probability, the algorithm gives an estimation within ϵn of $|M|$.
■

Claim 2 *The Algorithm runs in time* $D^{O(D)} \cdot \frac{1}{\epsilon^{O(1)}}$.

Proof We must first emphasize that the two first stages of the algorithm are done "on the fly", and so do not take additional time. The analysis only needs to involve the traversal of edges.

The probability of checking a path of length k from edge e is $\frac{1}{k!}$, since an increasing ordering of the permutation p is required. The number of paths of length k from edge e is $(2D)^k$.

$$P[\text{exploration around } e \text{ gets to radius } k] \leq \frac{(2D)^k}{k!}$$

decreases in k, and for $k = cD$:

$$P[\text{exploration around } e \text{ gets to radius } k] \leq \frac{(2D)^k}{k!} = \frac{(2D)^{cD}}{\sqrt{2\pi cD}\frac{(cD)^{cD}}{e^{cD}}} \leq (\frac{2e}{c})^{cD},$$

thus for a large enough c we can obtain a large probability that all edges traversed are within the $(2D)^{cD}$ edges closest to e. Thus, the total runtime is

$$O(\#\text{iterations times } \#\text{edges traversed}) = O(s \cdot (2D)^{cD+1}) = D^{O(D)} \cdot \frac{1}{\epsilon^{O(1)}}.$$

■

2 Property Testing

For a certain input, we wish to test if a property holds. We want a very efficient Tester algorithm that checks whether an object (Graph, List, Function) satisfies a certain property (2-colorable, sorted, linear etc), or is very "far" from having that property. For a problem of "Property Testing" we need to define:

1) What the object is?
2) What the property is?

3) What does it mean to be ϵ-far from the property?

A Tester algorithm is one that responds as follows:
1) If the object satisfies the property, w.h.p. the algorithm accepts.
2) If the object is ϵ-far from the propery, w.h.p the algorithm rejects.
3) Otherwise, the algorithm may respond either way.

3 Testing a list for monotonicity

Input: List of n distinct integers, allowing random access to its elements.
Goal: Decide if a list is monotone or ϵ-far from monotone.

A list is called monotone if it is increasing. A list is ϵ-close to monotone, if it can be made monotone by changing less than ϵn of its entries. Otherwise it is called ϵ-far from monotone.

Theorem 3 (Ergun-Kannan-Kumar-Rubinfeld-Viswanathan) *: There is a tester for the monotonicity problem that works in time $O(\frac{1}{\epsilon}\log n)$.*

We present the following algorithm:
Algorithm - Test Monotonicity
1) Repeat $\frac{2}{\epsilon}$ times:
   1.1. Choose random $i \in [n]$ and perform binary search for x_i.
2) If all binary searches succeed, accept. Otherwise, reject.

Analysis:
Runtime: clearly, the algorithm simply runs $O(\frac{1}{\epsilon})$ binary searches.
Runtime is $O(\frac{1}{\epsilon} \cdot \log n)$.

Correctness: Let $I = \{i \in [n] : \text{ binary search on } x_i \text{ succeeds}\}$.

Lemma 4 *I is a monotone subsequence.*

Proof Suppose $i, i' \in I$, and $i < i'$. Then the binary searches reach some x_j that separates x_i from $x_{i'}$. Since both binary searches succeed, it holds that $x_i \leq x_j \leq x_{i'} \Rightarrow x_i < x'_i$. ■

Claim 5 *The algorithm is a tester for monotonicity of a list (i.e. If $\mathbf{x}$ is monotone, w.h.p the algorithm accepts, and if $\mathbf{x}$ is ϵ-far from monotone, w.h.p the algorithm rejects.)*

Proof If $\mathbf{x}$ is monotone, all searches succeed, and the algorithm accepts with probability=1. Assume henceforth that x is ϵ-far from monotone.

Suppose towards contradiction that $|I| > (1-\epsilon)n$. From the Lemma, I is an increasing subsequence, so all it takes to make $\mathbf{x}$ monotone is to change the values at coordinates $i \notin I$. There are at most ϵn such coordinates $\Rightarrow \mathbf{x}$ is ϵ-close to monotone, in contradiction to our previous assumption.

Therefore, $|I| \leq (1-\epsilon)n$. This implies that

$$P[\text{ALG accepts}] \;=\; (\frac{|I|}{n})^{\frac{2}{\epsilon}} \;\leq\; (1-\epsilon)^{\frac{2}{\epsilon}} \;\leq\; e^{-2} \;<\; \frac{1}{3}.$$

Thus if $\mathbf{x}$ is ϵ-far from monotone, the algorithm rejects with probability $> \frac{2}{3}$, which proves the correctness of both the claim and the theorem. ■