**CSE 522: Sublinear (and Streaming) Algorithms** **Spring 2014**

# Lecture 15: Sortedness, Connectivity, MST Weight, Components

May 21, 2014

*Lecturer: Paul Beame* *Scribe: Paul Beame*

## 1 Property Testing

**Sortedness** Given input $x_1, \ldots, x_n$ we wish to test whether

1. $x$ is sorted in increasing order
2. $x$ has more than $\varepsilon n$ elements that need to be removed before $x$ is sorted in increasing order (i.e., the longest increasing (non-decreasing) subsequence in $x$ has length $< (1-\varepsilon)n$.

Before discussing an efficient solution for this problem, which will require only $O(\varepsilon^{-1} \log n)$ queries, we consider the kinds of sampling algorithms we have considered so far for property testing:

Independently sample $s$ uniformly random indices in $[n]$ in $x$ and accept iff those elements of $x$ are in sorted order.

For certain sequences we see that this will not yield a particularly efficient algorithm. For example, if $n$ is a multiple of $m$, consider the sequence

$$n/m, n/m-1, \ldots, 1, 2n/m, \ldots, n/m+1, \ldots, n-n/m, n \ldots, n-n/m+1$$

which consists of an increasing sequence of $m$ decreasing sequences, each of length $n/m$. The longest increasing subsequence has length $m$ but but the only way for the sample to imply that the input is not sorted is if two chosen elements land in the same decreasing sequence. The probability that this happens is at most $\binom{s}{2}/m \leq s^2/(2m)$ which is $o(1)$ if $s$ is $o(\sqrt{m})$. In particular, for $\varepsilon = 1/2$ and $m = n/2$, $s = \Omega(\sqrt{n})$ is required.

An alternative might be to query randomly chosen pairs of adjacent elements, as well as checking overall the sorted order, which would easily detect problems with the previous example. However, in this case, we replace each decreasing sequence with a number, say $m$ again, if $m^2$ divides $n$, of increasing sequences, each of which contains elements smaller than the previous one, resulting in an input with $n/m^2$ increasing sequences, each of length $m$. Then, again, the longest increasing sequence would be of length $m^2$ and there would only be an extra $O(s/m)$ chance of catching the local decrease using the negative elements.

Instead, we run the following algorithm. In this algorithm we assume without loss of generality that no two elements are equal by breaking ties using the indices:

**Testing Algorithm for Sortedness**

1: **for** $t = 1$ to $\lceil 2/\varepsilon \rceil$ **do**
2: Choose $i$ uniformly at random from $[n]$
3: Run binary search for $x_i$ in the input
4: **if** $x_i$ is not found at location $i$ or binary search is not consistent* **then**
5: Output "NOT SORTED" and Halt
6: **end if**
7: **end for**
8: Output "SORTED"

*: in binary search, at each step there is a rangle of possible values, initially $[-\infty, +\infty]$, and a range of indices in which the search is continuing; the check for consistency is simply to check that the value at the midpoint of each interval is contained within the interval of possible values.

**Analysis:** Clearly the algorithm uses only $O(\varepsilon^{-1} \log n)$ queries to the input since each binary search pass uses $O(\log n)$ time. The algorithm also requires only a constant number of registers.

It remains to show that the output is correct with probability at least 2/3, say. Obviously, the algorithm will be correct on any sorted input. To see that it will be correct on inputs that are far from sorted, we make the following observation.

**Lemma 1.1.** *If $x_i$ and $x_j$ are both found in their respective binary searches then $x_i < x_j$ iff $i < j$.*

*Proof.* Since the values are different, the binary search must have diverged while comparing $x_i$ and $x_j$ to the least common ancestor of the two of them in the binary search tree. The search for the larger one must have gone right and on the smaller one must have gone left; hence $i' < j'$ iff $x_i < x_j$. □

It follows that the subsequence of consisting of the elements that are found by binary search is itself an increasing sequence. Therefore, if the input is $\varepsilon$-far from sorted, less than a $1 - \varepsilon$ fraction of the input elements can be found by binary search and hence each iteration passes with probability less than $1 - \varepsilon$. Therefore the total probability that an input that is $\varepsilon$-far from sorted will get an incorrect answer is less than

$$(1 - \varepsilon)^t < (1 - \varepsilon)^{2/\varepsilon} < (e^{-\varepsilon})^{2/\varepsilon} = e^{-2} < 1/7.$$

**Property Testing Background** Property testing ideas were introduced in the late 1980s by Blum, Luby and Rubinfeld with the goal of checking the output of a program was self-consistent. In particular, they analyzed a simple test which determines whether or not a function is $\mathbb{F}_2$-linear (more generally, homomorphic) in its inputs by checking that $f(x+y) = f(x)+f(y)$ for randomly chosen $x$ and $y$.

The ideas of property testing for linearity were extended to specific low-degree tests and were critical pieces in the proof of the PCP Theorem in the early 1990s. Property testing algorithms for functions began to be considered as objects in themselves in the work of Rubinfeld and Sudan and others.

In the mid-1990s, Goldreich, Goldwasser, and Ron introduced many property testing algorithms for dense graphs and relaxed the notions to allow 2-sided error. In this work, the graph is input as an adjacency matrix and the notion of $\varepsilon$-closeness is based on the fraction of entries in the adjacency matrix that need to be flipped to change one graph to another; i.e., the Hamming distance betweent the adjacency matrices.

There were many property testing algorithms shown for this model that have complexity of the form $(1/\varepsilon)^{O(1)}$ independent of the size of the input graph. Examples include bipartiteness and $k$-colorability. There is a nice normal form theorem due to Goldreich and Trevisan for property testing of dense graphs:

**Lemma 1.2.** *If a graph property is testable on dense $n$ vertex graphs with $q = q(n, \varepsilon)$ queries then there is a test with the same error that makes fewer than $2q^2$ queries by choosing $2q$ vertices uniformly at random and querying all edges on these vertices*

A property is called *testable on dense graphs* iff there is a property tester that makes $O_\varepsilon(1)$ queries, independent of the size of the graph. Alon, Shapira and others produced a characterization of the properties that are testable on dense graphs based on a Regularity Lemma due to Szemeredi which shows how graphs can be partitioned into a finite number of pieces so that the edges are distributed in a very "regular" way between almost all of the parts of the partition.

One drawback is the fact that the number of pieces can be enormous as a function of $1/\varepsilon$ which yields huge values for the $O_\varepsilon(1)$ quantity, of the form $2^{2^{2^{\cdots^2}}}$ where the height of the tower of 2's is $\Omega(1/\varepsilon)$. Though constant, unlike the algorithms for properties such as bipartiteness, such algorithms are surely hugely impractical.

We focus here on property testing for bounded-degree graphs rather than dense graphs.

**Property Testing for Connectivity in Bounded-degree Graphs** The general idea for property testing on bounded-degree graphs is to view the input as being in adjacency list form, with lists of length up to some fixed degree bound $d$, which means that the input size is at most $2dn$. The distance between graphs is the fraction of these $2dn$ potential adjacency list entries that must be

changed to convert one graph into the other. Therefore, two graphs of degree at most $d$ are $\varepsilon$-close iff they differ by at most $\varepsilon dn$ edges.

The property testing problem for connectivity is then simply to determine whether or not a graph $G$ of degree at most $d$ is

- connected, or
- is $\varepsilon$-far from connected; i.e., it differs from every connected graph of degree at most $d$ by more than $\varepsilon dn$ edges.

Note that if $\varepsilon d \geq 1$ then every graph is $\varepsilon$-close to being connected since we can add the edges of a spanning tree, so we can assume without loss of generality that $\varepsilon d < 1$.

**Connectivity Testing Algorithm**

```
r ← ⌈8/(εd)⌉
for t = 1 to r do
    Choose random node s
    Run a limited Breadth-First Search from s, stopping if ≥ 4/(εd) distinct nodes are seen
    if number of nodes BFS found < 4/(εd) then
        Output "NOT CONNECTED" and Halt
    end if
end for
Output "CONNECTED"
```

Since the graph has degree at most $d$, the Breadth-First Search will complete in $O(1/\varepsilon)$ steps and therefore the whole algorithm will make at most $O(1/(\varepsilon^2 d))$ queries. Clearly, the algorithm will accept any connected graph.

**Lemma 1.3.** *If $G$ is $\varepsilon$-far from connected, then it has $> \varepsilon dn/2$ connected components.*

*Proof.* Any two components can be connected by a single edge, so if there are $c$ connected components then they can be joined by adding at most $c - 1$ edges. The only issue is to make sure that no degree gets too large. We simply repeat the following: If two components each have a vertex of degree $< d$ add an edge between those vertices. If a component has all its vertices of degree $d$ then remove some edge not in the spanning tree of the component. The total number of edges changed is less than $2c$. □

Suppose that $G$ is $\varepsilon$-far from connected. Since $G$ has $> \varepsilon dn/2$ connected components, the average size of these connected components is $< 2/(\varepsilon d)$. Therefore, by Markov's inequality, at least 1/2 of the connected components of $G$ have size $< 4/(\varepsilon d)$. Thus, $G$ has $> \varepsilon dn/4$ connected components of size $< 4/(\varepsilon d)$. In particular, this means that $G$ has $> \varepsilon dn/4$ vertices in connected components

of size $< 4/(\varepsilon d)$. Therefore the probability that an iteration will output "NOT CONNECTED" is $> \varepsilon d/4$. Therefore the overall probability that the algorithm incorrectly outputs "CONNECTED" is less than

$$(1 - \varepsilon d/4)^r < (1 - \varepsilon d/4)^{8/(\varepsilon d)} < e^{-2} < 1/7.$$

# 2 Sublinear Algorithms on Bounded-Degree Graphs

Property testing algorithms are not the only kinds of sublinear algorithms of interest. while the typical objects we might want to compute for problems such as optimization problems would take too long to write down, we can sometimes estimate things like their objective function values.

**Minimum Spanning Tree Value** Suppose that we are given an $n$ vertex weighted graph $G$ of degree at most $d$ and weights from $\{1, \ldots, w\}$. We wish to compute (or estimate) the weight of the minimum weight spanning tree (MST) of $G$.

First consider the case when $w = 2$ so all the edges are of weight 1 or 2. The MST weight would then be $n - 1$ plus the number of weight 2 edges used. The MST would use as many edges of weight 1 as possible and would only use an edge of weight 2 to connect components that could not be connected by weight 1 edges. Therefore the number of weight 2 edges would be 1 less than the number $c^{(1)}$ of connected components in the subgraph given only by the weight 1 edges of $G$. So the total weight would be $n - 1 + c^{(1)} - 1 = n - 2 + c^{(1)}$.

More generally, define $c^{(i)}$ = the number of connected components induced by edges of weight $\leq i$ in $G$.

By a similar argument to the case $w = 2$, we see that the number of edges of weight at least $i + 1$ in the MST for $G$ is then precisely $c^{(i)} - 1$. Therefore, since an edge in the MST of weight exactly $j + 1$ will be at least $i + 1$ for $i = 1, \ldots, j$, we have

**Lemma 2.1.** *The weight of the MST for $G$ is precisely* $n - 1 + \sum_{i=1}^{w}(c^{(i)} - 1) = n - w + \sum_{i=1}^{w} c^{(i)}$.

Since the MST weight is at least $n - 1$, the overall algorithm to estimate the MST weight within a $1 \pm \varepsilon$ factor will be simple: Compute an estimate $\tilde{c}^{(i)}$ to each $c^{(i)}$ within an additive $\varepsilon_0 n$ for $\varepsilon_0 = \varepsilon/w$ and then output $n - w + \sum_{i=1}^{w} \tilde{c}^{(i)}$. The overall error will be at most $\varepsilon n$ and the number of queries is at most $w$ times the maximum number of queries for the connected component estimation.

**Counting Connected Components** We wish to compute an estimate within $\varepsilon_0 n$ of the number of connected components $c$ of a bounded-degee graph $G$.

For each vertex $v$ in $G$, let $size(v)$ denote the size of the connected component containing $v$ and define $\alpha_v = 1/size(v)$.

For any connected component $A \subseteq V$, clearly $\sum_{v \in A} \alpha_v = 1$ and hence $\sum_v \alpha_v = c$.

The overall idea of the algorithm will be to estimate $c/n = \mathbb{E}_v(\alpha_v)$ by sampling vertices and estimating their $\alpha_v$ values. Since any $v$ for which $\alpha_v$ is very small will not have much impact on the total, it suffices to compute $\alpha_v$ exactly when it is relatively large, that is, when $size(v)$ is small. We will compute it exactly when $\alpha_v \geq \varepsilon_0/2$.

Let $\tilde{\alpha}_v = \max(\alpha_v, \varepsilon_0/2)$. Clearly $|\tilde{\alpha}_v - \alpha_v| < \varepsilon_0/2$ and

$$x = \sum_v \alpha_v \leq \sum_v \tilde{\alpha}_v = \sum_v \max(\alpha_v, \varepsilon_0/2) \leq sum_v \alpha_v + \varepsilon_0 n/2 = c + \varepsilon_0 n/2.$$

**Connected Component Estimation**

1: $r \leftarrow O(1/\varepsilon_0^3)$
2: $c' \leftarrow 0$
3: **for** $r$ times **do**
4: Choose random vertex $v$
5: Compute $\tilde{\alpha}_v$ by running a limited Breadth-First Search from $v$, stopping after at most $2/\varepsilon_0$ new nodes.
6: $c' \leftarrow c' + \alpha_v$
7: **end for**
8: Output $\tilde{c} = nc'/r$.

Since the graph has maximum degree $d$, the limited BFS takes $O(d/\varepsilon_0)$ for a total running time of $O(d/\varepsilon_0^4)$.

Now $c'$ is the sum of $r$ independent random variables and $\mathbb{E}(c') = r\mathbb{E}_v(\tilde{\alpha}_v)$. Morever the deviations of these variables from their mean is bounded since $\varepsilon_0/2 \leq \tilde{\alpha}_v \leq 1$ for all $v$, so we can use a Bernstein's/Azuma's inequalities (more general forms of Chernoff bounds) to say that with probability near 1, $c'$ is within $\varepsilon_0 r/2$ of $r\mathbb{E}_v(\tilde{\alpha}_v)$. Therefore with probability near 1,

$$|\tilde{c} - \sum_v \tilde{\alpha}_v| = |\tilde{c} - n\mathbb{E}_v(\tilde{\alpha}_v)| \leq \varepsilon_0 n/2.$$

Finally, using the fact that $c \leq \sum_v \tilde{\alpha}_v \leq c + \varepsilon_0 n/2$, we get that $|\tilde{c} - c| \leq \varepsilon_0 n$ as required.

**Putting it all together** As described this would yield an algorithm for estimating the MST value within a $1 \pm \varepsilon$ factor with $O(wd/\varepsilon_0^4) = O(w^5 d/\varepsilon^4)$ queries.

A more careful algorithm due to Chazelle, Rubinfeld, and Trevisan does this in $O(dw\varepsilon^{-3}\log(dw/\varepsilon))$ queries and works even if the *average* degree is at most $d$.