# Parallel Algorithms for Graphs on a Very Large Number of Nodes

**Krzysztof Onak**

IBM T.J. Watson Research Center

# Outline

1. Model of Computation
2. Sample Algorithms and Their Limitations
3. Efficiently Estimating MST Weight
4. Computing MST in Geometric Setting

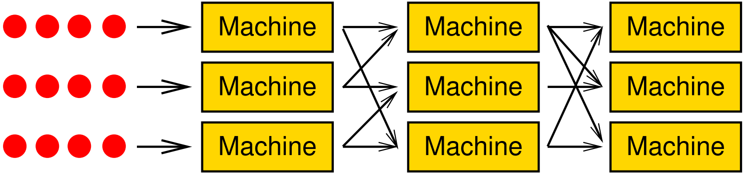
# Outline

1. Model of Computation
2. Sample Algorithms and Their Limitations
3. Efficiently Estimating MST Weight
4. Computing MST in Geometric Setting

# Model: Massive Parallel Computation

[Karloff, Suri, Vassilvitskii 2010; Beame, Koutris, Suciu 2013; . . . ]

$n$ items on input $\qquad$ $m$ machines

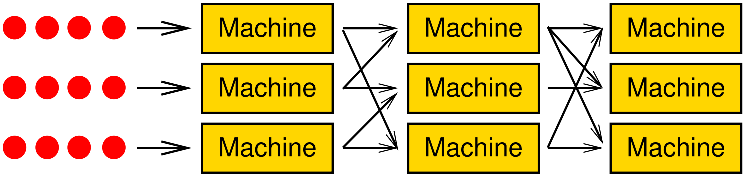


# Model: Massive Parallel Computation

[Karloff, Suri, Vassilvitskii 2010; Beame, Koutris, Suciu 2013; ...]

$n$ items on input $m$ machines

space per machine: $s = \frac{n}{m} \cdot \text{small-factor}$

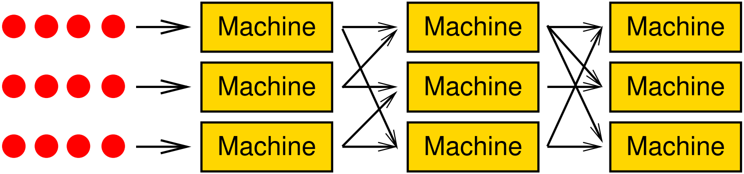


# Model: Massive Parallel Computation

[Karloff, Suri, Vassilvitskii 2010; Beame, Koutris, Suciu 2013; . . . ]

$n$ items on input $m$ machines

space per machine: $s = \frac{n}{m} \cdot$ small-factor



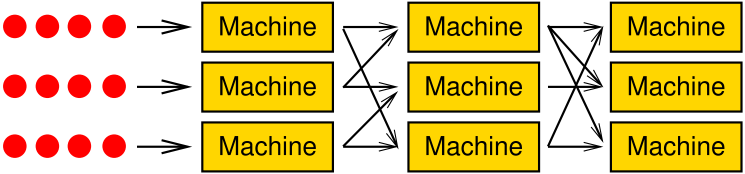

- Initially: each machine receives $n/m$ items

# Model: Massive Parallel Computation

[Karloff, Suri, Vassilvitskii 2010; Beame, Koutris, Suciu 2013; ...]

$n$ items on input $\qquad$ $m$ machines

space per machine: $s = \frac{n}{m} \cdot \text{small-factor}$




- Initially: each machine receives $n/m$ items
- Single round:
  1. Each machine performs computation
  2. Each machine sends and receives at most $O(s)$ data

# Resources

$n$ items on input $\qquad$ $m$ machines

space per machine: $s = \frac{n}{m} \cdot \text{small-factor}$

# Resources

$n$ items on input $\qquad$ $m$ machines

space per machine: $s = \dfrac{n}{m} \cdot$ small-factor

- Popular assumption:

$$m = O(n^{\alpha}) \text{ for } \alpha \in (0,1) \quad \Longrightarrow \quad s = n^{\Omega(1)}$$

# Resources

$n$ items on input $\qquad$ $m$ machines

space per machine: $s = \frac{n}{m} \cdot \text{small-factor}$

- Popular assumption:

$$m = O(n^\alpha) \text{ for } \alpha \in (0,1) \quad \Longrightarrow \quad s = n^{\Omega(1)}$$

- Likely to happen: $\quad s \gg m$

# Resources

$n$ items on input    $m$ machines

space per machine: $s = \frac{n}{m} \cdot \text{small-factor}$

- Popular assumption:

$$m = O(n^{\alpha}) \text{ for } \alpha \in (0, 1) \quad \Longrightarrow \quad s = n^{\Omega(1)}$$

- Likely to happen: $s \gg m$

Goals:

- Minimize the number of rounds

# Resources

$n$ items on input $\qquad$ $m$ machines

space per machine: $s = \frac{n}{m} \cdot \text{small-factor}$

- Popular assumption:

$$m = O(n^{\alpha}) \text{ for } \alpha \in (0, 1) \quad \Longrightarrow \quad s = n^{\Omega(1)}$$

- Likely to happen: $\quad s \gg m$

Goals:

- Minimize the number of rounds
- Optimize running time

# Resources

$n$ items on input $m$ machines

space per machine: $s = \frac{n}{m} \cdot$ small-factor

- Popular assumption:

$$m = O(n^{\alpha}) \text{ for } \alpha \in (0, 1) \implies s = n^{\Omega(1)}$$

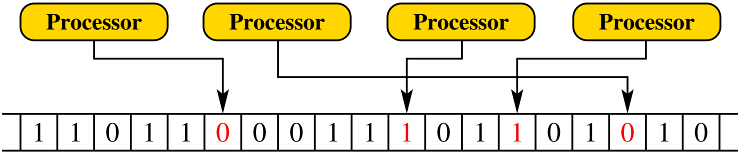
- Likely to happen: $s \gg m$

Goals:

- Minimize the number of rounds
- Optimize running time
- Use amount of memory as close to linear as possible

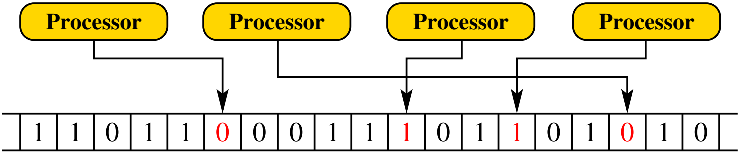
# Comparison to PRAM

- PRAM: classic parallel model
  - *m* processors
  - processors access common memory



# Comparison to PRAM

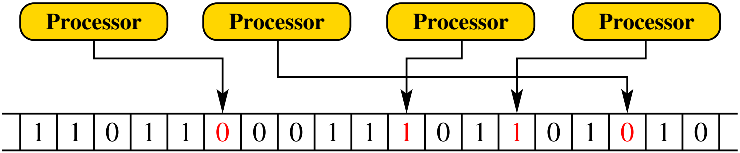
- PRAM: classic parallel model
  - $m$ processors
  - processors access common memory
- Many problems require $\tilde{\Omega}(\log n)$ rounds in PRAM



# Comparison to PRAM

- PRAM: classic parallel model
  - $m$ processors
  - processors access common memory
- Many problems require $\tilde{\Omega}(\log n)$ rounds in PRAM

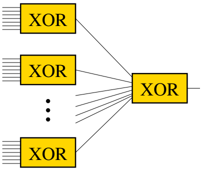
  Example: computing XOR of $n$ bits requires $\Omega(\log n / \log\log n)$ time in strongest PRAM model [Beame, Håstad 1989]



# Comparison to PRAM

- PRAM: classic parallel model
  - $m$ processors
  - processors access common memory
- Many problems require $\tilde{\Omega}(\log n)$ rounds in PRAM

  Example: computing XOR of $n$ bits requires $\Omega(\log n / \log\log n)$ time in strongest PRAM model [Beame, Håstad 1989]
- Our model: $O(\log_s n)$ rounds for XOR



# Comparison to PRAM

- PRAM: classic parallel model
  - $m$ processors
  - processors access common memory
- Many problems require $\tilde{\Omega}(\log n)$ rounds in PRAM

  Example: computing XOR of $n$ bits requires $\Omega(\log n / \log\log n)$ time in strongest PRAM model [Beame, Håstad 1989]
- Our model: $O(\log_s n)$ rounds for XOR

  If $s = n^{\Omega(1)}$, number of rounds is constant

# Comparison to PRAM

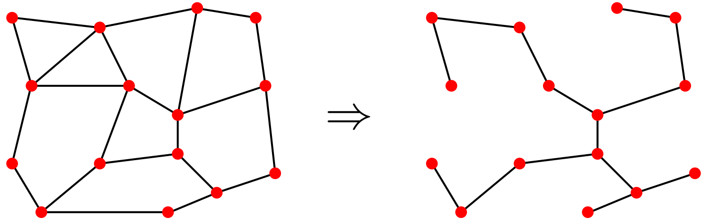
- PRAM: classic parallel model
  - $m$ processors
  - processors access common memory
- Many problems require $\tilde{\Omega}(\log n)$ rounds in PRAM

  Example: computing XOR of $n$ bits requires $\Omega(\log n / \log\log n)$ time in strongest PRAM model [Beame, Håstad 1989]
- Our model: $O(\log_s n)$ rounds for XOR

  If $s = n^{\Omega(1)}$, number of rounds is constant
- Our goal: constant number of communication rounds

# Outline

# Main Subject of Study: Minimum Spanning Tree



Select the subset of edges of minimum weight

that connects all vertices

# Filtering Technique

[Karloff, Suri, Vassilvitskii 2010]

[Lattanzi, Moseley, Suri, Vassilvitskii 2011]

# Filtering Technique

[Karloff, Suri, Vassilvitskii 2010]
[Lattanzi, Moseley, Suri, Vassilvitskii 2011]

- Input: weighted edges of a graph on $N$ vertices

# Filtering Technique

[Karloff, Suri, Vassilvitskii 2010]
[Lattanzi, Moseley, Suri, Vassilvitskii 2011]

- Input: weighted edges of a graph on $N$ vertices
- Main idea:

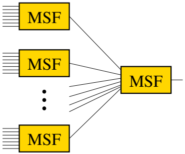
1. Find minimum spanning forest for subset of edges
2. Remove edges not in the forest

# Filtering Technique
[Karloff, Suri, Vassilvitskii 2010]
[Lattanzi, Moseley, Suri, Vassilvitskii 2011]

- Input: weighted edges of a graph on $N$ vertices
- Main idea:

1. Find minimum spanning forest for subset of edges
2. Remove edges not in the forest

- Algorithm: repeat the process until problem solved



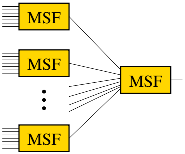
# Filtering Technique

[Karloff, Suri, Vassilvitskii 2010]
[Lattanzi, Moseley, Suri, Vassilvitskii 2011]

- Input: weighted edges of a graph on $N$ vertices
- Main idea:

  1. Find minimum spanning forest for subset of edges
  2. Remove edges not in the forest

- Algorithm: repeat the process until problem solved




- Caveat: $\geq N$ space per machine required

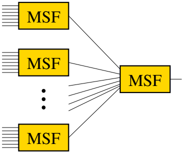
# Filtering Technique

[Karloff, Suri, Vassilvitskii 2010]
[Lattanzi, Moseley, Suri, Vassilvitskii 2011]

- Input: weighted edges of a graph on $N$ vertices
- Main idea:

  1. Find minimum spanning forest for subset of edges
  2. Remove edges not in the forest

- Algorithm: repeat the process until problem solved




- Caveat: $\geq N$ space per machine required
- Complexity: $s = N^{1+\Omega(1)} \quad \Rightarrow \quad O(1)$ rounds

# $N^{1-\Omega(1)}$ Space in $O(1)$ Rounds?

# $N^{1-\Omega(1)}$ Space in $O(1)$ Rounds?

- Unlikely to be possible in general

# $N^{1-\Omega(1)}$ Space in $O(1)$ Rounds?

- Unlikely to be possible in general
- Can reduce from Sparse Connectivity:

Do edges span a connected graph?

# $N^{1-\Omega(1)}$ Space in $O(1)$ Rounds?

- Unlikely to be possible in general
- Can reduce from Sparse Connectivity:

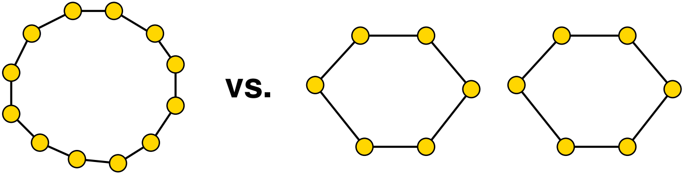
  Do edges span a connected graph?
- Conjecture: superconstant number of rounds with $N^{1-\Omega(1)}$ memory

# $N^{1-\Omega(1)}$ Space in $O(1)$ Rounds?

- Unlikely to be possible in general
- Can reduce from Sparse Connectivity:

  Do edges span a connected graph?

- Conjecture: superconstant number of rounds with $N^{1-\Omega(1)}$ memory
- Is this instance hard?



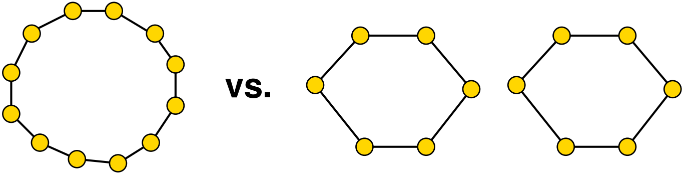
# $N^{1-\Omega(1)}$ Space in $O(1)$ Rounds?

- Unlikely to be possible in general
- Can reduce from Sparse Connectivity:

  Do edges span a connected graph?

- Conjecture: superconstant number of rounds
  with $N^{1-\Omega(1)}$ memory
- Is this instance hard?

(solvable in $O(\log N)$ rounds)



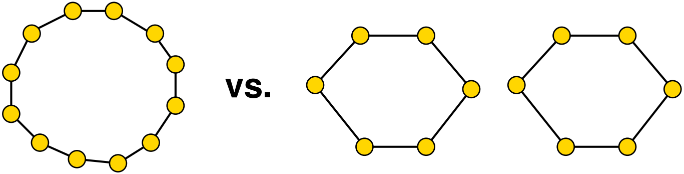
# $N^{1-\Omega(1)}$ Space in $O(1)$ Rounds?

- Unlikely to be possible in general
- Can reduce from Sparse Connectivity:

  Do edges span a connected graph?

- Conjecture: superconstant number of rounds with $N^{1-\Omega(1)}$ memory
- Is this instance hard?

(solvable in $O(\log N)$ rounds)



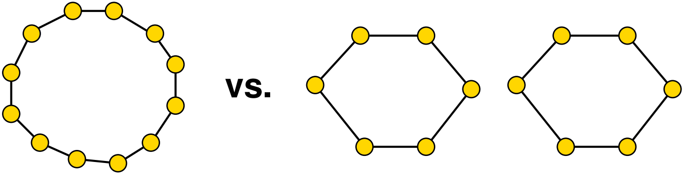

- Reduction: connect select vertex to all vertices with heavy edges

# $N^{1-\Omega(1)}$ Space in $O(1)$ Rounds?

- Unlikely to be possible in general
- Can reduce from Sparse Connectivity:

  Do edges span a connected graph?
- Conjecture: superconstant number of rounds
  with $N^{1-\Omega(1)}$ memory
- Is this instance hard?

(solvable in $O(\log N)$ rounds)




- Reduction: connect select vertex to all vertices
  with heavy edges
- This talk: algorithms with $O(N^{\epsilon})$ space per machine

# Outline

1. Model of Computation
2. Sample Algorithms and Their Limitations
3. Efficiently Estimating MST Weight
4. Computing MST in Geometric Setting

# Result

[Łącki, Mądry, Mitrović, O., Sankowski]

- Input: $M$ edges, weights in $\{1, 2, \ldots, W\}$
  (#nodes $N \leq$ #edges $M$)

# Result

[Łącki, Mądry, Mitrović, O., Sankowski]

- Input: $M$ edges, weights in $\{1, 2, \ldots, W\}$
  (#nodes $N \leq$ #edges $M$)

- Algorithm:
  - Computes $(1 + \epsilon)$-approximation to MST weight

# Result

[Łącki, Mądry, Mitrović, O., Sankowski]

- Input: $M$ edges, weights in $\{1, 2, \ldots, W\}$
  (#nodes $N \leq$ #edges $M$)

- Algorithm:
  - Computes $(1+\epsilon)$-approximation to MST weight
  - Space per machine:
    $$O\left(\frac{M}{m} + \frac{N}{m} \cdot \left(\frac{W}{\epsilon}\right)^2\right) \text{ for } M/m = M^{\Omega(1)}$$

# Result

[Łącki, Mądry, Mitrović, O., Sankowski]

- Input: $M$ edges, weights in $\{1, 2, \ldots, W\}$

  (#nodes $N \leq$ #edges $M$)

- Algorithm:
  - Computes $(1 + \epsilon)$-approximation to MST weight
  - Space per machine:

    $$O\left(\frac{M}{m} + \frac{N}{m} \cdot \left(\frac{W}{\epsilon}\right)^2\right) \text{ for } M/m = M^{\Omega(1)}$$

  - Number of rounds: $O(\log(W/\epsilon))$

# Result

[Łącki, Mądry, Mitrović, O., Sankowski]

- Input: $M$ edges, weights in $\{1, 2, \ldots, W\}$
  (#nodes $N \leq$ #edges $M$)

- Algorithm:
  - Computes $(1+\epsilon)$-approximation to MST weight
  - Space per machine:
    $$O\left(\frac{M}{m} + \frac{N}{m} \cdot \left(\frac{W}{\epsilon}\right)^2\right) \text{ for } M/m = M^{\Omega(1)}$$
  - Number of rounds: $O(\log(W/\epsilon))$

- Note: No dependence on $W$ would disprove
  Sparse Connectivity Conjecture

# Approach

Use techniques of Chazelle, Rubinfeld, Trevisan (2005)

# Approach

Use techniques of Chazelle, Rubinfeld, Trevisan (2005):

- $G_i$ = graph restricted to edges of weight $< i$

# Approach

Use techniques of Chazelle, Rubinfeld, Trevisan (2005):

- $G_i$ = graph restricted to edges of weight $< i$
- $T_i$ = #connected components in $G_i$

# Approach

Use techniques of Chazelle, Rubinfeld, Trevisan (2005):

- $G_i$ = graph restricted to edges of weight $< i$
- $T_i$ = #connected components in $G_i$
- Number of edges of weight $\geq i$ in MST = $T_i - 1$

# Approach

Use techniques of Chazelle, Rubinfeld, Trevisan (2005):

- $G_i$ = graph restricted to edges of weight $< i$
- $T_i$ = #connected components in $G_i$
- Number of edges of weight $\geq i$ in MST = $T_i - 1$

$$\Rightarrow \quad \text{weight(MST)} = \sum_{i=1}^{W} (T_i - 1)$$

# Approach

Use techniques of Chazelle, Rubinfeld, Trevisan (2005):

- $G_i$ = graph restricted to edges of weight $< i$
- $T_i$ = #connected components in $G_i$
- Number of edges of weight $\geq i$ in MST = $T_i - 1$

$$\Rightarrow \quad \text{weight(MST)} = \sum_{i=1}^{W} (T_i - 1)$$

- $C_i(v)$ = size of the component of $v$ in $G_i$

$$T_i = \sum_v 1/C_i(v)$$

# Approach

Use techniques of Chazelle, Rubinfeld, Trevisan (2005):

- $G_i$ = graph restricted to edges of weight $< i$
- $T_i$ = #connected components in $G_i$
- Number of edges of weight $\geq i$ in MST = $T_i - 1$

$$\Rightarrow \quad \text{weight(MST)} = \sum_{i=1}^{W} (T_i - 1)$$

- $C_i(v)$ = size of the component of $v$ in $G_i$

$$T_i = \sum_{v} 1/C_i(v)$$

- Good approximation:
  - Compute sizes of small components
  - Replace $1/C_i(v)$ with 0 if $C_i(v) \geq W/\epsilon$

# Implementation

- Reachability sets $R_v$ for each node $v$:
  - Set of $W/\epsilon$ nodes accessible via cheapest edges

# Implementation

- Reachability sets $R_v$ for each node $v$:
  - Set of $W/\epsilon$ nodes accessible via cheapest edges
  - Initially: collect cheapest incident edges

# Implementation

- Reachability sets $R_v$ for each node $v$:
  - Set of $W/\epsilon$ nodes accessible via cheapest edges
  - Initially: collect cheapest incident edges
  - Repeat $O(\log(W/\epsilon))$ times:
    Ask nodes $u$ on $R_v$ for their $R_u$ and update

# Implementation

- Reachability sets $R_v$ for each node $v$:
    - Set of $W/\epsilon$ nodes accessible via cheapest edges
    - Initially: collect cheapest incident edges
    - Repeat $O(\log(W/\epsilon))$ times:
      Ask nodes $u$ on $R_v$ for their $R_u$ and update

- $O(\log(W/\epsilon))$ updates suffice to explore useful nodes up to distance $W/\epsilon$

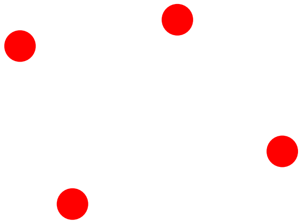
# Implementation

- Reachability sets $R_v$ for each node $v$:
  - Set of $W/\epsilon$ nodes accessible via cheapest edges
  - Initially: collect cheapest incident edges
  - Repeat $O(\log(W/\epsilon))$ times:
    Ask nodes $u$ on $R_v$ for their $R_u$ and update
- $O(\log(W/\epsilon))$ updates suffice to explore useful nodes up to distance $W/\epsilon$
- Use QuickSort-like sorting algorithm of Goodrich, Sitchinava, Zhang (2011) to organize communication

# Outline

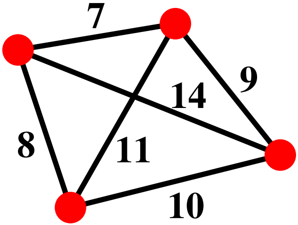
# Geometric Setting

Input: set of points in low dimensional metric space



# Geometric Setting

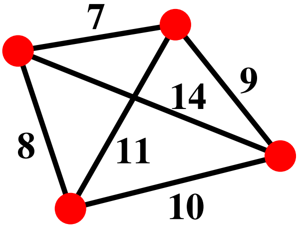
Input: set of points in low dimensional metric space




- Points induce a weighted graph

# Geometric Setting

Input: set of points in low dimensional metric space




- Points induce a weighted graph
- Graph problems to consider:
  - Minimum Spanning Tree
  - Earth Mover Distance
  - Transportation Problem
  - Travelling Salesman Problem
  - ...

# Result

[Andoni, Nikolov, O., Yaroslavtsev 2014]

- Input: $N$ points in low dimensional metric space
  - Example: $\mathbb{R}^2$
  - Generalizes to bounded doubling dimension

# Result

[Andoni, Nikolov, O., Yaroslavtsev 2014]

- Input: $N$ points in low dimensional metric space
  - Example: $\mathbb{R}^2$
  - Generalizes to bounded doubling dimension
- Algorithm:
  - Computes $(1 + \epsilon)$-approximate MST

# Result

[Andoni, Nikolov, O., Yaroslavtsev 2014]

- Input: $N$ points in low dimensional metric space
  - Example: $\mathbb{R}^2$
  - Generalizes to bounded doubling dimension
- Algorithm:
  - Computes $(1 + \epsilon)$-approximate MST
  - Space per machine: roughly $O(N/m)$
    (as long as it fits subproblems)

# Result

[Andoni, Nikolov, O., Yaroslavtsev 2014]

- Input: $N$ points in low dimensional metric space
  - Example: $\mathbb{R}^2$
  - Generalizes to bounded doubling dimension
- Algorithm:
  - Computes $(1+\epsilon)$-approximate MST
  - Space per machine: roughly $O(N/m)$
    (as long as it fits subproblems)
  - Number of rounds: $O(1)$

# Result

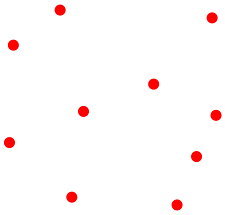
[Andoni, Nikolov, O., Yaroslavtsev 2014]

- Input: $N$ points in low dimensional metric space
  - Example: $\mathbb{R}^2$
  - Generalizes to bounded doubling dimension
- Algorithm:
  - Computes $(1 + \epsilon)$-approximate MST
  - Space per machine: roughly $O(N/m)$
    (as long as it fits subproblems)
  - Number of rounds: $O(1)$
  - Running time: near-linear

# Random Gridding

We reuse the Arora-Mitchell approach:

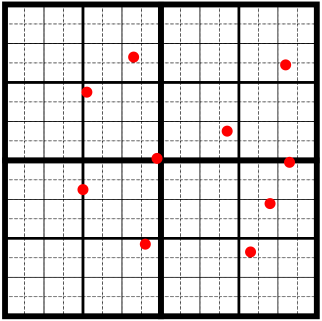
Apply a randomly shifted grid



# Random Gridding

We reuse the Arora-Mitchell approach:

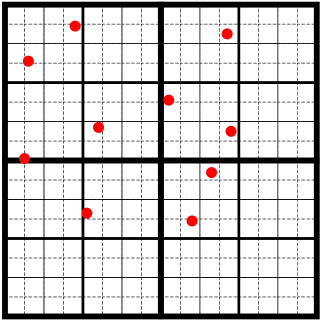
Apply a randomly shifted grid



# Random Gridding

We reuse the Arora-Mitchell approach:

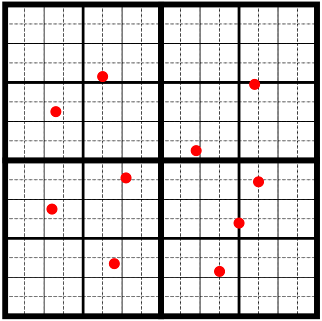
Apply a randomly shifted grid



# Random Gridding

We reuse the Arora-Mitchell approach:

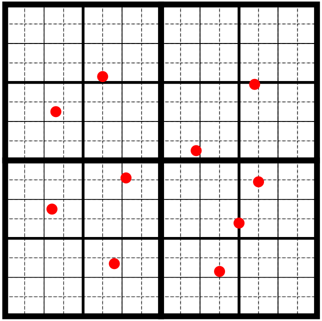
Apply a randomly shifted grid



# Random Gridding

We reuse the Arora-Mitchell approach:

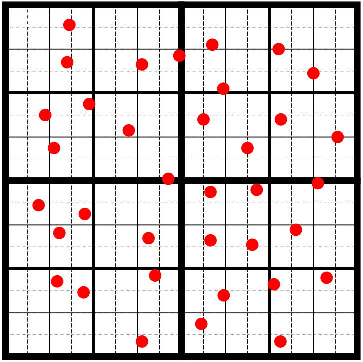
Apply a randomly shifted grid



Key property: cell of side $\Delta$ separates points $x$ and $y$ w.p. $O(1) \cdot \frac{\rho(x,y)}{\Delta}$

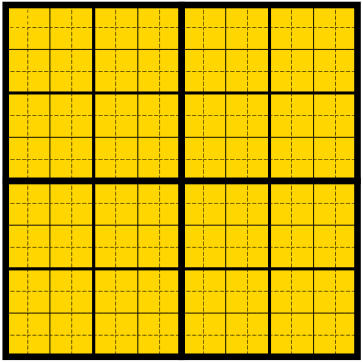
# Using Random Gridding

Typical usage: Recursive dynamic program
for approximately solving problem



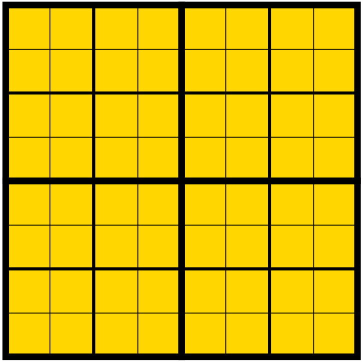
# Using Random Gridding

Typical usage: Recursive dynamic program
for approximately solving problem



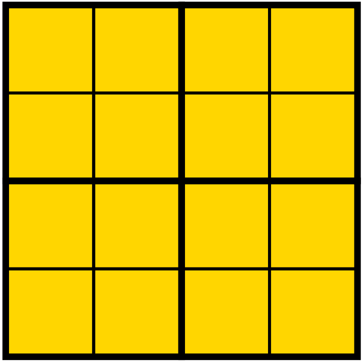
# Using Random Gridding

Typical usage: Recursive dynamic program
for approximately solving problem



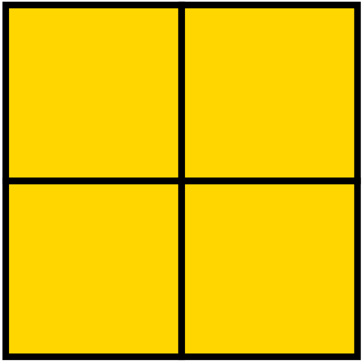
# Using Random Gridding

Typical usage: Recursive dynamic program
for approximately solving problem



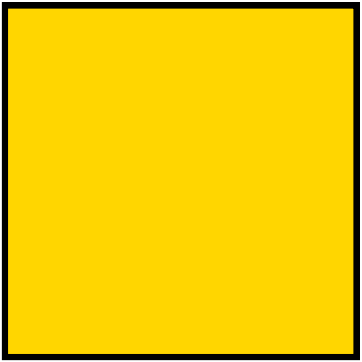
# Using Random Gridding

Typical usage: Recursive dynamic program
for approximately solving problem



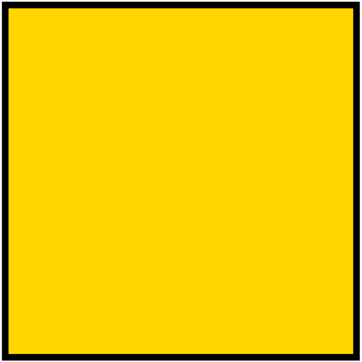
# Using Random Gridding

Typical usage: Recursive dynamic program
for approximately solving problem



# Using Random Gridding

Typical usage: Recursive dynamic program
for approximately solving problem



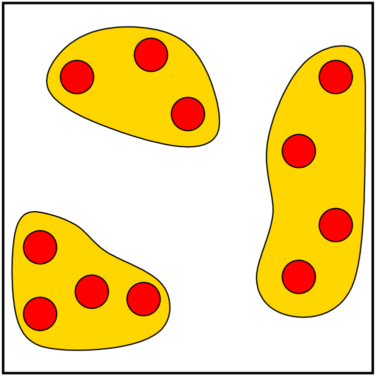
Can partially isolate what happens inside a cell

# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily

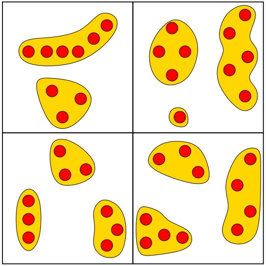
# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:
  $\epsilon^2\Delta$-covering with induced components



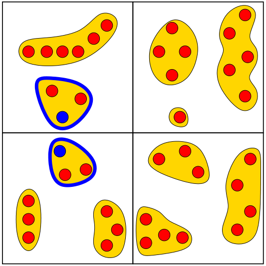
# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:
  $\epsilon^2\Delta$-covering with induced components
- Combining sub-solutions:
  Truncated version of Kruskal's algorithm



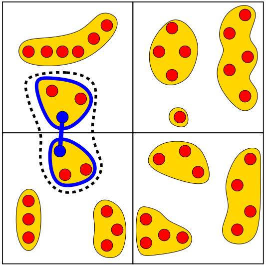
# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:
  $\epsilon^2 \Delta$-covering with induced components
- Combining sub-solutions:
  Truncated version of Kruskal's algorithm
  1. Find two closest clusters



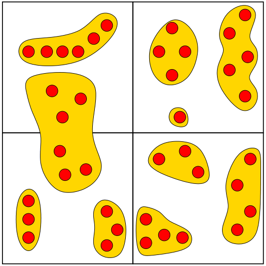
# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:
  $\epsilon^2\Delta$-covering with induced components
- Combining sub-solutions:
  Truncated version of Kruskal's algorithm
  1. Find two closest clusters
  2. If their distance less than $\epsilon\Delta$, connect them



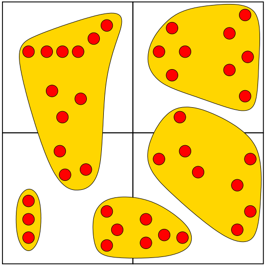
# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:
  $\epsilon^2 \Delta$-covering with induced components
- Combining sub-solutions:
  Truncated version of Kruskal's algorithm
  1. Find two closest clusters
  2. If their distance less than $\epsilon \Delta$, connect them



# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:
  $\epsilon^2\Delta$-covering with induced components
- Combining sub-solutions:
  Truncated version of Kruskal's algorithm
  1. Find two closest clusters
  2. If their distance less than $\epsilon\Delta$, connect them and repeat

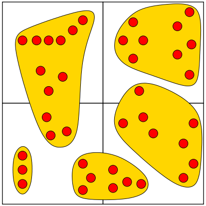


# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:
  $\epsilon^2\Delta$-covering with induced components
- Combining sub-solutions:
  Truncated version of Kruskal's algorithm
  1. Find two closest clusters
  2. If their distance less than $\epsilon\Delta$, connect them and repeat
- Pass up $\epsilon^2\Delta$-covering with information about connected components

# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:
  $\epsilon^2\Delta$-covering with induced components
- Combining sub-solutions:
  Truncated version of Kruskal's algorithm
  1. Find two closest clusters
  2. If their distance less than $\epsilon\Delta$, connect them and repeat
- Pass up $\epsilon^2\Delta$-covering with information about connected components



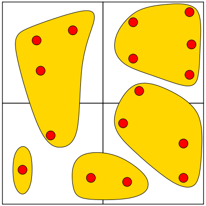
# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:

  $\epsilon^2\Delta$-covering with induced components
- Combining sub-solutions:

  Truncated version of Kruskal's algorithm

  1. Find two closest clusters
  2. If their distance less than $\epsilon\Delta$, connect them and repeat
- Pass up $\epsilon^2\Delta$-covering with information about connected components



# Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot N}$ arbitrarily
- Sub-solution for cell of side $\Delta$:
  $\epsilon^2\Delta$-covering with induced components
- Combining sub-solutions:
  Truncated version of Kruskal's algorithm
  1. Find two closest clusters
  2. If their distance less than $\epsilon\Delta$, connect them and repeat
- Pass up $\epsilon^2\Delta$-covering with information about connected components
- Expected cost of solution: optimum $\cdot$ $(1 + \epsilon \cdot \#\text{levels})$

# Select Implementation Details

- Merge $N^{\Omega(1)} \times N^{\Omega(1)}$ cells at once

# Select Implementation Details

- Merge $N^{\Omega(1)} \times N^{\Omega(1)}$ cells at once
- Sub-solutions for all subcells should fit on a single machine

# Select Implementation Details

- Merge $N^{\Omega(1)} \times N^{\Omega(1)}$ cells at once
- Sub-solutions for all subcells should fit on a single machine
- Use sorting [Goodrich, Sitchinava, Zhang 2011] for grouping points and subcells that are close

# Select Implementation Details

- Merge $N^{\Omega(1)} \times N^{\Omega(1)}$ cells at once
- Sub-solutions for all subcells should fit on a single machine
- Use sorting [Goodrich, Sitchinava, Zhang 2011] for grouping points and subcells that are close
- Near-linear time:
  - Relax Kruskal's algorithm
  - Efficient nearest neighbor data structure [Krauthgamer, Lee 2004], [Cole, Gottlieb 2006]

# Lower Bounds for MST

- Natural questions to ask:
  - Can generalize to unbounded dimension?
  - Can compute exact solution?

# Lower Bounds for MST

- Natural questions to ask:
  - Can generalize to unbounded dimension?
  - Can compute exact solution?

- Query complexity:
  - Model: distance queries
  - Our algorithm can be adapted to arbitrary bounded doubling dimensional metric in this model
  - Lower bound: $N^{\Omega(1)}$ rounds

# Lower Bounds for MST

- Natural questions to ask:
  - Can generalize to unbounded dimension?
  - Can compute exact solution?
- Query complexity:
  - Model: distance queries
  - Our algorithm can be adapted to arbitrary bounded doubling dimensional metric in this model
  - Lower bound: $N^{\Omega(1)}$ rounds
- We give a conditional lower bound based on Sparse Connectivity

# Reduction

In constant number of rounds:

Computing exact MST in $\ell_\infty^d$ for $d = 100 \log N$
$\Rightarrow$ deciding Sparse Connectivity

# Reduction

In constant number of rounds:

Computing exact MST in $\ell_\infty^d$ for $d = 100 \log N$
$\Rightarrow$ deciding Sparse Connectivity

Construction:

- For each vertex, pick a random vector $v_i$ in $\{-1, +1\}^d$
- For each edge $e = (i, j)$, add point $f(e) = v_i + v_j$

# Reduction

In constant number of rounds:

Computing exact MST in $\ell_\infty^d$ for $d = 100 \log N$
$\Rightarrow$ deciding Sparse Connectivity

Construction:

- For each vertex, pick a random vector $v_i$ in $\{-1, +1\}^d$
- For each edge $e = (i, j)$, add point $f(e) = v_i + v_j$

Distances (whp.):

- Adjacent edges: $\|f(e) - f(e')\|_\infty \leq 2$
- Non-adjacent edges: $\|f(e) - f(e')\|_\infty = 4$

# Reduction

In constant number of rounds:

Computing exact MST in $\ell_\infty^d$ for $d = 100 \log N$
$\Rightarrow$ deciding Sparse Connectivity

Construction:

- For each vertex, pick a random vector $v_i$ in $\{-1, +1\}^d$
- For each edge $e = (i, j)$, add point $f(e) = v_i + v_j$

Distances (whp.):

- Adjacent edges: $\|f(e) - f(e')\|_\infty \leq 2$
- Non-adjacent edges: $\|f(e) - f(e')\|_\infty = 4$

MST weight:

- Connected: $\leq 2(M - 1)$
- Not connected: $\geq 2M$

# Other Results

[Andoni, Nikolov, O., Yaroslavtsev 2014]

- Algorithm for approximating Earth-Mover Distance
- A new way of partitioning the instance into subproblems
- Resolves an open question of Sharathkumar & Agarwal (2012) about the transportation problem:

  First near-linear time algorithm

# Summary

- Main goal:

Efficient algorithms
for the Massive Parallel
Computation Model

# Summary

- Main goal:

  Efficient algorithms
  for the Massive Parallel
  Computation Model

- Important efficiency measure: number of rounds

  When can it be made $O(1)$ with low memory?

# Summary

- Main goal:

  Efficient algorithms
  for the Massive Parallel
  Computation Model

- Important efficiency measure: number of rounds

  When can it be made $O(1)$ with low memory?

- Well known obstacle: Sparse Connectivity

# Summary

- Main goal:

  Efficient algorithms
  for the Massive Parallel
  Computation Model

- Important efficiency measure: number of rounds

  When can it be made $O(1)$ with low memory?

- Well known obstacle: Sparse Connectivity
- This talk: efficient algorithms for MST

# Summary

- Main goal:

  Efficient algorithms
  for the Massive Parallel
  Computation Model

- Important efficiency measure: number of rounds

  When can it be made $O(1)$ with low memory?

- Well known obstacle: Sparse Connectivity
- This talk: efficient algorithms for MST
- Future research:
  - More such algorithms
  - Better understanding of our limitations

# Questions?