CIS 700:
"algorithms for Big Data"

Lecture 10:
Massively Parallel Algorithms

Slides at http://grigory.us/big-data-class.html

Grigory Yaroslavtsev

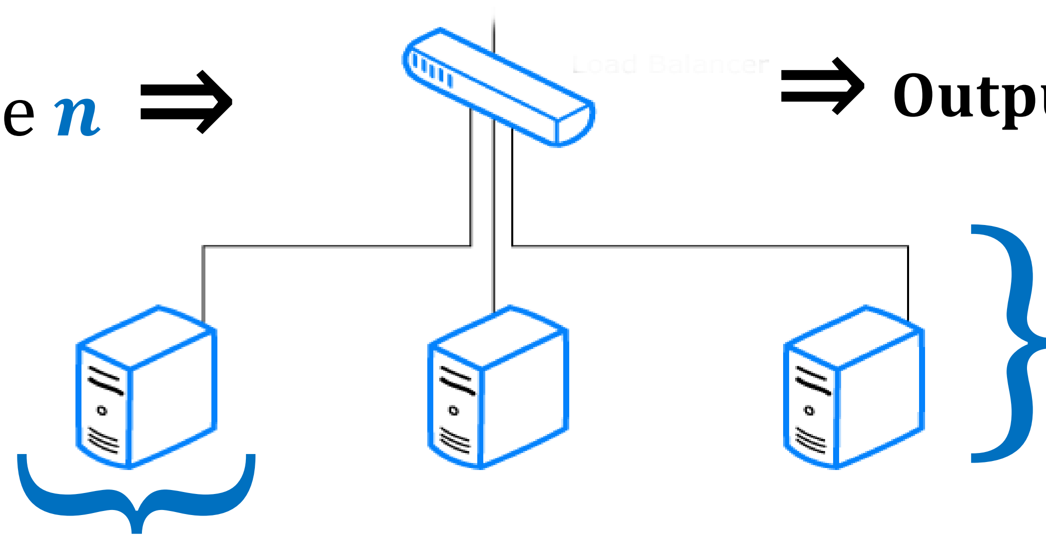
http://grigory.us

Penn
UNIVERSITY of PENNSYLVANIA

Computational Model

- **Input**: size **n**
- $\boldsymbol{M}$ machines, space $\boldsymbol{S}$ on each ($\boldsymbol{S} = \boldsymbol{n}^{1-\epsilon}$, $0 < \epsilon < 1$)
  – 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

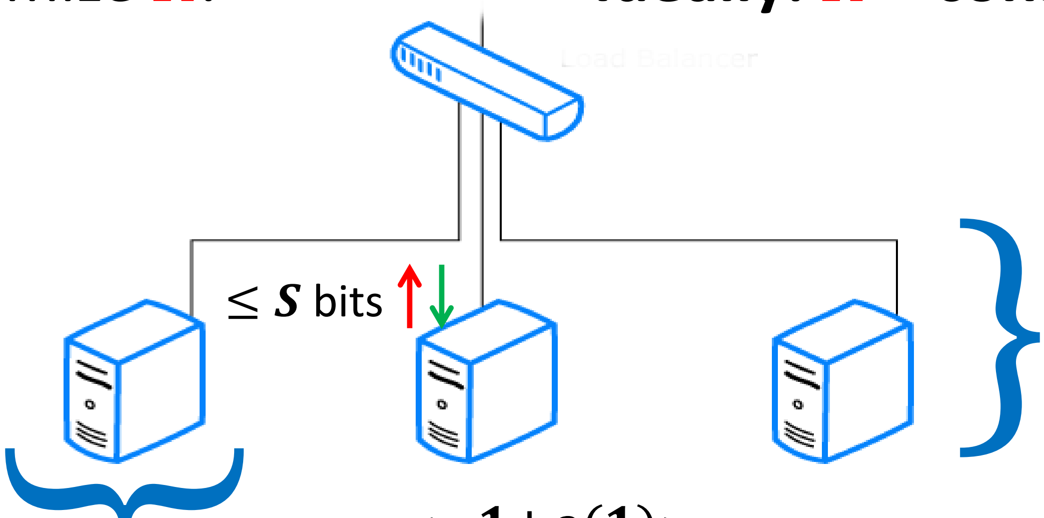
S space

Computational Model

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

Goal: Minimize $\boldsymbol{R}$.

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

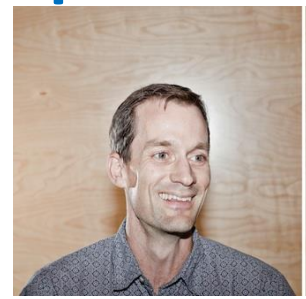


$\boldsymbol{M}$ machines

S space

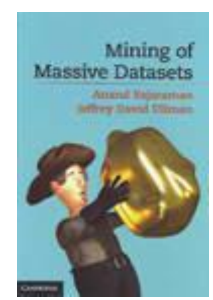
$\boldsymbol{O(S^{1+o(1)})}$ **time**

MapReduce-style computations



What I won’t discuss today

- PRAMs (**shared memory**, multiple processors) (see e.g. [Karloff, Suri, Vassilvitskii‘10])
  - Computing XOR requires $\widetilde{\Omega}(\log n)$ 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**

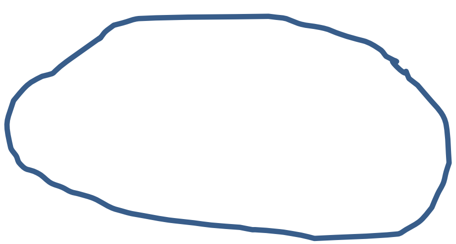
Sorting: Terasort

- Sorting $\boldsymbol{n}$ keys on $\boldsymbol{M} = \boldsymbol{O}(\boldsymbol{n}^{\boldsymbol{1}-\boldsymbol{\alpha}})$ 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 $\boldsymbol{M} * \boldsymbol{s} \leq \boldsymbol{S} = \boldsymbol{n}^{\boldsymbol{\alpha}}$ samples are sorted locally
  - Blocks are computed based on the samples
- By Chernoff bound $\mathbf{M} * \boldsymbol{s} = O\left(\frac{\boldsymbol{log\, n}}{\boldsymbol{\epsilon^2}}\right)$ samples suffice to compute al block sizes with $\pm \epsilon \boldsymbol{n}$ error
- Take $\epsilon = \frac{\boldsymbol{n}^{\alpha-1}}{2}$: error $\mathrm{O}(\boldsymbol{S})$; $\mathbf{M} * \boldsymbol{s} = \widetilde{O}(\boldsymbol{n}^{2-\boldsymbol{2\alpha}}) = \boldsymbol{O}(\boldsymbol{M}^{\boldsymbol{2}}) \leq \boldsymbol{O}(\boldsymbol{n}^{\boldsymbol{\alpha}})$ for $\alpha \geq 2/3$

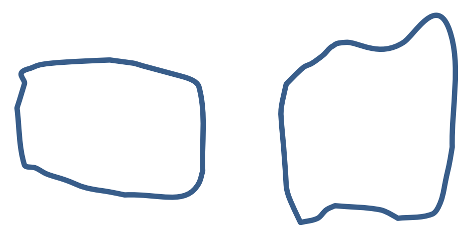
Algorithms for Graphs

- **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 $\mathrm{o}(\log n)$ rounds?)



vs.



Algorithm for Connectivity

- 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 $i = 1,2, \ldots, O(\log N)$ 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

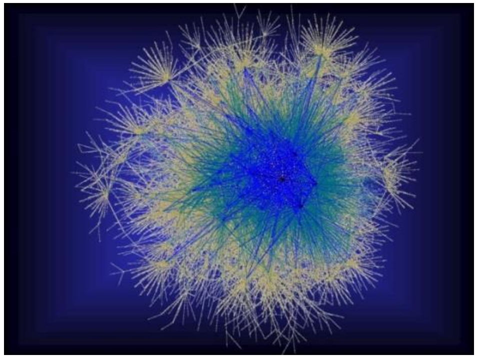
- Distributed data structure of size $O(|V|)$ to maintain labels, ids, leader/non-leader status, etc.
  - O(1) rounds per stage to update the data structure
- Edges stored locally with all auxiliary info
  - Between stages: use distributed data structure to update local info on edges
- For every **active** **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]

Applications

- Using same reductions as in streaming:
  - Bipartiteness
  - k-connectivity
  - Cut-sparsification

Approximating Geometric Problems in Parallel Models

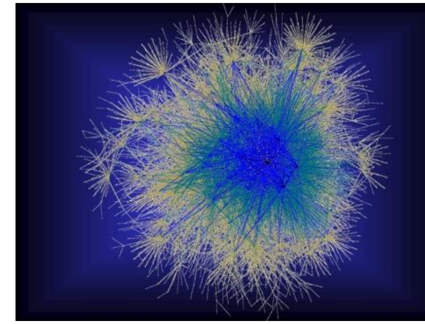
Geometric graph (implicit):
Euclidean distances between **n** points in $\mathbb{R}^{d}$

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

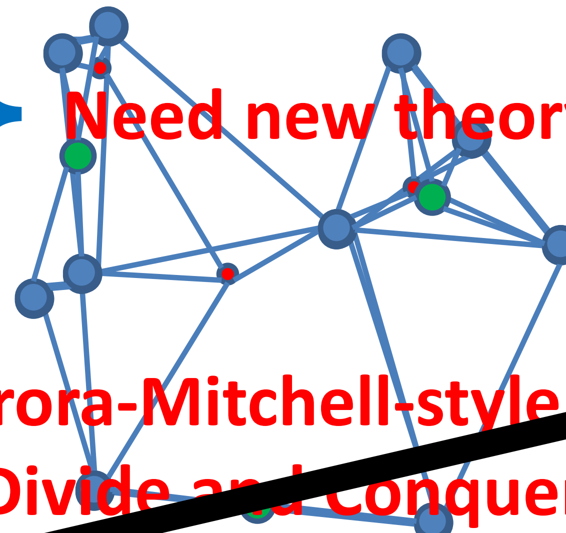


Combinatorial problems on graphs in $\mathbb{R}^d$

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.)~~

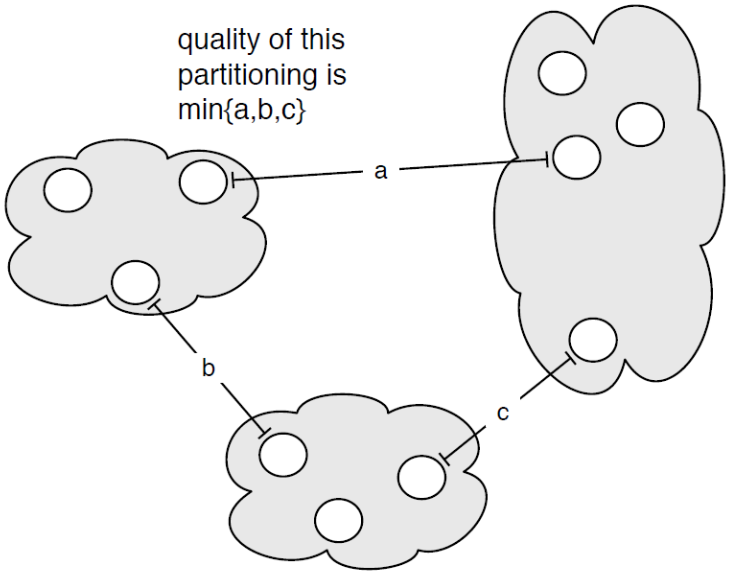
~~**Arora-Mitchell-style "Divide and Conquer", easy to implement in Massively Parallel Computational Models, but bad running time**~~

MST: Single Linkage Clustering

- [Zahn'71] **Clustering** via MST (Single-linkage):

k clusters: remove $\boldsymbol{k-1}$ longest edges from MST

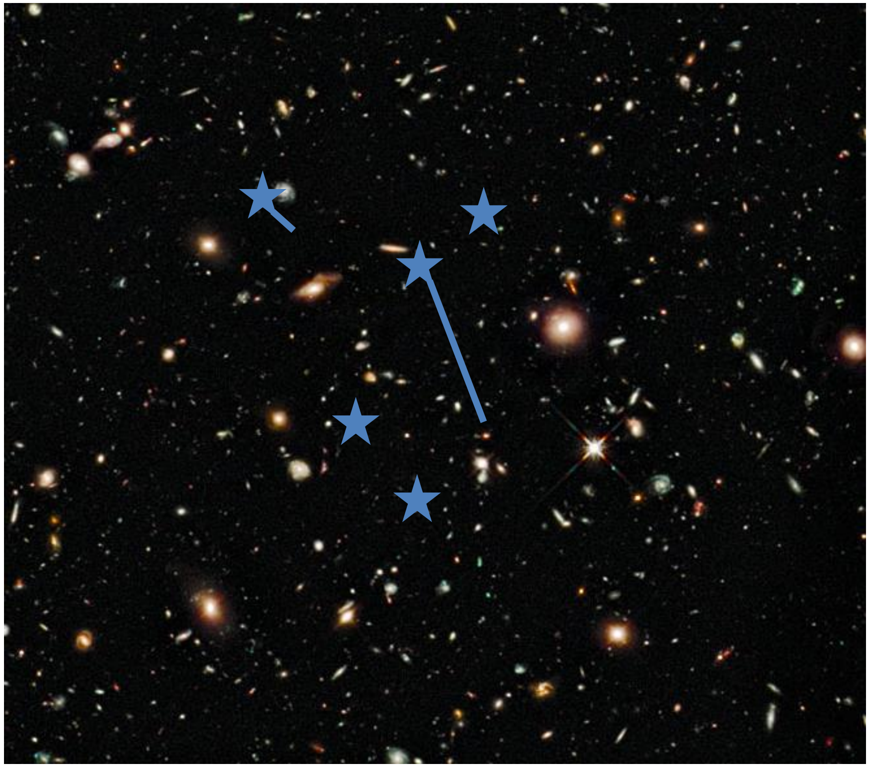
- Maximizes **minimum** intercluster distance



[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: $\boldsymbol{S} \ll |V|$.

- <u>**Our setting:**</u>
  - 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 ($\boldsymbol{S} = \boldsymbol{n}^{\boldsymbol{\Omega(1)}}$)

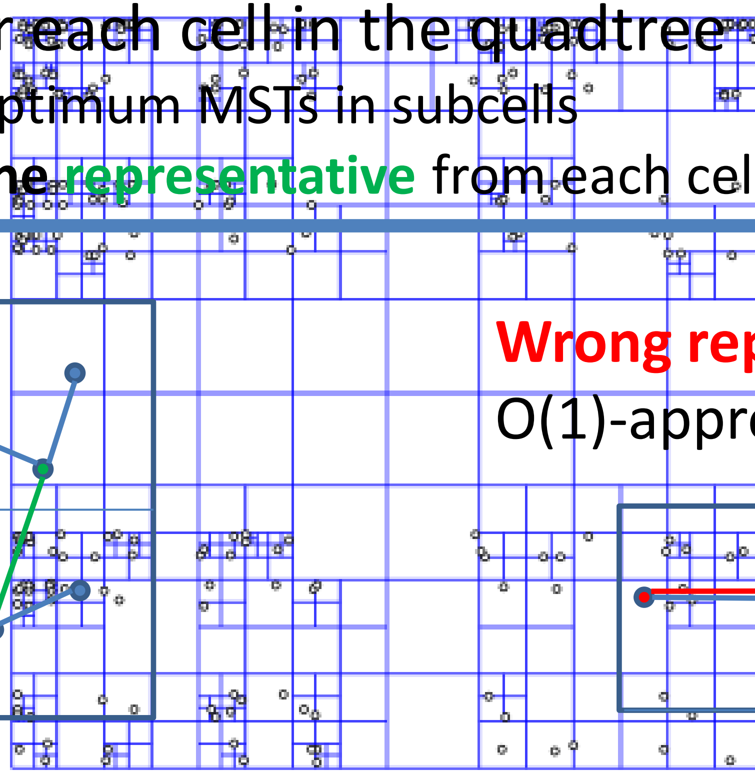
$O(\log n)$-MST in $R = O(\log n)$ rounds

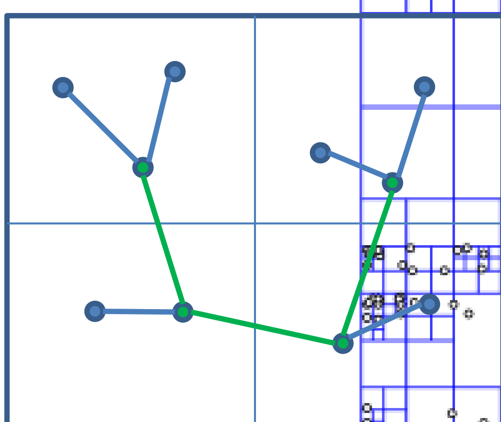
- Assume points have integer coordinates $[0, \ldots, \Delta]$, where $\Delta = O(n^2)$.

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

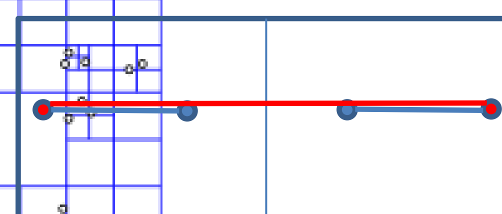
Bottom-up: For each cell in the quadtree

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





Wrong representative:
O(1)-approximation per level



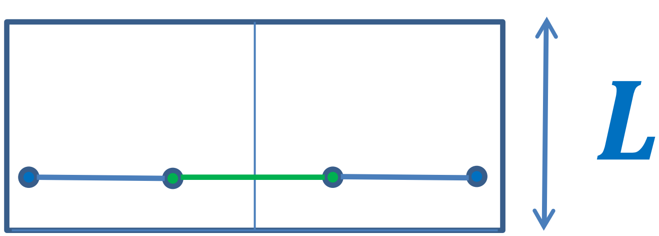
ϵL-nets

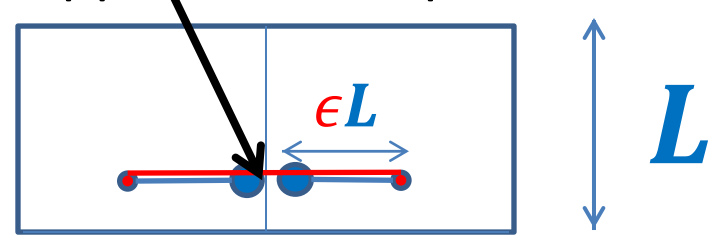
- ϵL-net for a cell C with side length L:
  Collection **S** of vertices in C, every vertex is at distance <= ϵL from some vertex in **S**. (Fact: Can efficiently compute ϵ-net of size $O\left(\frac{1}{\epsilon^2}\right)$)

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

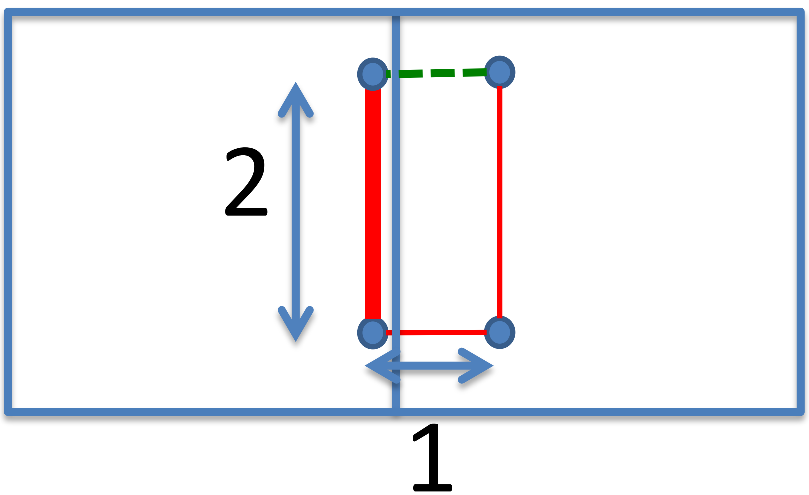
- Top cell shifted by a random vector in $[0, \boldsymbol{L}]^2$

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 ϵL-net from each cell on the next level



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

Bad Cut

Pr[Bad Cut] = Ω(1)

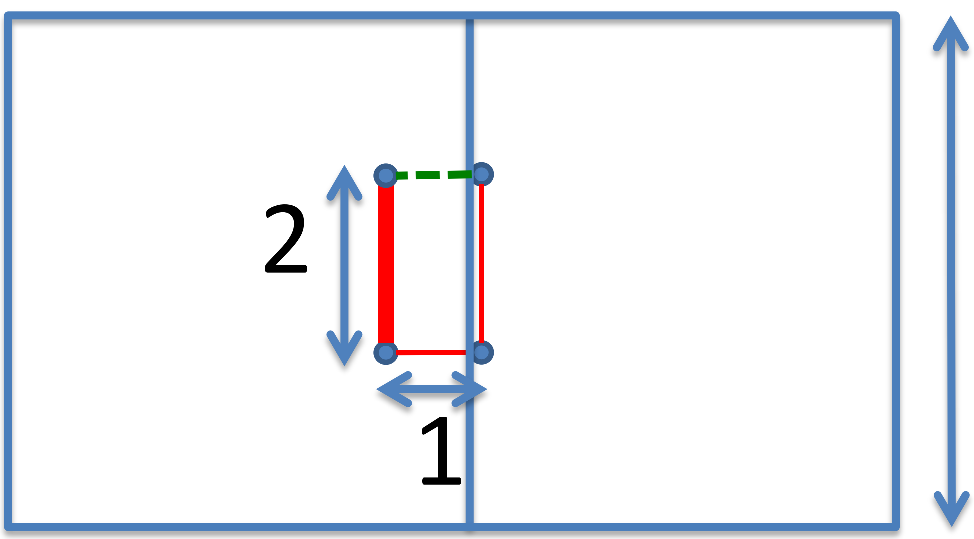
$(1+\epsilon)$-MST in $\mathbf{R} = O(\log\ n)$ rounds

- **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

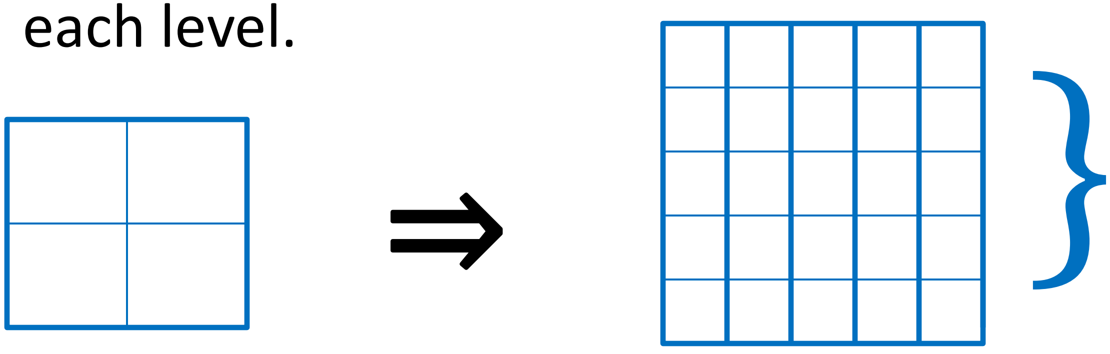


$$L = \Omega\left(\frac{1}{\epsilon}\right)$$

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

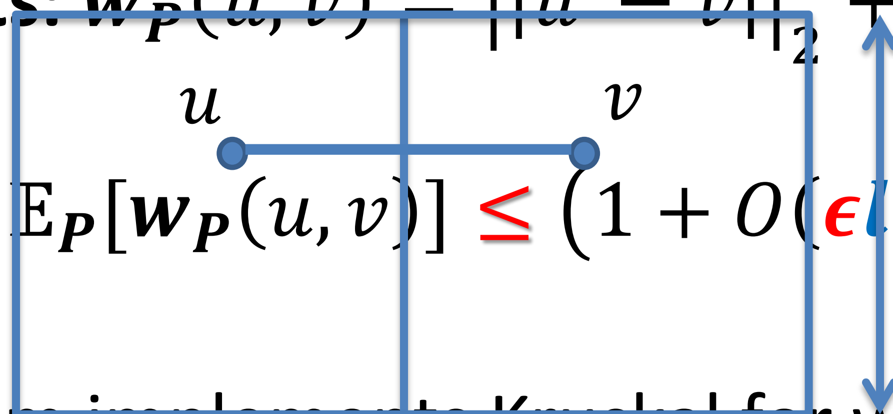
$(1+\epsilon)$-MST in $\mathbf{R} = O(1)$ rounds

Theorem: Let $\boldsymbol{l} =$ # levels in a random tree $\boldsymbol{P}$

$$\mathbb{E}_{\boldsymbol{P}}[\mathbf{ALG}] \le \big(1 + O(\boldsymbol{\epsilon l d})\big)\mathbf{OPT}$$

Proof (sketch):

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



$$||u-v||_2 \le \mathbb{E}_{\boldsymbol{P}}[\boldsymbol{w_P}(u,v)] \le \big(1 + O(\boldsymbol{\epsilon l d})\big)||u-v||_2$$

$\boldsymbol{\Delta_P}(u,v)$

- Our algorithm implements Kruskal for weights $\boldsymbol{w_P}$

“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 $O_\epsilon\left(\boldsymbol{n}\log^{O(1)}\boldsymbol{n}\right)$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

- $F\colon \mathbb{R}^{\boldsymbol{\tau}-1} \to \mathbb{R}$ = 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

Open problems:

- Exetension to high dimensions?
  - Probably no, reduce from connectivity => conditional lower bound : $\Omega(\log n)$ rounds for MST in ℓ_∞^n
  - The difficult setting is $d = \Theta(\log \boldsymbol{n})$ (can do JL)
- Streaming alg for **EMD** and **Transporation Cost**?
- Our work:
  - First near-linear time algorithm for **Transportation Cost**
  - Is it possible to reconstruct the solution itself?

Class Project

- Survey of 3-5 research papers
  - Closely related to the topics of the class
    - Streaming
    - MapReduce
    - Convex Optmization
    - Sublinear Time Algorithms
  - Office hours if you need suggestions
  - Individual or groups of 2 people
  - **Deadline**: December 18, 2015 at 23:59 EST
- Submission by e-mail grigory@grigory.us
  - Submission Email Title: Project + Space + "Your Name"
  - One submission per group listing participants
  - Submission format
    - PDF from LaTeX (best)
    - PDF

Example: Gradient Descent in TensorFlow

- Gradient Descent (covered in class)
- Adagrad: http://www.magicbroom.info/Papers/DuchiHaSi10.pdf
- Momentum (stochastic gradient descent + tweaks): http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf
- Adam (Adaptive + momentum): http://arxiv.org/pdf/1412.6980.pdf
- FTRL: http://jmlr.org/proceedings/papers/v15/mcmahan11b/mcmahan11b.pdf
- RMSProp: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf