Lecture 13: More examples of Sub-linear Time Algorithms

6.893 Spring 2013

Lecturer: Ronitt Rubinfeld
Today:

• Classical approximation problems
  ▪ Diameter of a point set
  ▪ Minimum Spanning Tree

• Property testing: approximation for decision problems
  ▪ “Sortedness” of a list
  ▪ Connectedness of a graph
What can we hope to do without viewing most of the data?

• Change our goals?
  ▪ for most interesting problems: algorithm must give approximate answer

• we know we can answer *some* questions...
  ▪ e.g., sampling to approximate average, median values
What types of approximation?

• “Classical” approximation for optimization problems:
  output is number that is close to value of the optimal solution for given input.
  (not enough time to construct a solution)

• Property testing for decision problems:
  output is correct answer for given input, or at least for some other input “close” to it.
I. Classical Approximation Problems
Last time:

• approximate average degree of a graph in sublinear time!
Another example today:

• Approximate the diameter of a point set
  ▪ Simple
  ▪ Deterministic
Approximate the diameter of a point set

- Given: \( m \) points, described by a distance matrix \( D \), s.t.
  - \( D_{ij} \) is the distance from \( i \) to \( j \).
  - \( D \) satisfies triangle inequality and symmetry.
  (note: input size \( n = m^2 \))

- Let \( i, j \) be indices that maximize \( D_{ij} \) then \( D_{ij} \) is the *diameter*.

- Output: \( k,l \) such that \( D_{kl} \geq D_{ij}/2 \)
Algorithm

- Algorithm:
  - Pick $k$ arbitrarily
  - Pick $l$ to maximize $D_{kl}$
  - Output $D_{kl}$
- Why does it work?
  - $D_{ij} \leq D_{ik} + D_{kj}$ (triangle inequality)
  - $\leq D_{kl} + D_{kl}$ (choice of $l$ + symmetry of $D$)
  - $\leq 2D_{kl}$
- Running time? $O(m) = O(n^{1/2})$
II. Property testing
Main Goal:

- **Quickly** distinguish inputs that **have** specific property from those that are **far from having** the property

![Diagram]

- all inputs
- inputs with the property
- close to having property
Property Testing

• Properties of any object, e.g.,
  ▪ Functions
  ▪ Graphs
  ▪ Strings
  ▪ Matrices
  ▪ Codewords

• Model must specify
  ▪ representation of object and allowable queries
  ▪ notion of close/far, e.g.,
    ▪ number of bits/words that need to be changed
    ▪ edit distance
A simple property tester
Sortedness of a sequence

- Given: list $y_1 y_2 \ldots y_n$
- Question: is the list sorted?
  - Clearly requires $n$ steps – must look at each $y_i$
Sortedness of a sequence

- Given: list $y_1 y_2 \ldots y_n$

- Question: can we quickly test if the list close to sorted?
What do we mean by ``quick’’?

• query complexity measured in terms of list size $n$

• Our goal (if possible):
  ▪ *Very small* compared to $n$, will go for $clog n$
What do we mean by “close”?  

Definition: a list of size $n$ is $\varepsilon$-close to sorted if can delete at most $\varepsilon n$ values to make it sorted. Otherwise, $\varepsilon$-far.

($\varepsilon$ is given as input, e.g., $\varepsilon=1/10$)

Sorted:  1   2   4   5   7   11   14   19   20   21   23   38   39   45  
Close:    1   4   2   5   7   11   14   19   20   39   23   21   38   45  
          1   4   5   7   11   14   19   20          23       38   45  
Far:       45   39   23   1   38   4   5   21   20   19   2   7   11   14  
           1   4   5                          7   11   14
Requirements for algorithm:

• Pass sorted lists

• Fail lists that are $\varepsilon$-far.
  - Equivalently: if list likely to pass test, can change at most $\varepsilon$ fraction of list to make it sorted

  Probability of success $> \frac{3}{4}$
  (can boost it arbitrarily high by repeating several times and outputting “fail” if ever see a “fail”, “pass” otherwise)

• Can test in $O(1/\varepsilon \log n)$ time
  (and can’t do any better!)
An attempt:

• Proposed algorithm:
  - Pick random $i$ and test that $y_i \leq y_{i+1}$

• Bad input type:
  - $1, 2, 3, 4, 5, \ldots n/4, 1, 2, \ldots n/4, 1, 2, \ldots n/4, 1, 2, \ldots, n/4$
  - Difficult for this algorithm to find “breakpoint”
  - But other tests work well…

![Graph](image-url)
A second attempt:

- Proposed algorithm:
  - Pick random $i<j$ and test that $y_i \leq y_j$
- Bad input type:
  - $n/4$ groups of 4 decreasing elements:
    - 4, 3, 2, 1, 8, 7, 6, 5, 12, 11, 10, 9..., 4k, 4k-1, 4k-2, 4k-3,...
  - Largest monotone sequence is $n/4$
  - must pick $i,j$ in same group to see problem
  - need $\Omega(n^{1/2})$ samples
A minor simplification:

- Assume list is distinct (i.e. \( x_i \neq x_j \))

- Claim: this is not really easier
  - Why?
    - Can “virtually” append \( i \) to each \( x_i \)
      - \( x_1, x_2, \ldots, x_n \) → (\( x_1, 1 \), \( x_2, 2 \), \ldots, \( x_n, n \))
      - e.g., 1,1,2,6,6 → (1,1),(1,2),(2,3),(6,4),(6,5)
    - Breaks ties without changing order
A test that works

• The test:

Test $O(1/\varepsilon)$ times:
  ▪ Pick random $i$
  ▪ Look at value of $y_i$
  ▪ Do binary search for $y_i$
  ▪ Does the binary search find any inconsistencies? If yes, FAIL
  ▪ Do we end up at location $i$? If not FAIL
  ▪ Pass if never failed

• Running time: $O(\varepsilon^{-1} \log n)$ time
• Why does this work?
Behavior of the test:

- Define index $i$ to be good if binary search for $y_i$ successful

- $O(1/\varepsilon \log n)$ time test (restated):
  - pick $O(1/\varepsilon)$ $i$’s and pass if they are all good

- Correctness:
  - If list is sorted, then all $i$’s are good (uses distinctness)
    - So test always passes
  - If list likely to pass test,
    - Then at least $(1-\varepsilon)n$ $i$’s are good.
    - Main observation: good elements form increasing sequence
      - Proof: for $i<j$ both good need to show $y_i < y_j$
        - let $k =$ least common ancestor of $i,j$
        - Search for $i$ went left of $k$ and search for $j$ went right of $k \Rightarrow y_i < y_k < y_j$
      - Thus list is $\varepsilon$-close to monotone (delete $< \varepsilon n$ bad elements)
Testing connectedness of a graph

• Given graph G
  ▪ n vertices
  ▪ Max degree d
  ▪ Adjacency list representation

• Is G connected?
Connected world phenomenon

• Is the underlying graph close to connected?
Close to connected

- Def: $G$ is $\epsilon$–close to connected if can add $< \epsilon dn$ edges and transform it to connected
  - Today: ok to violate max degree $d$ requirement in transformed graph
More formally: $\epsilon$-close to connected

- Let $P$ be the set of degree $\leq d$ connected graphs
- $G$ is $\epsilon$-close to $P$ if $G$ has degree $\leq d$ and there is a $G'$ in $P$ such that $G$ and $G'$ differ on at most $\epsilon dn$ edges.
Property tester:

• Input: $\epsilon$ and $G$

• Output:
  
  - If $G$ connected, output “PASS”
  
  - If $G$ not $\epsilon$-close to connected, output “FAIL” with probability $\geq 3/4$

  (note: if $G$ not connected, but is close, then ok to output either “PASS” or “FAIL”)

Idea:

- If G far from connected, lots of nodes must be in small components!
- More specifically...
  - Will show that if G far from connected
  - Then must have many connected components
  - So many components must be small
  - And there must be many nodes in small components
Algorithm:

• Do $O\left(\frac{1}{\varepsilon d}\right)$ times:
  ▪ Pick random node $s$, and run BFS from $s$ until:
    ▪ $\geq \frac{2}{\varepsilon d}$ distinct nodes seen
    ▪ OR see that $s$ is component of size $< \frac{2}{\varepsilon d}$ nodes, in which case output “FAIL” and halt

• If reach this point, output “PASS”

Runtime: $O\left(\frac{1}{\varepsilon d}\right)$ loops, each does $O\left(\frac{1}{\varepsilon d}\right)$ steps of BFS, using $O(d)$ time per step – total is $O\left(\frac{1}{\varepsilon^2 d}\right)$
Behavior

- Lemma 1: If $G$ $\epsilon$-far from connected, then has $\geq \epsilon d n$ components
- Lemma 2: If $\geq \epsilon d n$ components then $\geq \epsilon d n / 2$ components of size $< \frac{2}{\epsilon d}$
- Observation: If $\geq \epsilon d n / 2$ components of size $< \frac{2}{\epsilon d}$ then $\geq \epsilon d n / 2$ nodes in components of size $< \frac{2}{\epsilon d}$

These cause tester to FAIL!
Putting it together: If $G$ $\epsilon$-far from connected, then $\geq \epsilon d/2$ fraction of nodes cause algorithm to fail!

- So $\text{Prob}[\text{tester fails in one of } \frac{c}{\epsilon d} \text{ loops}]$ is

$$\geq 1 - \left(1 - \frac{\epsilon d}{2}\right)^\left\lceil \frac{2c}{2\epsilon d} \right\rceil \geq 1 - e^{c/2} \geq 3/4 \quad \text{(for big enough } c)$$
Lemma 1

If $G \epsilon$-far from connected, then has $\geq \epsilon dn$ components.

Proof: if $<\epsilon dn$ components, can add $<\epsilon dn$ edges to connect (but this might make $G$ have degree $>d$ if not careful).
Lemma 2

If $\geq \epsilon dn$ components then $\geq \epsilon dn/2$ components of size $< \frac{2}{\epsilon d}$

(see notes for proof)
Observation:

If $\geq \epsilon dn/2$ components of size $< \frac{2}{\epsilon d}$ then
$\geq \epsilon dn/2$ nodes in components of size $< \frac{2}{\epsilon d}$

Why? Each small component has at least one node.
Minimum spanning tree (MST)

• What is the cheapest way to connect all the dots?

• Best known:
  ▪ Deterministic $O(m\alpha(m))$ time [Chazelle]
  ▪ Randomized $O(m)$ time [Karger Klein Tarjan]
A sublinear time algorithm: [Chazelle R. Trevisan]

Given input graph with
- weights in $[1..w]$
- average degree $d$
- adjacency list representation

outputs $(1+\varepsilon)$-approximation to MST in time $O\left(\frac{dw}{\varepsilon^3 \log \frac{dw}{\varepsilon}}\right)$

Remarks: (1) sublinear when $dw=o(m)$
constant when $d,w$ bounded
(2) $\Omega(dw \varepsilon^{-2})$ required
(3) case of integral weights, max degree $d$ can be done in $O(dw \varepsilon^{-2} \log w/\varepsilon)$ time
Idea behind algorithm:

• characterize MST weight in terms of number of connected components in certain subgraphs of $G$

• show that number of connected components can be estimated quickly
Suppose all weights are 1 or 2. Then

\[ \text{MST weight} = \# \text{ weight 1 edges} + 2 \cdot \# \text{ weight 2 edges} \]
\[ = n - 1 + \# \text{ of weight 2 edges} \]
\[ = n - 2 + \# \text{ of conn. comp. induced by weight 1 edges} \]
• For integer weights 1..w let
  \[ c^{(i)} = \# \text{ of connected components induced by edges of weight at most } i \]

• Then MST weight is
  \[ n - w + \sum_{i=1,\ldots,w-1} c^{(i)} \]

• additive approximation of \( c^{(i)} \)'s to within \( \varepsilon n/w \)
gives additive approx of MST to within \( \varepsilon n \)
  ▪ Since MST > n-1, also gives multiplicative approximation of MST to within \( 1 \pm \varepsilon \)
Approximating number of connected components:

• Given input graph with
  ▪ max degree $d$
  ▪ adjacency list representation

• outputs additive approximation to within $\varepsilon_0 n$ of the number of connected components in time $O(d \varepsilon_0^{-2} \log 1/\varepsilon_0)$

• Can show $\Omega(d \varepsilon_0^{-2})$ time is required
Approximating # of connected components

• Let $c = \text{number of components}$

• For every vertex $u$, define $n_u := 1 / \text{size of component of } u$
  - for any connected component $A \subseteq V$, $\sum_{u \in A} n_u = 1$
  - so $\sum u n_u = c$
Main idea

• Estimate sum of approximations of $n_u$’s via sampling

• To estimate $n_u = 1 / \text{size of component of } u$ quickly:
  - If size of component is big, then $n_u$ is small so easy to estimate (similar to property tester for connectivity [Goldreich Ron])
  - Suffices to compute $n_u$ exactly only for small components
Some more details:

Estimating \( n_u \equiv 1 / \text{size of u’s component} \):

- let \( \tilde{n}_u := \max \{ n_u, \varepsilon_0/2 \} \)
  - When size of u’s component is \(< 2/\varepsilon_0 \), \( \tilde{n}_u = n_u \)
  - Else \( \tilde{n}_u = \varepsilon_0/2 \)

- \( |n_u - \tilde{n}_u| < \varepsilon_0/2 \) so \( c = \sum_u n_u = \sum_u \tilde{n}_u \pm \varepsilon_0 n / 2 \)

- can compute \( \tilde{n}_u \) quickly
  - in time \( O(d/\varepsilon_0) \) with BFS
Not quite optimal algorithm:

CC-APPROX($\varepsilon_0$):

Repeat $O(1/\varepsilon_0^3)$ times
  pick a random vertex $v$
  compute $\tilde{n}_v$ via BFS from $v$, stopping after at most $2/\varepsilon_0$ new nodes
return (average of the values $\tilde{n}_v$) $\cdot n$

Run time: $O(d/\varepsilon_0^4)$
Correctness

• Chernoff bounds $\implies$ algorithm gives good estimate of average/sum of the $\tilde{n}_v$ values

• Previous arguments $\implies$ good estimate of average of the $\tilde{n}_v$ values gives (slightly less) good estimate of average/sum of the $n_v$ values

• Estimate of sum of the $n_v$ values gives estimate of number of connected components via $\sum_u n_u = c$
Improvement for MST:

This gives MST algorithm with runtime $O(dw^2 \varepsilon^{-4})$

Can do better:

- $O(dw\varepsilon^{-3} \log dw/\varepsilon)$ algorithm
Further work:

• Euclidean MST approximation algorithm
  [Czumaj Ergun Fortnow Newman Magen Rubinfeld Sohler]
  ▪ Given access to certain data structures can do better

• Metric MST [Czumaj Sohler]
  ▪ (1+ $\varepsilon$)-approximation in time $\tilde{O}(n)$