Sub-linear Time Algorithms

Ronitt Rubinfeld
Tel Aviv University
Fall 2014

Scribe?

Big data?

Really Big data

- Impossible to access all of it
- Potentially accessible data is too enormous to be viewed by a single individual
- Once accessed, data can change

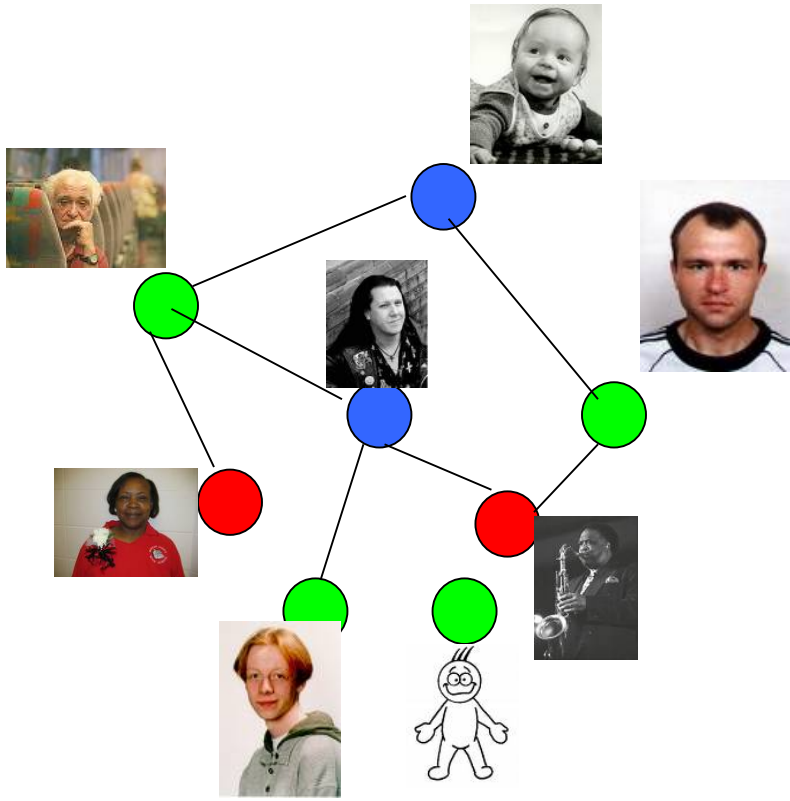
Approaches

- Ignore the problem

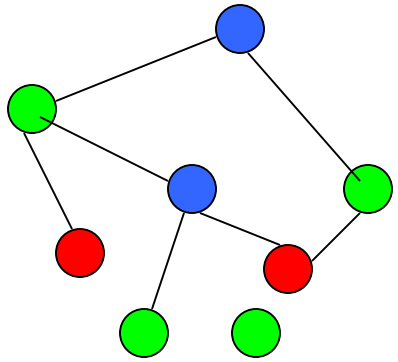
- Develop algorithms for dealing with such data

Small world phenomenon

- each “node’’ is a person
- “edge’’ between people that know each other



Small world property



- *"connected"* if every pair can reach each other
- *"distance''* between two people is the minimum number of edges to reach one from another
- *"diameter''* is the maximum distance between any pair

6 degrees of separation property

In our language:

diameter of the world population is 6

Does earth have the small world property?

- How can we know?
  - data collection problem is immense
  - unknown groups of people found on earth
  - births/deaths

The Gold Standard

- linear time algorithms
  - Inadequate…

What can we hope to do without viewing most of the data?

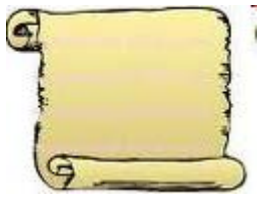
- Can't answer "for all" or "there exists" and other "exactly" type statements:
  - are *all* individuals connected by at most 6 degrees of separation?
  - *exactly* how many individuals on earth are left-handed?
- Maybe can answer?
  - is there a *large* group of individuals connected by at most 6 degrees of separation?
  - is the *average* pairwise distances of a graph roughly 6?
  - *approximately* how many individuals on earth are left-handed?

What can we hope to do without viewing most of the data?

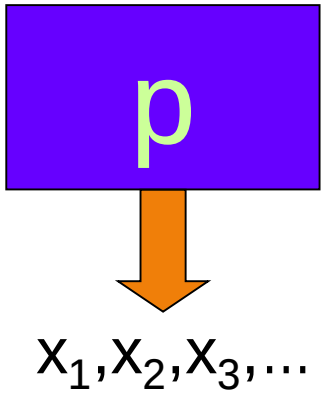
- Must change our goals:
  - for most interesting problems: algorithm must give *approximate* answer
- we know we can answer *some* questions...
  - e.g., sampling to approximate average, median values

Sublinear time models:

- Random Access Queries
  - Can access any word of input in one step



- Samples
  - Can get sample of a distribution in one step,
  - Alternatively, can only get random word of input in one step
    - When computing functions depending on frequencies of data elements
    - When data in random order



Course requirements

- Scribing: 20%
  - Signup on web
  - Must be in latex

- Problem sets: 45%
- Project: 20%
- Class participation: 15%

Project Possibilities

- Read a paper or two or three
  - Suggest some open problems
  - Even better -- Make some progress on them
- Implement an algorithm or two or three

Can work in groups of 2-3

Plan for this lecture

- Introduce sublinear time algorithms

- Basic algorithms
  - Estimating diameter of a point set
  - Property testing of monotonicity
  - Property testing of distinctness

- Sublinear time sampling (if time remains)

I. Classical Approximation Problems

First:

- A very simple example –
  - Deterministic
  - Approximate answer
  - And (of course)…. sub-linear time!

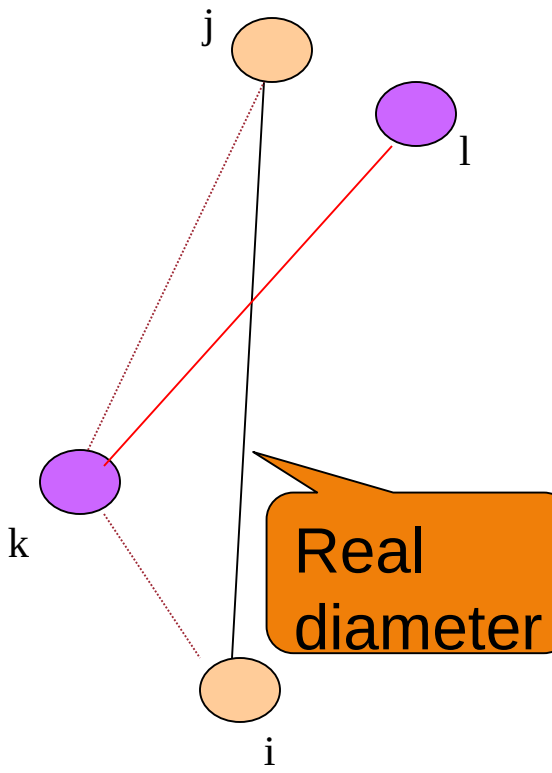
Approximate the diameter of a point set

- 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 $D_{kl} \geq D_{ij}/2$

2-multiplicative approximation

Algorithm



- Algorithm:
  - Pick *k* arbitrarily
  - Pick *l* to maximize D_{kl}
  - Output D_{kl}
- Running time? $O(m) = O(n^{1/2})$
- Why does it work?

$D_{ij} \leq D_{ik} + D_{kj}$ (triangle inequality)

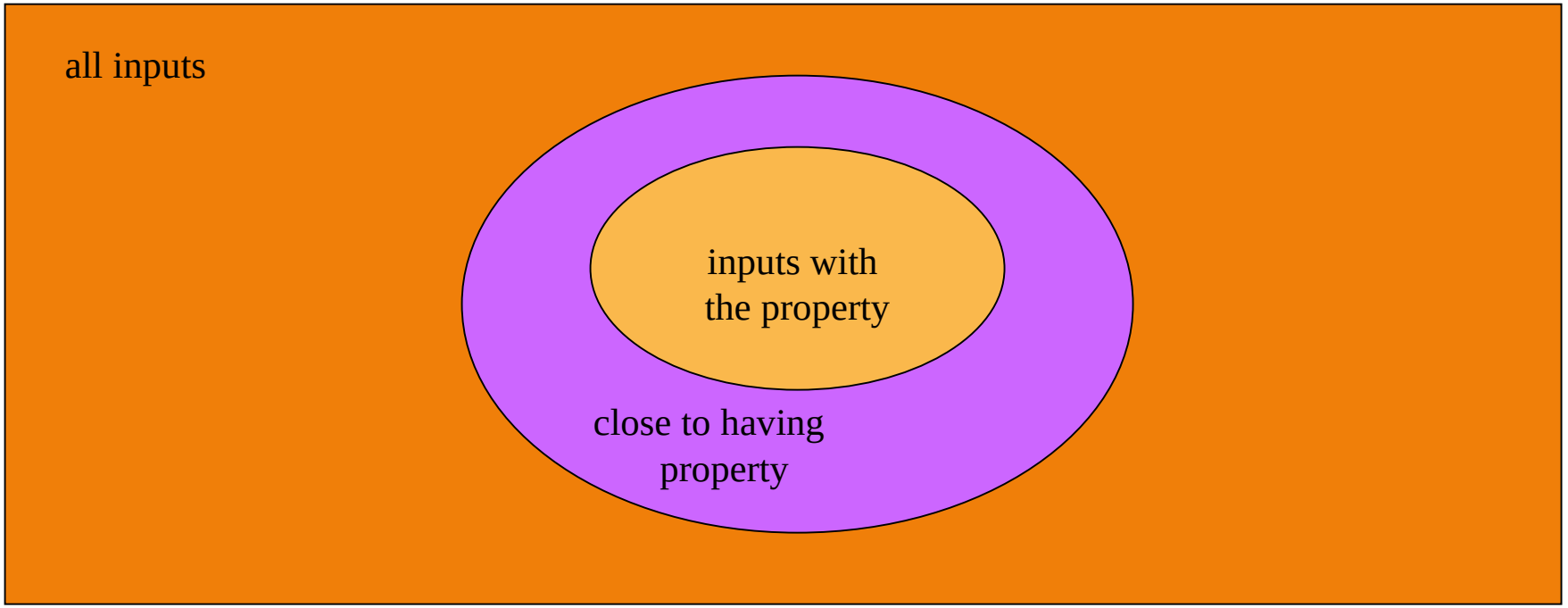
$\leq D_{kl} + D_{kl}$ (choice of *l* + symmetry of *D*)

$\leq 2D_{kl}$

II. Property testing

Main Goal:

- Quickly distinguish inputs that have specific property from those that are far from having the 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 ε-close to sorted if can delete at most εn values to make it sorted. Otherwise, ε-far.

(ε is given as input, e.g., ε=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 ε-far.
  - Equivalently: if list likely to pass test, can change at most ε fraction of list to make it sorted

What if list not sorted, but not far?

Probability of success > ¾

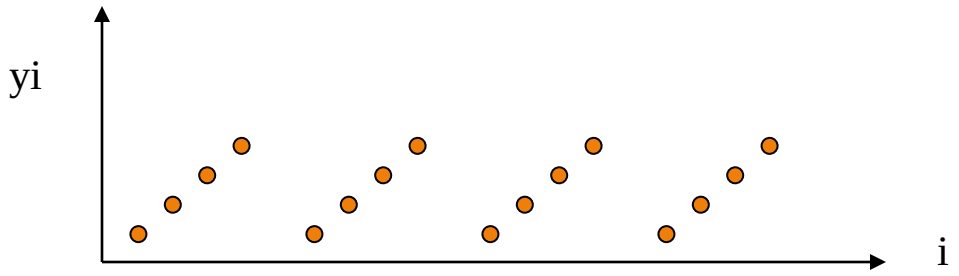
(can boost it arbitrarily high by repeating several times and outputting "fail" if ever see a "fail", "pass" otherwise)

- Can test in $O(1/\varepsilon \log n)$ time

  (and can't do any better!)

An attempt:

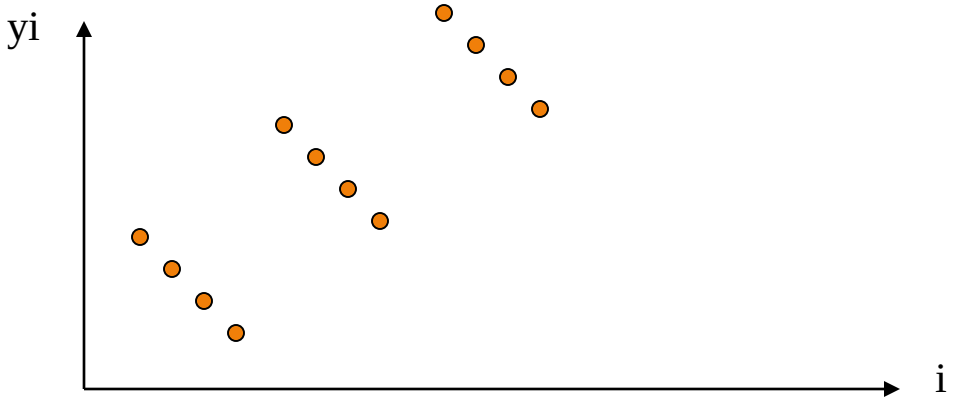
- Proposed algorithm:
  - Pick random *i* and test that $y_i \leq y_{i+1}$
- Bad input type:
  - 1,2,3,4,5,…n/4, 1,2,….n/4, 1,2,…n/4, 1,2,…,n/4
  - Difficult for this algorithm to find “breakpoint”
  - But other tests work well…



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

    $x_1, x_2, \ldots x_n \rightarrow (x_1,1), (x_2,2), \ldots, (x_n,n)$

    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 good (uses distinctness) → 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 → $y_i < y_k < y_j$
    - Thus list is ε-close to monotone (delete $< \varepsilon n$ bad elements)

Another simple property tester

Are all words distinct?

Element distinctness

- Task:
  - Given inputs $x_1,\ldots,x_n$
  - Are all x_i distinct?

- Complexity:
  - Requires at least linear time and queries

    e.g. 1,2,3,4,5,6,7,8,1,9,10,11

Property testing “approximation”

- New task:

  Given *n* elements, distinguish two cases:
  - all distinct
  - number of distinct elements < $(1-\varepsilon)n$

  (If neither case holds, algorithm can output arbitrarily)

- Can we do it in sublinear time?
  - Deterministic sublinear time?
  - What is a good algorithmic idea?

Property testing: element distinctness

Task: Given n elements, distinguish two cases

1. all distinct 
2. number of distinct elements < $(1-\varepsilon)n$

Proposed algorithm:

take several independent samples

if there is a *duplicate* output "fail"

else output "pass"

Analysis for case 1:

If all elements distinct, will always output "pass"

Property testing: element distinctness

Case 2: number of distinct elements < $(1-\varepsilon)n$

Proposed algorithm:

take several samples and "fail" if there is a duplicate

After how many samples will you see a duplicate?

e.g., input 1, 1, 2, 2, 3, 3, …,n/2, n/2 in random order

- is *O(sqrt(n))* enough for this input?
- is *O(sqrt(n))* enough in general?
- can we do better than sampling?

Some analytical challenges:

- Do we sample with or without replacement?

- How are duplicates distributed?
  - All same symbol? e.g., 1,1,1,1,1,1,1,2,3,4,5,6,7,8,9,10,11,…
  - All different symbols? e.g., 1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,…

Plan:

- Analysis:
  - Pair off the duplicates
  - Argue that O(sqrt(n)) samples likely to hit both members of some pair
- Gives lower bound on probability of detecting some duplicate

Recall from probability courses…

- Indicator variables, expectations, expectations of sums of indicators…
- How likely are you to be close to the expectation?
  - Tool #1: Markov's inequality:
    - applies to positive variables
    - Probability of exceeding expectation by more than factor of *c* at most *1/c*

$$\Pr[x > cE[x]] < \frac{1}{c}$$

Recall from probability courses… (cont.)

- How likely are you to be close to the expectation? (cont.)
  - Tool #2: Chebyshev's inequality
    - need upper bound B on standard deviation
    - Probability of deviating from expectation by more than $(c \times B)$ is at most $1/c^2$

$$\Pr[\,|X - E[x]| > cB] < \frac{1}{c^2}$$

Recall from probability courses… (cont.)

- How likely are you to be close to the expectation? (cont.)
  - Tool #3: Chernoff/Hoeffding Bounds
    - Bounds probability that the sum of independent variables deviates from its expectation
    - See course website for pointers
    - Chernoff Bound:
      - Sum = $x_1 + x_2 + \ldots + x_n$ where x_i's are independent 0/1 variables
      - $\Pr[\text{Sum} > (1+\delta)\,E[\text{Sum}]] < e^{-\delta^2 E[\text{Sum}]/3}$
      - $\Pr[\text{Sum} < (1-\delta)\,E[\text{Sum}]] < e^{-\delta^2 E[\text{Sum}]/2}$

Plan:

- Analysis:
  - Pair off the duplicates (order matters!)
  - Argue that O(sqrt(n)) samples likely to hit both of some pair
- More details:
  - Imagine samples divided in two phases
  - Phase 1 samples likely to hit the first member of lots of pairs
  - Phase 2 samples likely to hit the second member of a pair hit in phase 1

Property testing: element distinctness (when number of distinct elements < (1-ε)n) (cont.)

- Slightly modified version of algorithm:
  - $S_1 \leftarrow$ sqrt(n) samples
    - If see duplicate, stop and output "fail"
  - $S_2 \leftarrow c \cdot \text{sqrt}(n)/\varepsilon$ samples
    - If see duplicate with samples in S_1 output "fail" else "pass"

(note: duplicate refers to *same* value in *different* location)

Why do we need *sqrt(n)?*

- Claim: No algorithm can do better than sampling
  - Proof?
  - Recall example

Related question:

Estimating support size of a distribution

Coming soon to theatres near you!