CIS 399: "Foundations of Data Science"

Massively Parallel Algorithms

Grigory Yaroslavtsev

Warren Center for Network and Data Sciences

http://grigory.us

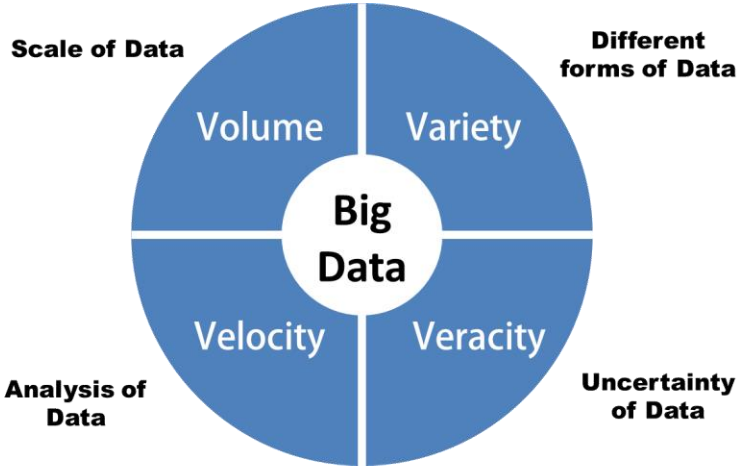
Penn
UNIVERSITY of PENNSYLVANIA

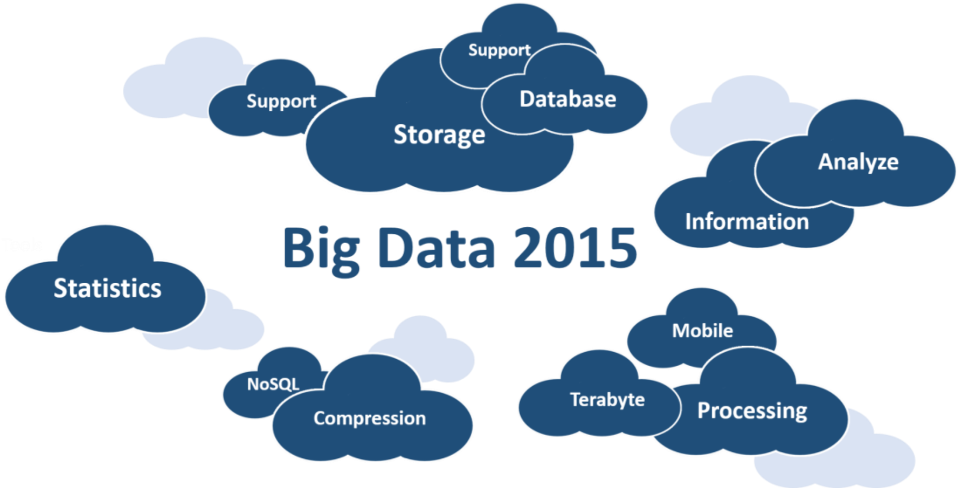
Big Data = buzzword

- **Non-experts, media**:
  - a lot of spreadsheets, medical data,
  - electropop band
  - …

Big Data = buzzword

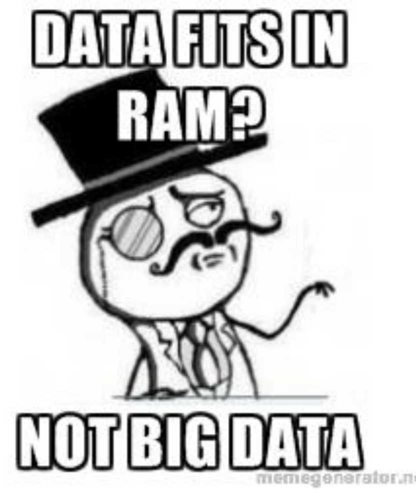
- **Business experts, analysts, data scientists**:
  - Volume, velocity, variety, (veracity)
  - Databases, statistics, cloud computing, machine learning, privacy, …



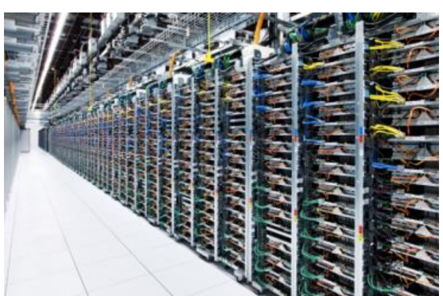


Big Data: technical definition

- **“Big Data” = “Data that doesn’t fit in RAM”**
  - Massively parallel computing: MapReduce/Hadoop/Apache Spark
  - Streaming: Apache Storm, etc.
  - “**algorithms for Big Data**” class at Penn: http://grigory.us/big-data-class.html



hadoop Spark STORM



Algorithms for Big Data

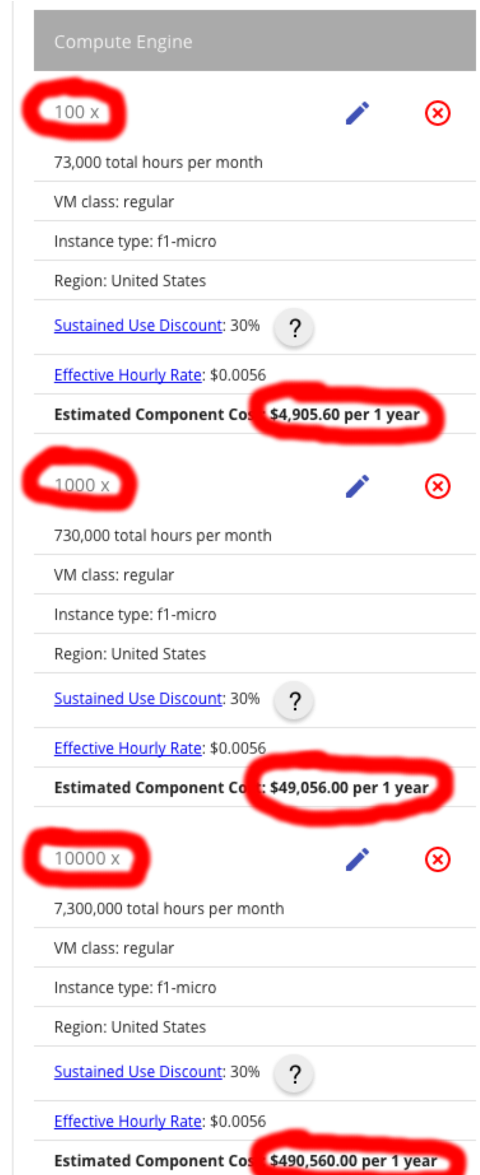
- **Algorithms/theory perspective:** a fundamental challenge
  - Data fits into RAM ⇒ decades of previous work
  - Data doesn’t fit into RAM ⇒ **algorithmic challenges** are **qualitative**, not quantitative

Algorithms for Big Data

- **User's perspective**: paradigm shift brought by cloud services
  - Outsourcing computation and data storage is great for both businesses and **researchers**
  - **Cloud service providers**: Amazon EC2, Google Compute Engine, …
  - **Open source stacks/frameworks**: MapReduce/Hadoop, Apache Spark, etc.

Business perspective

- Pricings:
  - https://cloud.google.com/pricing/
  - https://aws.amazon.com/pricing/
- ~Linear with **space** and **time** usage
  - 100 machines: 5K $/year
  - 10000 machines: 0.5M $/year
- You pay **a lot more** for using provided algorithms
  - https://aws.amazon.com/machine-learning/pricing/

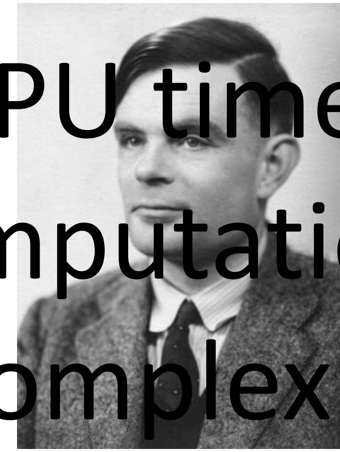


Getting hands dirty

- Cloud computing platforms (all offer free trials):
  - Amazon EC2 (1 CPU/12mo)
  - Microsoft Azure ($200/1mo)
  - Google Compute Engine ($200/2mo)
- Distributed Google Code Jam
  - First time in 2015: https://code.google.com/codejam/distributed_index.html
  - Caveats:
    - Very basic aspects of distributed algorithms (few rounds)
    - Small data ($\sim 1\ GB$, with hundreds MB RAM)
    - Fast query access ($\sim 0.01\ ms$ per request), “data with queries”

“Big Data Theory” = Turing meets Shannon

=



CPU time / Computational Complexity

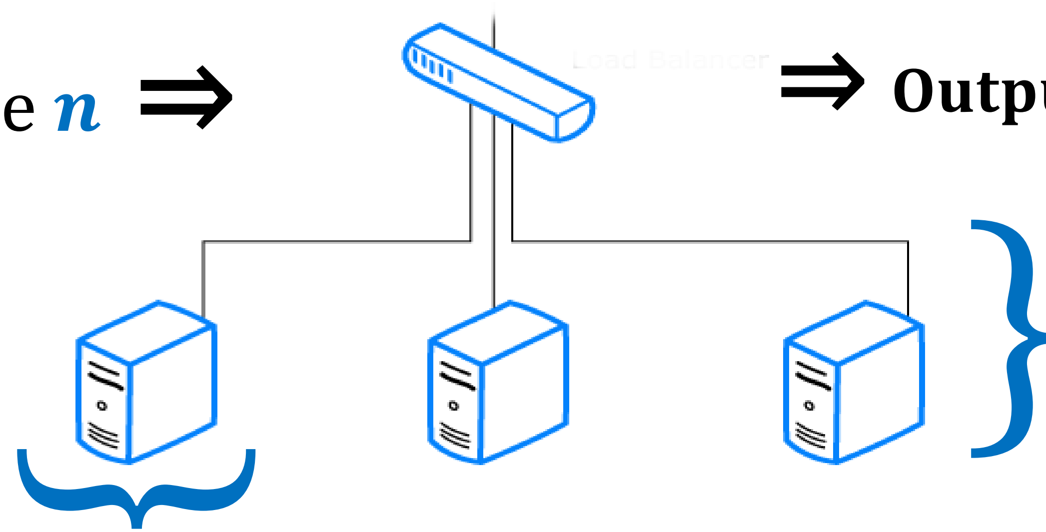
+

Network Time / Information and Communication Complexity

Computational Model

- **Input**: size **n**
- $\boldsymbol{M}$ machines, space $\boldsymbol{S}$ on each ($\boldsymbol{S} = \boldsymbol{n}^{\epsilon}$, $0 < \epsilon < 1$)
  - Constant overhead in total space: $\boldsymbol{M} \cdot \boldsymbol{S} = O(\boldsymbol{n})$
- **Output**: solution to a problem (often size O($\boldsymbol{n}$))
  - Doesn't fit on a single machine ($\boldsymbol{S} \ll \boldsymbol{n}$)

Input: size $\boldsymbol{n}$ $\Rightarrow$



$\Rightarrow$ **Output**: *size* $O(\boldsymbol{n})$

$\boldsymbol{M}$ machines

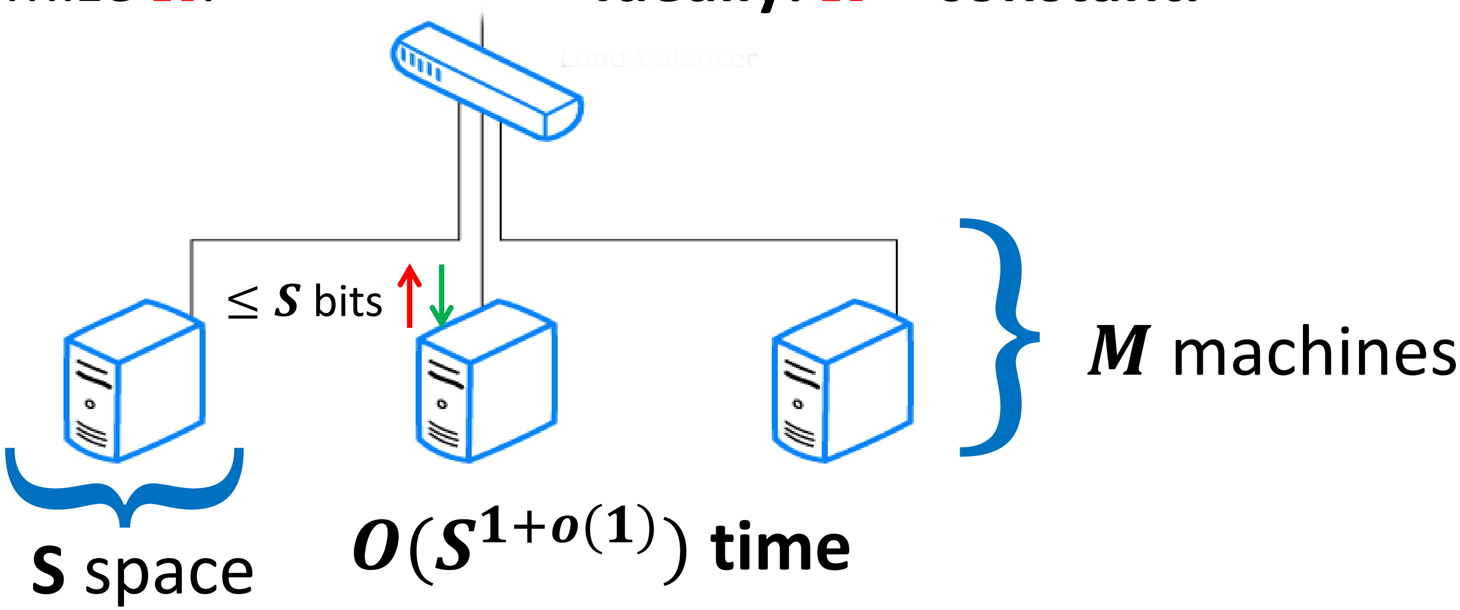
S space

Computational Model

- Computation/Communication in $\boldsymbol{R}$ rounds:
  - Every machine performs a **near-linear time** computation => Total user time $O(\boldsymbol{S}^{1+o(1)}\boldsymbol{R})$
  - Every machine **sends/receives at most $\boldsymbol{S}$ bits** of information => Total communication $O(\boldsymbol{nR})$.

Goal: Minimize $\boldsymbol{R}$.

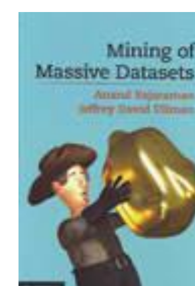
Ideally: $\boldsymbol{R}$ = constant.



MapReduce-style computations

What I won't discuss today

- PRAMs (**shared memory**, multiple processors) (see e.g. [Karloff, Suri, Vassilvitskii'10])
  - Computing XOR requires $\tilde{\Omega}(\log n)$ rounds in CRCW PRAM
  - Can be done in $O(\log_s n)$ rounds of MapReduce
- Pregel-style systems, Distributed Hash Tables (see e.g. Ashish Goel's class notes and papers)
- Lower-level implementation details (see e.g. Rajaraman-Leskovec-Ullman book)



Models of parallel computation

- **Bulk-Synchronous Parallel Model** (BSP) **[Valiant,90]**

  Pro: Most general, generalizes all other models

  Con: Many parameters, hard to design algorithms

- **Massive Parallel Computation** [Feldman-Muthukrishnan-Sidiropoulos-Stein-Svitkina'07, Karloff-Suri-Vassilvitskii'10, **Goodrich-Sitchinava-Zhang'11, ..., Beame, Koutris, Suciu'13**]

  Pros:

  - Inspired by **modern** systems (Hadoop, MapReduce, Dryad, …)
  - Few parameters, **simple** to design algorithms
  - **New algorithmic ideas**, robust to the exact model specification
  - **# Rounds** is an information-theoretic measure => can prove unconditional lower bounds
  - Between **linear sketching** and **streaming with sorting**

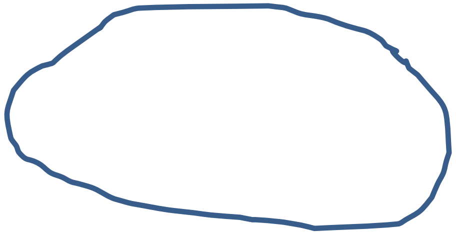
Sorting: Terasort

- Sort Benchmark: http://sortbenchmark.org/
- Sorting $\boldsymbol{n}$ keys on $\boldsymbol{M} = \boldsymbol{O}(\boldsymbol{n}^{1-\epsilon})$ machines
  - Would like to partition keys uniformly into blocks: first $\boldsymbol{n}/\boldsymbol{M}$, second $\boldsymbol{n}/\boldsymbol{M}$, etc.
  - Sort the keys locally on each machine
- Build an approximate histogram:
  - Each machine takes a sample of size $\boldsymbol{s}$
  - All $\boldsymbol{M} * \boldsymbol{s} \leq \boldsymbol{S} = \boldsymbol{n}^{\epsilon}$ samples are sorted locally
  - Blocks are computed based on the samples
- By Chernoff: $\mathbf{M} * \boldsymbol{s} = O\left(\frac{\log n}{\alpha^2}\right)$ samples suffice to compute all block sizes up to $\pm\boldsymbol{\alpha n}$ error with high probability
- Take $\alpha = \frac{\boldsymbol{n}^{\epsilon-1}}{2}$: error $\mathrm{O}(\boldsymbol{S})$
- $\mathbf{M} * \boldsymbol{s} = \widetilde{O}(\boldsymbol{n}^{2-2\epsilon}) = \boldsymbol{O}(\boldsymbol{M}^2) \leq \boldsymbol{O}(\boldsymbol{n}^{\epsilon})$ for $\epsilon \geq 2/3$

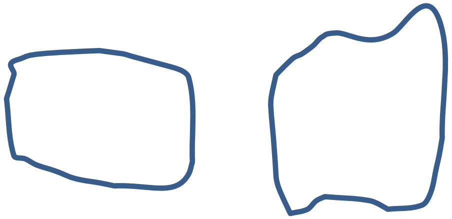
Algorithms for Graphs

- **Dense graphs** vs. sparse graphs
  - **Dense**: $\boldsymbol{S} \gg |V|$
    - Linear sketching: one round
    - "Filtering" (Output fits on a single machine) [Karloff, Suri Vassilvitskii, SODA'10; Ene, Im, Moseley, KDD'11; Lattanzi, Moseley, Suri, Vassilvitskii, SPAA'11; Suri, Vassilvitskii, WWW'11]
  - Sparse: $\boldsymbol{S} \ll |V|$ (or $\boldsymbol{S} \ll$ solution size)

    Sparse graph problems appear hard (**Big open question**: connectivity in $o(\log n)$ rounds?)



vs.



Algorithm for Connectivity

- Blog: http://grigory.us/blog/mapreduce-model/
- Version of Boruvka's algorithm
- Repeat $O(\log n)$ times:
  - Each component chooses a neighboring component
  - All pairs of chosen components get merged
- How to avoid **chaining**?
- If the graph of components is bipartite and only one side gets to choose then no chaining
- **Randomly** assign components to the sides

Algorithm for Connectivity: Setup

Data: **N** edges of an undirected graph.

Notation:

- For $v \in V$ let $\pi(v)$ be its id in the data
- $\Gamma(S) \equiv$ set of neighbors of a subset of vertices S⊆V.

Labels:

- Algorithms assigns a label $\ell(v)$ to each v.
- Let $L_v \subseteq V$ be the set of vertices with the label $\ell(v)$ (invariant: subset of the connected component containing v).

Active vertices:

- Some vertices will be called **active**.
- Every set L_v will have exactly one active vertex.

Algorithm for Connectivity

- Mark every vertex as **active** and let $\ell(v) = \pi(v)$.
- For phases $i = 1,2, \dots, O(\log N)$ do:
  - Call each **active** vertex a **leader** with probability 1/2. If v is a **leader**, mark all vertices in L_v as **leaders**.
  - For every **active** **non-leader** vertex w, find the smallest **leader**(with respect to π) vertex $\mathrm{w}^\star \in \Gamma(L_w)$.
  - If $\mathrm{w}^\star$ is not empty, mark w **passive** and relabel each vertex with label w by $\mathrm{w}^\star$.
- Output the set of CCs, where vertices having the same label according to ℓ are in the same component.

Algorithm for Connectivity: Analysis

- If $\ell(u) = \ell(v)$ then u and v are in the same CC.
- Unique labels w.h.p after $O(\log N)$ phases.
- For every CC # active vertices reduces by a constant factor in every phase.
  - Half of the active vertices declared as non-leaders.
  - Fix an active **non-leader** vertex $\boldsymbol{v}$.
  - If at least two different labels in the CC of v then there is an edge $(\boldsymbol{v'}, \boldsymbol{u})$ such that $\ell(\boldsymbol{v}) = \ell(\boldsymbol{v'})$ and $\ell(\boldsymbol{v'}) \neq \ell(\boldsymbol{u})$.
  - $\boldsymbol{u}$ marked as a **leader** with probability 1/2; in expectation half of the active non-leader vertices will change their label.
  - Overall, expect 1/4 of labels to disappear.
  - By Chernoff after $O(\log N)$ phases # of active labels in every connected component will drop to one w.h.p.

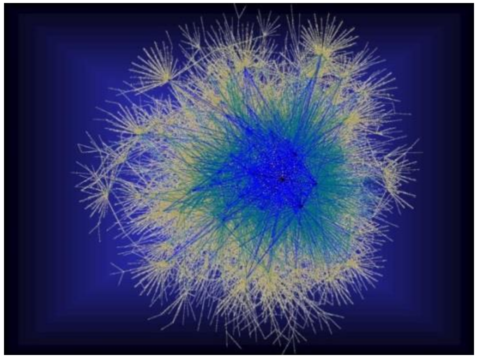
Algorithm for Connectivity: Implementation Details

- Distributed data structure of size $O(|V|)$ to maintain labels, ids, leader/non-leader status, etc.
  - O(1) rounds per stage to update the data structure
- Edges stored locally with all auxiliary info
  - Between stages: use distributed data structure to update local info on edges
- For every **active** **non-leader** vertex w, find the smallest **leader** (w.r.t π) vertex $\mathrm{w}^\star \in \Gamma(L_w)$
  - Each (**non-leader,** **leader**) edges sends an update to the distributed data structure
- Much faster with Distributed Hash Table Service (DHT) [Kiveris, Lattanzi, Mirrokni, Rastogi, Vassilvitskii'14]

Approximating Geometric Problems in Parallel Models

Geometric graph (implicit):

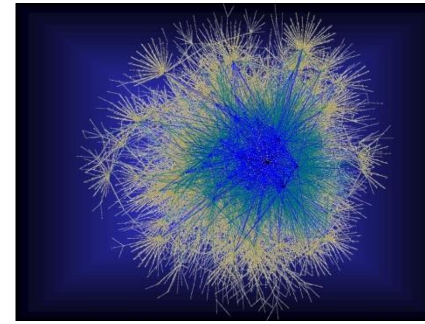
Euclidean distances between **n** points in $\mathbb{R}^{d}$



Already have solutions for old NP-hard problems (Traveling Salesman, Steiner Tree, etc.)

- Minimum Spanning Tree (clustering, vision)
- Minimum Cost Bichromatic Matching (vision)

Geometric Graph Problems

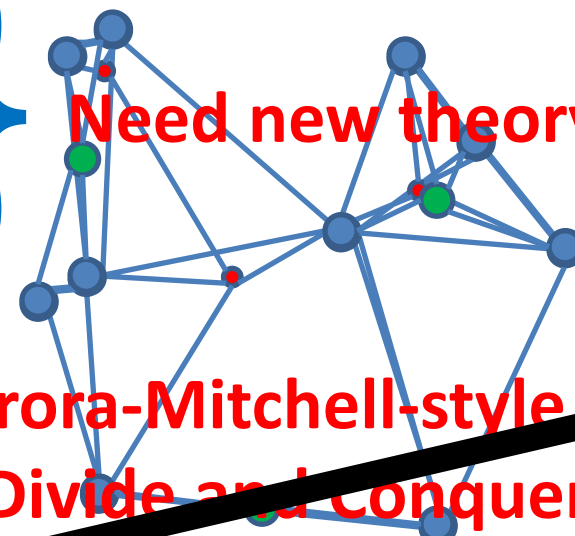


Combinatorial problems on graphs in $\mathbb{R}^d$

Polynomial time ("easy")

- Minimum Spanning Tree
- Earth-Mover Distance =

Min Weight Bi-chromatic Matching



Need new theory!

NP-hard ("hard")

- Steiner Tree
- Traveling Salesman
- Clustering (k-medians, facility location, etc.)

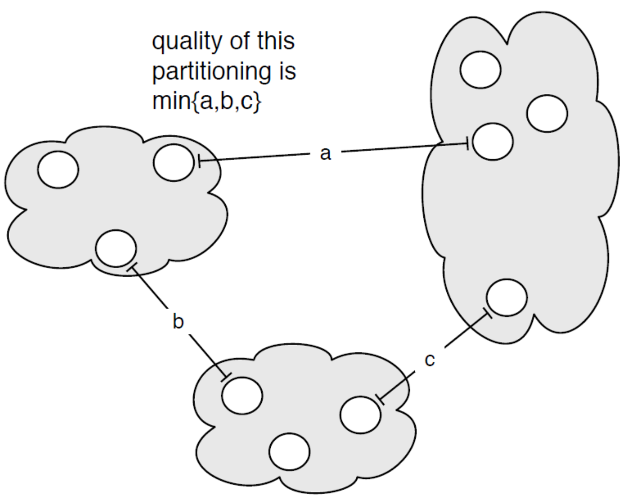
Arora-Mitchell-style "Divide and Conquer", easy to implement in Massively Parallel Computational Models, but bad running time

MST: Single Linkage Clustering

- Blog: http://grigory.us/blog/mapreduce-clustering/
- [Zahn'71] **Clustering** via MST (Single-linkage):

k clusters: remove $k - 1$ longest edges from MST

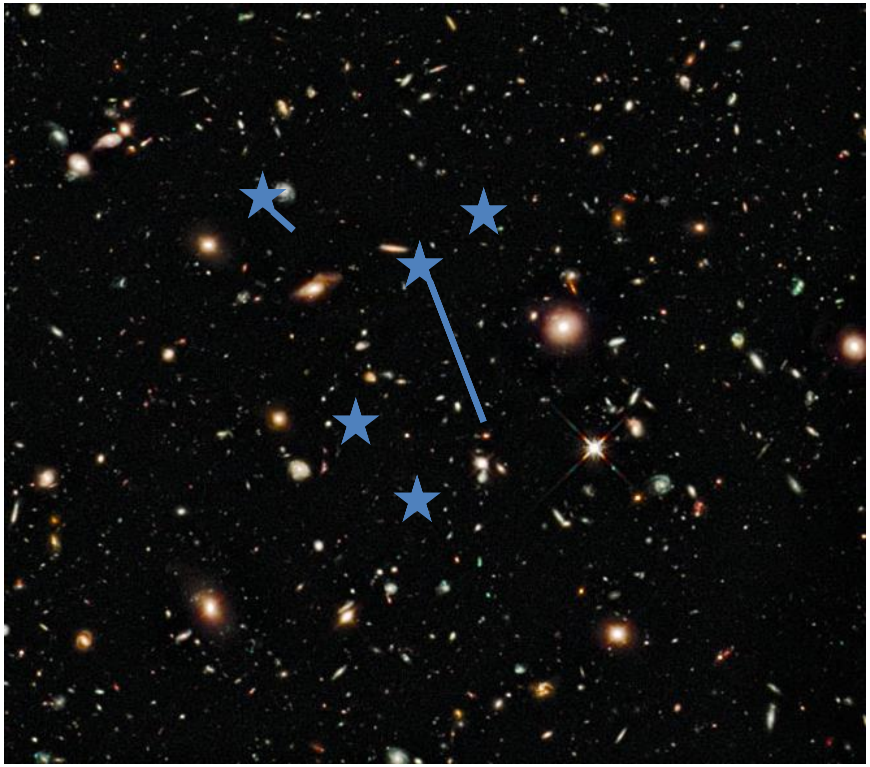
- Maximizes **minimum** intercluster distance



[Kleinberg, Tardos]

Earth-Mover Distance

- Computer vision: compare two pictures of moving objects (stars, MRI scans)



Large geometric graphs

- Graph algorithms: **Dense graphs** vs. sparse graphs
  - **Dense**: $\boldsymbol{S} \gg |V|$.
  - Sparse: $\boldsymbol{S} \ll |V|$.

- **Our setting:**
  - Dense graphs, sparsely represented: O($\mathbf{n}$) space
  - Output doesn't fit on one machine ($\boldsymbol{S} \ll \boldsymbol{n}$)
- **Today:** $(1+\epsilon)$-approximate MST
  - $\boldsymbol{d} = 2$ (easy to generalize)
  - $\boldsymbol{R} = \log_{\boldsymbol{S}} \boldsymbol{n} =$ O(1) rounds ($\boldsymbol{S} = \boldsymbol{n}^{\boldsymbol{\Omega(1)}}$)

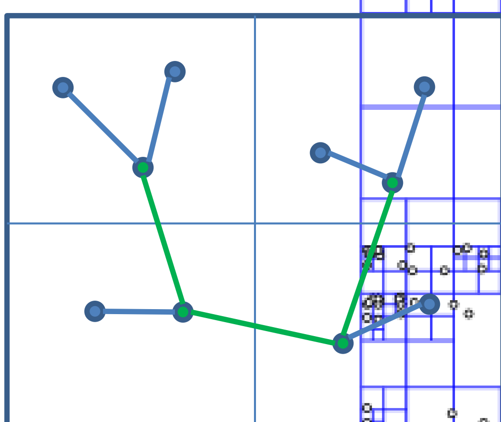
$O(\log n)$-MST in $R = O(\log n)$ rounds

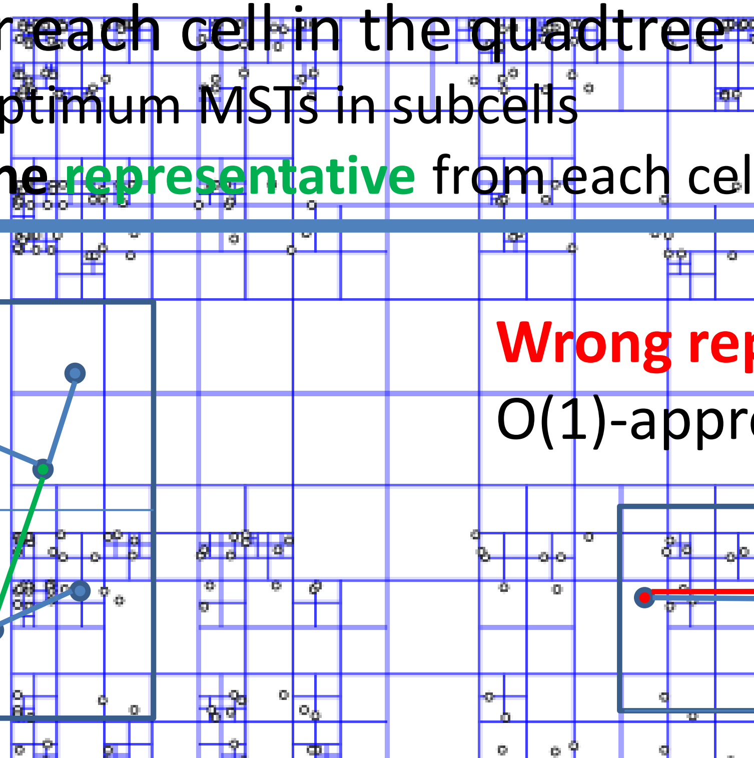
- Assume points have integer coordinates $[0, \dots, \Delta]$, where $\Delta = O(n^2)$.

Impose an $O(\log n)$-depth quadtree

Bottom-up: For each cell in the quadtree

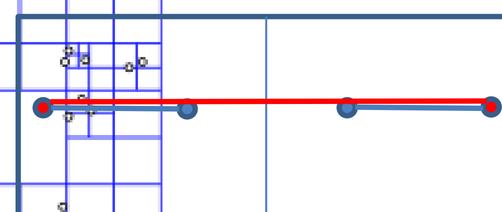
- compute optimum MSTs in subcells
- Use only **one** representative from each cell on the next level





Wrong representative:

O(1)-approximation per level



ϵL-nets

- ϵL-net for a cell C with side length L:

  Collection **S** of vertices in C, every vertex is at distance <= ϵL from some vertex in **S**. (Fact: Can efficiently compute ϵ-net of size $O\left(\frac{1}{\epsilon^2}\right)$)

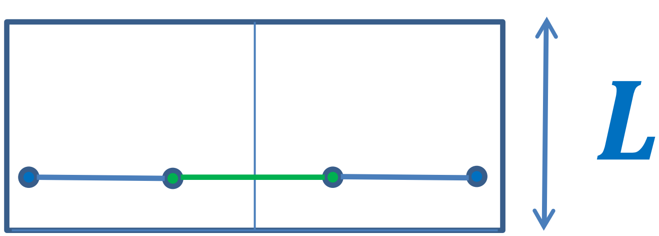
> Bottom-up: For each cell in the quadtree
> – Compute optimum MSTs in subcells
> – Use ϵL-net from each cell on the next level

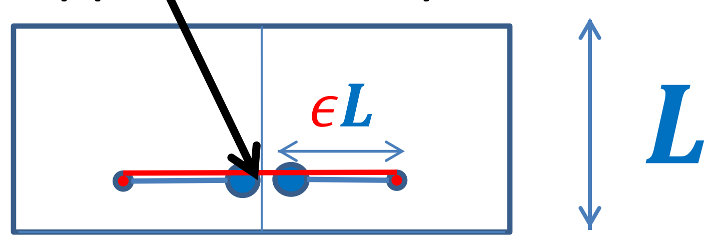
- **Idea**: Pay only O(ϵL) for an **edge** cut by cell with side L
- Randomly shift the quadtree:

  $\Pr[cut\ edge\ of\ length\ \ell\ by\ L] \sim \ell/L$ – charge errors

  Wrong representative

  O(1)-approximation per level





Randomly shifted quadtree

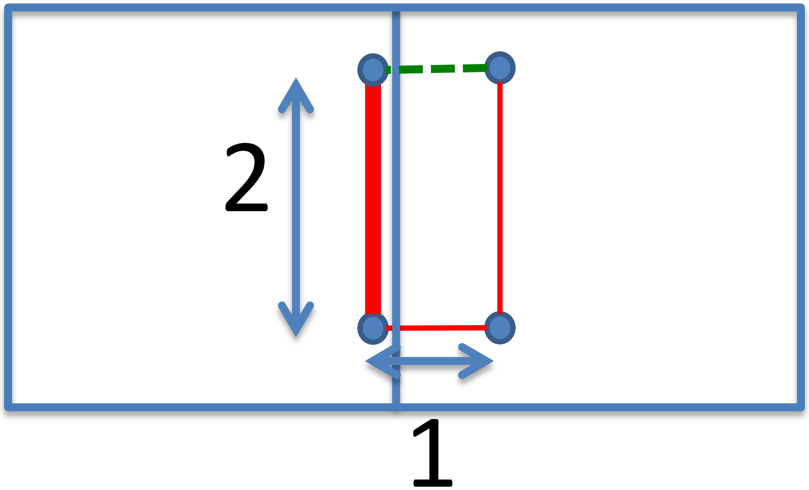
- Top cell shifted by a random vector in $[0, \boldsymbol{L}]^2$

Impose a **randomly shifted** quadtree (top cell length $\mathbf{2\Delta}$)

Bottom-up: For each cell in the quadtree

– Compute optimum MSTs in subcells

– Use $\boldsymbol{\epsilon L}$-net from each cell on the next level



Pay **5** instead of **4**

Bad Cut

$\mathbf{Pr[Bad\ Cut] = \Omega(1)}$

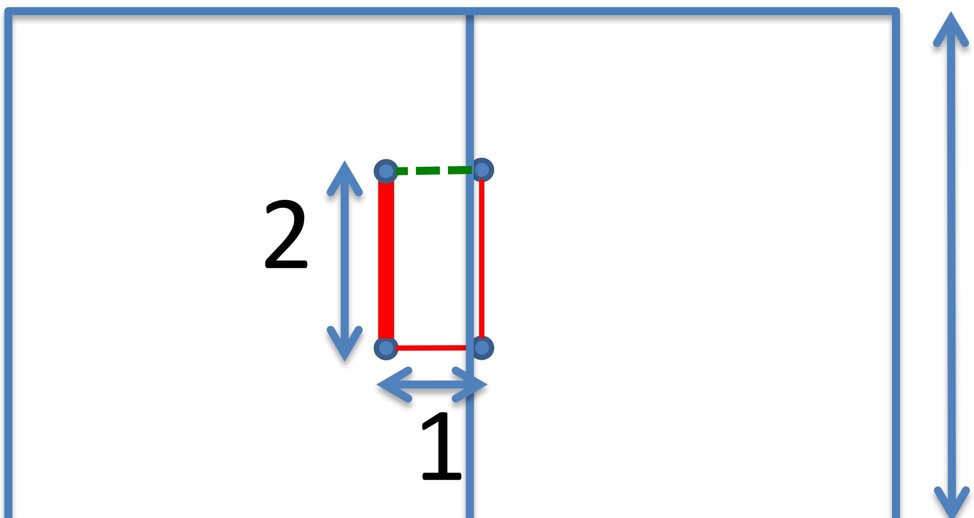
$(1+\epsilon)$-MST in $\mathbf{R} = O(\log\ n)$ rounds

- **Idea:** Only use short edges inside the cells

Impose a **randomly shifted** quadtree (top cell length $\frac{2\Delta}{\epsilon}$)

Bottom-up: For each node (cell) in the quadtree

- compute optimum Minimum Spanning **Forests** in subcells, **using edges of length** $\leq \epsilon L$
- Use only $\epsilon^2 L$-net from each cell on the next level

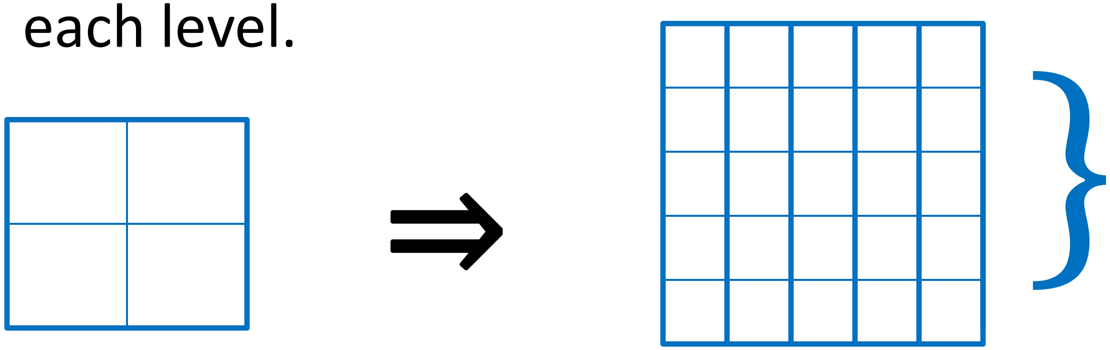


$$L = \Omega\left(\frac{1}{\epsilon}\right)$$

Pr[Bad Cut] $= O(\epsilon)$

$(1+\epsilon)$-MST in $\mathbf{R} = O(1)$ rounds

- $O(\log n)$ rounds => $O(\log_s n)$ = O(1) rounds
  - Flatten the tree: $(\sqrt{M} \times \sqrt{M})$-grids instead of (2x2) grids at each level.



$\sqrt{M} = n^{\Omega(1)}$

Impose a **randomly shifted** $(\sqrt{M} \times \sqrt{M})$-tree

Bottom-up: For each node (cell) in the tree

- compute optimum MSTs in subcells via edges of length $\leq \epsilon L$
- Use only $\epsilon^2 L$-net from each cell on the next level

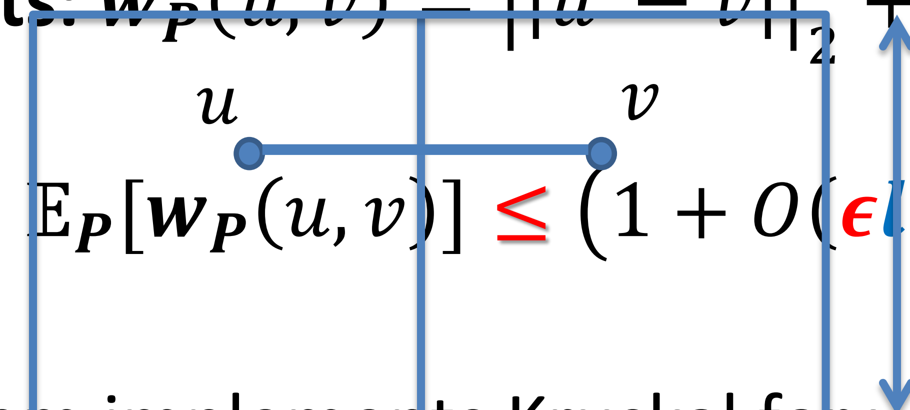
$(1+\epsilon)$-MST in $\mathbf{R} = O(1)$ rounds

Theorem: Let l = # levels in a random tree P

$$\mathbb{E}_P[\mathbf{ALG}] \le \left(1 + O(\epsilon l d)\right)\mathbf{OPT}$$

Proof (sketch):

- $\Delta_P(u,v)$ = cell length, which first partitions (u,v)
- **New weights:** $w_P(u,v) = \|u-v\|_2 + \epsilon \Delta_P(u,v)$



$$\|u-v\|_2 \le \mathbb{E}_P[w_P(u,v)] \le \left(1 + O(\epsilon l d)\right)\|u-v\|_2$$

- Our algorithm implements Kruskal for weights w_P

"Solve-And-Sketch" Framework

$(1+\epsilon)$-**MST**:

- "**Load balancing**": partition the tree into parts of the same size
- **Almost linear time locally**: Approximate Nearest Neighbor data structure [Indyk'99]
- Dependence on dimension **d** (size of ϵ-net is $O\left(\left(\frac{d}{\epsilon}\right)^{d}\right)$)
- Generalizes to bounded **doubling dimension**
- Implementation in MapReduce

“Solve-And-Sketch” Framework

$(1+\epsilon)$-**Earth-Mover Distance, Transportation Cost**

- No simple “divide-and-conquer” Arora-Mitchell-style algorithm (unlike for general matching)
- Only recently sequential $(1+\epsilon)$-apprxoimation in $O_\epsilon(\boldsymbol{n}\log^{O(1)}\boldsymbol{n})$time [Sharathkumar, Agarwal ‘12]

Our approach (convex sketching):

- Switch to the flow-based version
- In every cell, send the flow to the closest net-point until we can connect the net points

"Solve-And-Sketch" Framework

Convex sketching the cost function for $\boldsymbol{\tau}$ net points

- $F: \mathbb{R}^{\boldsymbol{\tau}-1} \to \mathbb{R}$ = the cost of routing fixed amounts of flow through the net points
- Function $F' = F$ + "normalization" is monotone, convex and Lipschitz, $(1+\boldsymbol{\epsilon})$-approximates F
- We can $(1+\boldsymbol{\epsilon})$-sketch it using a lower convex hull

Thank you! http://grigory.us

- More in the CIS 700 class: http://grigory.us/big-data-class.html