Massively Parallel Communication and Geometric Graph Problems

Krzysztof Onak
IBM T.J. Watson Research Center

Joint work with **Alexandr Andoni** (Microsoft),
Aleksandar Nikolov (Rutgers),
and **Grigory Yaroslavtsev** (Brown)

Modern Systems for Big Data

- Modern data sets can be large

Modern Systems for Big Data

- Modern data sets can be large
- One way to deal with large data:

  Large Parallel Computation Systems

Modern Systems for Big Data

- Modern data sets can be large
- One way to deal with large data:

  Large Parallel Computation Systems

- Examples:
  - MapReduce (Google) [Dean, Ghemawat 2004]
  - Pregel (Google) [Malewicz et al. 2009]
  - Hadoop (Yahoo!, Facebook, IBM)
  - Dryad (Microsoft) [Isard et al. 2007]

Modeling Attempts

- Multiple attempts at modeling (MapReduce as main focus):
  [Feldman, Muthukrishnan, Sidiropoulos, Stein, Svitkina 2008]
  [Karloff, Suri, Vassilvitskii 2010]
  [Goodrich, Sitchinava, Zhang 2011]
  [Beame, Koutris, Suciu 2013 & 2014]

Modeling Attempts

- Multiple attempts at modeling (MapReduce as main focus):
  [Feldman, Muthukrishnan, Sidiropoulos, Stein, Svitkina 2008]
  [Karloff, Suri, Vassilvitskii 2010]
  [Goodrich, Sitchinava, Zhang 2011]
  [Beame, Koutris, Suciu 2013 & 2014]
- Our model closest to Beame et al.'s:
  Massively Parallel Communication (MPC)

Modeling Attempts

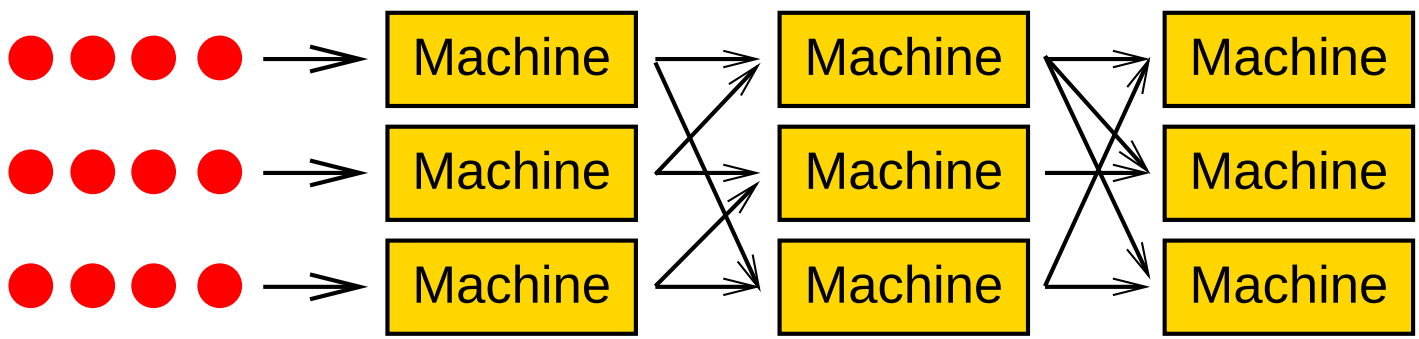
- Multiple attempts at modeling (MapReduce as main focus):
  [Feldman, Muthukrishnan, Sidiropoulos, Stein, Svitkina 2008]
  [Karloff, Suri, Vassilvitskii 2010]
  [Goodrich, Sitchinava, Zhang 2011]
  [Beame, Koutris, Suciu 2013 & 2014]
- Our model closest to Beame et al.'s:
  Massively Parallel Communication (MPC)
- Can be seen as a specific case of BSP
  (Bulk Synchronous Parallel) [Valiant 1990]

Modeling Attempts

- Multiple attempts at modeling (MapReduce as main focus):
  [Feldman, Muthukrishnan, Sidiropoulos, Stein, Svitkina 2008]
  [Karloff, Suri, Vassilvitskii 2010]
  [Goodrich, Sitchinava, Zhang 2011]
  [Beame, Koutris, Suciu 2013 & 2014]
- Our model closest to Beame et al.'s:
  Massively Parallel Communication (MPC)
- Can be seen as a specific case of BSP
  (Bulk Synchronous Parallel) [Valiant 1990]
  - Multiple parameters
  - Never thoroughly explored

Our Model

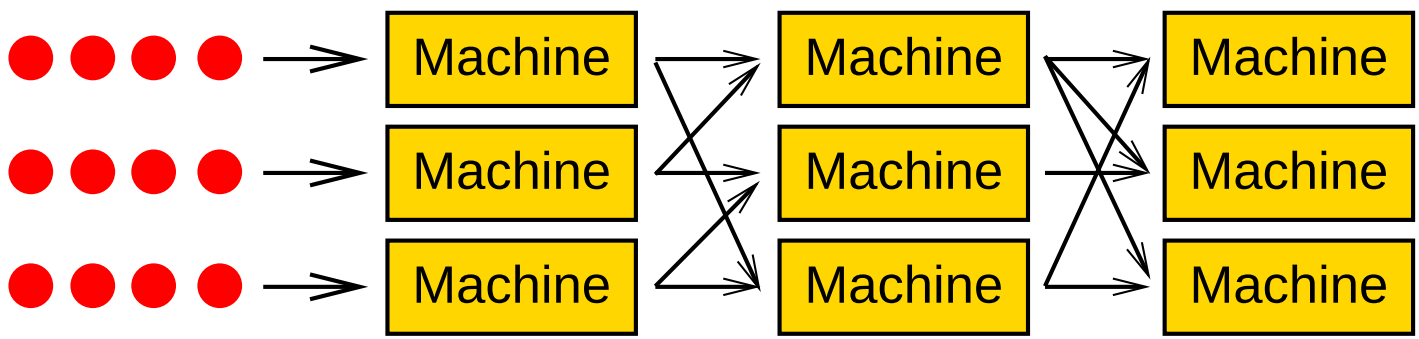
- n items on input
- m machines
- s space per machine, $s = O(n/m)$



Our Model

- n items on input
- m machines
- s space per machine, $s = O(n/m)$

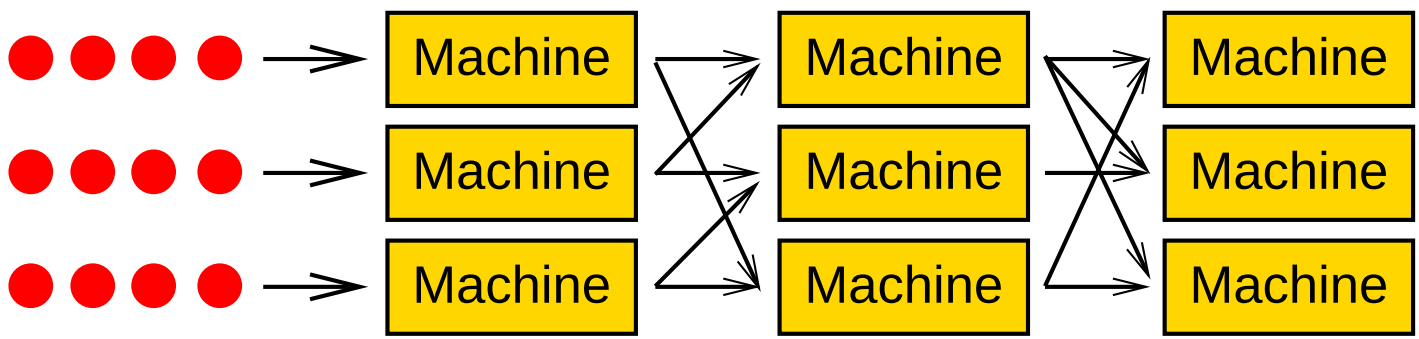
- Initially: each machine receives $1/m$ fraction of data



Our Model

- n items on input
- m machines
- s space per machine, $s = O(n/m)$

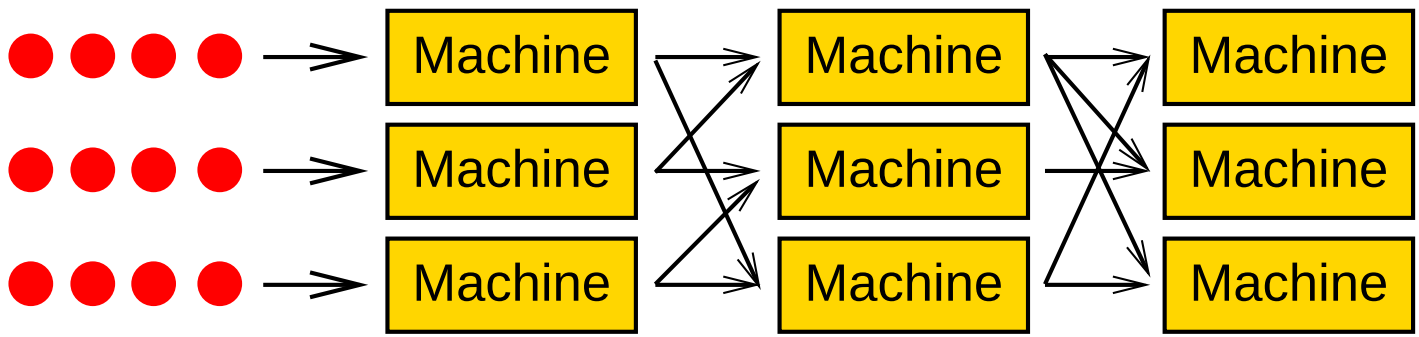
- Initially: each machine receives $1/m$ fraction of data
- Single round of computation:
  1. Each machine performs computation
     Hopefully: time poly in s
     Ideally: time near-linear in s



Our Model

- n items on input
- m machines
- s space per machine, $s = O(n/m)$

- Initially: each machine receives $1/m$ fraction of data
- Single round of computation:
  1. Each machine performs computation
     Hopefully: time poly in s
     Ideally: time near-linear in s
  2. Each machine sends and receives
     at most $O(s)$ information



Our Model

- n items on input
- m machines
- s space per machine, $s = O(n/m)$

- Initially: each machine receives $1/m$ fraction of data
- Single round of computation:
  1. Each machine performs computation
     Hopefully: time poly in s
     Ideally: time near-linear in s
  2. Each machine sends and receives
     at most $O(s)$ information
- Reasonable assumption:
  $s = \Omega(n^\alpha)$ for constant $\alpha \in (0, 1)$

Our Model

- n items on input
- m machines
- s space per machine, $s = O(n/m)$

- Initially: each machine receives $1/m$ fraction of data
- Single round of computation:
  1. Each machine performs computation
     Hopefully: time poly in s
     Ideally: time near-linear in s
  2. Each machine sends and receives
     at most $O(s)$ information
- Reasonable assumption:
  $s = \Omega(n^\alpha)$ for constant $\alpha \in (0, 1)$

  Example: if $s = \omega(m)$, then $\alpha \geq 1/2$

A Few Comments on the Model

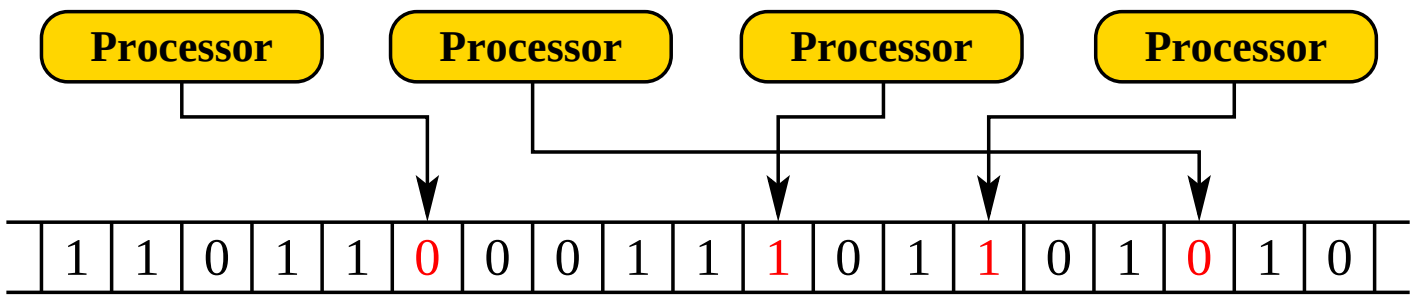
- Main focus: communication (i.e., moving data around)
- Want to minimize the number of communication rounds
- But running time of single machine important as well

A Few Comments on the Model

- Main focus: communication (i.e., moving data around)
- Want to minimize the number of communication rounds
- But running time of single machine important as well
- Relatively easy to implement in the MapReduce model

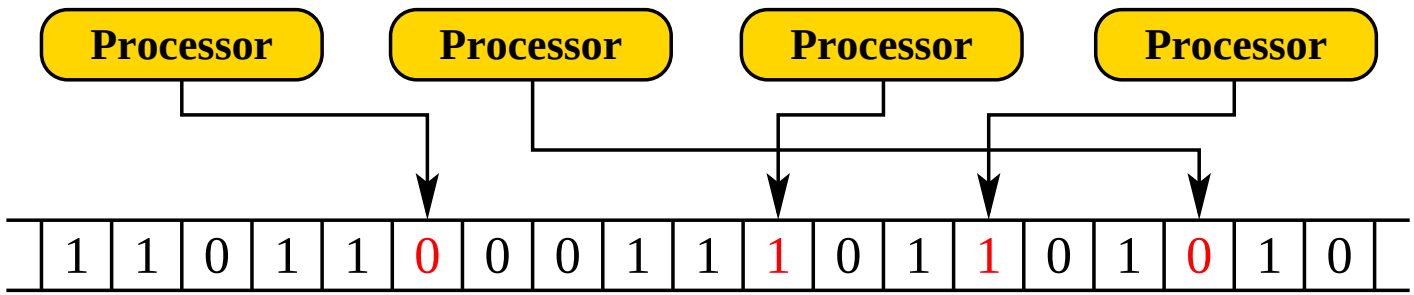
Comparison to PRAM

- PRAM: classic attempt at modeling parallelism
  - m processors
  - processors access common memory



Comparison to PRAM

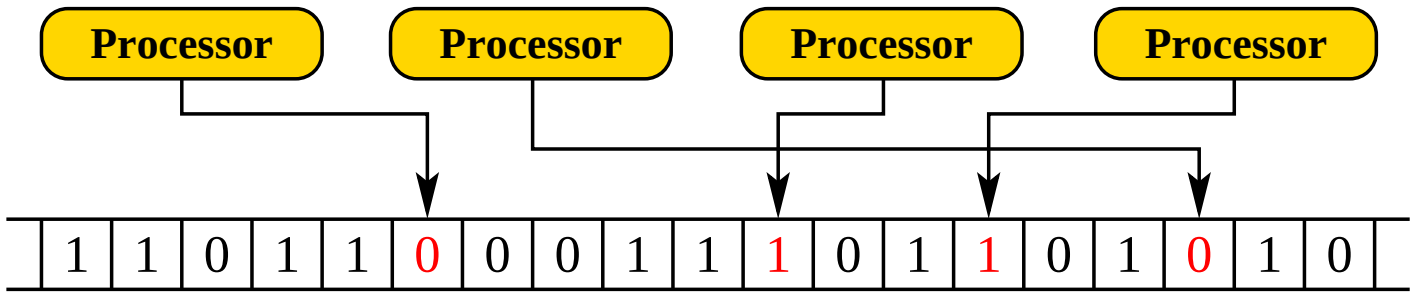
- PRAM: classic attempt at modeling parallelism
  - m processors
  - processors access common memory
- Many problems require $\tilde{\Omega}(\log n)$ rounds in PRAM



Comparison to PRAM

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

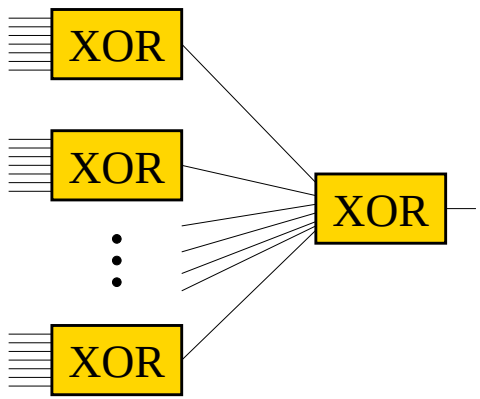
  Example: computing XOR of n bits requires $\Omega(\log n / \log \log n)$ time in strongest PRAM model [Beame, Håstad 1989]



Comparison to PRAM

- PRAM: classic attempt at modeling parallelism
  - 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 attempt at modeling parallelism
  - 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

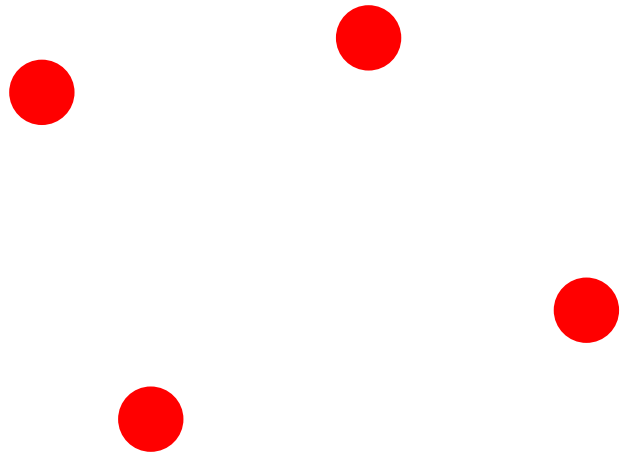
Comparison to PRAM

- PRAM: classic attempt at modeling parallelism
  - 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

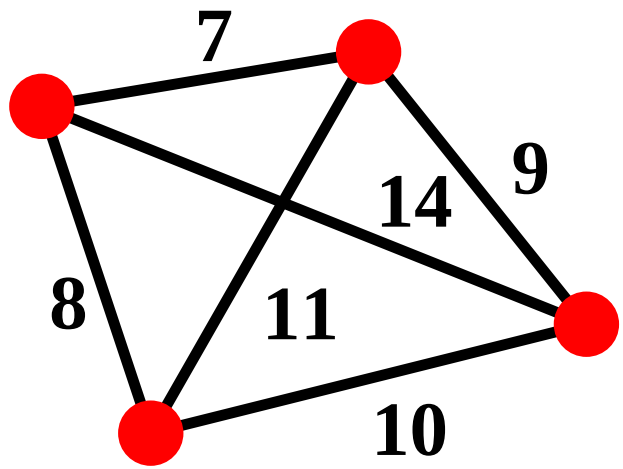
This Talk: Geometric Graph Problems

- Input: set of points in low dimensional metric space



This Talk: Geometric Graph Problems

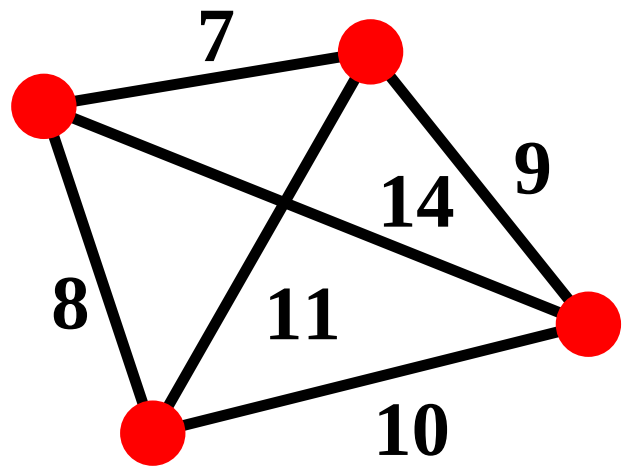
- Input: set of points in low dimensional metric space



- Points induce a weighted graph

This Talk: Geometric Graph Problems

- Input: set of points in low dimensional metric space

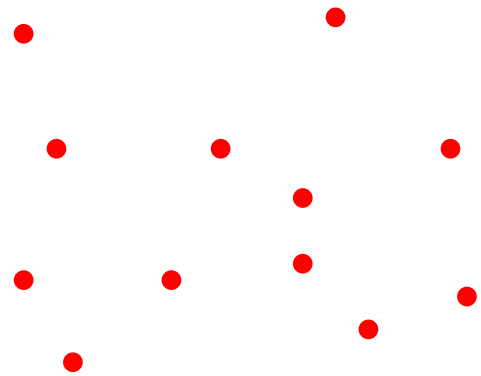


- Points induce a weighted graph
- Graph problems to consider:
  - Minimum Spanning Tree
  - Earth Mover Distance
  - Transportation Problem
  - Travelling Salesman Problem
  - ...

Minimum Spanning Tree

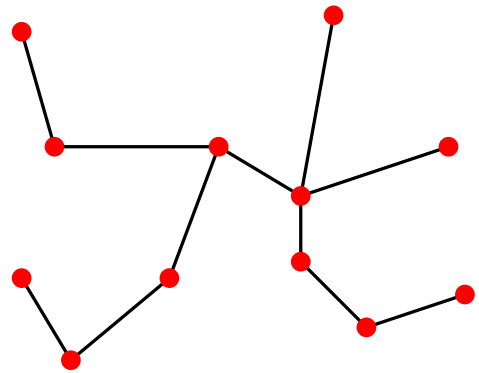
Minimum Spanning Tree

Input: n points in $\mathbb{R}^2$



Minimum Spanning Tree

Input: n points in $\mathbb{R}^2$

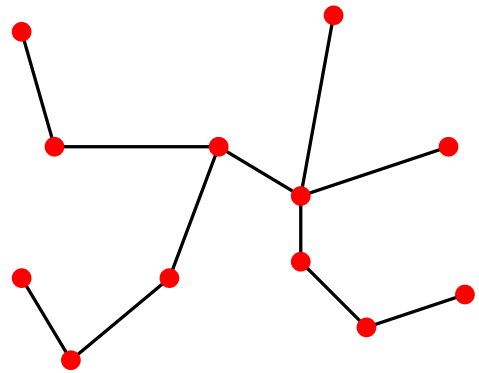


Goal:

Minimize total length of edges to connect all points

Minimum Spanning Tree

Input: n points in $\mathbb{R}^2$



Goal:

Minimize total length of edges to connect all points

Applications:

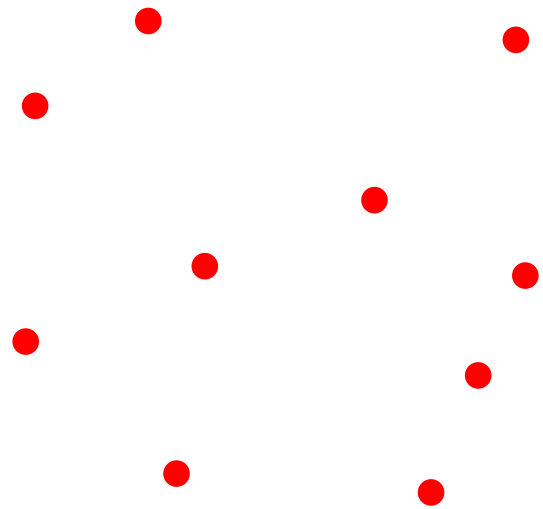
- Related to hierarchical agglomerative clustering with single linkage
- Classic practical clustering algorithms
  [Zahn 1971] [Kleinberg, Tardos]

  Remove heaviest $k-1$ edges to create k clusters

Random Gridding

We reuse the Arora-Mitchell approach:

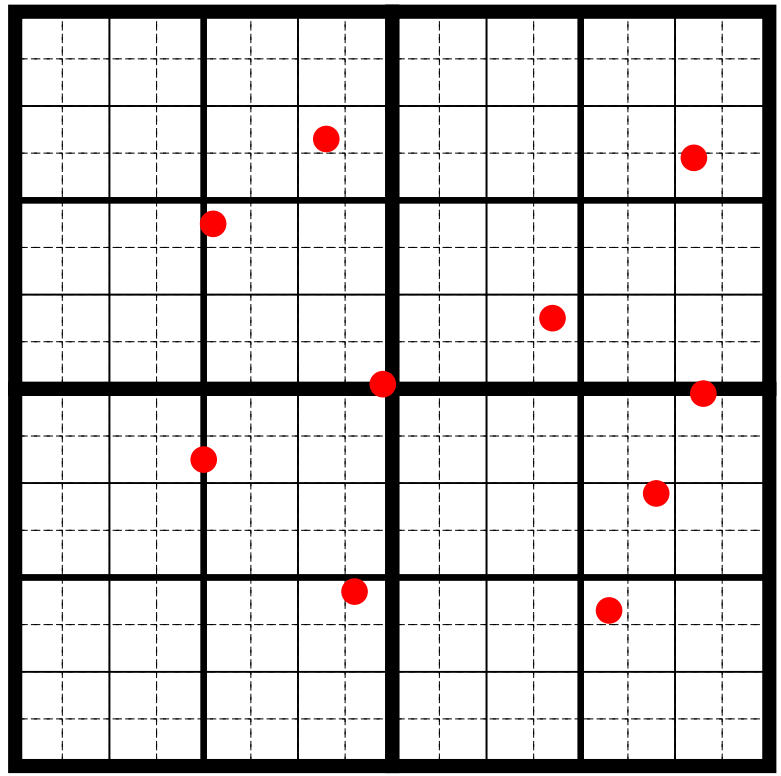
Apply a randomly shifted grid



Random Gridding

We reuse the Arora-Mitchell approach:

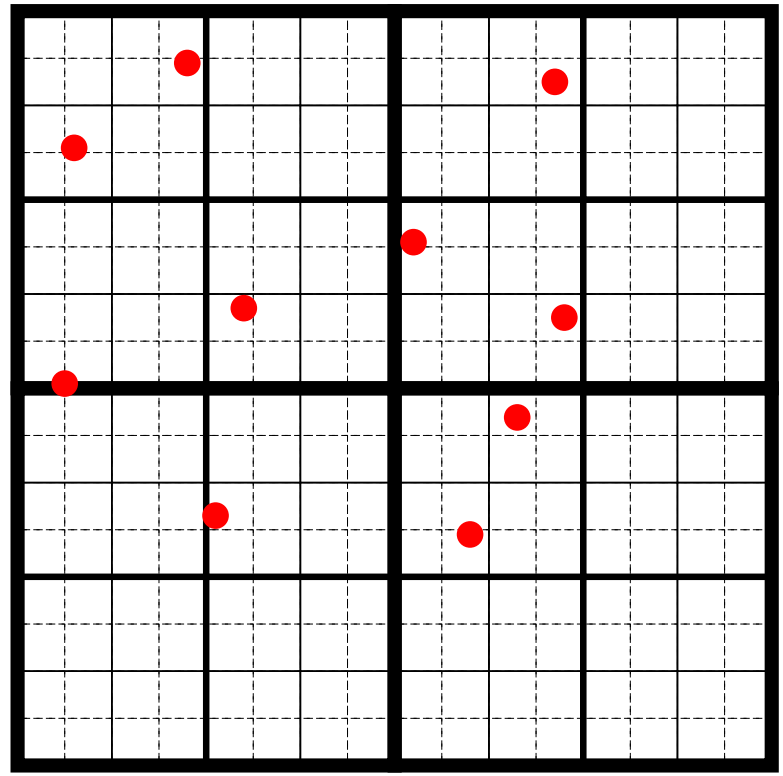
Apply a randomly shifted grid



Random Gridding

We reuse the Arora-Mitchell approach:

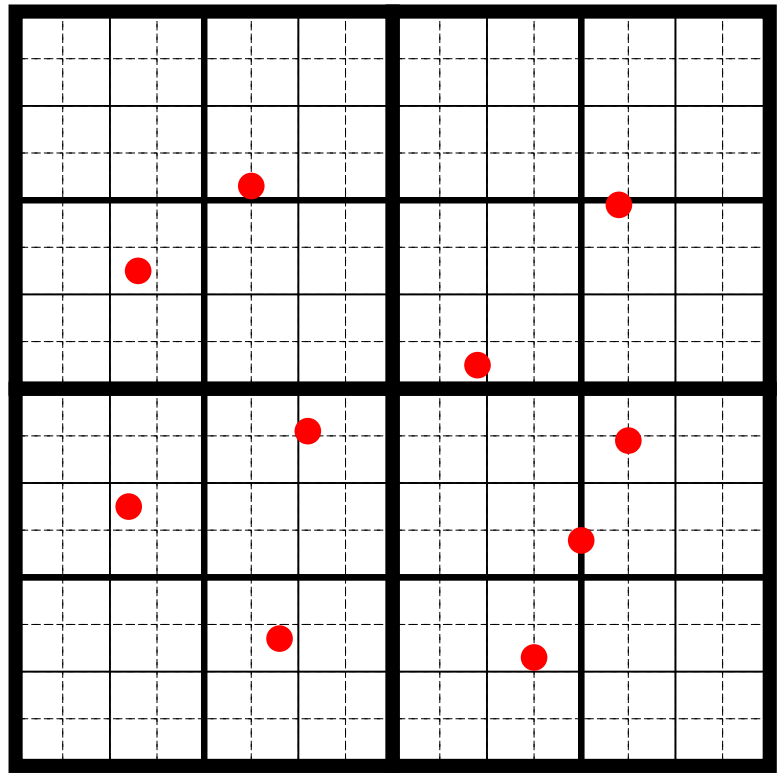
Apply a randomly shifted grid



Random Gridding

We reuse the Arora-Mitchell approach:

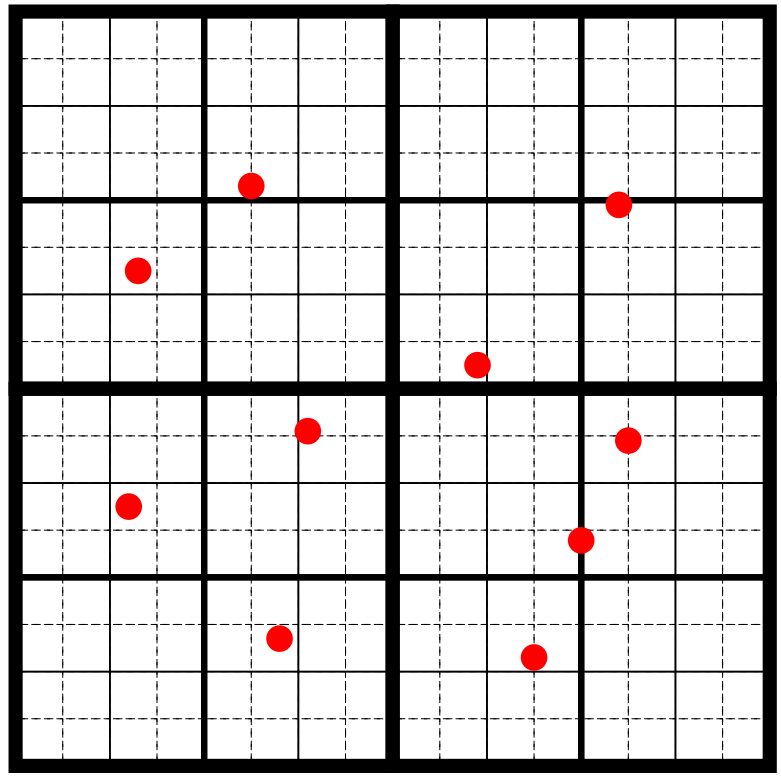
Apply a randomly shifted grid



Random Gridding

We reuse the Arora-Mitchell approach:

Apply a randomly shifted grid



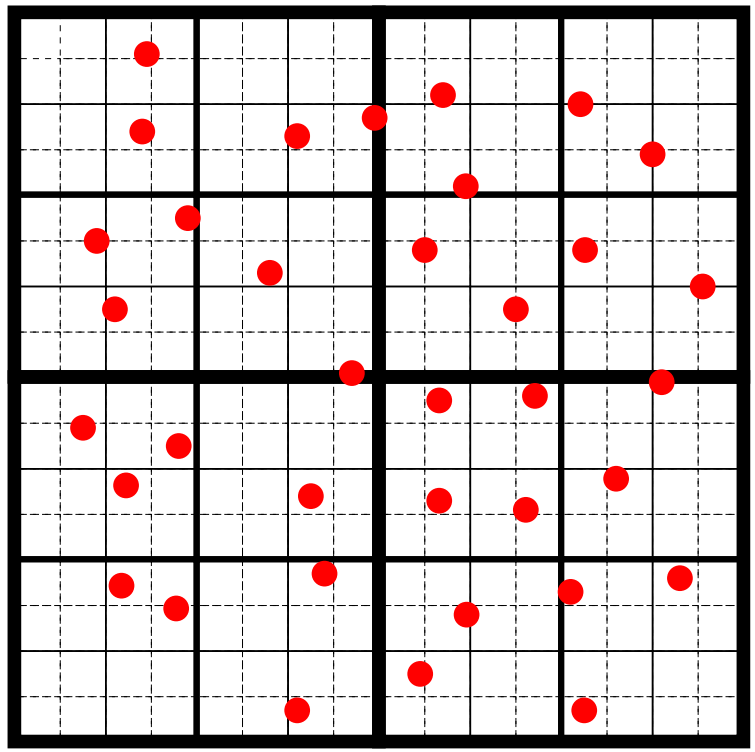
Key fact: cell of side Δ separates points x and y w.p. $O(1) \cdot \frac{\rho(x,y)}{\Delta}$

Using Random Gridding

- Typical usage: Recursive dynamic program for approximately solving problem

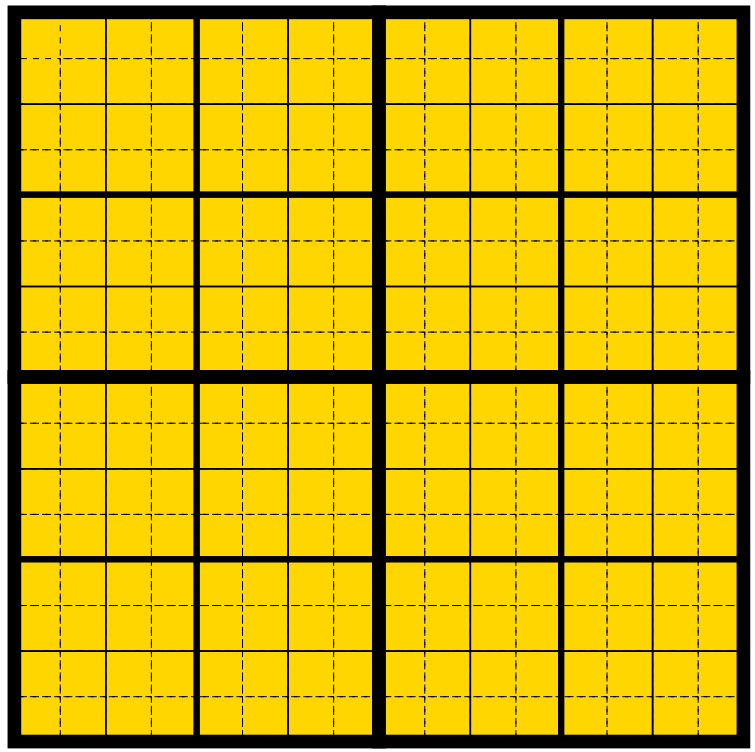
Using Random Gridding

- Typical usage: Recursive dynamic program for approximately solving problem



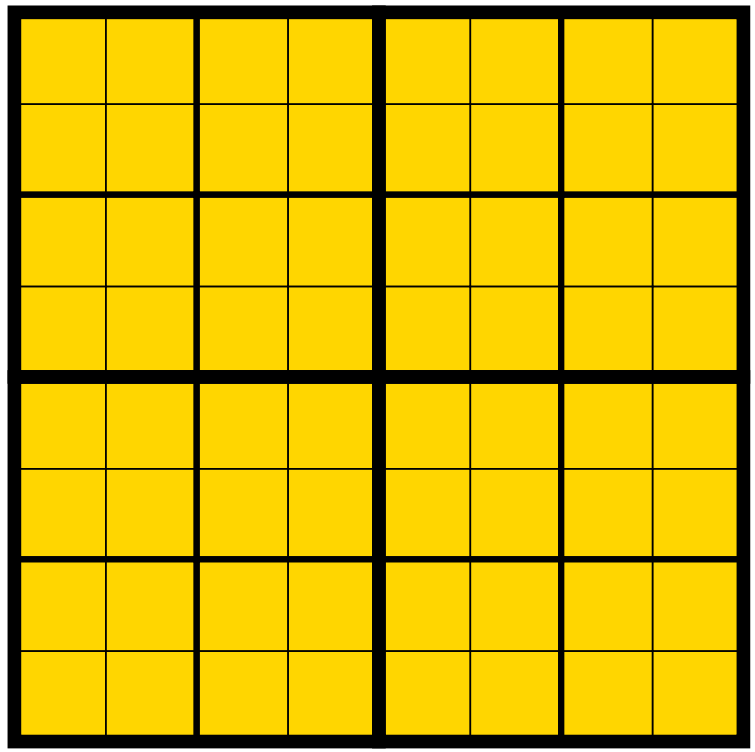
Using Random Gridding

- Typical usage: Recursive dynamic program for approximately solving problem



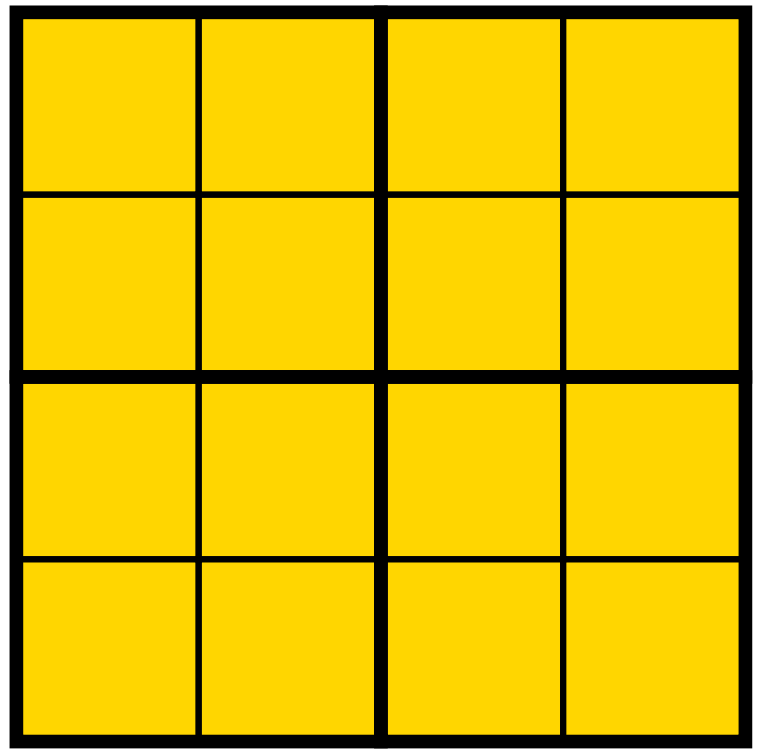
Using Random Gridding

- Typical usage: Recursive dynamic program for approximately solving problem



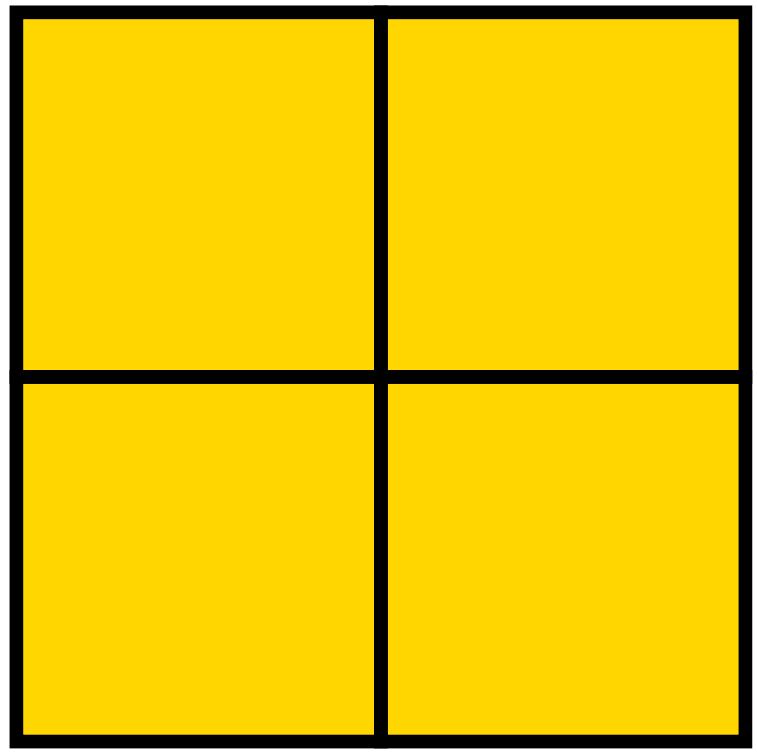
Using Random Gridding

- Typical usage: Recursive dynamic program for approximately solving problem



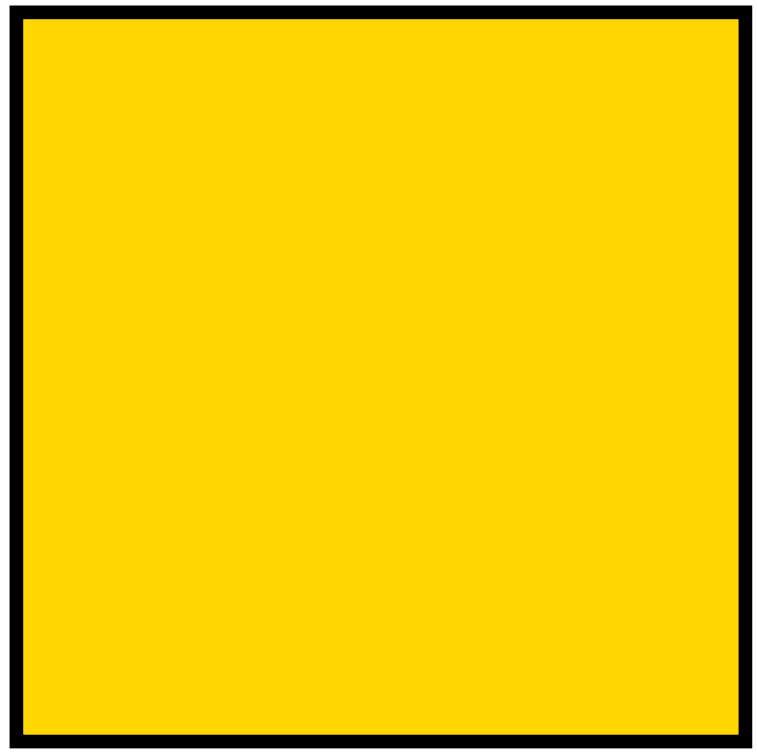
Using Random Gridding

- Typical usage: Recursive dynamic program for approximately solving problem



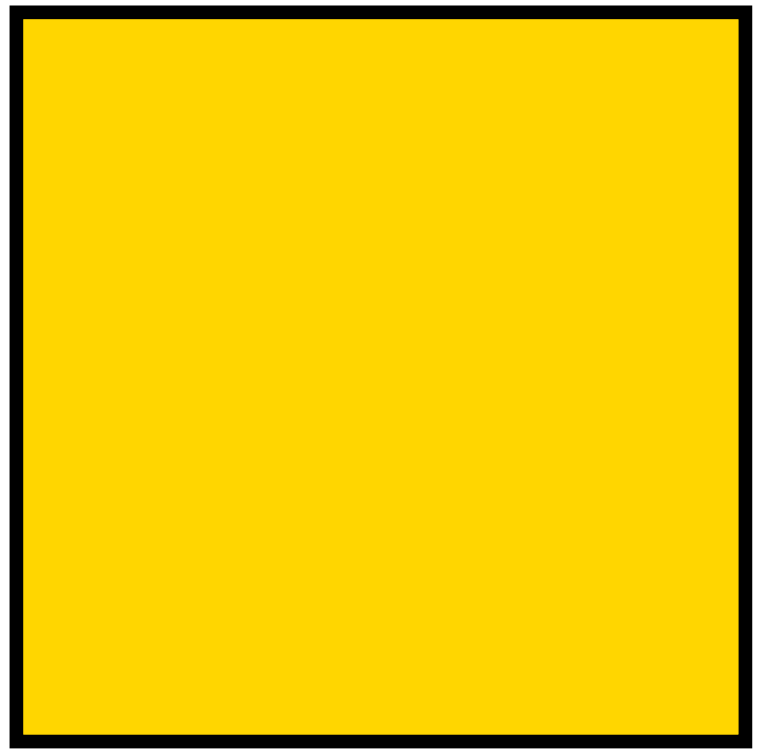
Using Random Gridding

- Typical usage: Recursive dynamic program for approximately solving problem



Using Random Gridding

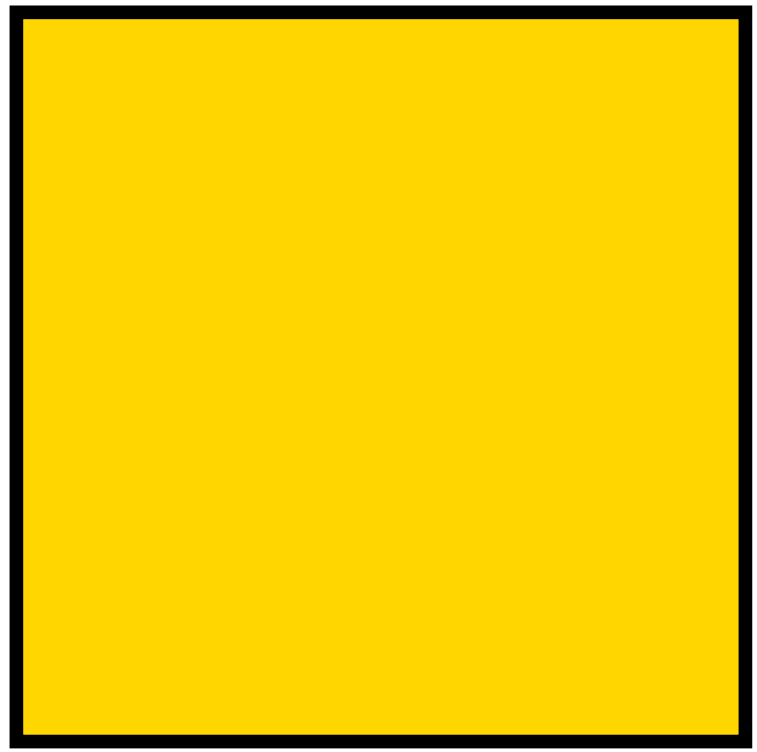
- Typical usage: Recursive dynamic program for approximately solving problem



- Classic application: Traveling Salesman Problem

Using Random Gridding

- Typical usage: Recursive dynamic program for approximately solving problem



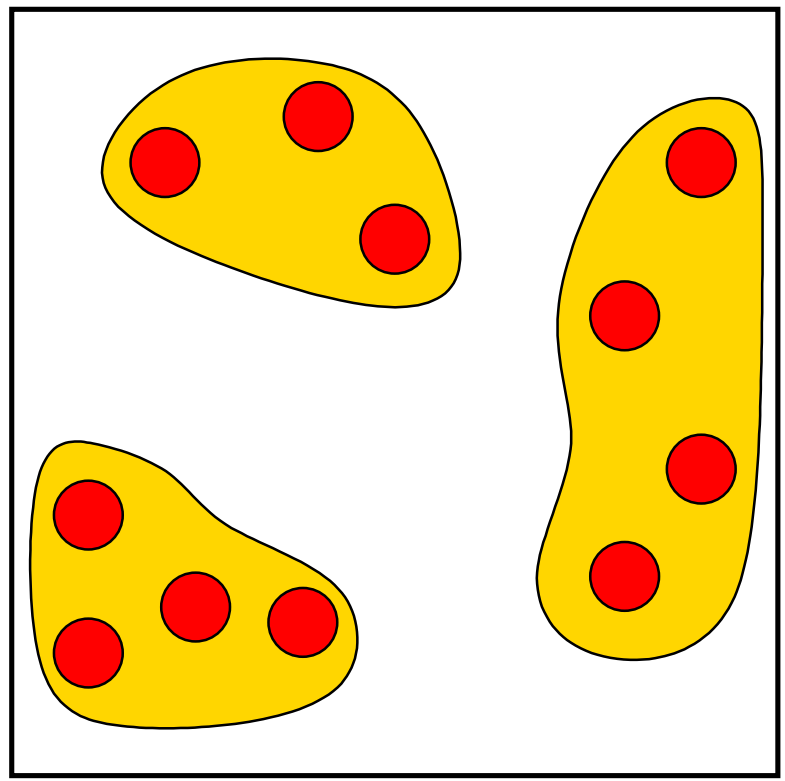
- Classic application: Traveling Salesman Problem
- Can partially isolate what happens inside a cell

Our Algorithm

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

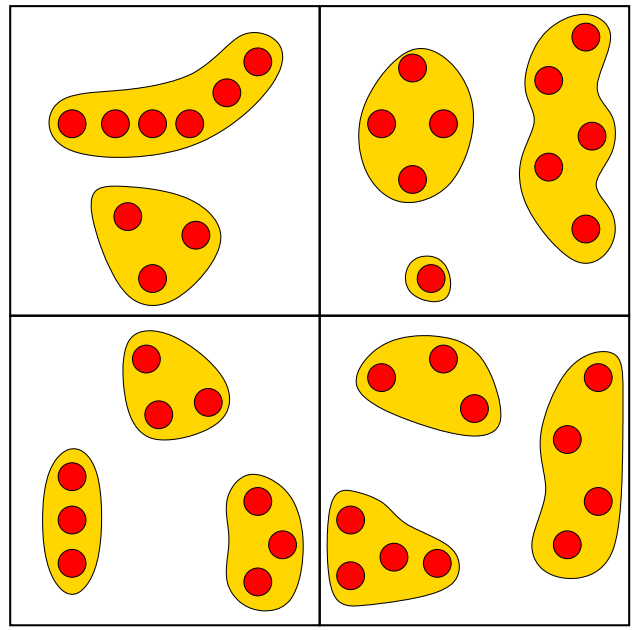
Our Algorithm

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



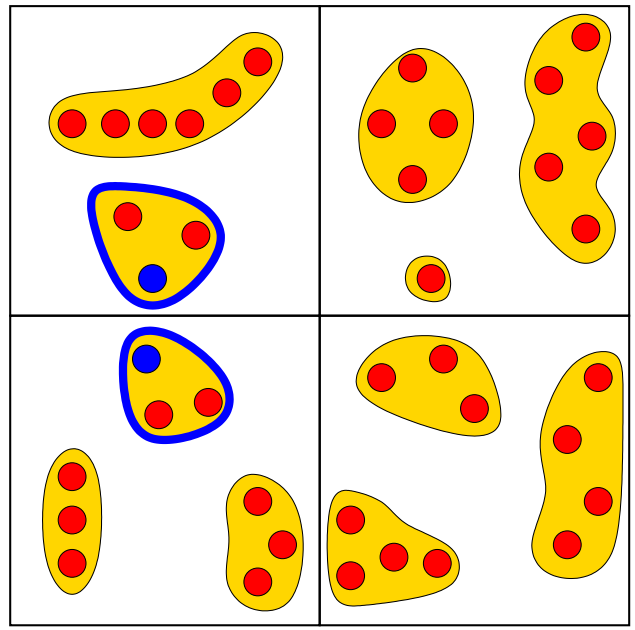
Our Algorithm

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



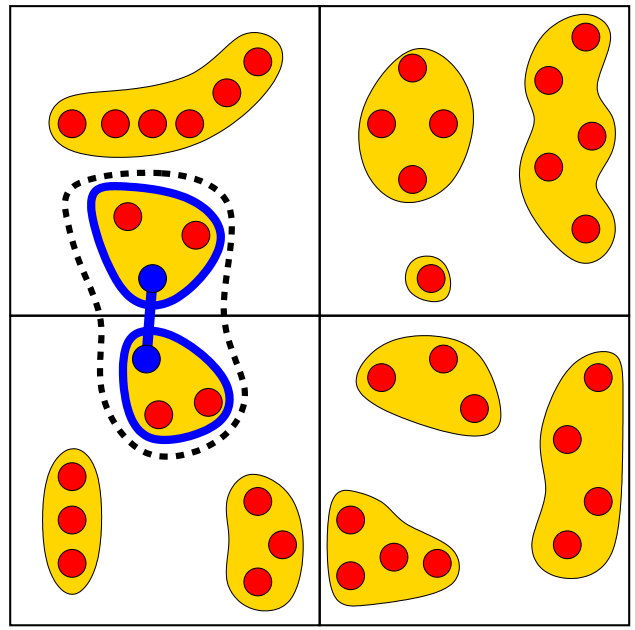
Our Algorithm

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



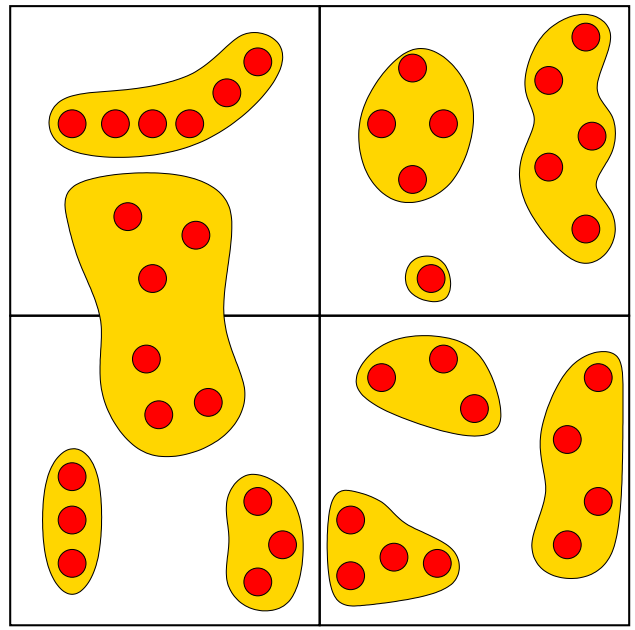
Our Algorithm

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



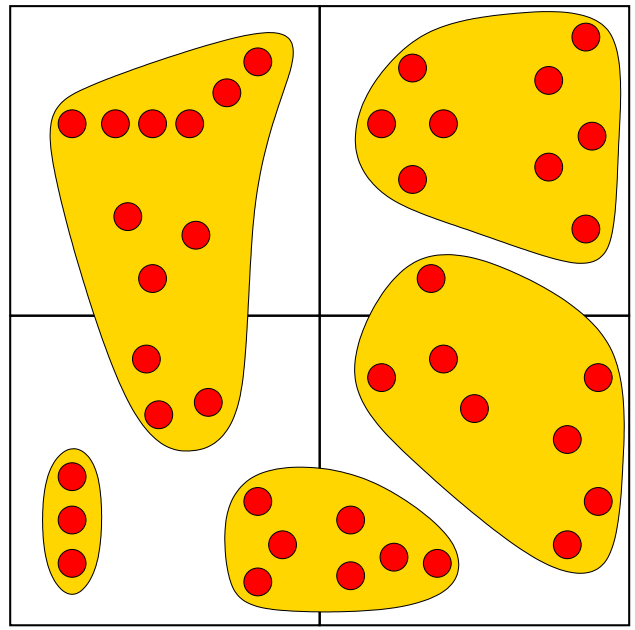
Our Algorithm

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

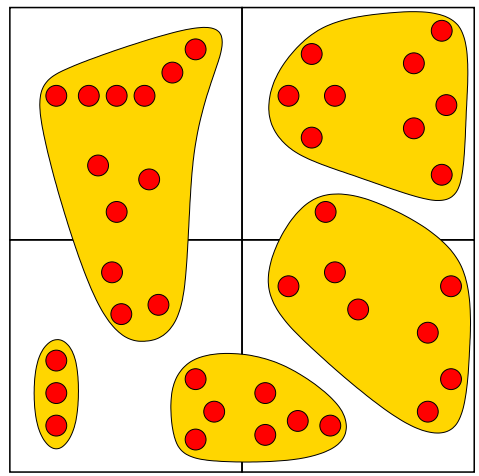


Our Algorithm

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

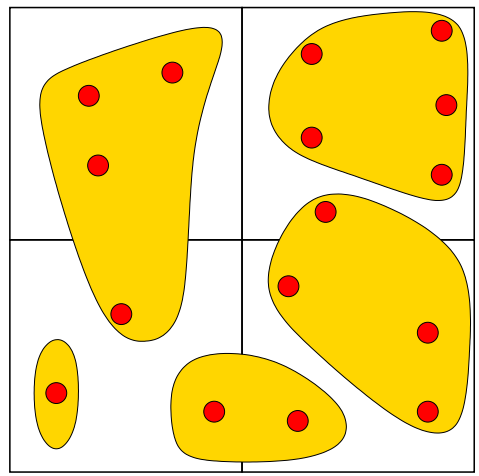
Our Algorithm

- Connect points closer than $\frac{\epsilon \cdot \mathrm{diam}(S)}{100 \cdot n}$ arbitrarily
- Sub-solution for cell of side Δ:
  $\epsilon^2\Delta$-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 Δ:
  $\epsilon^2\Delta$-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 Δ:
  $\epsilon^2\Delta$-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 $\cdot\ (1 + \epsilon \cdot$ #levels$)$

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

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 conditional lower bound based on Sparse Connectivity

Sparse Connectivity

- Input: collection of $O(n)$ edges on n vertices
- Space per machine: n^{α}, where $\alpha \in (0, 1)$
- Question: Do edges span connected graph?

Sparse Connectivity

- Input: collection of $O(n)$ edges on n vertices
- Space per machine: n^{α}, where $\alpha \in (0, 1)$
- Question: Do edges span connected graph?

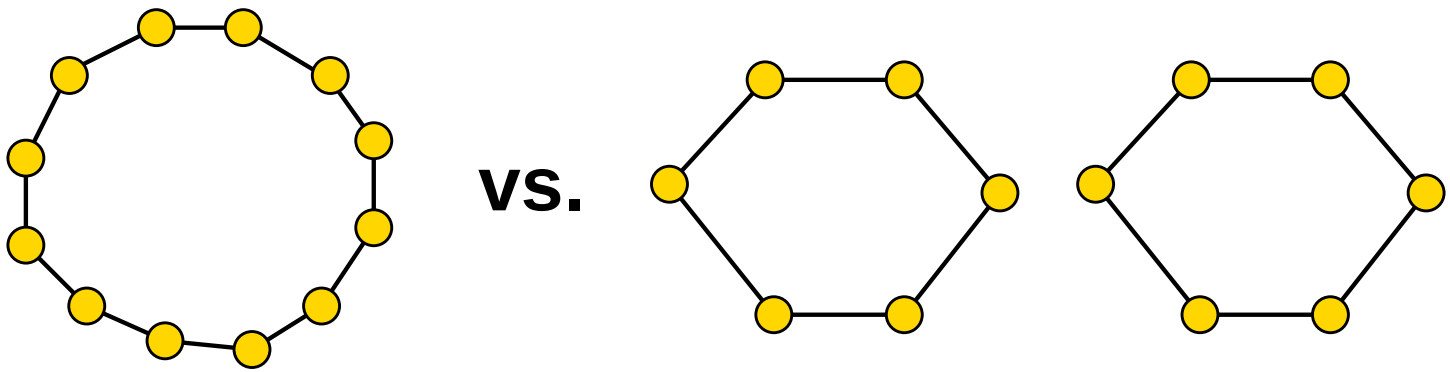
- Conjecture: superconstant number of rounds

Sparse Connectivity

- Input: collection of $O(n)$ edges on n vertices
- Space per machine: n^{α}, where $\alpha \in (0, 1)$
- Question: Do edges span connected graph?

- Conjecture: superconstant number of rounds

- Is this instance hard?

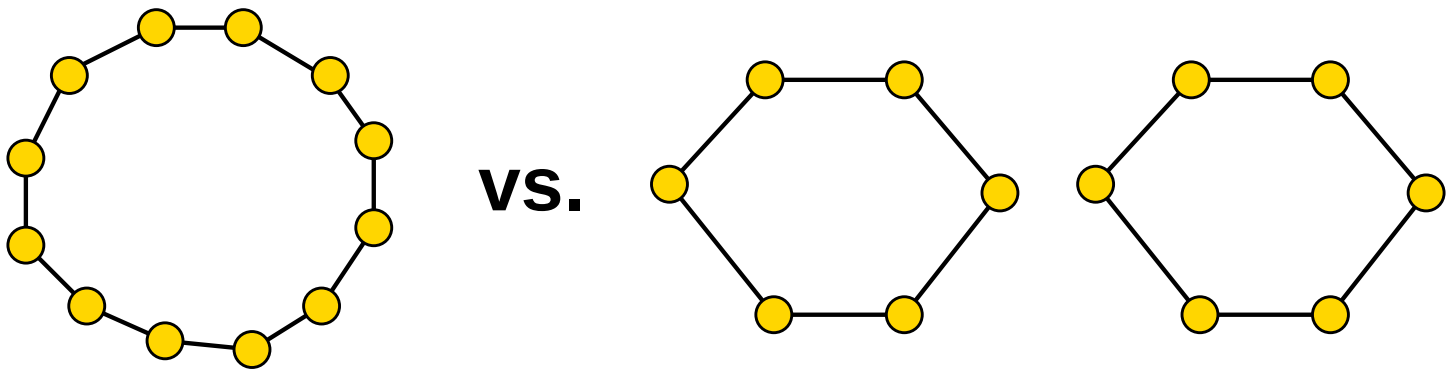


Sparse Connectivity

- Input: collection of $O(n)$ edges on n vertices
- Space per machine: n^{α}, where $\alpha \in (0, 1)$
- Question: Do edges span connected graph?

- Conjecture: superconstant number of rounds

- Is this instance hard?



- Best known algorithm: $O(\log n)$ rounds

Reduction

In constant number of rounds:

Computing exact MST in ℓ_∞^d for $d = 100 \log n$

$\Rightarrow$ deciding Sparse Connectivity

Reduction

In constant number of rounds:

Computing exact MST in ℓ_∞^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 ℓ_∞^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 ℓ_∞^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: $\|f(e) - f(e')\|_\infty = 4$

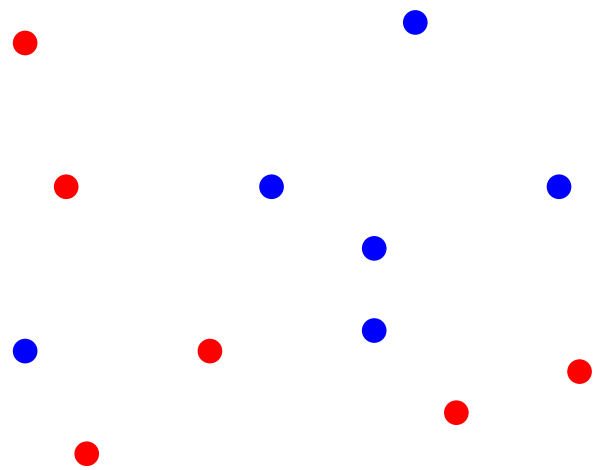
MST weight:

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

The Earth Mover Distance

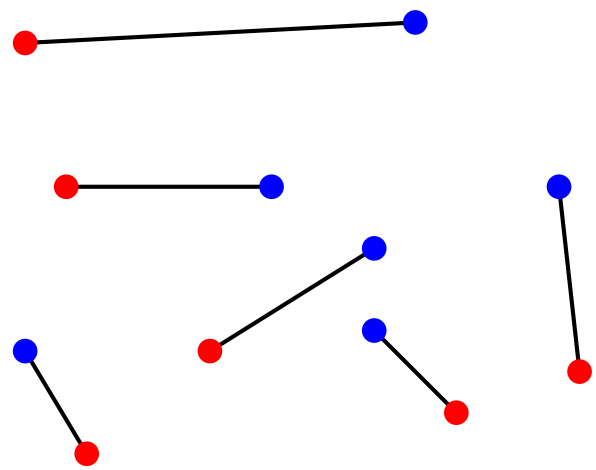
The Earth Mover Distance (EMD)

Input: n blue and n red points in $\mathbb{R}^2$



The Earth Mover Distance (EMD)

Input: n blue and n red points in $\mathbb{R}^2$

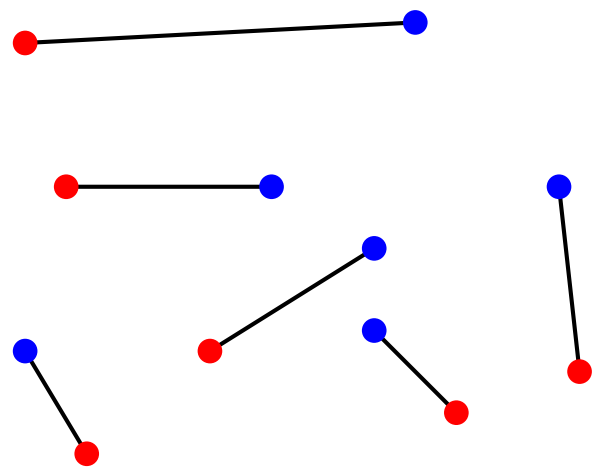


Goal:

Minimize the cost of matching blue points to red points

The Earth Mover Distance (EMD)

Input: n blue and n red points in $\mathbb{R}^2$



Goal:

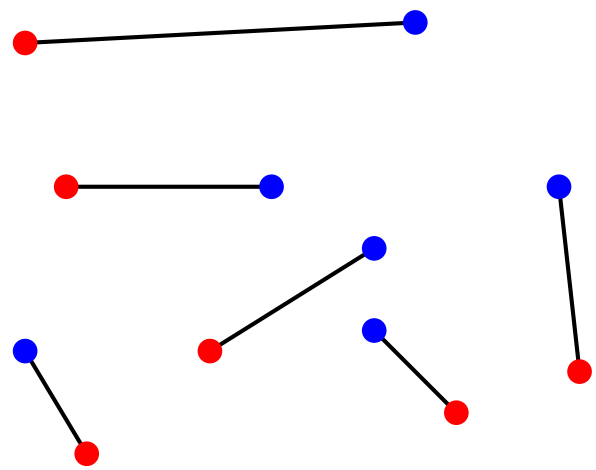
Minimize the cost of matching blue points to red points

Application:

- How similar are two images?
- Discover characteristic features
- See how much it costs to match them

The Earth Mover Distance (EMD)

Input: n blue and n red points in $\mathbb{R}^2$



Goal:

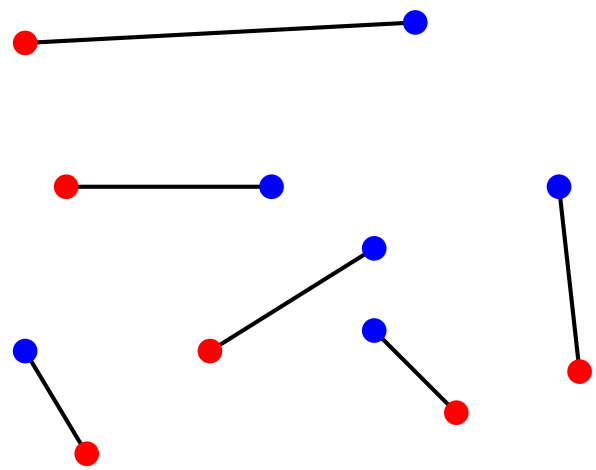
Minimize the cost of matching blue points to red points

How difficult is it?

- Not easily amenable to Arora-Mitchell approach (1996)
- Near-linear time approximation [Sharathkumar, Agarwal 2012]

The Earth Mover Distance (EMD)

Input: n blue and n red points in $\mathbb{R}^2$



Goal:

Minimize the cost of matching blue points to red points

How difficult is it?

- Not easily amenable to Arora-Mitchell approach (1996)
- Near-linear time approximation [Sharathkumar, Agarwal 2012]
- Streaming complexity is open

Tweak Problem

1. Switch to flow (transportation problem):
   - Minimum cost flow from red to blue points
   - Edge cost $=$ flow along the edge $\times$ distance

Tweak Problem

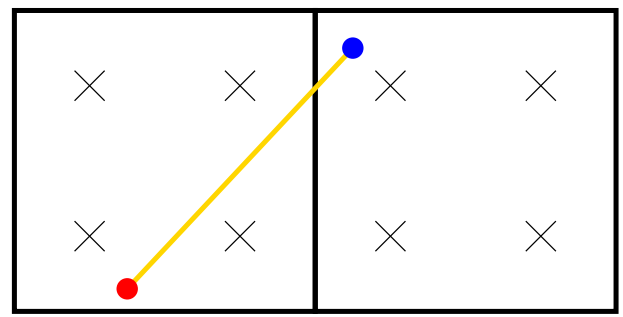
1. Switch to flow (transportation problem):
   - Minimum cost flow from red to blue points
   - Edge cost $=$ flow along the edge $\times$ distance
   - Best flow not better than best matching

Tweak Problem

1. Switch to flow (transportation problem):
   - Minimum cost flow from red to blue points
   - Edge cost $=$ flow along the edge $\times$ distance
   - Best flow not better than best matching
2. Modify metric:
   - Don't send flow directly between points

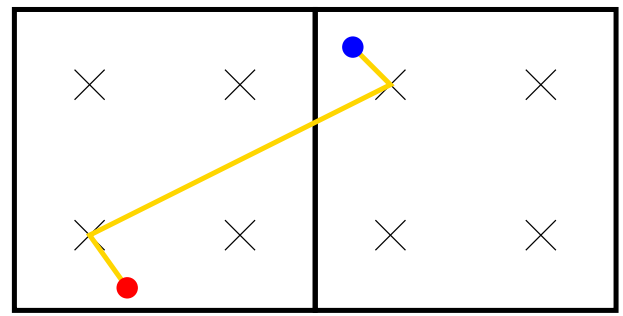
Tweak Problem

1. Switch to flow (transportation problem):
   - Minimum cost flow from red to blue points
   - Edge cost $=$ flow along the edge $\times$ distance
   - Best flow not better than best matching
2. Modify metric:
   - Don't send flow directly between points
   - Each cell covered with net of points
   - Go up the partition and send to the closest net point until can connect net points



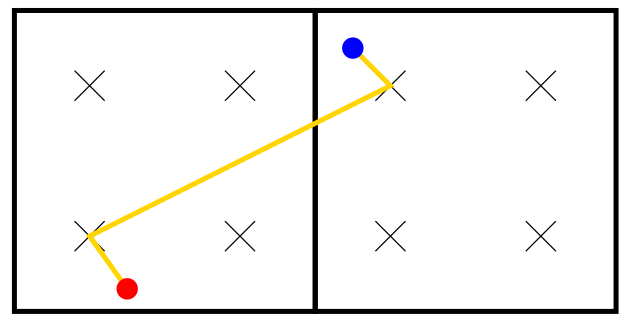
Tweak Problem

1. Switch to flow (transportation problem):
   - Minimum cost flow from red to blue points
   - Edge cost $=$ flow along the edge $\times$ distance
   - Best flow not better than best matching
2. Modify metric:
   - Don't send flow directly between points
   - Each cell covered with net of points
   - Go up the partition and send to the closest net point until can connect net points



Tweak Problem

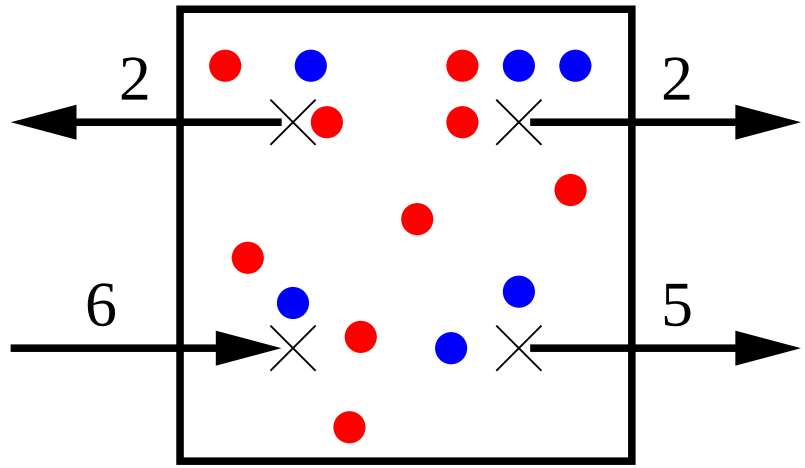
1. Switch to flow (transportation problem):
   - Minimum cost flow from red to blue points
   - Edge cost $=$ flow along the edge $\times$ distance
   - Best flow not better than best matching
2. Modify metric:
   - Don't send flow directly between points
   - Each cell covered with net of points
   - Go up the partition and send to the closest net point until can connect net points



   - For random gridding and dense net small change in value

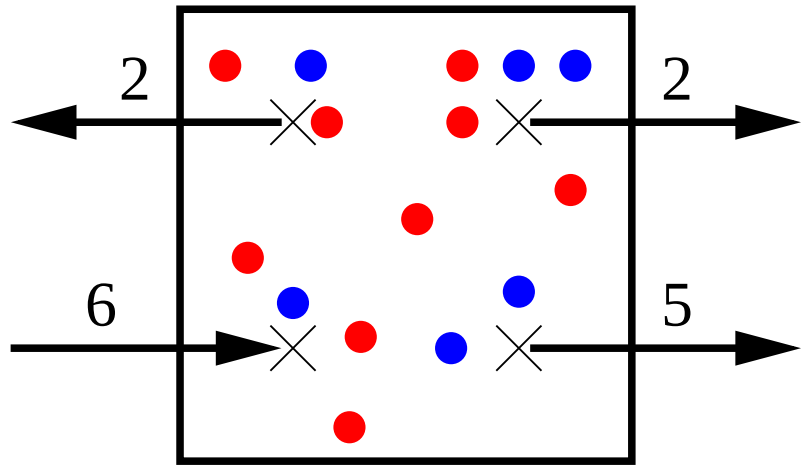
Cost Function for Subcell

- How much does it cost to send/receive given amounts of flow through net points?



Cost Function for Subcell

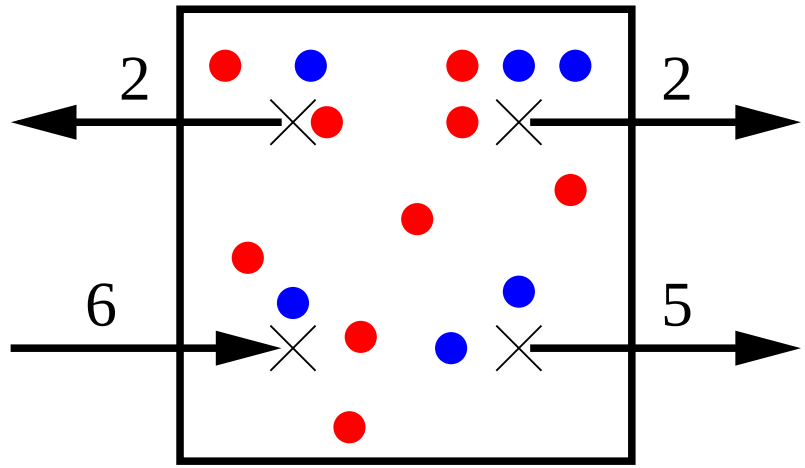
- How much does it cost to send/receive given amounts of flow through net points?



- The total (incoming/outgoing flow and supply/demand by points) has to be zero

Cost Function for Subcell

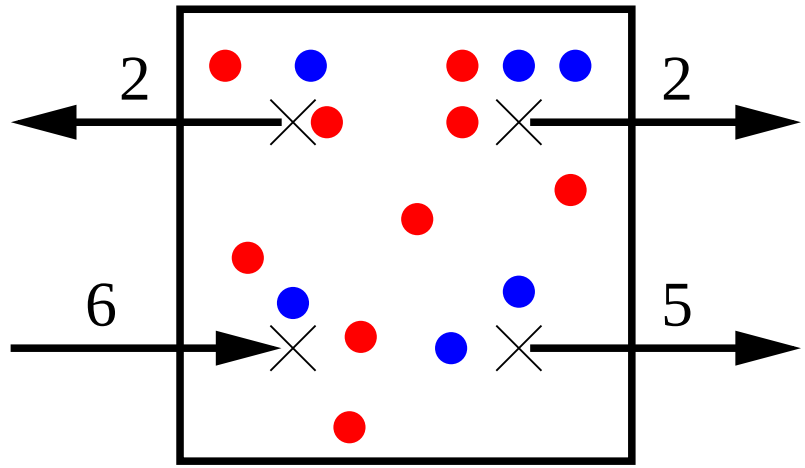
- How much does it cost to send/receive given amounts of flow through net points?



- The total (incoming/outgoing flow and supply/demand by points) has to be zero
- Cost function $F : \mathbb{R}^{\tau-1} \to \mathbb{R}$ for τ net points

Cost Function for Subcell

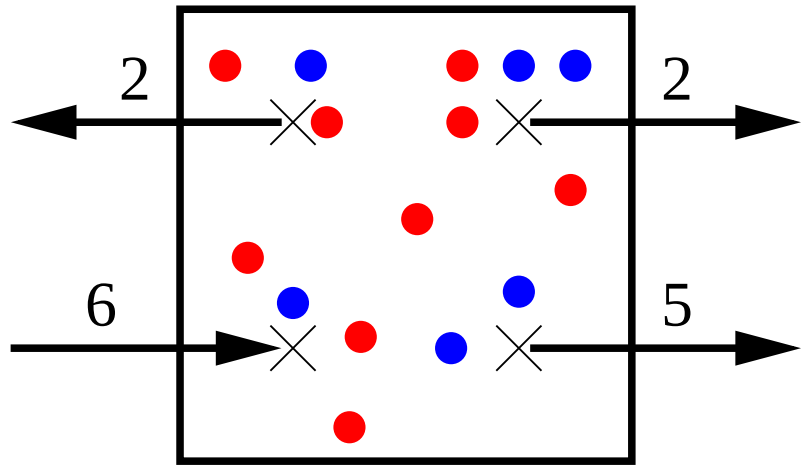
- How much does it cost to send/receive given amounts of flow through net points?



- The total (incoming/outgoing flow and supply/demand by points) has to be zero
- Cost function $F : \mathbb{R}^{\tau-1} \to \mathbb{R}$ for τ net points
- Describes all we need to know recursively

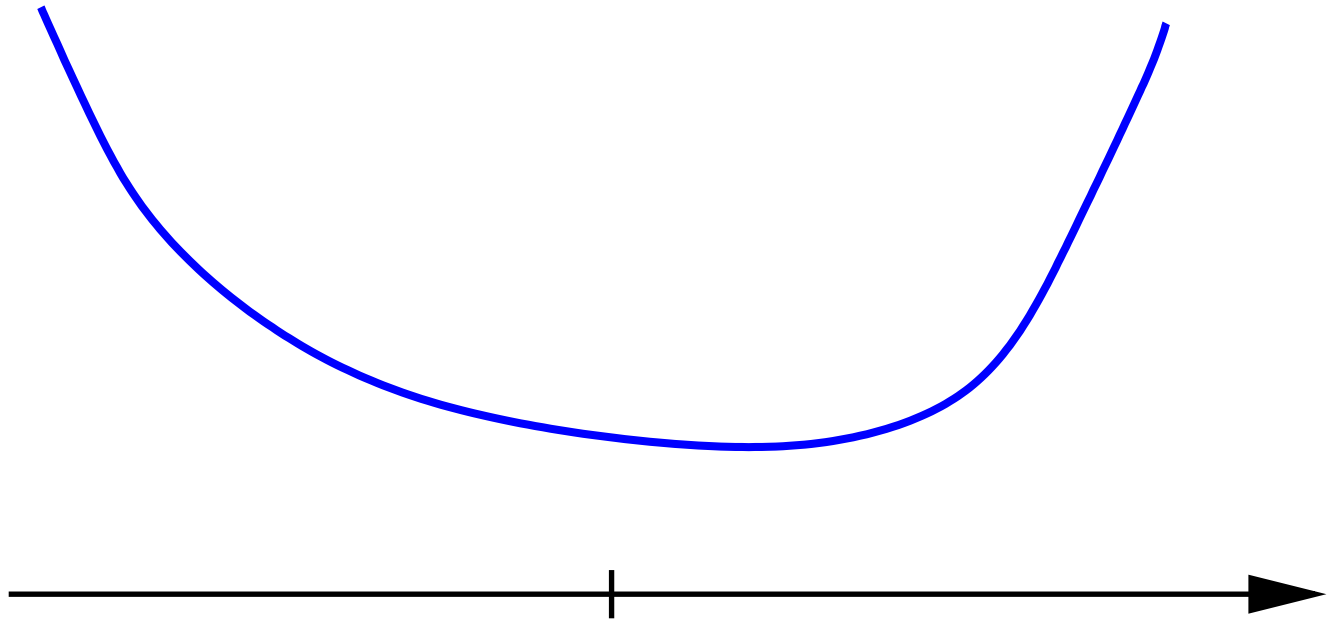
Cost Function for Subcell

- How much does it cost to send/receive given amounts of flow through net points?



- The total (incoming/outgoing flow and supply/demand by points) has to be zero
- Cost function $F : \mathbb{R}^{\tau-1} \to \mathbb{R}$ for τ net points
- Describes all we need to know recursively
- Problem: potentially a lot of information

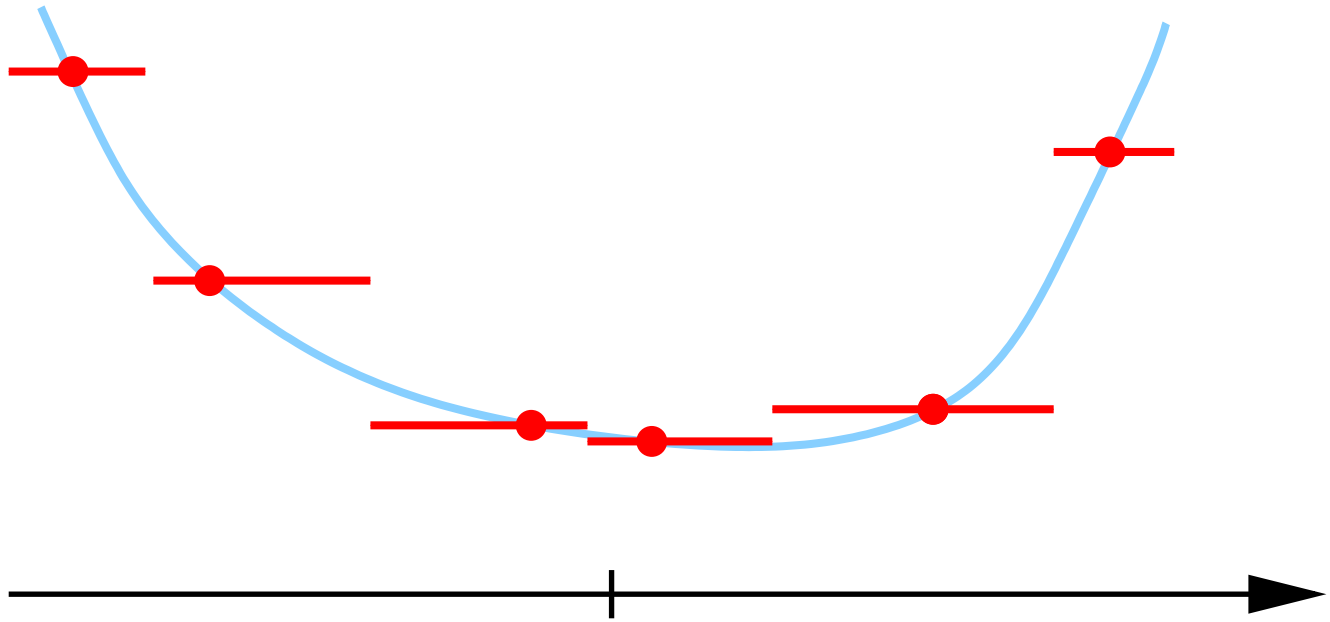
Compressed Form



- For $x, x' \in \mathbb{R}^{\tau_i}$ s.t. all $1 - \epsilon^{O(1)} \leq x_i/x'_i \leq 1 + \epsilon^{O(1)}$,

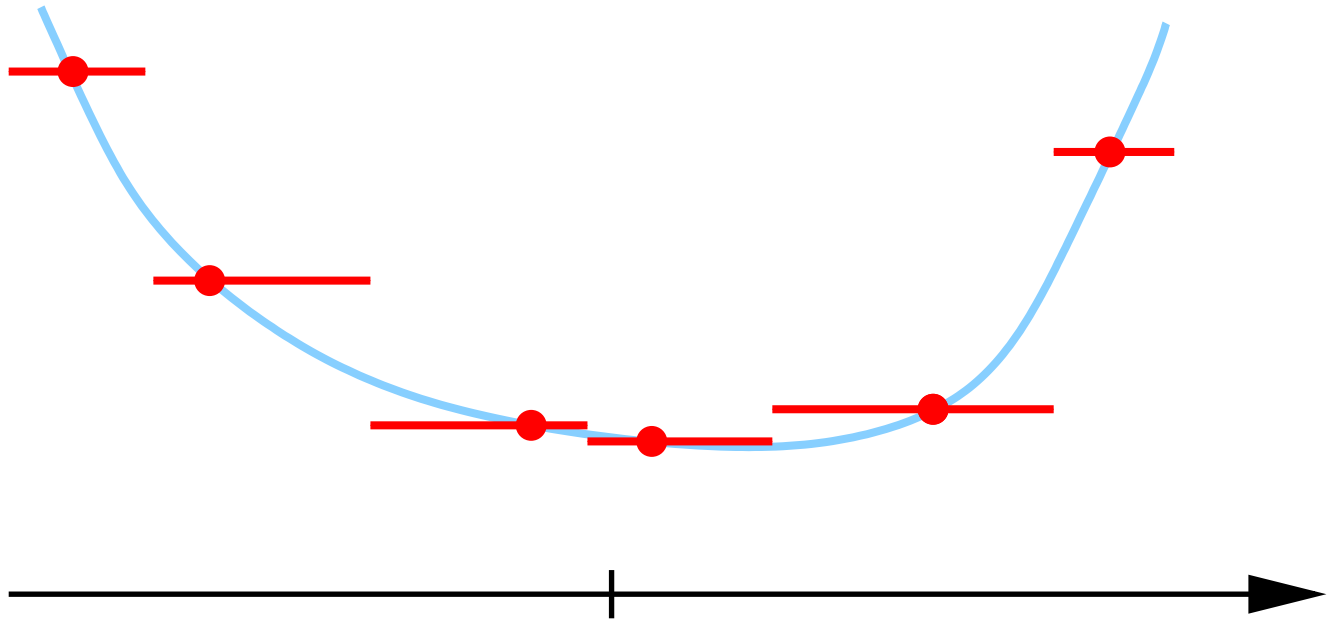
$$|F(x) - F(x')| \leq \epsilon F(x)$$

Compressed Form



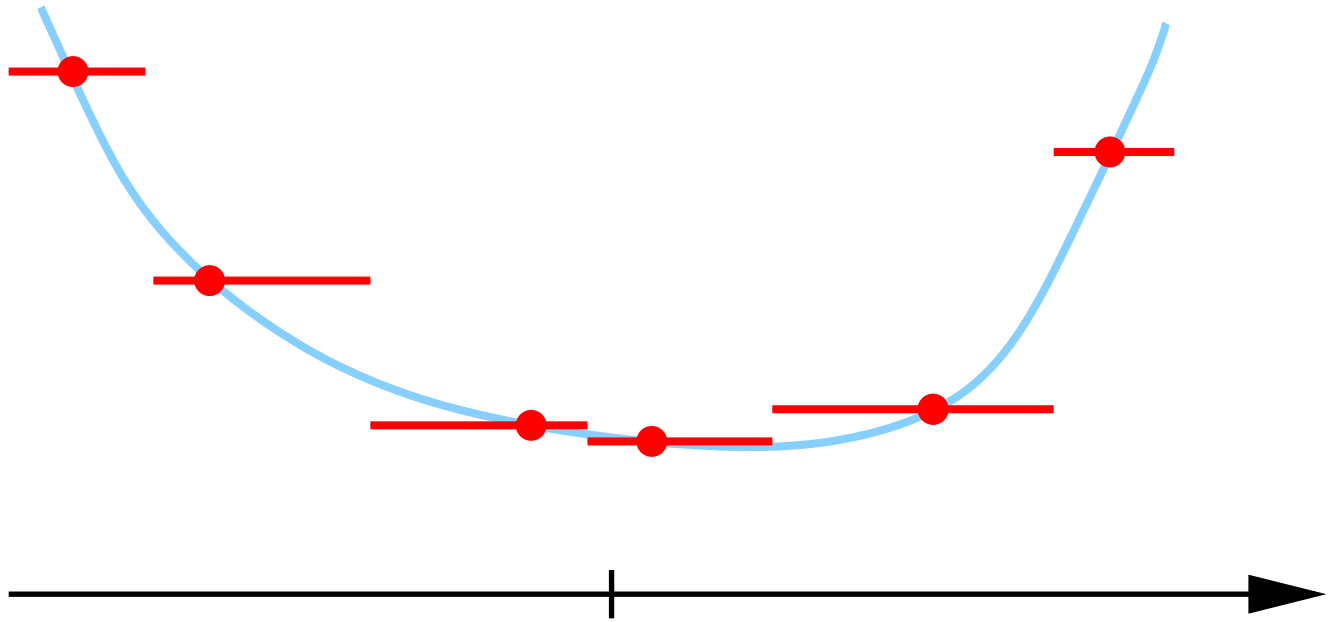
- For $x, x' \in \mathbb{R}^{\tau_i}$ s.t. all $1 - \epsilon^{O(1)} \leq x_i/x'_i \leq 1 + \epsilon^{O(1)}$,
$$|F(x) - F(x')| \leq \epsilon F(x)$$
- Suffices to keep values for coordinates $(1 + \epsilon^{O(1)})^i$

Compressed Form



- For $x, x' \in \mathbb{R}^{\tau_i}$ s.t. all $1 - \epsilon^{O(1)} \leq x_i/x'_i \leq 1 + \epsilon^{O(1)}$,
$$|F(x) - F(x')| \leq \epsilon F(x)$$
- Suffices to keep values for coordinates $(1 + \epsilon^{O(1)})^i$
- Number of values: $(\log n)^{\mathrm{poly}(1/\epsilon)}$

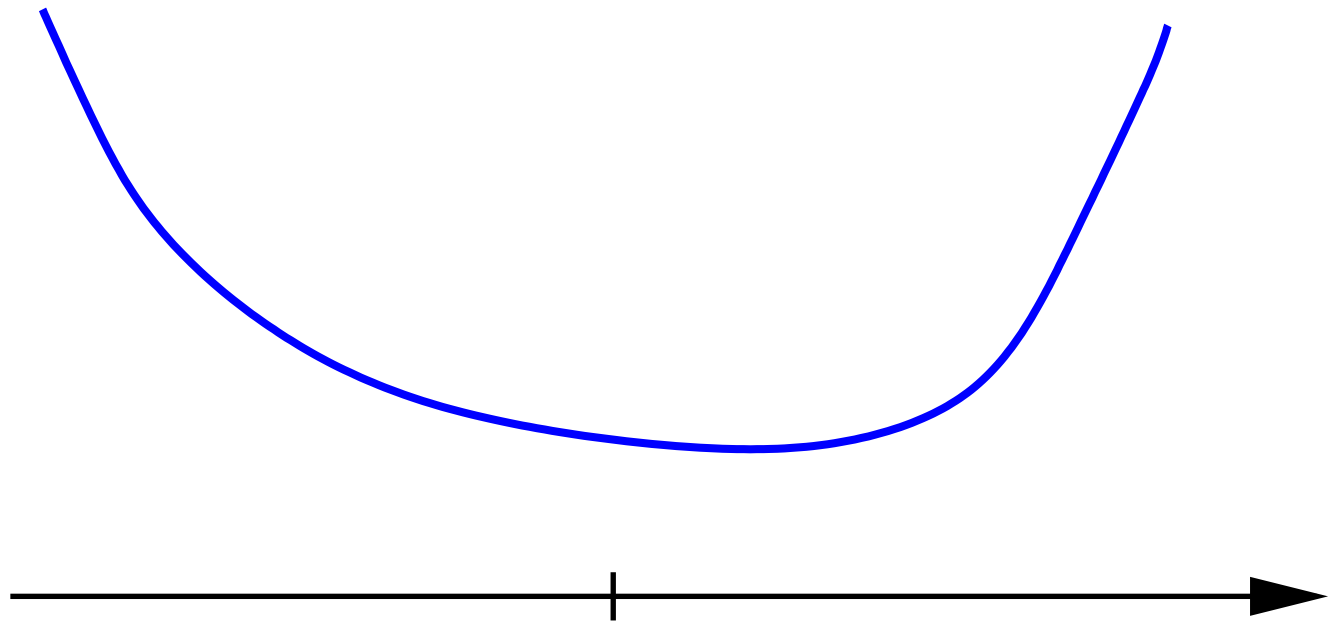
Compressed Form



- For $x, x' \in \mathbb{R}^{\tau_i}$ s.t. all $1 - \epsilon^{O(1)} \leq x_i/x'_i \leq 1 + \epsilon^{O(1)}$,
$$|F(x) - F(x')| \leq \epsilon F(x)$$
- Suffices to keep values for coordinates $(1 + \epsilon^{O(1)})^i$
- Number of values: $(\log n)^{\mathrm{poly}(1/\epsilon)}$
- New challenge: Efficiently combine in recursion

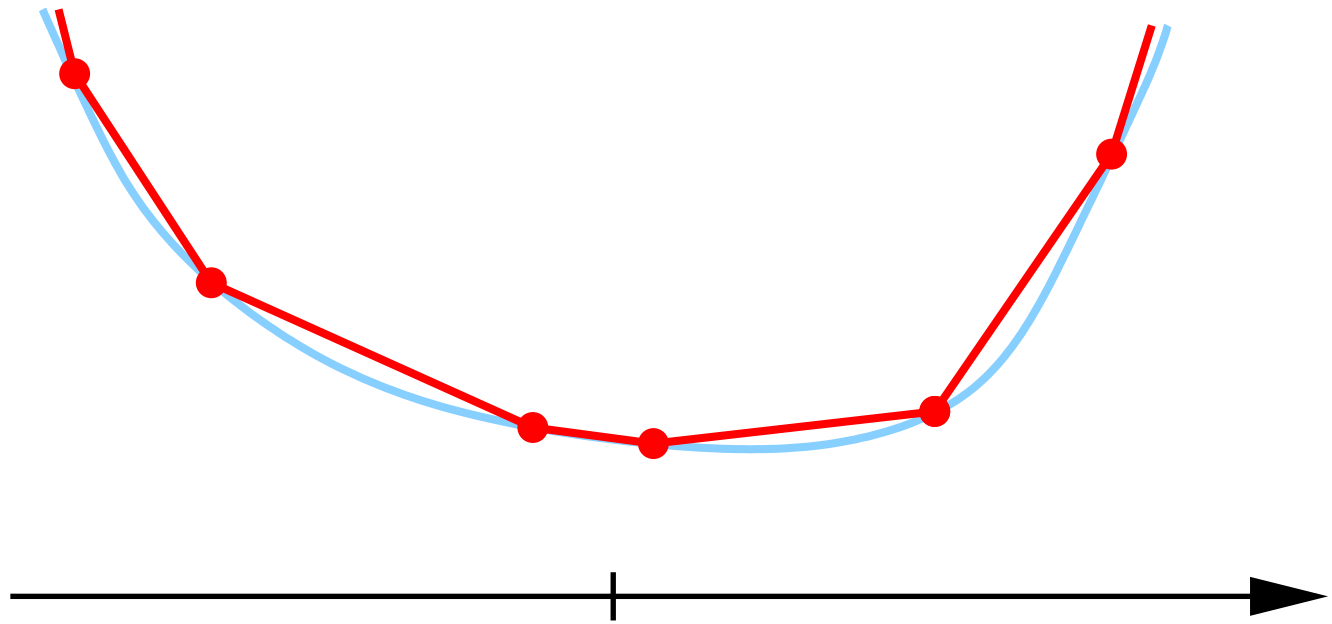
Recursive Step

- Key observation: F is convex!



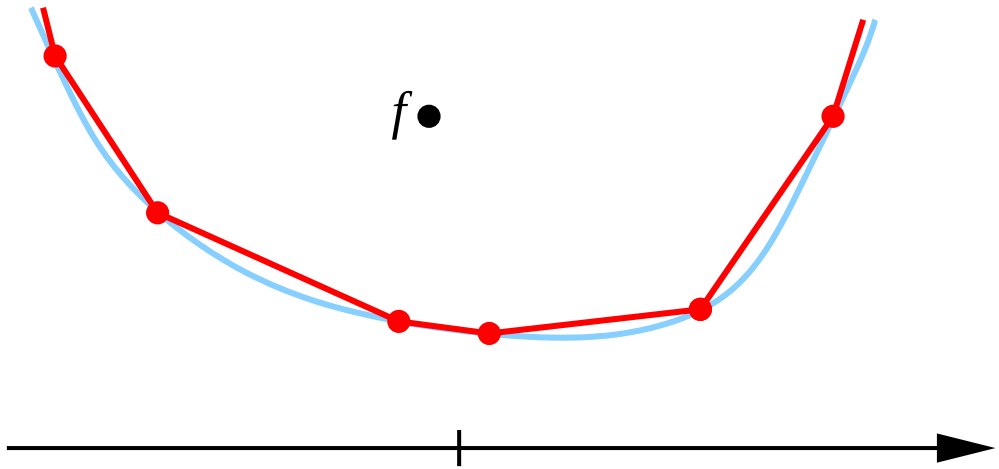
Recursive Step

- Key observation: F is convex!
- Use lower convex hull to approximate cost functions



Recursive Step

- Key observation: F is convex!
- Use lower convex hull to approximate cost functions
- Combining via a large linear system:
  - subfunctions $F_i(y_i)$'s can be rewritten as new variables f_i: $f_i \geq$ convex hull as function of y_i



  - To compute $F(x)$, solve

$$F(x) = \min(\sum f_i + Ax + \sum B_i y_i)$$

    with some linear constraints on x, y_i's and f_i's

Computing a Matching

- This suffices to approximate the cost in constant number of rounds and polynomial work

Computing a Matching

- This suffices to approximate the cost in constant number of rounds and polynomial work
- The approach also works for the transportation problem with different demands/supplies
  - Solution is a fractional matching

Computing a Matching

- This suffices to approximate the cost in constant number of rounds and polynomial work
- The approach also works for the transportation problem with different demands/supplies
  - Solution is a fractional matching
- Can we compute a matching, not just a cost?
  - Have to turn the flow into a (fractional) matching

Computing a Matching

- This suffices to approximate the cost in constant number of rounds and polynomial work
- The approach also works for the transportation problem with different demands/supplies
  - Solution is a fractional matching
- Can we compute a matching, not just a cost?
  - Have to turn the flow into a (fractional) matching
- Multi-round decomposition of flow

Computing a Matching

- This suffices to approximate the cost in constant number of rounds and polynomial work
- The approach also works for the transportation problem with different demands/supplies
  - Solution is a fractional matching
- Can we compute a matching, not just a cost?
  - Have to turn the flow into a (fractional) matching
- Multi-round decomposition of flow
  1. Compute flow for each subcell

Computing a Matching

- This suffices to approximate the cost in constant number of rounds and polynomial work
- The approach also works for the transportation problem with different demands/supplies
  - Solution is a fractional matching
- Can we compute a matching, not just a cost?
  - Have to turn the flow into a (fractional) matching
- Multi-round decomposition of flow
  1. Compute flow for each subcell
  2. Modify the flow go up to plateau point and then down

Computing a Matching

- This suffices to approximate the cost in constant number of rounds and polynomial work
- The approach also works for the transportation problem with different demands/supplies
  - Solution is a fractional matching
- Can we compute a matching, not just a cost?
  - Have to turn the flow into a (fractional) matching
- Multi-round decomposition of flow
  1. Compute flow for each subcell
  2. Modify the flow go up to plateau point and then down
  3. Decompose into flow to/from plateau points

Computing a Matching

- This suffices to approximate the cost in constant number of rounds and polynomial work
- The approach also works for the transportation problem with different demands/supplies
  - Solution is a fractional matching
- Can we compute a matching, not just a cost?
  - Have to turn the flow into a (fractional) matching
- Multi-round decomposition of flow
  1. Compute flow for each subcell
  2. Modify the flow go up to plateau point and then down
  3. Decompose into flow to/from plateau points
  4. Reconnect red and blue points assigned to the same plateau point

Computing a Matching

- This gives a fractional matching of size $n^{1+\delta}$ for small δ

Computing a Matching

- This gives a fractional matching of size $n^{1+\delta}$ for small δ
- Open: Can turn efficiently into a matching of size $O(n)$?

Computing a Matching

- This gives a fractional matching of size $n^{1+\delta}$ for small δ
- Open: Can turn efficiently into a matching of size $O(n)$?
- EMD:
  - We want a large (integral) matching

Computing a Matching

- This gives a fractional matching of size $n^{1+\delta}$ for small δ
- Open: Can turn efficiently into a matching of size $O(n)$?
- EMD:
  - We want a large (integral) matching
  - Always use integral intermediate solutions
  - Potentially super-poly work

Computing a Matching

- This gives a fractional matching of size $n^{1+\delta}$ for small δ
- Open: Can turn efficiently into a matching of size $O(n)$?
- EMD:
  - We want a large (integral) matching
  - Always use integral intermediate solutions
  - Potentially super-poly work

- Note:
  - Corollary: First near-linear time algorithm for the transportation problem
  - Open question in Sharathkumar & Agarwal (2012)

Open Questions

Open Questions

- Can communication complexity help prove lower bounds in this model?

Open Questions

- Can communication complexity help prove lower bounds in this model?
- Superconstant lower bound for Sparse Connectivity?

Open Questions

- Can communication complexity help prove lower bounds in this model?
- Superconstant lower bound for Sparse Connectivity?
- Can our techniques yield an efficient streaming algorithm for EMD?

Questions?