Sub-linear Algorithms

Lecture 20
This course so far

**Tractable**

solvable in polynomial time

**Intractable**

not solvable in polynomial time

**Linear Time**

– $\Omega(n)$ 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 $O(n)$ time
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?
Can such algorithms possibly be correct?

- For most problems, they cannot, in the strict sense of the word.

- Relax the correctness requirements from sub-linear algorithms
  - often use randomized algorithms
Today’s Outline

Sub-Linear Time algorithms for

• Decision problems, so called property Testers
  – List Monotonicity
  – Connectivity of Graphs
Warm up 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 \( \epsilon>0 \), asked to:
  - accept \( a \), if \( a=b \)
  - reject \( a \), if \( a \neq b \) in more than \( \epsilon \) fraction of the bits
    (say then \( a \) is \( \epsilon \)-far from \( b \))

Alternatively phrased: number of bit positions
to change \( a \) to get \( b \) is greater than \( \epsilon n \)
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 $\Pr(\text{accept } a) < (1-\varepsilon)^t = 1/e^c$
  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

Deciding $\rightarrow$ Property-Testing

Let $P$ be a decision problem

- A deterministic algorithm for $P$: for every input $x$ the algorithm must answer $P(x)$ correctly
- A probabilistic algorithm for $P$: for every input $x$ the algorithm must answer $P(x)$ correctly with high probability

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

Call these Algorithms: Property Testers

Test accepts $x$

Test rejects $x$
Let $P$: Y/N property (Decision problem)

Need to Specify
- $distance(x,y)$ function between any two inputs
- parameter $\varepsilon > 0$

Property Tester for $P$ with parameter $\varepsilon$:

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

- Notion of distance
  - Hamming Distance: number of bits that differ
  - Edit distance

- Relaxed Requirements to gain efficiency:
  algorithms do not even read the entire input but how
  Do they access it?

- Must specify what is the input representation and what is a
  query to the input

- Complexity Measure for Property Testers:
  - Number of Queries
  - Running Time
Remarks 2

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

• **1-sided error testers:**
  In general could only require that if \( P(x) = \text{yes}, \) \( \text{Prob} \{\text{Tester accepts } x\} > 2/3.\)
  This lecture require probability 1.
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
- 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
- Input is **not too** large but Problems are too hard
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 from monotone (increasing) if must delete more than $\varepsilon n$ values to make it monotone, otherwise, it is $\varepsilon$-close to monotone

Query $i$: get value $X_i$

Requirements for property tester:
• Accept monotone lists
• Reject the list if need to delete more than most $\varepsilon$ fraction of the list to make it monotone
Pick random \( i < j \) and test that \( x_i \leq x_j \)

Consider sequence
\[
c, c-1, c-2, \ldots, 1, 2c, 2c-1, \ldots, c+1, 3c, 3c-1, \ldots, 2c+1, \ldots, n, \ldots, n-c+1
\]

- There are \( n/c \) blocks each size \( c \)

- Longest monotone subsequence has length \( n/c \) (can’t pick from the same group twice)

- Exercise: Using birthday paradox can show a lower bound of \( \Omega(n) \) on the number of queries when \( c = \sqrt{n} \)
Test Attempt 2:
Pick random $i$ and test that $x_i \leq x_{i+1}$

Let the input sequence be:
$(1, 2, \ldots, n/c, 1, 2, \ldots n/c, \ldots, 1, 2, \ldots, n/c)$

- The longest sub-sequence is $c-1+n/c$ long
- $\text{Prob}(\text{Test accepts}) = (1-c/n)$ (always unless pick border points $i = n/c, 2n/c, \ldots$)
- Should repeat $n/c$ times to get prob $< 2/3$
- For $c = \sqrt{n}$, $\Omega(\sqrt{n})$
- This is SUBLINEAR, but actually 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 $x_i$:
  - 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 $X_k > X_k'$

Test with parameter $\varepsilon$

- Repeat $O(1/\varepsilon)$ times:
  - pick a random index $i$, look up $X_i$
  - Do a binary search for $X_i$ in the input sequence
- If every index $i$ was found to be good, output `Yes',
- Else output `No: sequence is not monotone'.
Comments

- Example: 1 3 5 4 7 2 6 10
- Run the algorithm to pick 3 indices: 2,3,8
  _ search for $x_2=3$ in index 2 (2 is a good index),
  _ search for $x_3=5$ in index 3 (do not find it, 3 is bad index)
  _ search for $x_8=10$ in index 8 (find it in location 8 but detect out of order elements $x_4=4$ and $x_7=2$ during the binary search for 10)

To detect that elements are out of order: Keep a pair [smallest seen, largest seen]
Complexity and Correctness

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

**Correctness:**
- If list is sorted, then all i’s are good
- If Claim: If the list is $\varepsilon$-far from monotone, then more than $\varepsilon n$ indices are bad
  
  then $\text{Prob } [\text{test accept } \varepsilon\text{-far list}] = \text{Prob } [\text{always choose good indices}] < (1-\varepsilon)^{c/\varepsilon} < e^c$

Need to prove claim: By counter-positive
It $\leq \varepsilon n$ indices are bad $\Rightarrow$ list is $\varepsilon$-close to monotone
This can be done by proving that elements in good indices form a monotone increasing subsequence
Main Technical Claim: Good indices form a monotone increasing sub-sequence

Proof: Let i,j be two good indices, i<j. Consider the paths in the binary search for $X_i$ and $X_j$. These paths have some longest common prefix of elements which they both compare to starting with $x_{n/2}$. Take index $k$ to be the last location which they both compare to $x_k$. Note that as $i < j$, $x_i$ is found at $i$, and $x_j$ is found at $j$, it must be that $i \leq k \leq j$. We now claim that $x_i \leq x_k \leq x_j$ which $\Rightarrow x_i \leq x_j$

Case 1: If the path to $x_i$ is a prefix of the path to $x_j$ then $k=i$, and to go right at $k$, must be that $x_i = x_x \leq x_j$

Case 2: If not, and since we do not encounter inconsistencies during the binary search, then we claim that $x_i < x_x$ and $x_k < x_j$. 
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 $d = o(n)$: $d = m/n$ (can think of as a constant)
- **Query**: what is $i$-th neighbor of $u$?
Connectivity in Sparse Graphs

- n Adjacency Lists: one per vertex in V,
- **Query:** what is i-th neighbor of u ?

- **Question:** Is Graph Connected?
- **Best Possible Classically:** $\Omega(m)$ (BSF)

- **Show** Sub-Linear Property Tester for Connectivity
- **Actually, constant in** $1/\epsilon$ and $1/d$
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 probability $>2/3$, if G is $\varepsilon$-far from connected
Don’t care otherwise
Intuition for Tester

- Far from connected ⇒
- Many connected components ⇒
- Many small connected components ⇒
- 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 (2)

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 (\(< 2/ed \) 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 4/ed times

1. Choose Vertex s at random
2. Ran BFS(s) until either
   2a. Either Visit small +1 (=2/εd) 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: 4/εd iterations. Each BSF in this model is \(O(1/(\epsilon d)^2)\). Total = \(O(poly(n/\epsilon m))\). = SUBLINEAR

Theorem: If G is connected, Tester always accept
If G is \(\epsilon\)-far from connected then, then Tester Rejects with probability > 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 \( \varepsilon m/2n = \varepsilon d/2 \),

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

• Representation: $nxn$ Adjacency Matrix $A$
  
  $A_{ij} = 1$ if edge $(i,j)$ exists in graph
  $0$ otherwise

• 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

8 2 1 9 1 9 2 4 6 3 9 4 2 3 4 2 3 8 5 2 5 6 ...
Many, Many Extensions

• To Testers which approximate distance to input

• To Search Approximation

• To properties of distributions

• In Section: Streaming