Sub-linear Algorithms

Lecture 21
Shafi Goldwasser
This course so far

Tractable
solvable in polynomial time

Intractable
not solvable in polynomial time

Linear Time
- $\Omega(n \log n)$ an easy lower bound for most problems
- comparison based sorting $\Omega(n \log n)$
The Gold Standard

• linear time algorithms:
  - for inputs encoded by $n$ bits/words, allow $cn$ time steps (constant $c$)
Many of Today Applications: Massive Data

Examples of inputs:

WEB pages ≈ 2 billion pages (2002), 19.5 billion (2005),

Human Genome ≈ 3 billion base pairs

Even Linear Time is too long:

What can we do?
New Goal: Sub-Linear Algorithms

Algorithms which inspect only a small fraction of the input/data

Am I a carnivore?

How can such algorithms possibly be correct?

- Can’t, in the strict sense of the word.

- Relax the correctness requirements from sub-linear algorithms

- Necessarily randomized algorithms
Warmup Example: string identity

• Fix n-bit string b.
• Problem: On input for n-bit string a, is a=b?

• Requires Linear Time to decide exactly

• But suppose we are given parameter \( \varepsilon > 0 \), asked to:
  - accept a, if a=b
  - reject a, if a \( \neq b \) in more than \( \varepsilon \) fraction of the bit locations (a is \( \varepsilon \)-far from b)
Warmup Example: string identity

Sub-Linear Algorithm with parameter $\varepsilon$:

Sample $t$ bits of $a$ at random,
if find $i$ such that $a_i \neq b_i$, then reject $a$ (say no)
else accept $a$ (say yes)

What does this algorithm guarantee on input $a$:

- If $a=b$, always accepts
- If $a$ is $\varepsilon$-far from $b$, then $\text{Prob (accept } a) < (1-\varepsilon)^t = 1/e^c$
  for $t=c/\varepsilon = O(1/\varepsilon)$
- Runtime $O(1/\varepsilon)$, make $O(1/\varepsilon)$ queries to the input
- No guarantees if $a$ is $\varepsilon$-close to $b$
In general, Relaxed Requirements

Decision Problem $\rightarrow$ Property-Testing

Decision P: for every input $x$ the algorithm must answer $P(x)$ correctly.

Property Testing for P with parameter $\varepsilon$: for every input $x$ the algorithm must with high probability:
- answer yes, if $P(x) =$ yes
- answer no, if $x$ is $\varepsilon$-far from any $x'$ for which $P(x') =$ yes

Need to fix notion of distance between inputs.
Property Tester: More Formal

Need to Specify
- \textit{distance}(,\,) function between inputs
- parameter \(\varepsilon > 0\)
- P: Y/N property (Decision problem)

\textbf{Property Tester for P with parameter} \(\varepsilon\): for every input \(x\)
- if \(P(x) = \text{yes},\) then \(\text{Prob [Tester accepts } x\text{]} = 1\)
- if \(\text{distance}(x,x') > \varepsilon \ \forall x'\) such that \(P(x') = \text{yes},\) then \(\text{Prob [Tester reject } x\text{]} > 2/3\)
Remarks I

- **Must specify the notion of distance/close/far,**
  - Hamming Distance: number of bits that differ
  - Edit distance

- **Probability of error** can be made arbitrary small, by usual amplification tricks

- **1-sided error testers:**
  In general could only require that if \( P(x) = \text{yes}, \)
  \[ \text{Prob [Tester accepts } x] > 2/3. \]
  This lecture require probability 1.
Remarks 2

- Relaxed Requirements to gain efficiency: algorithms do not even read the entire input.
- Must specify what is the representation of the input, and what are the allowed queries to the input.

- **Complexity Measure for Property Testers:**
  - Number of Queries
  - Running Time
Property Testing: Wide applicability

- Properties of
  - Strings
  - Functions
  - Graphs
  - Matrices
  - Code-words
When do these Relaxed Requirements make sense?

When:
- Linear time is too long due to input size
- Input is not too large but Problems are too hard
- Applications tolerate inputs which are close to having a property if algorithms are much faster
- As a fast pre-processing step before running an slow classical decision algorithm
Outline for Today

• Two examples of Sub-Linear Time Property Testers
  – List Monotonicity
  – Connectivity of Graphs

• Sub-Linear Space Streaming Algorithm
  – Element Distinctness (in recitation)
Monotonicity of a List

• Given: list $x_1 \, x_2 \ldots \, x_n$
• Question: is the list sorted?

• Clearly requires $\Omega(n)$ time to decide
Monotonicity of a List

• Given: list \( x_1 x_2 \ldots x_n \)

• Question: can we test with sub-linear of queries and time if the list close to sorted?

• Show: An \( O(1/\varepsilon \log n) \) time and query tester
Definition: a list of size $n$ is $\varepsilon$-far to monotone if must delete more than $\varepsilon n$ values to make it monotone. Otherwise, $\varepsilon$-close.

Query $i$: get value $X_i$.

Requirements for property tester:
- Accept monotone (i.e. sorted) lists.
- Reject the list if need to change more than most $\varepsilon$ fraction of the list to make it monotone.
Test Attempt 1:
Pick random $i \prec j$ and test that $x_i \leq x_j$

Here is the worst example, fix $c$

c, c-1, c-2, ... 1, 2c, 2c-1 ... c+1, ... n, n-1, ..., n-c+1

• Longest subsequence has length $n/c$ (can’t pick from the same group twice)

• Prob (Test accept) $< \frac{n(n-c)}{n^2} = \frac{(n-c)}{n} = \left(1 - \frac{1}{c}\right)$

• So, for $c = \sqrt{n}$, to get Prob(reject) $> 2/3$ must repeat $O(\sqrt{n})$ times. SUB LINEAR but we can do better
Test Attempt 2:
Pick random $i$ and test that $x_i \leq x_{i+1}$

Worst example, Fix $c$

$(1,2\ldots,n/c),(1,2\ldots n/c),\ldots,(1,2\ldots,n/c)$

- The longest sub-sequence is $c-1+n/c$ long:
- $\text{Prob (Test accepts)} = (1-c/n)$ (always unless pick border points)
- Should repeat $n/c$ times to get prob $<2/3$
- For $c=\sqrt{n}$, this SUBLINEAR. can do better
Sequence Monotonicity Test

• Without loss of generality, assume $x_1 \ x_2 \ldots \ x_n$ distinct
• Define: index $i$ as good if binary search for $x_i$ successful:
  - finds $X_i$ in $i$-th location
  - does not encounter any $X_k, X_k'$ that are out of order along the search: i.e. $k < k'$ but $X_k > X_k'$

Sequence Monotonicity Test with parameter $\varepsilon$

• Repeat $O(1/\varepsilon)$ times:
  - pick a random index $i$, and do a binary search for $X_i$ in the input sequence
• If all indices are good, output `Yes`,
• Else output `No: sequence is not monotone'.
Complexity and Correctness

- **Complexity:** $O(1/\varepsilon \log n)$ time and queries
  - Note: can determine goodness in $O(\log n)$ time

- If list is sorted, then all i’s are good
- **Need to show:** If the list is $\varepsilon$-far from sorted, then less than $(1-\varepsilon)n$ indices are good

  $\Rightarrow$ \text{Prob}[test accept $\varepsilon$-far list] < \text{Prob}[always choose good indices] < \left(1-\varepsilon\right)^{O(1/\varepsilon)} < e^c$ where $O(1/\varepsilon)=c/\varepsilon$

Counter positive: It at least $(1-\varepsilon)n$ indices i are good $\Rightarrow$ list is $\varepsilon$-close to sorted

- **Observation:** elements at good indices form an increasing sub-sequence
PROVE MAIN OBSERVATION:

*Good indices form an increasing sub-sequence*

PROOF:

Let i and j be two good indices. Consider the path in the binary search they both take.

These two paths have some longest common prefix i.e elements they both compare to.

Take index k the place where they part, \( i \leq k \leq j \)

Let \( z = x_k \)

We claim that \( x_i \leq z \leq x_j \) and thus \( x_i \leq x_j \)

If the path to \( x_i \) is a prefix of the path to \( x_j \) then \( k = i \), \( z = x_i \) and are done - since \( j < i \) and \( j \), so \( x_i \leq x_j \)

If not, then we claim that \( x_i \) will be a left descendent of \( z \) and \( x_j \) a right descendant.

This is the case as \( x_i \) will be found in location i and since \( i < k \) also \( x_i < x_i \). Similarity for \( x_j \)
Testing Properties of graphs

• Given graph $G = (V, E)$, $|V| = n$, $|E| = m$
• What questions can we ask about $G$?
  – Connected?
  – Bipartite?
  – $k$-colorable?
  – Large clique?
  – Large cut? Large conductance?

Goal: Design sub-linear Property Testers
Graph Representations

• Adjacency Matrices: appropriate for dense graphs

• **Adjacency Lists**: Appropriate for sparse graphs, average degree \( d = \frac{m}{n} = o(n) \):
  (can think of \( d \) as a constant)

• **Query**: what is \( i \)-th neighbor of \( u \) ?
Connectivity in Sparse Graphs

- \( n \) Adjacency Lists: one per vertex in \( V \),
- \textbf{Query}: what is \( i \)-th neighbor of \( u \) ?

- \textbf{Question}: Is Graph Connected?
- \textbf{Best Possible Classically}: \( \Omega (m) \) (BSF)

- Show \textbf{Sub-Linear Property Tester} for Connectivity
- Actually, constant in \( \frac{1}{\varepsilon} \) and \( \frac{1}{d} \)
Connectivity: Property Testing

We say that graph is $\varepsilon$-far from connected if need to add more than $\varepsilon m$ edges to be connected, and $\varepsilon$-close otherwise.

Goal for Tester with parameter $\varepsilon$:
Accept $G$, if $G$ is connected
Reject $G$ with prob $>2/3$, if $G$ is $\varepsilon$-far from connected
Don’t care otherwise
Intuition for Tester

- Far from connected implies
- Many connected components implies
- Many small connected components implies
- Many vertices in small connected components

TEST:
- Choose random vertices, and see if they are in a small connected component by doing BFS
- If so reject, else accept
Property Testing Connectivity

Let $CC(G) =$ # connected components in $G$

**Simple Observation:**
If graph is not connected, then the minimum number of edges to add to become connected is $CC(G)-1$

**Lemma:** Graph is $\varepsilon$-far from connected $\Rightarrow$ $CC(G) > \varepsilon m \Rightarrow$ more than half of the connected components are small ($< 2/\varepsilon d$ nodes)

**Proof:** If $G$ is $\varepsilon$-far, then there are more than $\varepsilon m$ components. And if we partition these into small and big components, then there are at most $n/big$ big-components. So, $\varepsilon m < CC(G) = #big + #small \leq (n/big) + # small \Rightarrow #small > \varepsilon m - n/big > \varepsilon m/2$ for small $= 2/\varepsilon d$
**Property Tester for Connectivity**

**Tester:** Repeat \( \frac{4}{\varepsilon d} \) times

1. Choose Vertex \( s \) at random
2. Run BFS(s) until either
   2a. Either Visit small +1 \((=\frac{2}{\varepsilon d})\) vertices
   2b. Or no more new vertices can be found
   (found small connected component!!)
3. If ever found a small connected component reject, else accept

**Complexity:** \( \frac{4}{\varepsilon d} \) iterations. Each BFS in this model is \( O\left(\frac{1}{(\varepsilon d)^2}\right) \). Total = \( O(poly(n/\varepsilon m)) \). = SUBLINEAR

**Theorem:** If \( G \) is connected, Tester always accept
If \( G \) is \( \varepsilon \)-far from connected then,
then Tester Rejects with probability > \( \frac{2}{3} \)
Theorem:
If $G$ is connected, Tester always accept
If $G$ is $\varepsilon$-far from connected then, Tester Rejects
with probability $> 2/3$

Proof:

⇒ Obviously, if $G$ is connected will accept.

⇐ the number of nodes in small components is at least
as many as the number of small components, which
by the lemma is $\varepsilon m/2$, i.e fraction $\varepsilon m/2n = \varepsilon d/2$,

$\text{Prob( algorithm accept if } G \text{ is } \varepsilon \text{-far}) <$
$(1-\text{prob(pick vertex in small component)})^{4/\varepsilon d} <$
$(1-\varepsilon d/2)^{4/\varepsilon d} < e^2 < 1/3$
Other Query Models: Dense Graph Model

• Representation: \( n \times n \) Adjacency Matrix \( A \)
  \[
  A_{ij} = \begin{cases} 
  1 & \text{if edge } (i,j) \text{ exists in graph} \\
  0 & \text{otherwise}
  \end{cases}
  \]

• Query model:
  for any \( i,j \) can query edge \( A_{ij} \) in one step

• Distance:
  \( A \) is \( \varepsilon \)-close from having property \( P \) if \( \cdot \varepsilon n^2 \) edges need to be modified
Properties of dense graphs

• Properties: bi-partite, colorability, not containing a forbidden sub-graph, conductance, max cut, partition...

• All above can be tested in constant time!!!

• **SURPRISE**: Some of these properties are NP-complete
Basic Data Streaming Model

- Single pass over the data: $i_1, i_2, \ldots, i_n$
  - Typically, we assume $n$ is known

- Bounded storage (typically $n^\alpha$ or $\log^c n$)
  - Units of storage: bits, words or „elements”
    - (e.g., points, nodes/edges)

- Fast processing time per element
- Randomness: almost always necessary
Many, Many Extensions

- To Testers which approximate distance to input
- To Search Approximation
- To properties of distributions
- In Section: Streaming