

CSCI B609: “Foundations of Data Science”

Lecture 21/22: Massively Parallel Algorithms

Slides at <http://grigory.us/data-science-class.html>

Grigory Yaroslavtsev

<http://grigory.us>

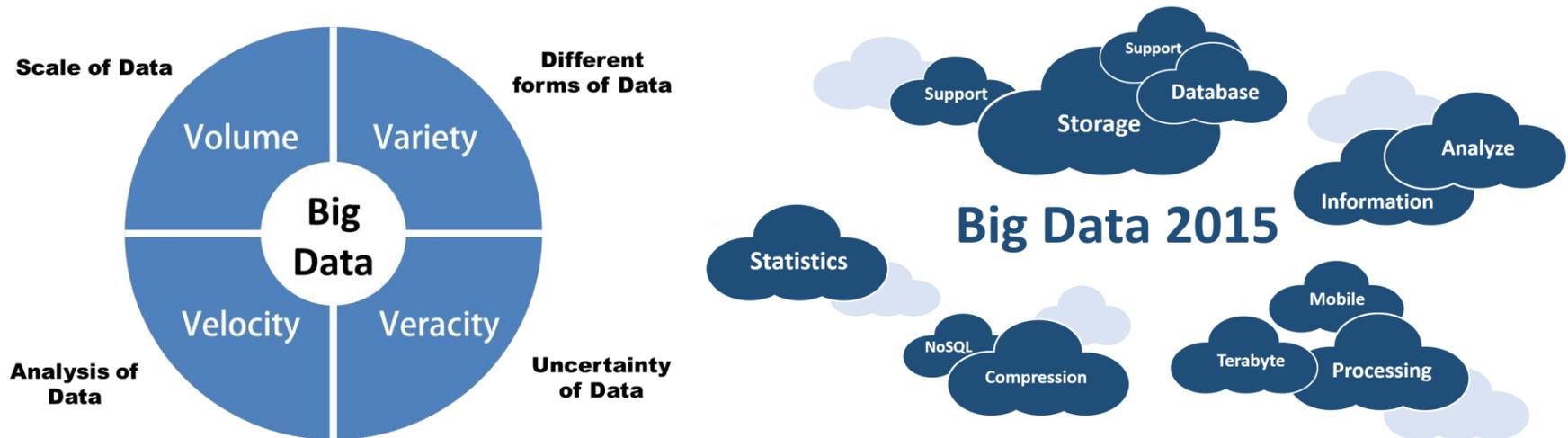
Big Data = buzzword

- **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, ...



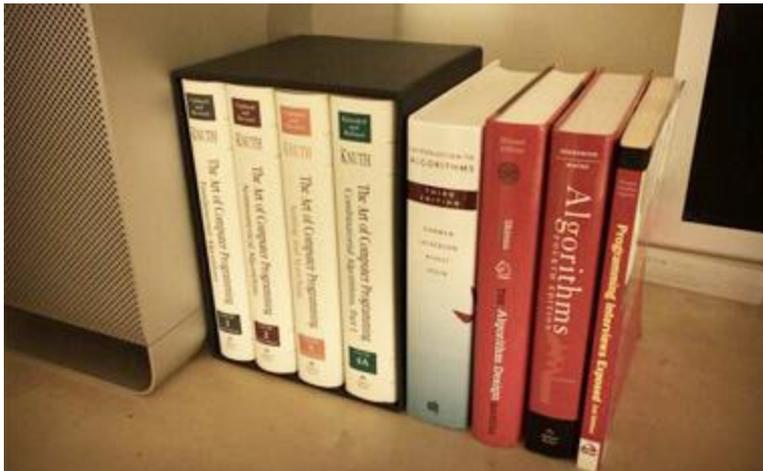
Big Data: technical definition

- **“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>



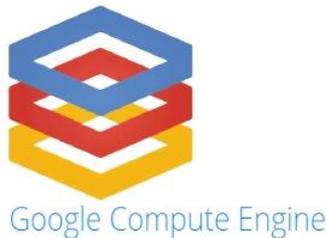
Algorithms for Big Data

- **Algorithms/theory perspective: a fundamental challenge**
 - Data fits into RAM \Rightarrow decades of previous work
 - Data doesn't fit into RAM \Rightarrow **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/>

Compute Engine	
100 x	 
73,000 total hours per month	
VM class: regular	
Instance type: f1-micro	
Region: United States	
Sustained Use Discount: 30% ?	
Effective Hourly Rate: \$0.0056	
Estimated Component Cost:	\$4,905.60 per 1 year
1000 x	 
730,000 total hours per month	
VM class: regular	
Instance type: f1-micro	
Region: United States	
Sustained Use Discount: 30% ?	
Effective Hourly Rate: \$0.0056	
Estimated Component Cost:	\$49,056.00 per 1 year
10000 x	 
7,300,000 total hours per month	
VM class: regular	
Instance type: f1-micro	
Region: United States	
Sustained Use Discount: 30% ?	
Effective Hourly Rate: \$0.0056	
Estimated Component Cost:	\$490,560.00 per 1 year

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 (~ 1 GB, with hundreds MB RAM)
- Fast query access (~ 0.01 ms per request), “data with queries”

“Big Data Theory” = Turing meets Shannon

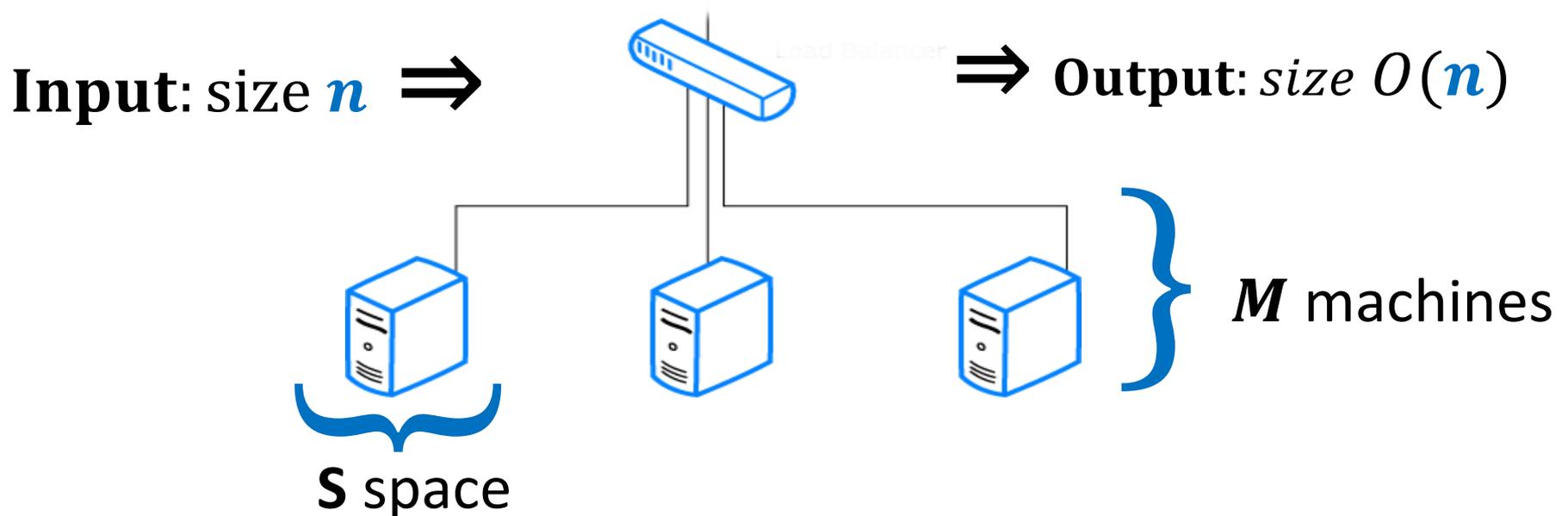


$$\begin{array}{l} \text{CPU time /} \\ \text{Computational} \\ \text{Complexity} \\ \\ = \quad + \\ \\ \text{Network Time /} \\ \text{Information and} \\ \text{Communication} \\ \text{Complexity} \end{array}$$



Computational Model

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

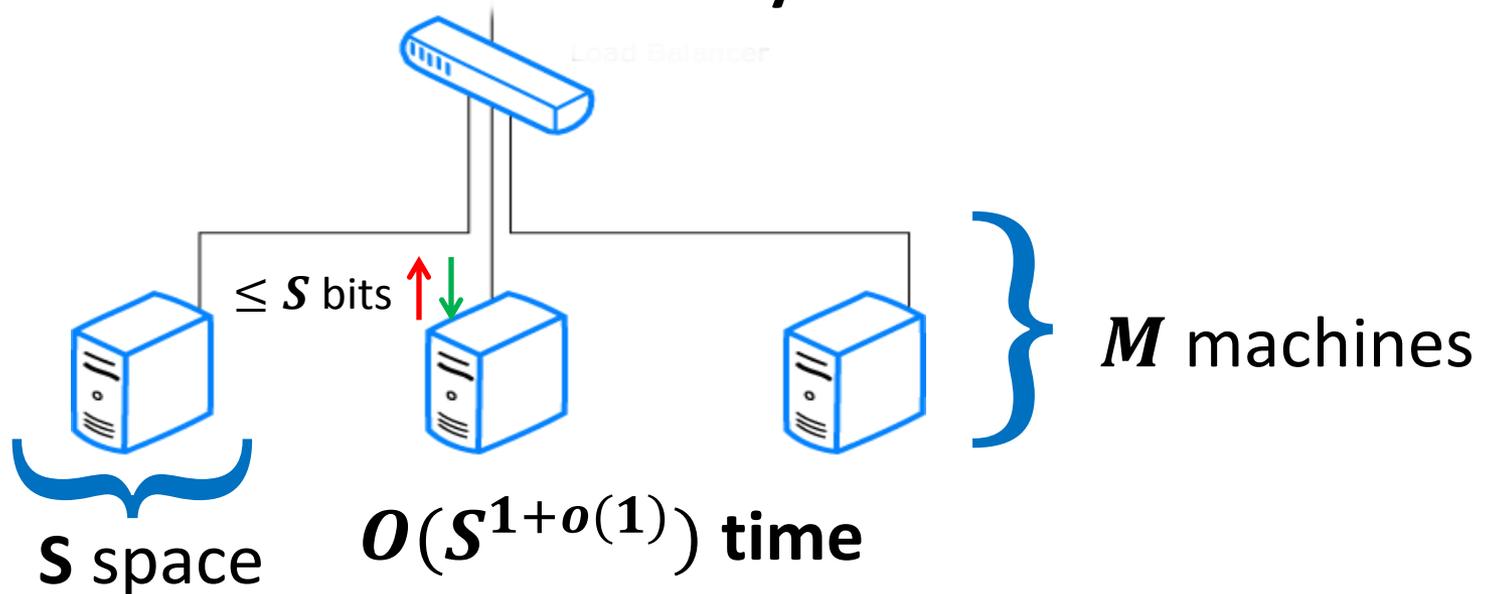


Computational Model

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

Goal: Minimize R .

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

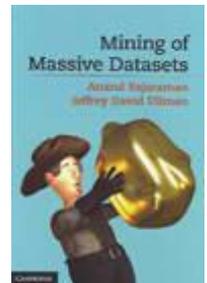


MapReduce-style computations



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

Algorithms for Graphs

- **Dense graphs vs. sparse graphs**

- **Dense: $S \gg |V|$**

- 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: $S \ll |V|$ (or $S \ll$ solution size)**

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



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: \mathbf{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 $S \subseteq V$.

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 $w^* \in \Gamma(L_w)$.
 - If w^* is not empty, mark w **passive** and relabel each vertex with label w by w^* .
- 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 v .
 - If at least two different labels in the CC of v then there is an edge (v', u) such that $\ell(v) = \ell(v')$ and $\ell(v') \neq \ell(u)$.
 - 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.

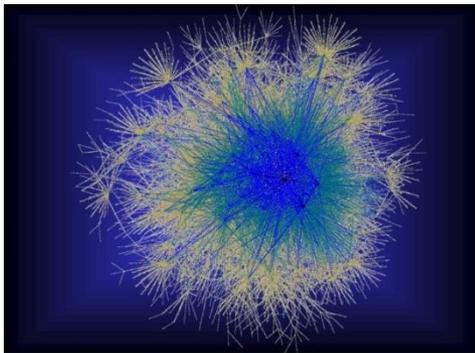
Algorithm for Connectivity: Implementation Details

- 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 $w^* \in \Gamma(L_w)$
 - 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):

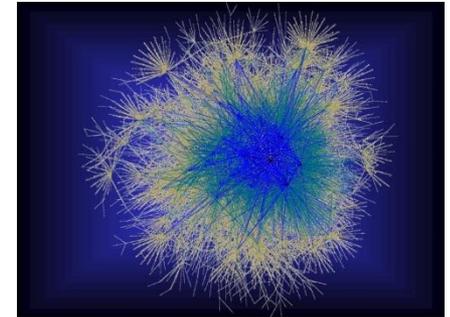
Euclidean distances between n points in \mathbb{R}^d



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

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

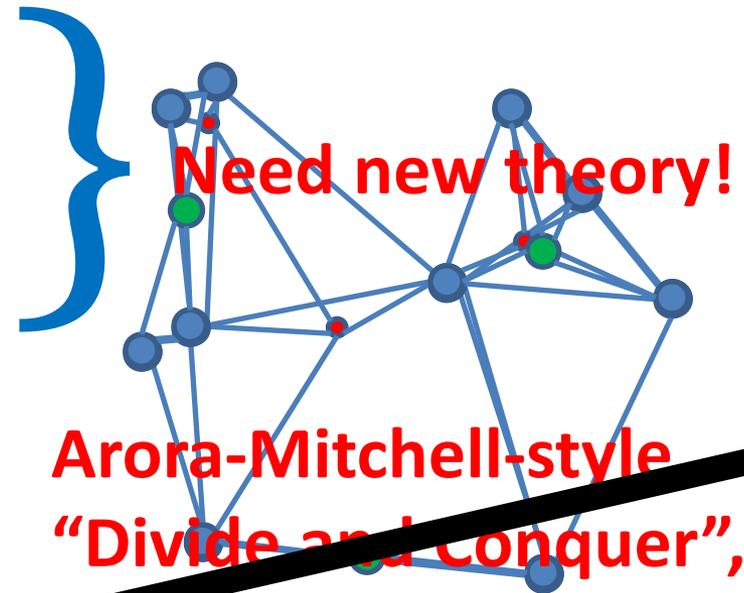
Geometric Graph Problems



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!

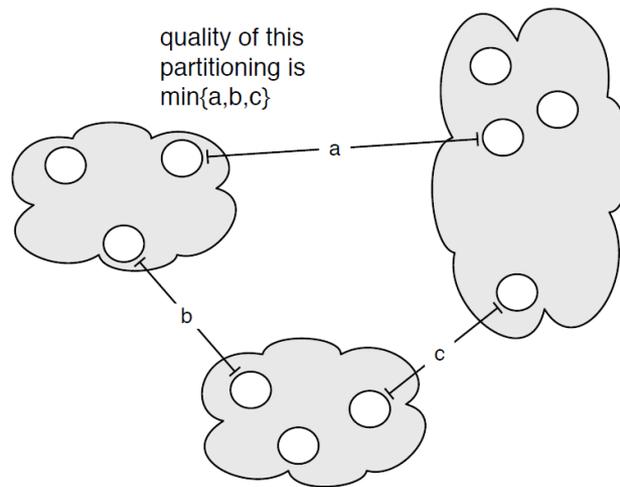
~~NP-hard (“hard”)~~

- ~~• 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

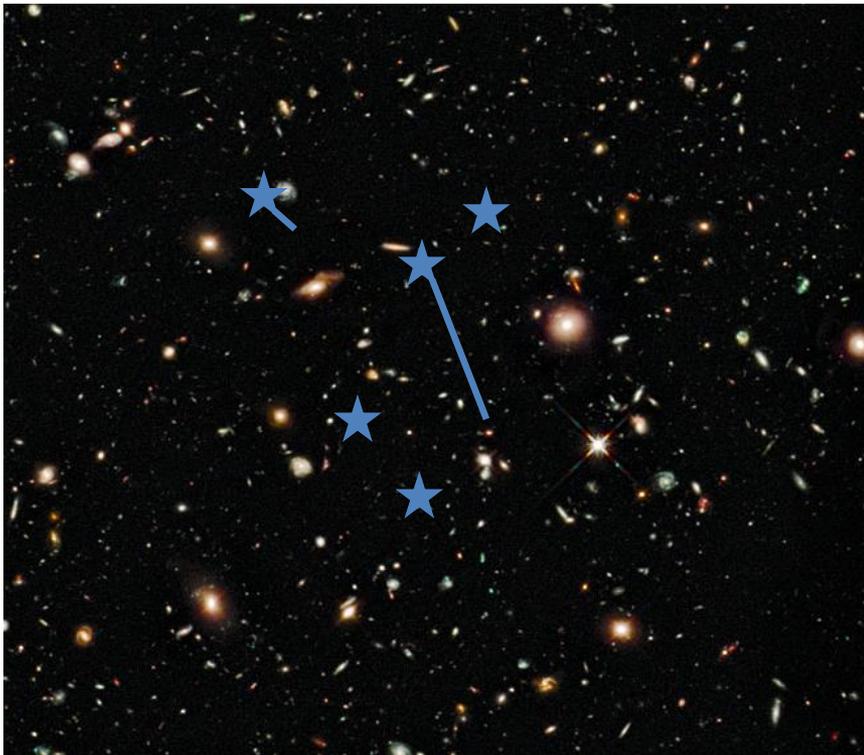
- Blog: <http://grigory.us/blog/mapreduce-clustering/>
- [Zahn'71] **Clustering** via MST (Single-linkage):
k clusters: remove **k - 1** longest edges from MST
- Maximizes **minimum** intercluster distance



[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:** $\mathcal{S} \gg |V|$.
 - **Sparse:** $\mathcal{S} \ll |V|$.

- Our setting:

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

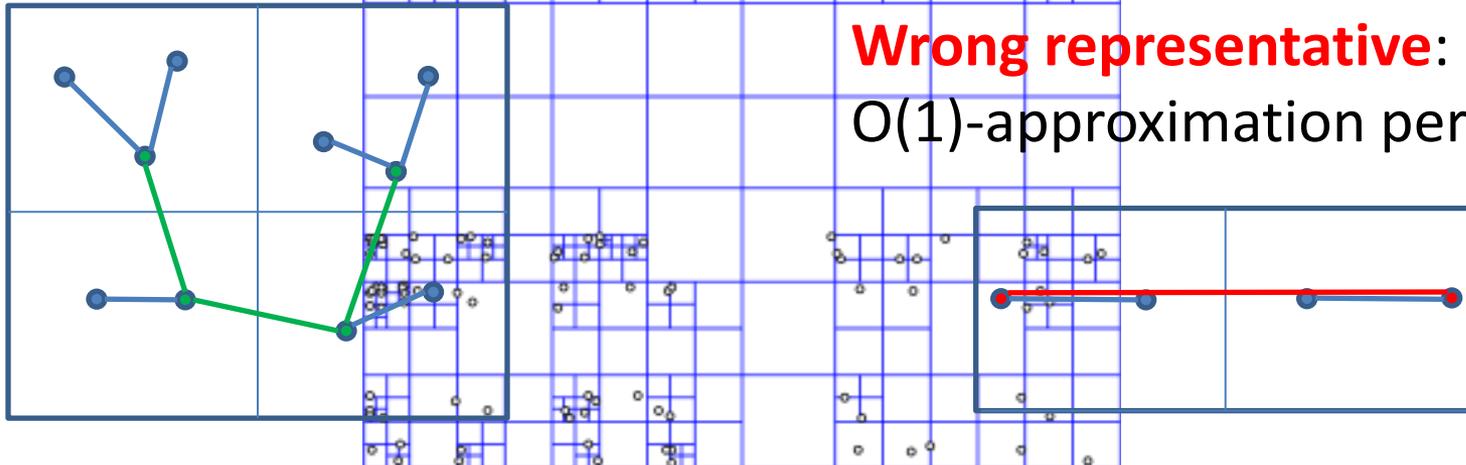
$O(\log n)$ -MST in $R = O(\log n)$ rounds

- 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



ϵL -nets

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

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

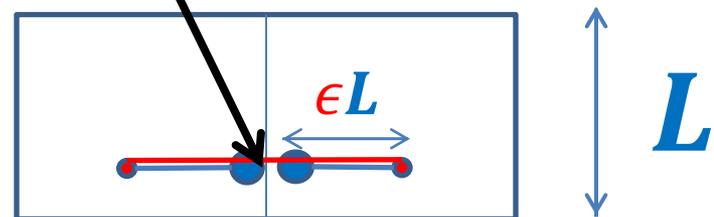
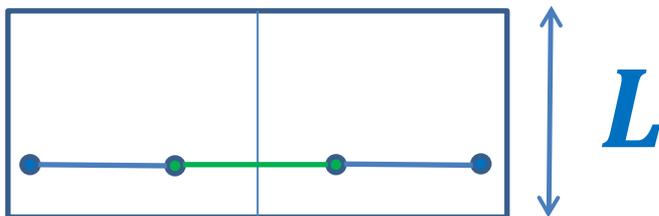
Bottom-up: For each cell in the quadtree

- Compute optimum MSTs in subcells
- Use ϵL -net from each cell on the next level

- **Idea:** Pay only $O(\epsilon L)$ for an **edge** cut by cell with side L

- Randomly shift the quadtree:

$\Pr[\text{cut edge of length } \ell \text{ by } L] \sim \ell/L$ – charge errors $O(1)$ -approximation per level



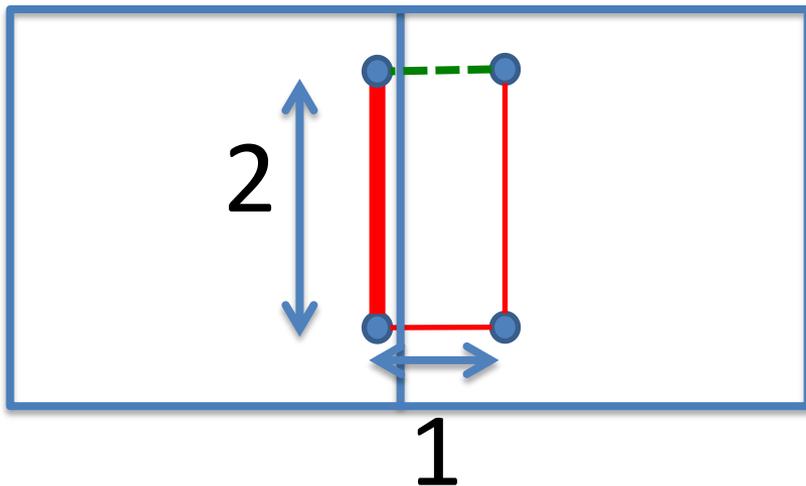
Randomly shifted quadtree

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

Impose a **randomly shifted** quadtree (top cell length 2Δ)

Bottom-up: For each cell in the quadtree

- Compute optimum MSTs in subcells
- Use ϵL -net from each cell on the next level



Pay **5** instead of **4**
Bad Cut
 $\Pr[\text{Bad Cut}] = \Omega(1)$

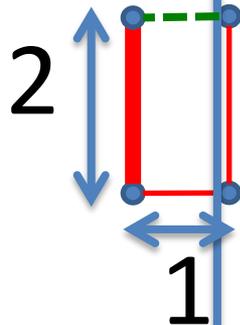
$(1 + \epsilon)$ -MST in $\mathbf{R} = O(\log n)$ rounds

- **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

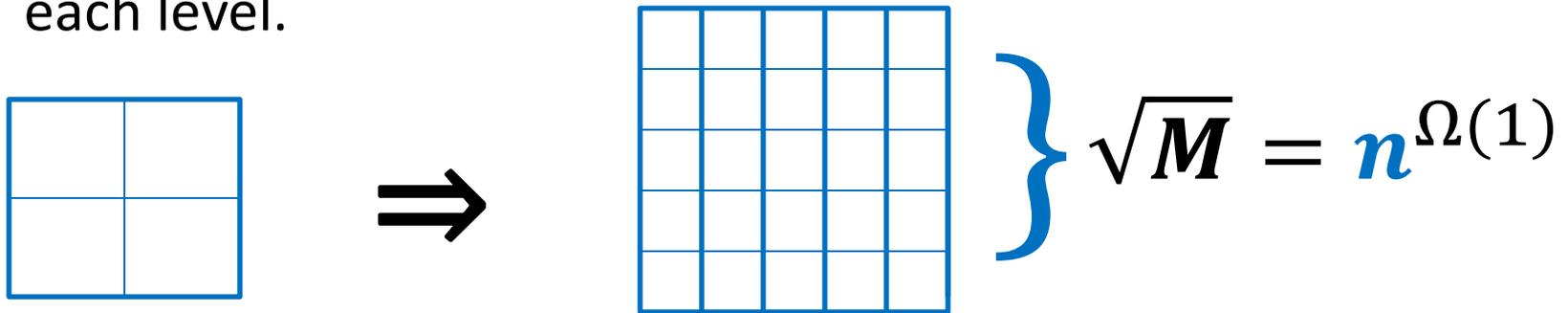


$$L = \Omega\left(\frac{1}{\epsilon}\right)$$

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

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

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



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

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

Proof (sketch):

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

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

- 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 $O\left(\frac{d}{\epsilon}\right)^d$)
- 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 $(1 + \epsilon)$ -approximation in $O_\epsilon(n \log^{O(1)} n)$ 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 τ net points

- $F: \mathbb{R}^{\tau-1} \rightarrow \mathbb{R}$ = the cost of routing fixed amounts of flow through the net points
- Function $F' = F$ + “normalization” is monotone, convex and Lipschitz, $(1 + \epsilon)$ -approximates F
- We can $(1 + \epsilon)$ -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>