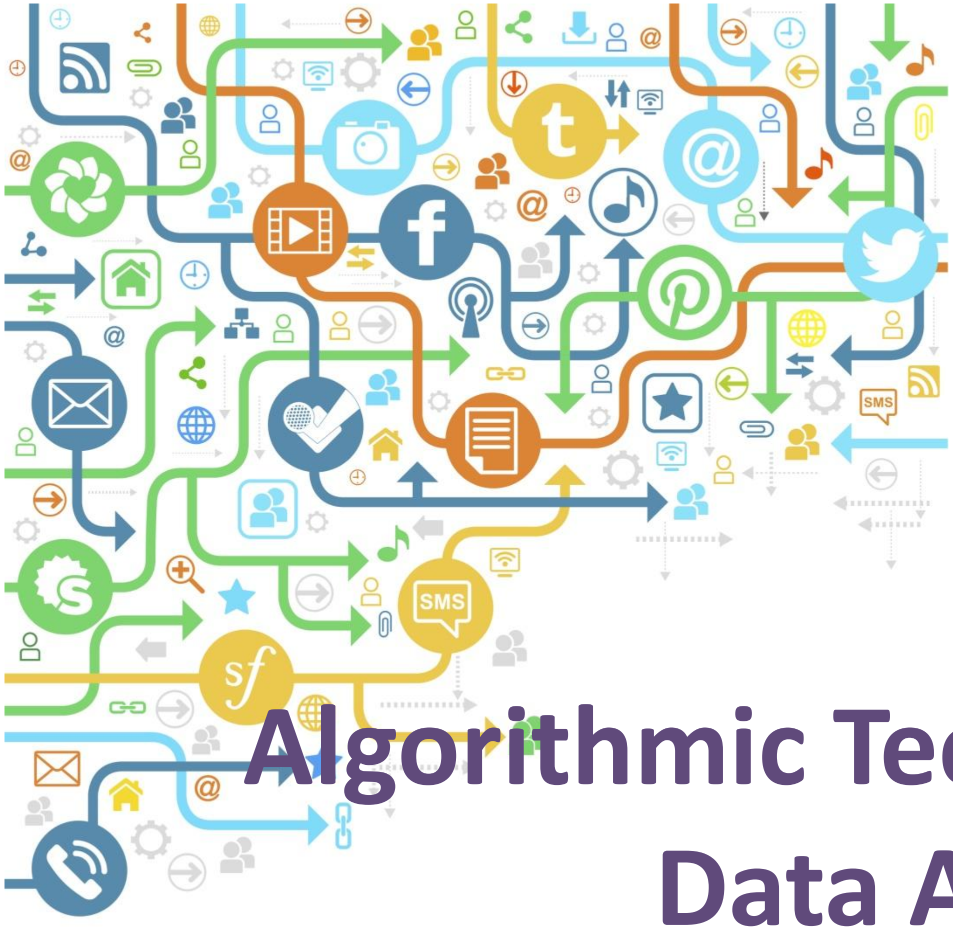
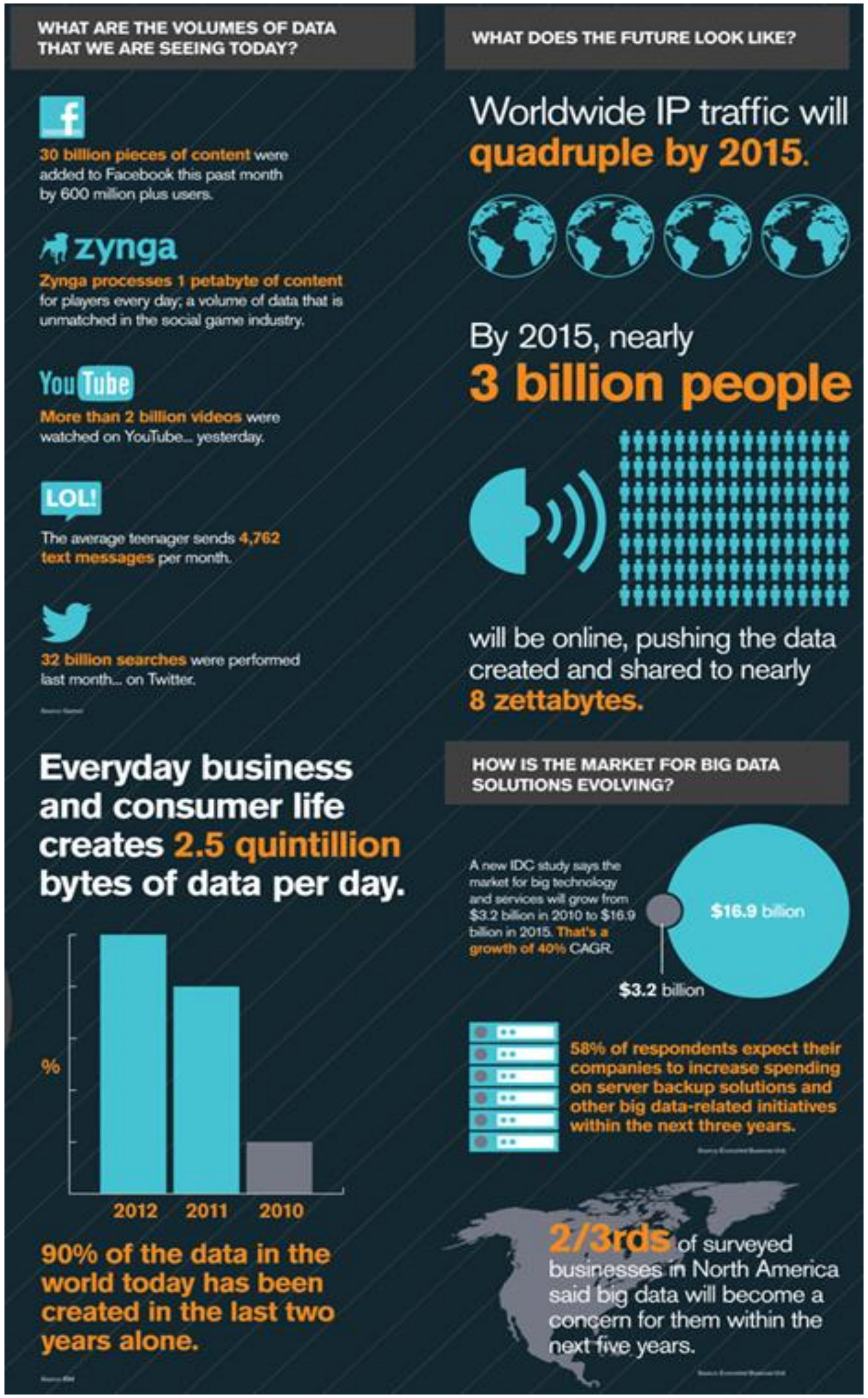


Algorithmic Techniques for Big Data Analysis

Barna Saha

AT&T Lab-Research



WHAT ARE THE VOLUMES OF DATA THAT WE ARE SEEING TODAY?

30 billion pieces of content were added to Facebook this past month by 600 million plus users.

Zynga processes 1 petabyte of content for players every day; a volume of data that is unmatched in the social game industry.

More than 2 billion videos were watched on YouTube... yesterday.

The average teenager sends 4,762 text messages per month.

32 billion searches were performed last month... on Twitter.

Everyday business and consumer life creates 2.5 quintillion bytes of data per day.

90% of the data in the world today has been created in the last two years alone.

WHAT DOES THE FUTURE LOOK LIKE?

Worldwide IP traffic will quadruple by 2015.

By 2015, nearly 3 billion people will be online, pushing the data created and shared to nearly 8 zettabytes.

HOW IS THE MARKET FOR BIG DATA SOLUTIONS EVOLVING?

A new IDC study says the market for big technology and services will grow from \$3.2 billion in 2010 to \$16.9 billion in 2015. That's a growth of 40% CAGR.

58% of respondents expect their companies to increase spending on server backup solutions and other big data-related initiatives within the next three years.

2/3rds of surveyed businesses in North America said big data will become a concern for them within the next five years.

Challenges of Big Data

- **VOLUME**
  —Large amount of data
- **VELOCITY**
  —Needs to be analyzed quickly
- **VARIETY**
  —Different types of structured and unstructured data
- **VERACITY**
  —Low quality data, inconsistencies

This course

- Develop algorithms to deal with such data
  - Emphasis on different models for processing data
  - Common techniques and fundamental paradigm
- Style: algorithmic/theoretical
  - Background in basic algorithms and probability
- However
  - Important problems and practical solutions (mostly)
  - Applied projects are OK and encouraged (more discussions to follow)

Grading

- Scribing once: 20%
- Participation (class and blog): 20%
  - http://csci8980bigdataalgo.wordpress.com/: all information, lecture notes will be posted there
  - Add references
  - Add datasets
  - Write posts on interesting articles
  - Discuss questions from the lecture
- Project: 60%
  - Survey: 30%
    - Read ~5 papers (~2 for background+~3 recent developments)
    - Out of class presentation/discussion
  - Project presentation+ Write-up: 30%
    - Goal to have a project worthy of publication in a good conference in theory/ data bases/ data mining.
- NO EXAMS

Timeline

- Scribe notes
  - Due before next class. Template will be provided.
- Oct 3rd : submit names of 5 survey papers
  - Encouraged to consult with me before
- Oct 17th: project proposal due
  - Max 2 pages + references
  - Must be in latex, single column, article 11 pt.
- Nov 14th : a one page write-up describing progress on project due
- Dec 5th: project final write-up due
  - At most 10 pages in latex + references+ appendix (if required to include omitted proofs/experiments)
  - Introduction with motivation + related works+ technical part + experimental section

Office Hours

- By appointment: Friday 1pm to 5pm
  - Send email to fix a time
- Email: barna@research.att.com, barna.cs@gmail.com
- Where: 6-198 KHKH
- Encouraged to discuss the progress on projects with me throughout the semester

Tentative Syllabus

- Models
  - Small memory algorithms
  - External memory algorithms
  - Distributed algorithms (Map Reduce)
  - Crowdsourcing (People assisted computing)
- Focus: Efficiency vs Quality
  - Near linear time algorithm design
  - Incremental and update efficient algorithm design
  - Sub-linear algorithm design / property testing
  - Sparse transformation
  - Dimensionality reduction
  - Metric embedding

Tentative Course Plan

Sept 5th	Overview, Introduction to Data Streaming
Sept 12th	
Sept 19th	Streaming: Sketching + Sampling, Dimensionality Reduction
Sept 26th	
Oct 3rd	Semi-streaming and External Memory Algorithms
Oct 10th	Map Reduce Algorithms
Oct 17th	
Oct 24th	Property testing
Oct 31st	Sparse transformation or near-linear time algorithm design
Nov 7th	Metric embedding
Nov 14th	Crowdsourcing
Nov 21st	Project Presentation
Nov 28th	Thanksgiving break
Dec 5th	Project Presentation

Plan for this lecture

- 1st half: Overview of the course topics
- 2nd half: Introduction to Data Streaming

Models

- Different models need different algorithms for the same problem
  - Default: Main Memory Model
  - External Memory Model
  - Streaming Model
  - MapReduce
  - Crowdsourcing

1. Do you have enough main memory ?

2. How much disk I/O are you performing ?

3. Is your data changing fast ?

4. Can you distribute your data to multiple servers for fast processing ?

5. Is your data ambiguous that it needs human power to process ?

Counting Distinct Elements

Given a sequence A= $a_1, a_2, \ldots, a_m$ where $a_i \in \{1\ldots n\}$, compute the number of distinct elements in A (denoted by $|A|$).

- Natural and popular statistics, eg.
  - Given the list of transactions, compute the number of different customers (i.e. credit card numbers)
  - What is the size of the web vocabulary ?

Example: 4 5 5 1 7 6 1 2 4 4 4 3 6 6

distinct elements=7

Counting Distinct Elements

- **Default model: Random Access Main Memory Model**
- Maintain an array of size n: B[1,…,n]—initially set to all “0”
- If item “i” arrives set B[i]=1
- Count the number of “1”s in B

Counting Distinct Elements

- **Default model: Random Access Memory Model**
- Maintain an array of size n: B[1,…,n]—initially set to all “0”
- If item “i” arrives set B[i]=1
- Count the number of “1”s in B

➢ O(m) running time ☺

➢ Requires random access to B **(?)**

➢ Requires space n even though the number of distinct elements is small or m < n –domain may be much larger ✖

Counting Distinct Elements

- **Default model: Random Access Memory Model**
- Initialize count=0, an array of lists B[1….O(m)] and a hash function h : {1….n} → {1…O(m)}
- For each a_i
  - Compute $j=h(a_i)$
  - Check if a_i occurs in the list pointed to by B[j]
  - If not, count=count+1 and add a_i to the list
- Return count

Assuming that h(.) is random enough, running time is O(m), space usage O(m).

PROVE IT !

Space is still O(m)

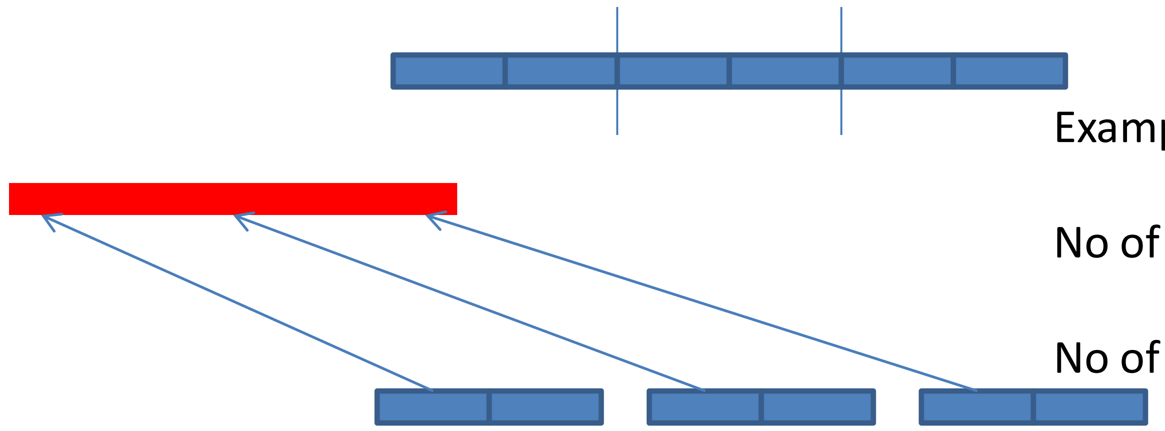
Random access to B for each input

Counting Distinct Elements

- **External Memory Model**
  - M units of main memory
  - Input size m, m >> M
  - Data is stored on disk:
    - Space divided into blocks, each of size B <=M
    - Transferring one block of data into the main memory takes unit time
  - Main memory operations for free but disk I/O is costly
  - **Goal is to reduce number of disk I/O**

Distinct Elements in External Memory

- Sorting in external memory
- External Merge sort
  - Split the data into M/B segments
  - Recursively sort each segment
  - Merge the segments using m/B block accesses



Example: M/B=3

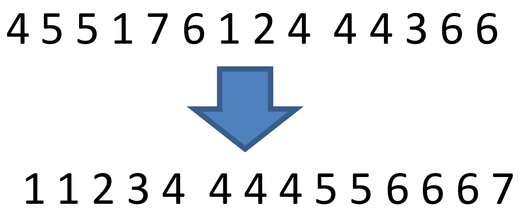
No of disk I/O/merging= m/B

No of recursion call=$\log_{M/B}$ m

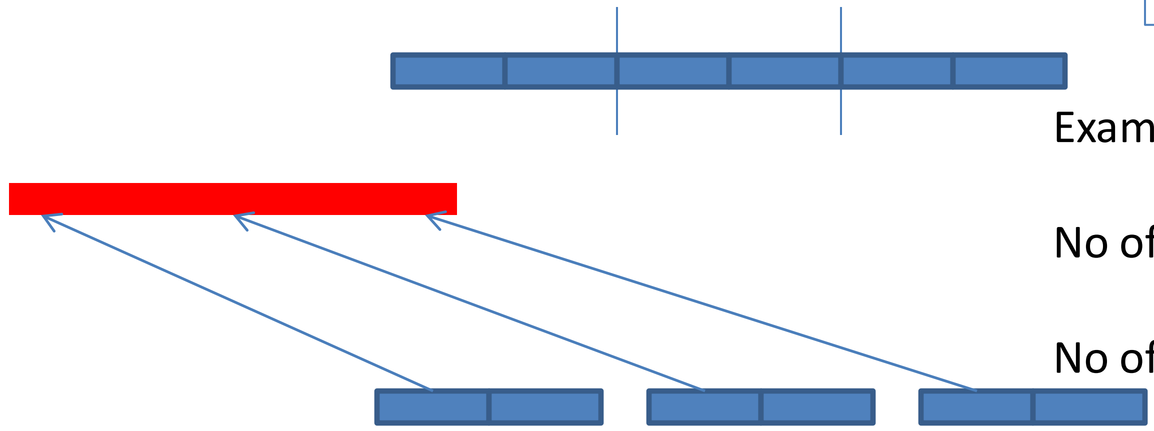
Total sorting time=m/B $\log_{M/B}$ m

Distinct Elements in External Memory

- Sorting in external memory
- External Merge sort
  - Split the data into M/B segments
  - Recursively sort each segment
  - Merge the segments using m/B block accesses



```
Count=1
For j=2,…,m
If $a_j > a_{j-1}$ count=count+1
```



Example: M/B=3

No of disk I/O/merging= m/B

No of recursion call=$\log_{M/B} m$

Total sorting time=$m/B \log_{M/B} m$

Distinct Elements in Streaming Model

- Streaming Model
  - Data comes in streaming fashion one at a time

    (suppose from CD-ROM or cash-register)
  - M units of main memory, M << m
  - Only one pass over data
    - Data not stored is lost

Distinct Elements in Streaming Model

- Suppose you want to know if the number of distinct elements is at least “t”
- Initialize a hash function h:{1,…n} → {1,…,t}
- Initialize the answer to NO
- For each a_i:
    - If $h(a_i)$ ==1, then set the answer to YES

The algorithm uses only 1 bit of storage ! (not counting the random bits for h)

Distinct Elements in Streaming Model

- Suppose you want to know if the number of distinct elements is at least "t"
- Initialize a hash function h:{1,…m} → {1,…,t}
- Initialize the answer to NO, count=0
- For each a_i:
  - If $h(a_i)$ ==1, then count++ (this run returns YES)
- Repeat the above procedure for log n different hash functions from the family
  - Set YES if count > log n (1-1/e) [Boosting the confidence]

The algorithm uses log n bit of storage ! (not counting the random bits for h)

Run log(n) algorithms in parallel using t=2,4,8,…n
Approximate answers with high probability > 1-1/n
Space usage $O(\log^2 n)$

Distinct Elements in Streaming Model

- Suppose you want to know if the number of distinct elements is at least "t"
- Initialize a hash function $h:\{1,\ldots m\} \rightarrow \{1,\ldots,t\}$
- Initialize the answer to NO, count=0
- For each a_i:
  - If $h(a_i) == 1$, then count++ (this run returns YES)
- Repeat the above procedure for log n different hash functions from the family
  - Set YES if count > log n (1-1/e) [Boosting the confidence]

The algorithm uses log n bit of storage ! (not counting the random bits for h)

Run log(n) algorithms in parallel using t=2,4,8,…n
Approximate answers with high probability > 1-1/n
Space usage $O(\log^2 n)$

Approximation and Randomization are essential !

MapReduce Model

- Hardware is relatively cheap
- Plenty of parallel algorithms designed but
  - Parallel programming is hard
    - Threaded programs are difficult to test, debug, synchronization issues, more machines mean more breakdown
- MapReduce makes parallel programming easy

MapReduce Model

- **MapReduce makes parallel programming easy**
  - Tracks the jobs and restarts if needed
  - Takes care of data distribution and synchronization
- But there is no free lunch:
  - Imposes a structure on the data
  - Only allows for certain kind of parallelism

MapReduce Model

- Data:
  - Represented as <Key, Value> pairs
- Map:
  - Data → List < Key, Value> [programmer specified]
- Shuffle:
  - Aggregate all pairs with the same key [handled by system]
- Reduce:
  - <Key, List(Value)>→ <Key, List(Value)> [programmer specified]

Distinct Elements in MapReduce

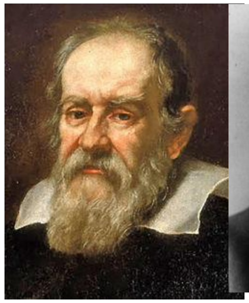
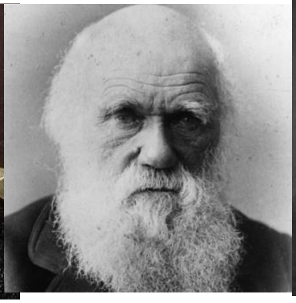
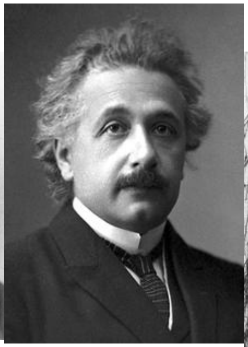
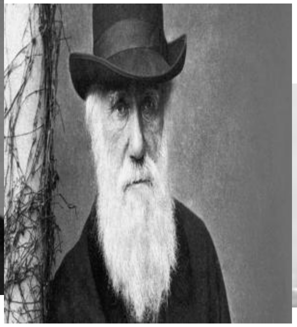
- r servers
- Data
  - $[1,a_1]$, $[2,a_2]$,…., $[n,a_m]$
- Map
  - $[1,a_1]$, $[2,a_2]$,…., $[n,a_m] \rightarrow [1,a_1]$, $[1,a_2]$,…$[1,a_{m/r}]$,$[2,a_{m/r+1}]$,…., $[2,a_{2m/r}]$,….,$[r,a_m]$
- Reduce
  - Reducer 1: $[1,a_1]$, $[1,a_2]$,…$[1,a_{m/r}] \rightarrow [1,a_1]$, $[1,a_2]$,…$[1,a_{m/r}]$ **,[1,h()]** **generates the hash function)**
  - Reducer 2: $[2,a_{m/r+1}]$, $[2,a_{m/r+2}]$,…$[2,a_{2m/r}] \rightarrow [2,a_{m/r+1}]$, $[2,a_{m/r+2}]$,…$[2,a_{2m/r}]$
  - …
- Map

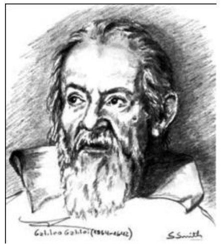
  $[1,a_1]$, $[1,a_2]$,…$[1,a_{m/r}]$,$[2,a_{m/r+1}]$,…., $[2,a_{2m/r}]$,….,$[r,a_m]$,$[1,h()] \rightarrow [1,a_1]$, $[1,a_2]$,…$[1,a_{m/r}]$,$[2,a_{m/r+1}]$,…., $[2,a_{2m/r}]$,….,$[r,a_m]$, $[1,h()]$, $[2,h()]$, …., $[r,h()]$ **makes multiple copies of the hash function for distribution**
- Reduce
  - Reducer 1: $[1,a_1]$, $[1,a_2]$,…$[1,a_{N/r}]$,$[1,h()]$, create sketch B_1, outputs $[1,B_1]$
  - ……
- Map
  - $[1,B_1]$, $[2,B_2]$,….,$[r,B_r] \rightarrow [1,B_1]$, $[1,B_2]$,….,$[1,B_r]$ **gathers all the sketches**
- Reduce
  - Reducer1: $[1,B_1]$, $[1,B_2]$,….,$[1,B_r]$, computes $B= B_1+B_2+\ldots..+B_{r,}$, Follows the Streaming Algorithm to compute distinct elements from the sketch

Crowdsourcing

- Incorporating human power for data gathering and computing
- People still outperform state-of-the-art algorithms for many data intensive tasks
  - Typically involve ambiguity, deep understanding of language or context or subjective reasoning

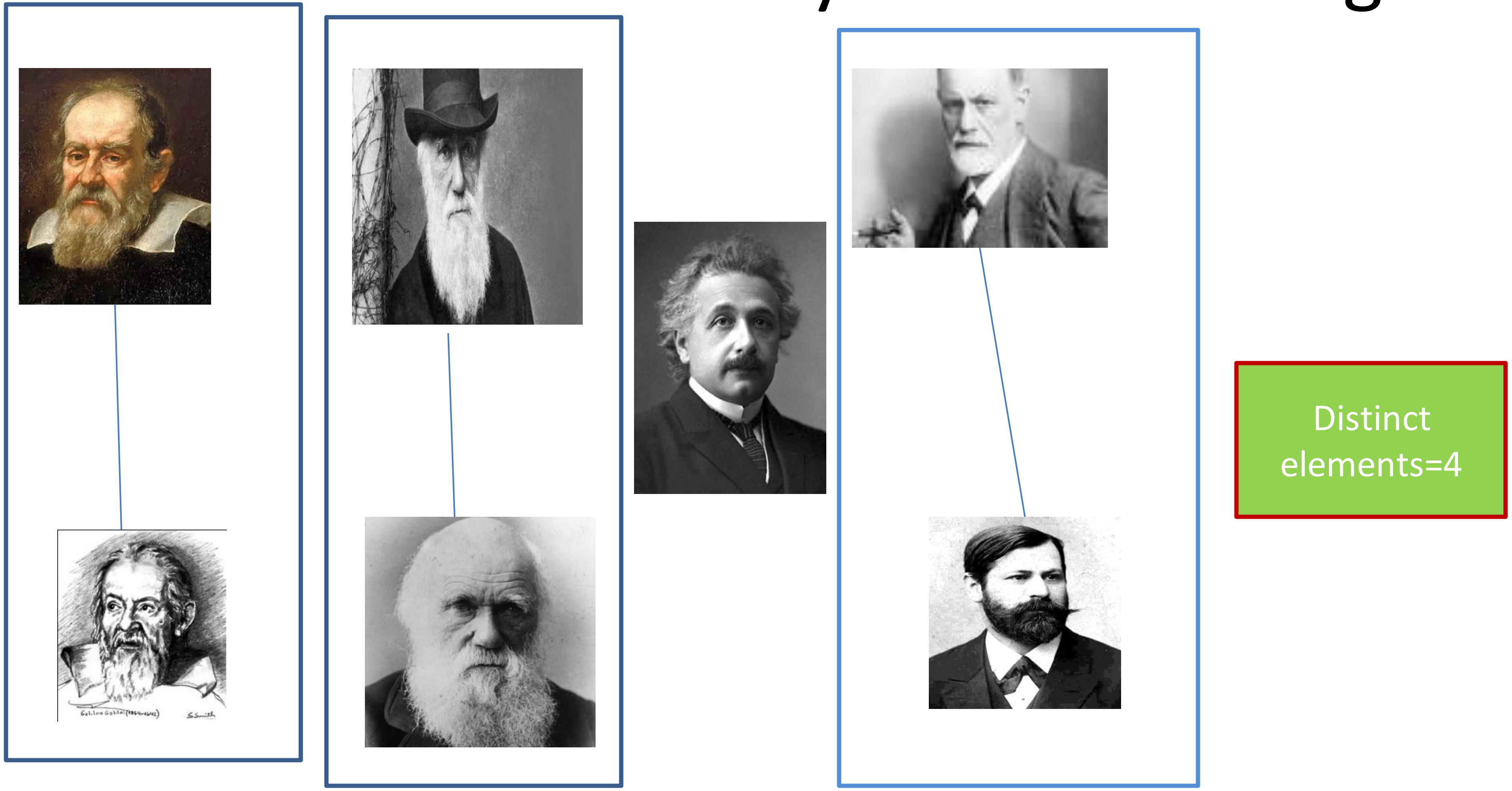
Distinct Elements by Crowdsourcing

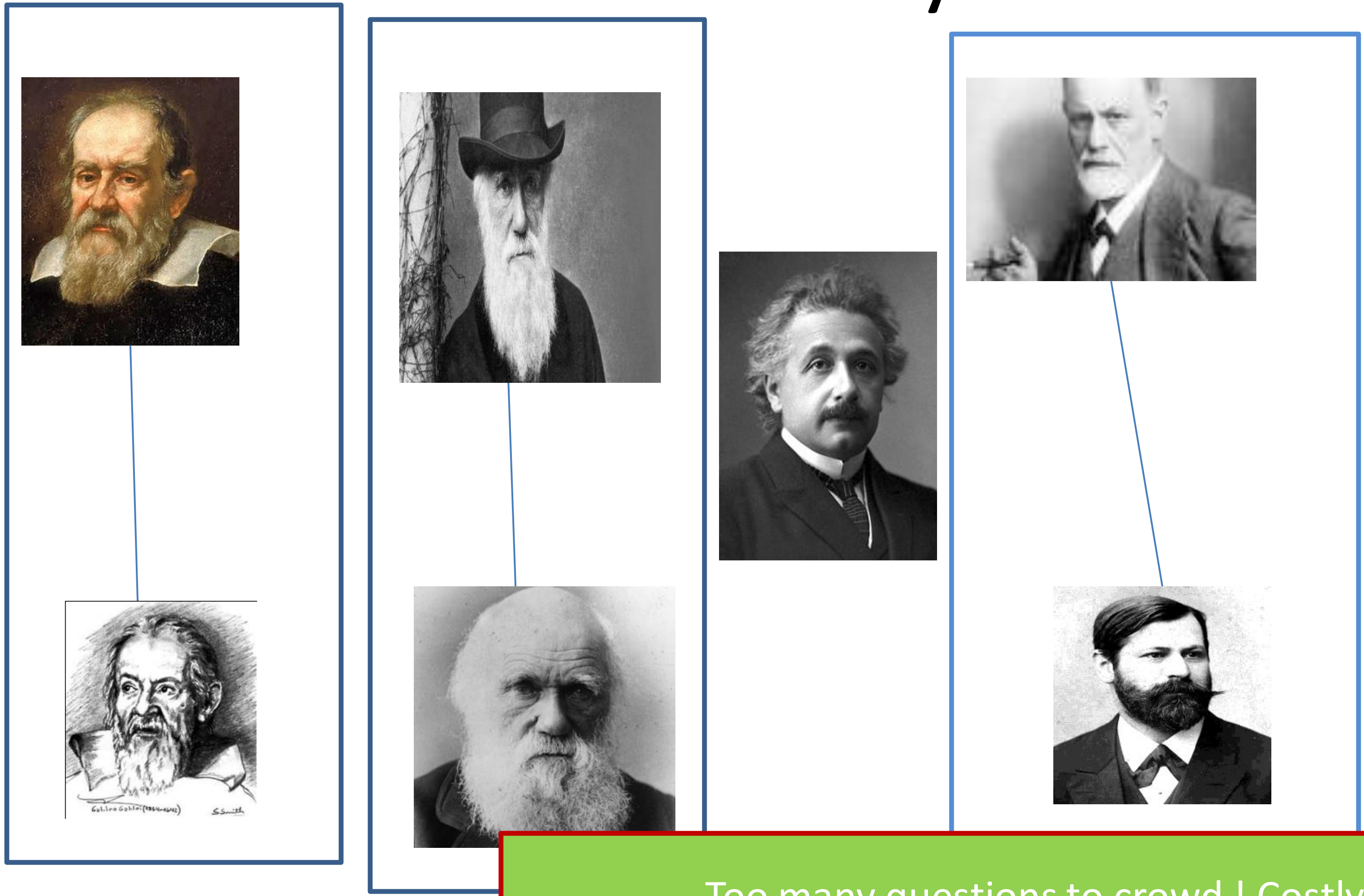


- Ask for each pair if they are equal
- Create a graph with each element as node
- Add an edge between two nodes if the corresponding pairs are returned to be equal
- Return number of connected components
- Also known as record linkage, entity resolution, deduplication

Distinct Elements by Crowdsourcing



Distinct Elements by Crowdsourcing



Distinct elements=4

Too many questions to crowd ! Costly.
Can we reduce the number of questions ?

Scalable Algorithm Design

Near linear time algorithm design

Seemingly best one can do since reading data needs linear time

Is linear time good enough ?

– Don't even read the entire data and return result !!
– Hope: do not require exact answer

Scalable Algorithm Design

Near linear time algorithm design

Seemingly best one can do since reading data needs linear time

Is linear time good enough ?

- Don’t even read the entire data and return result !!
- Hope: do not require exact answer

Property Testing

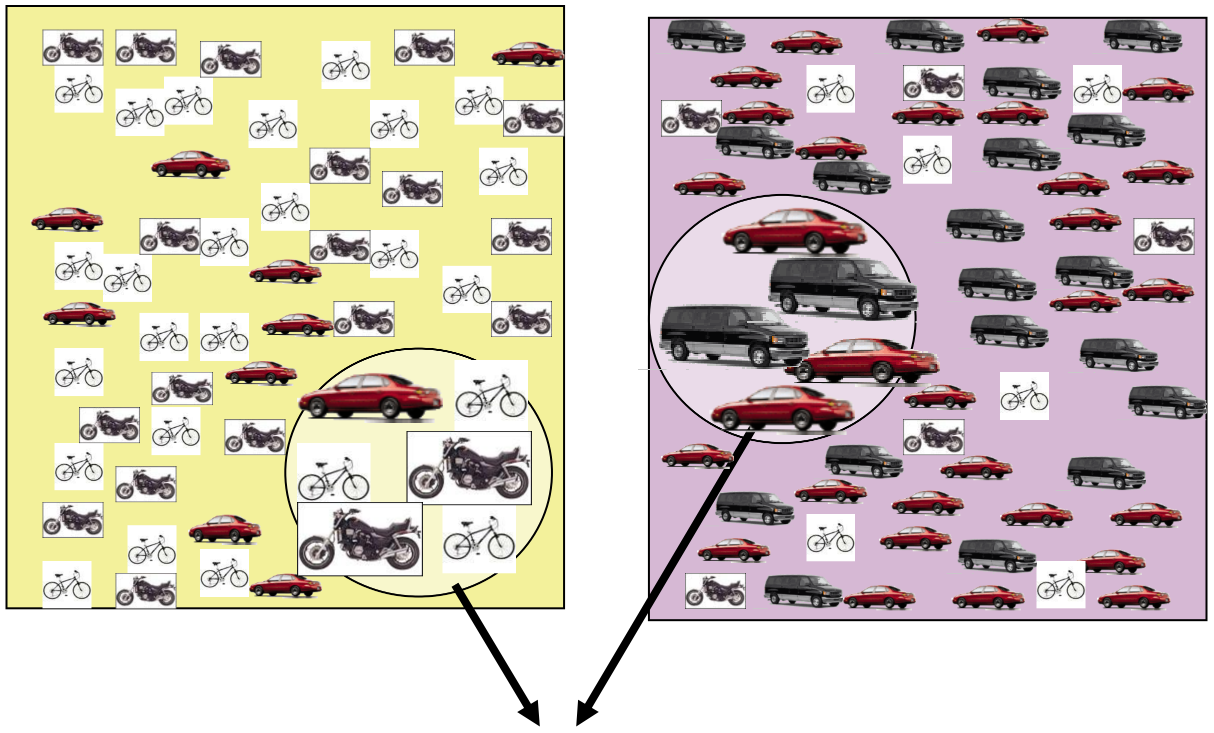
“In the ballpark” vs. “out of the ballpark” tests

- Distinguish inputs that have specific property from those that are far from having the property
- Benefits:
  - May be the natural question to ask
  - May be just as good when data constantly changing
  - Gives fast sanity check to rule out very “bad” inputs (i.e., restaurant bills) or to decide when expensive processing is worth it

Trend change analysis

Transactions of 20-30 yr olds

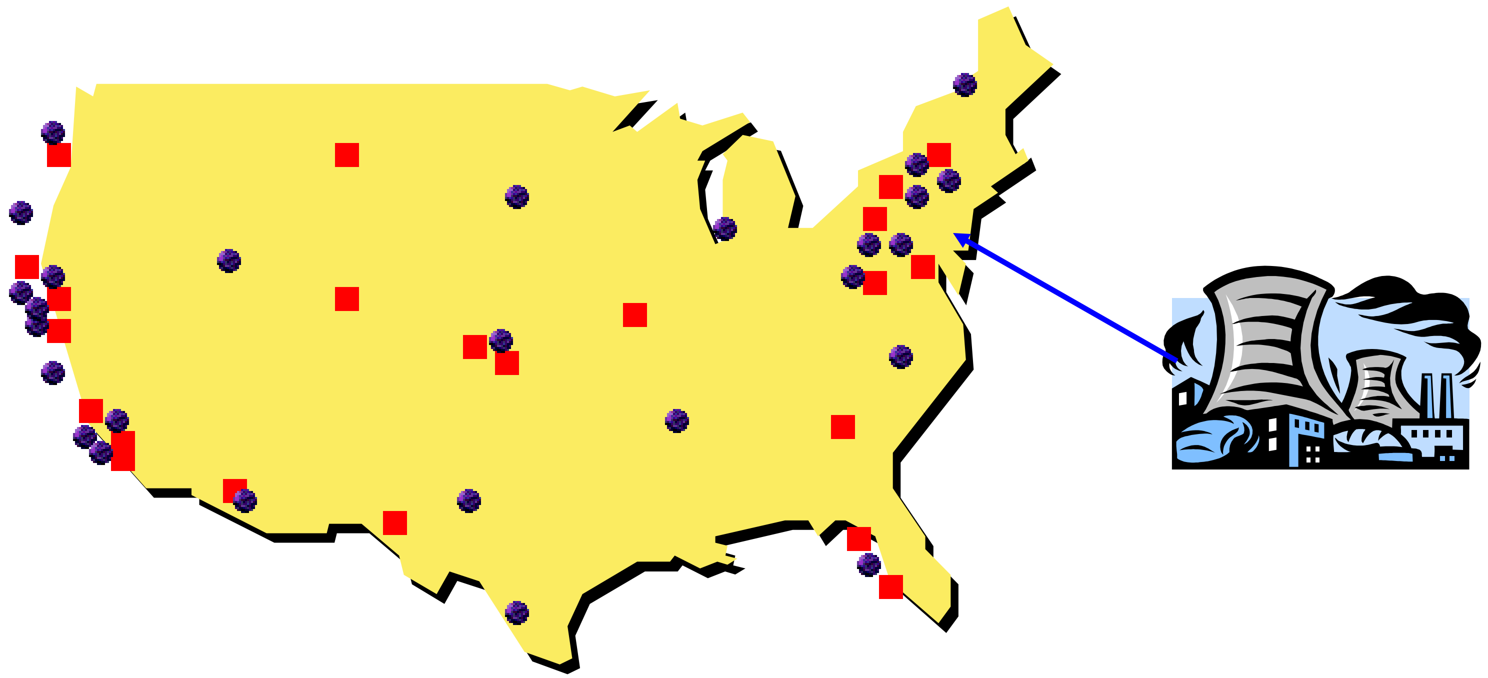
Transactions of 30-40 yr olds



trend change?

Outbreak of diseases

- Do two diseases follow similar patterns?
- Are they correlated with income level or zip code?
- Are they more prevalent near certain areas?



Is the lottery uniform?

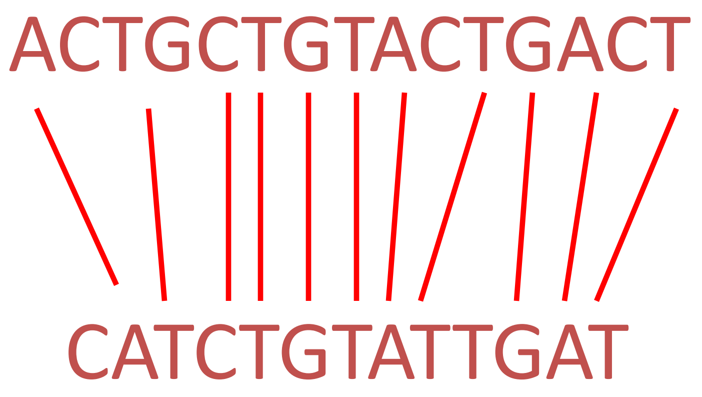
- New Jersey Pick-k Lottery (k =3,4)
  - Pick k digits in order.
  - 10^k possible values.
- Are all values equally likely to occur ?

Global statistical properties:

- Decisions based on samples of distribution
- Properties: similarities, correlations, information content, distribution of data,…

Another Example
Pattern matching on Strings

- Are two strings similar or not? (number of deletions/insertions to change one into the other)
  - Text
  - Website content
  - DNA sequences



(length 15)

(length 13)

match size =11

Pattern matching on Strings

- Previous algorithms using classical techniques for computing edit distance on strings of size *n* use at least n^2 time
  - For strings of size 1000, this is 1,000,000
  - Can you compute edit distance in near linear time ?
  - Can you test whether two strings have edit distance below c (small) or above c’ (big) in sub-linear time ?

Pattern matching on Strings

- Previous algorithms using classical techniques for computing edit distance on strings of size *n* use at least n^2 time
  - For strings of size 1000, this is 1,000,000
  - Can we compute edit distance in near linear time ?
    - Various methods, key is metric embedding
  - Can we test whether two strings have edit distance below c (small) or above c’ (big) in sub-linear time ?
    - Property testing

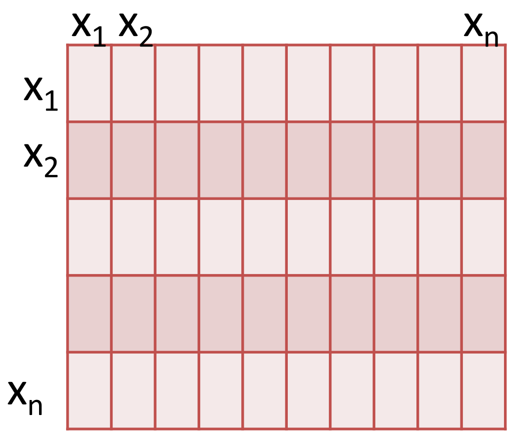
Metric Embedding

- Metric space is a pair (X, d) where X is a set and $d: X \times X \rightarrow [0,\infty]$ is a metric satisfying

  i.) $d(x,y)=0$ if and only if $x=y$

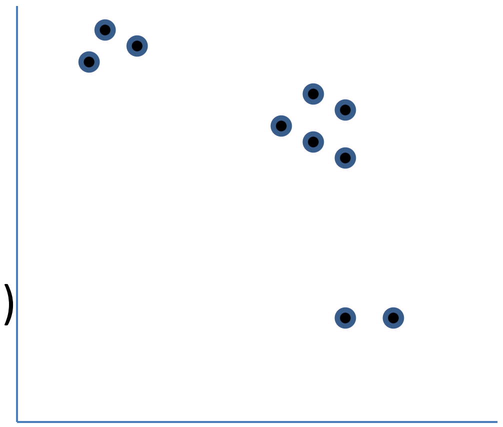
  ii.) $d(x,y)=d(y,x)$

  iii.) $d(x,y)+d(y,z) >= d(x,z)$



$f: X \rightarrow R^2$

$d(x_a, x_b)=d(f(x_a), f(x_b))$



Dissimilarity matrix for bacterial strain in microbiology

1.) succinct representation
2.) easy to understand structure
3.) efficient algorithms

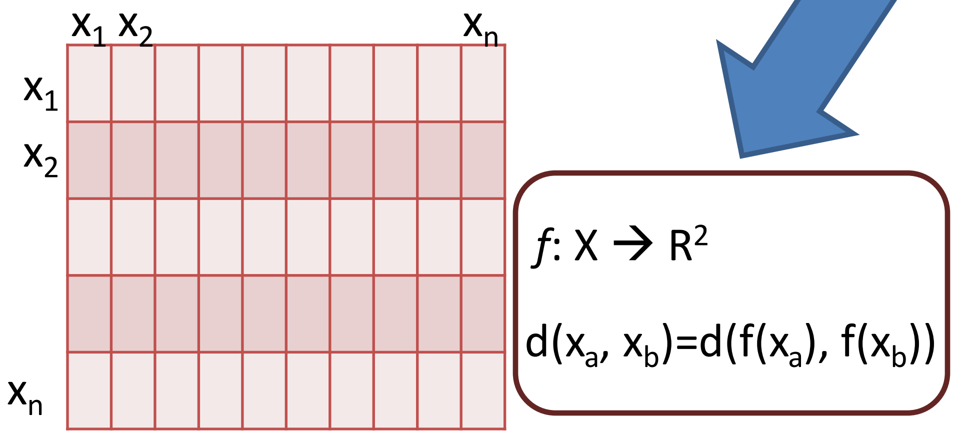
Metric Embedding

- Metric space is a pair (X, d) where X is a set and d: $X \times X \rightarrow [0,\infty]$ is a metric satisfying

  i.) $d(x,y)=0$ if and only if $x=y$

  ii.) $d(x,y)=d(y,x)$

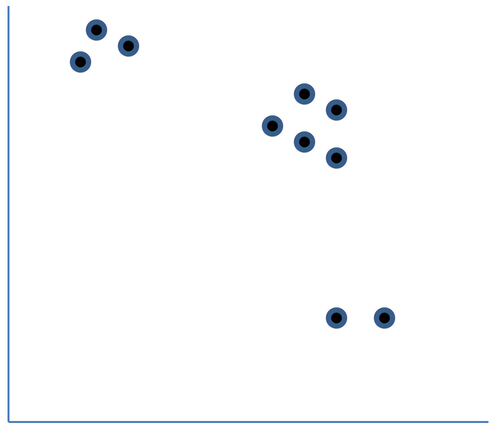
  iii.) $d(x,y)+d(y,z) >= d(x,z)$



f: $X \rightarrow R^2$

$d(x_a, x_b)=d(f(x_a), f(x_b))$

Isometric Embedding



Dissimilarity matrix for bacterial strain in microbiology

1.) succinct representation
2.) easy to understand structure
3.) efficient algorithms

Metric Embedding

- C-embedding: Embedding with distortion

f: X → X', (X,d) and (X,d') are metric

$\exists$ r $\in$ (0,∞) s.t r d(x_a, x_b) <=d(f(x_a), f(x_b))<= C r d(x_a, x_b)

Distortion=C

- Example
  - General n-point metric → Tree metric C=O(log n)
  - Specific metrics → normed space : edit distance (Levenshtein distance to low dimensional l_1)
  - Graphs → t-spanner C=t
  - High dimensional spaces → low dimensional spaces : Johnson-Lindenstrauss theorem: flattening in l_2 ,C=(1+epsilon) **[Dimensionality Reduction]**

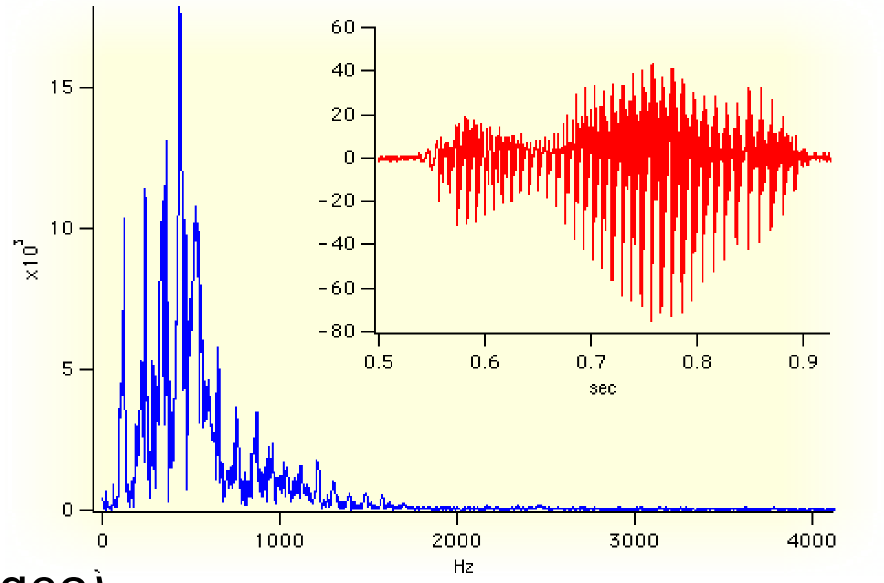
Sparse Transformation

- Sparse Fourier Transform
- Sparse Johnson-Lindenstrauss
  - Fast dimensionality reduction

Fourier Transform

- Discrete Fourier Transform:
  - Given: a signal x[1…n]
  - Goal: compute the frequency vector x' where

$$x'_f = \Sigma_t x_t e^{-2\pi i\, tf/n}$$



🟥 Sampled Audio Data (Time)

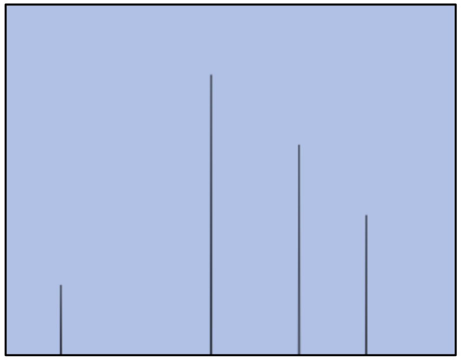
🟦 DFT of Audio Samples (Frequency)

- Very useful tool:
  - Compression (audio, image, video)
  - Data analysis
  - Feature extraction
  - …

- See SIGMOD'04 tutorial "Indexing and Mining Streams" by C. Faloutsos

Computing DFT

- Fast Fourier Transform (FFT) computes the frequencies in time O(n log n)
- But, we can do (much) better if we only care about small number k of “dominant frequencies”
  - E.g., recover   assume it is k-sparse (only k non-zero entries)
    - Exactly k-sparse signals: O(k log n)
    - Approx. k-sparse signals* : O(k log n * log(n/k))



Agenda

- Introduction to Data Streaming Model
- Finding Frequent Items Deterministically
- Lower bound for deterministic computation for distinct items
- Interlude to concentration inequality
- Analysis of counting distinct items algorithms