Massively Parallel Tree Embeddings for High Dimensional Spaces and Strong OV Conjecture

Mohammad Hajiaghayi

Wiki & Linkedin: @Mohammad Hajiaghayi
Twitter:@MTHajiaghayi
YouTube:@hajiaghayi [PLEASE SUBSCRIBE]
Instagram:@mhajiaghayi

AmirMohsen Ahanchi, Alexandr Andoni, MohammadTaghi Hajiaghayi, Marina Knittel, and Peilin Zhong (SPAA'23)

Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, Barna Saha (SPAA'24)

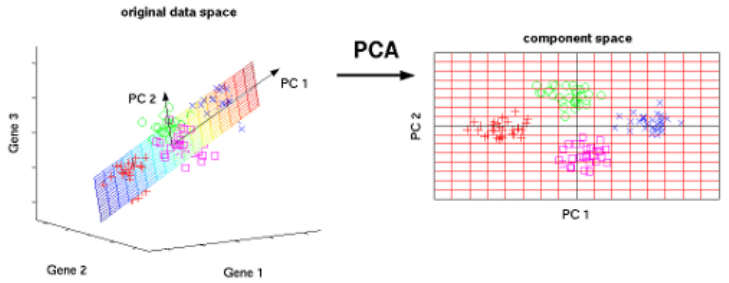
Motivation – Tree Embeddings

Dimensionality

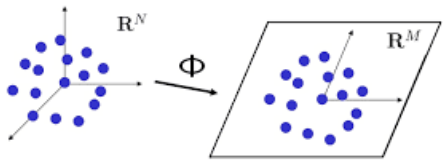
- Large-scale data: Potentially O(n)
- J-L embedding: O(log(n))
- Tree embeddings: O(1)

Problems that leverage distance preservation:

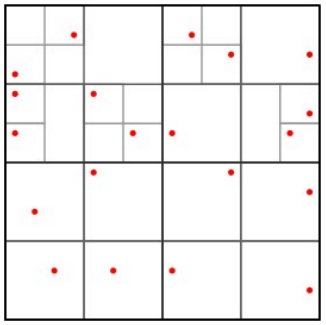
- Traveling Salesperson
- Minimum Spanning Tree
- Earth-Mover Distance
- Densest Ball
- Minimum Cut
- etc.



Principal Component Analysis

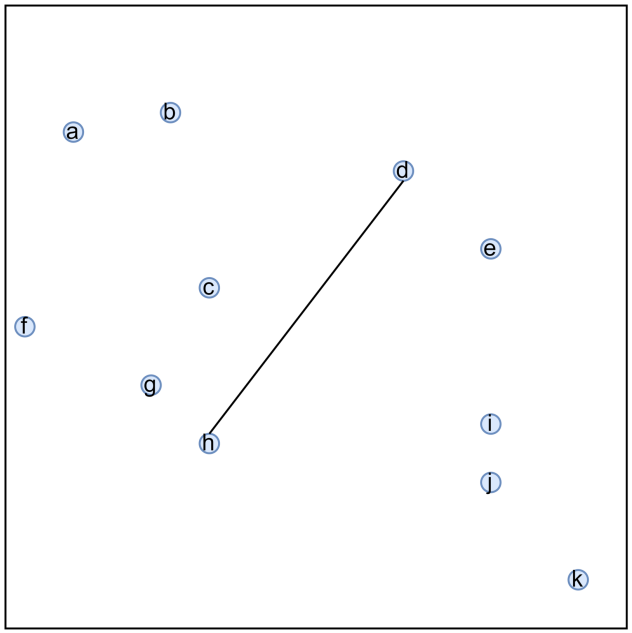


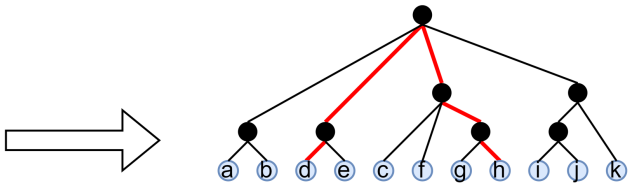
Johnson-Lindenstrauss Transform



Grid Partitioning

Tree Embeddings





We estimate distances:

$$dist(d,h) \approx w(path(d,h)) \qquad \text{(tree metric)}$$

Embedding has α *distortion* if:

$$dist(d,h) \leq w(path(d,h)) \leq \alpha \times dist(d,h)$$

Motivation – MPC in O(1) Rounds

MPC: "Massively Parallel Computation"

- Models the algorithmic framework in MapReduce, Hadoop, Spark, etc.
- Round Complexity: number of times machines must communicate
  - Most MPC algorithms are at most O(log(n)) rounds
  - O(1) is the gold standard, but most algorithms cannot achieve this
- More on this later…

Tree Embedding in MPC

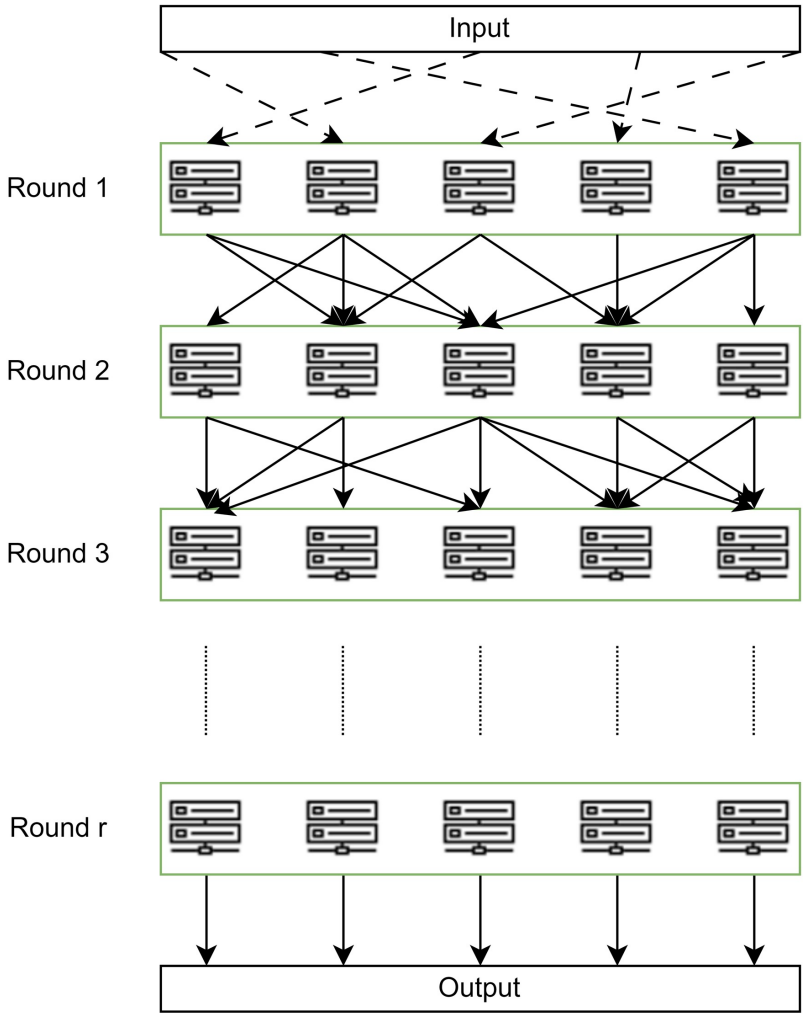
- Arora's Grid Partitioning: O(1) MPC rounds, but $O(\log^2(n))$ distortion
- Andoni et al.'s Ball Partitioning: $O(\log^{1.5}(n))$ distortion, but excessive space requirements

Can we combine these two methods to make a fast, low-memory, low-distortion, MPC tree embedding algorithm?

Massively Parallel Computation

Massively Parallel Computation (MPC) algorithms work as follows:

1. The problem is distributed across small machines.
2. In each round, machines execute local computations.
3. Between rounds, machines may send limited-sized messages.
4. Repeat.
5. In the final round, the solution may be found across machines.

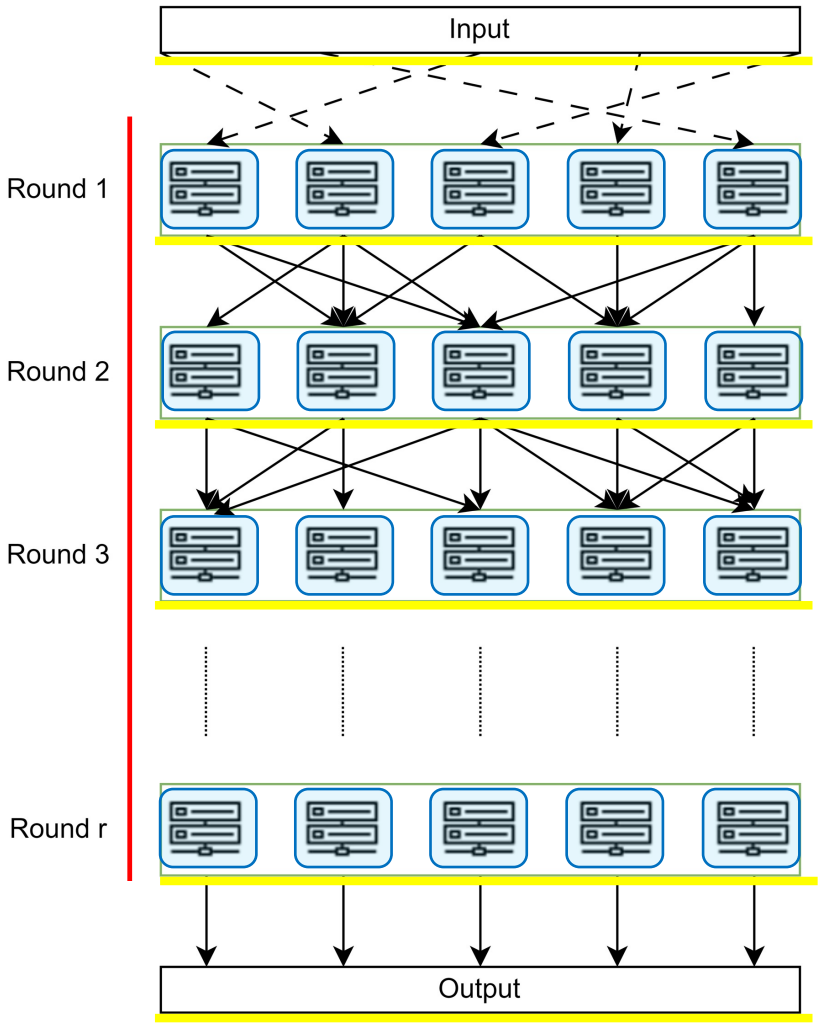


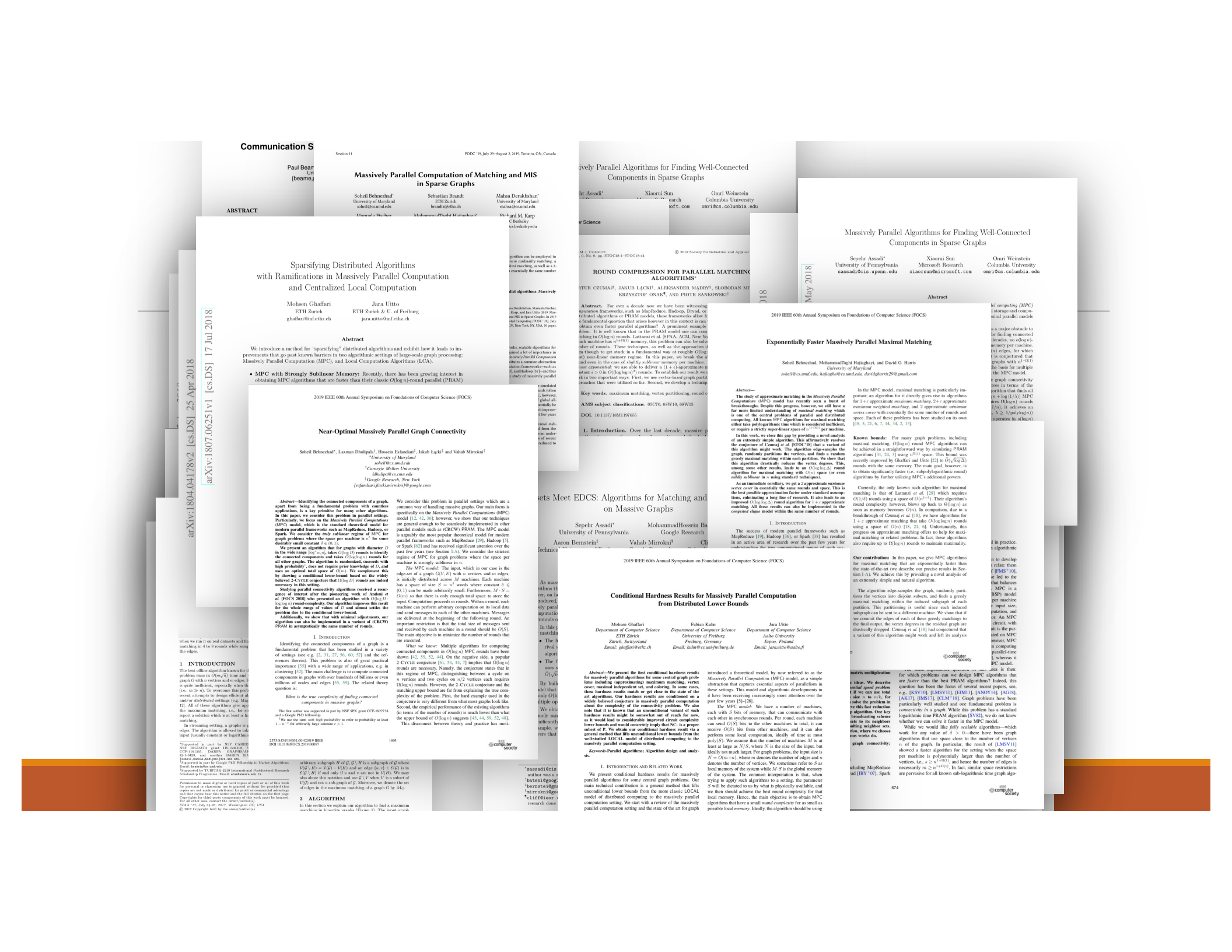
Massively Parallel Computation

Input: a d-dimensional geometric graph encoded as n, d-length vectors.

Complexities

- **Rounds**: The number of rounds to complete, r. We often optimize for this.
- **Memory**: The local space of each machine. This should be $O(n^{\varepsilon})$ for any given $0 < \varepsilon < 1$.
- **Total space**: The total space across all machines and hash tables at any time. This should be O(nd).

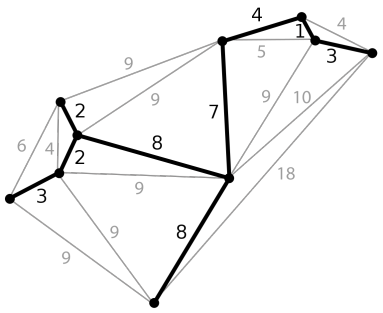



Massively Parallel Computation of Matching and MIS in Sparse Graphs
Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation
Near-Optimal Massively Parallel Graph Connectivity
Round Compression for Parallel Matching Algorithms
Exponentially Faster Massively Parallel Maximal Matching
Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds
Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs
arXiv:1804.04178v2 [cs.DS] 25 Apr 2018
arXiv:1807.06251v1 [cs.DS] 17 Jul 2018

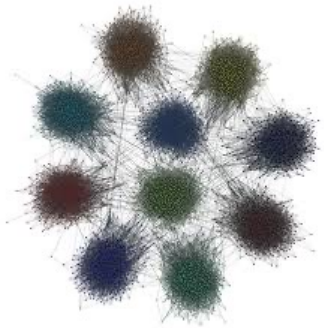
Our Results

Theorem: A set of n points in d-dimensional geometric space can be encoded into a tree embedding with distortion $O(\log^{1.5}(n))$ with probability at least 1-1/poly(n) by an MPC algorithm requiring: $\tilde{O}(nd)$ total space, $O(n^{\varepsilon})$ local space, and O(1) rounds (**a similar theorem for JL as well, but proved much easier and with optimal bounds**).

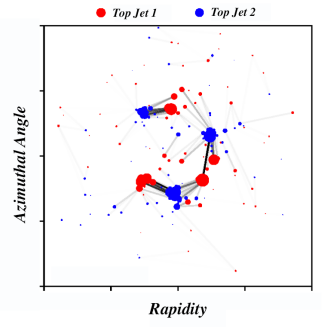
Corollary: Approximate minimum spanning tree, densest ball, and Earth-Mover distance can be solved with probability at least 1-1/poly(n) in MPC within these constraints.



Minimum spanning tree

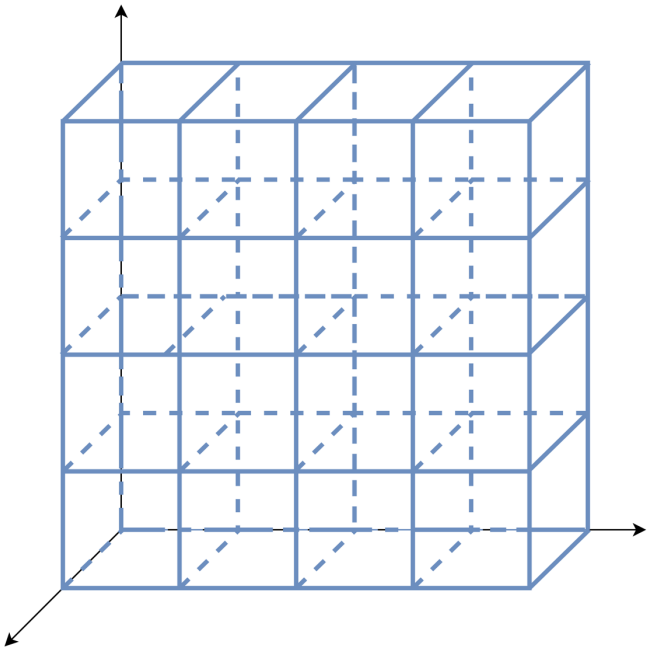


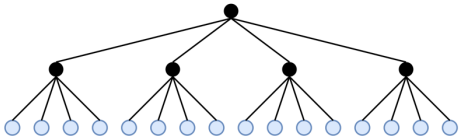
Densest subgraph on social networks



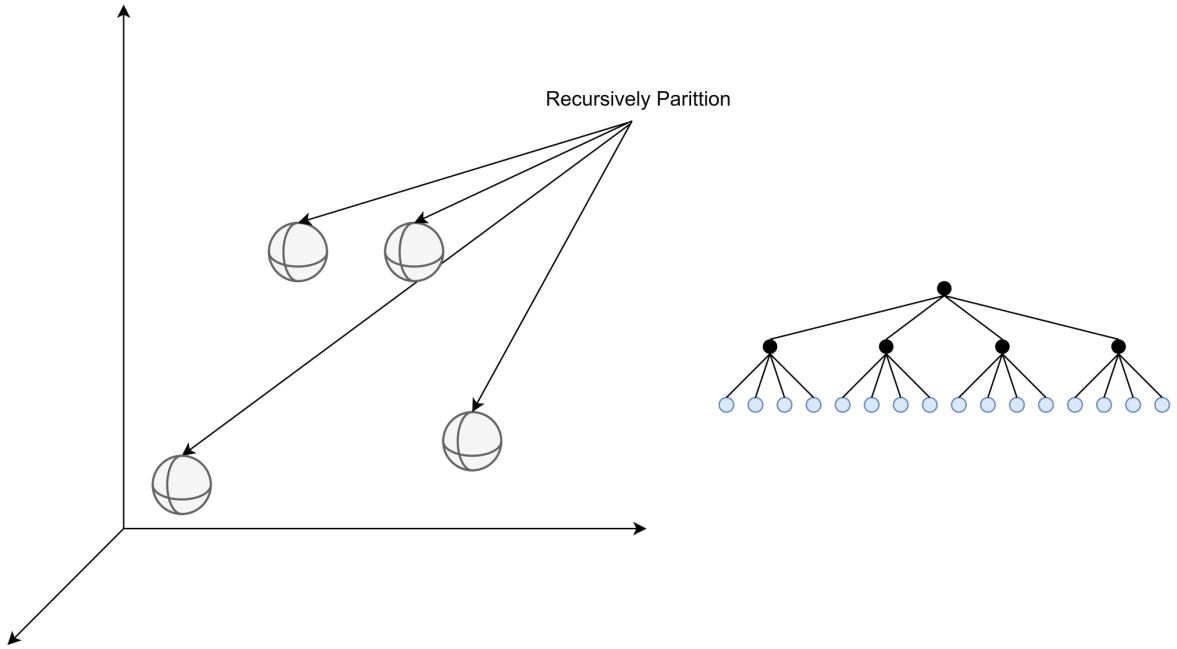
EMD for particle collision modeling

Arora's Grid Partitioning

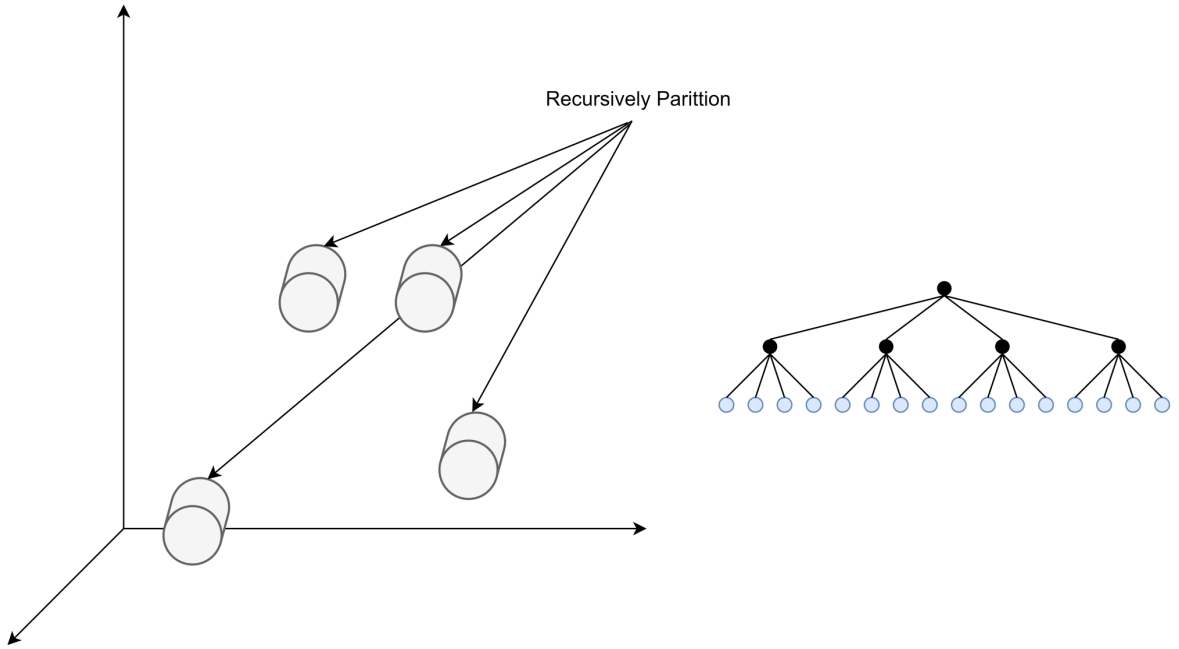




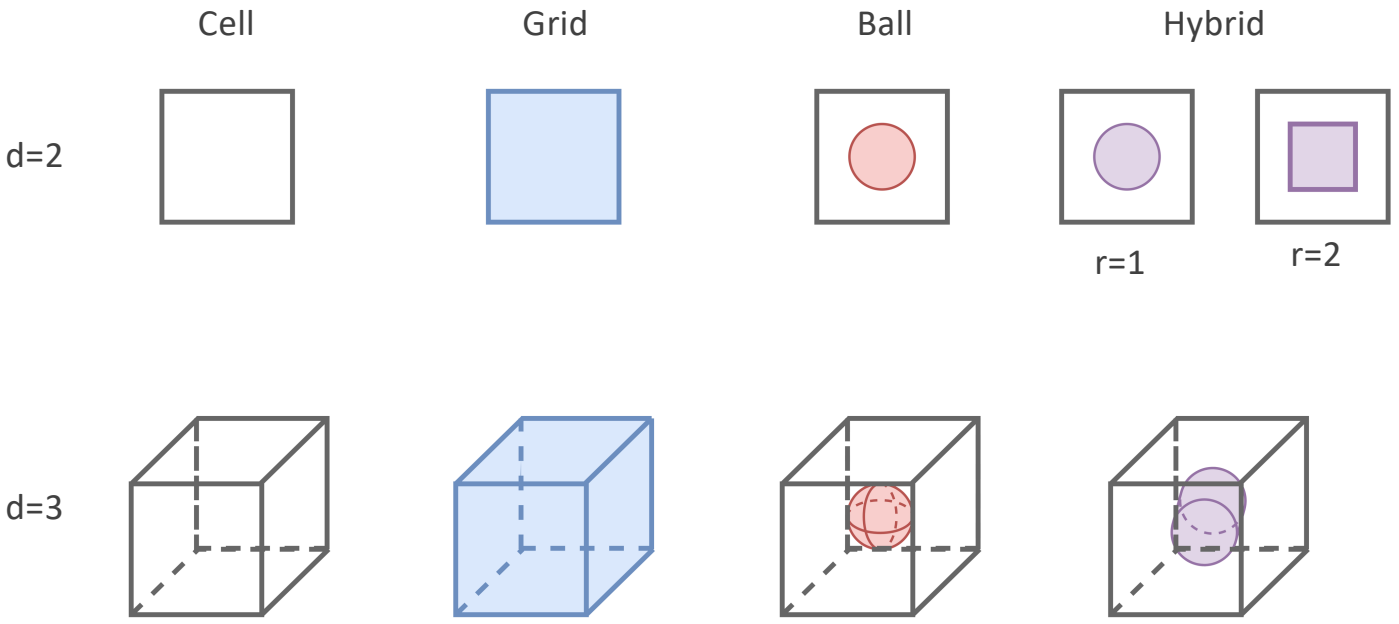
Charikar et al.’s Ball Partitioning



Our Hybrid Partitioning



Filling a Cell – All 3 methods



Bucketing Dimension

Say we are given data with dimensions [1, 2, …, d].
Let r be our number of buckets.
Then, our buckets will be: B = {[1,…,d/r], [d/r+1,…,2d/r], …, [(r-1)d/r, …, d]}

Given a point $x = (x_1, x_2, \ldots x_d)$ project it into each bucket:

$x^{(1)} = (x_1, \ldots x_{d/r})$ = the projection of x into B_1

$x^{(2)} = (x_{d/r+1}, \ldots x_{2d/r})$ = the projection of x into B_2

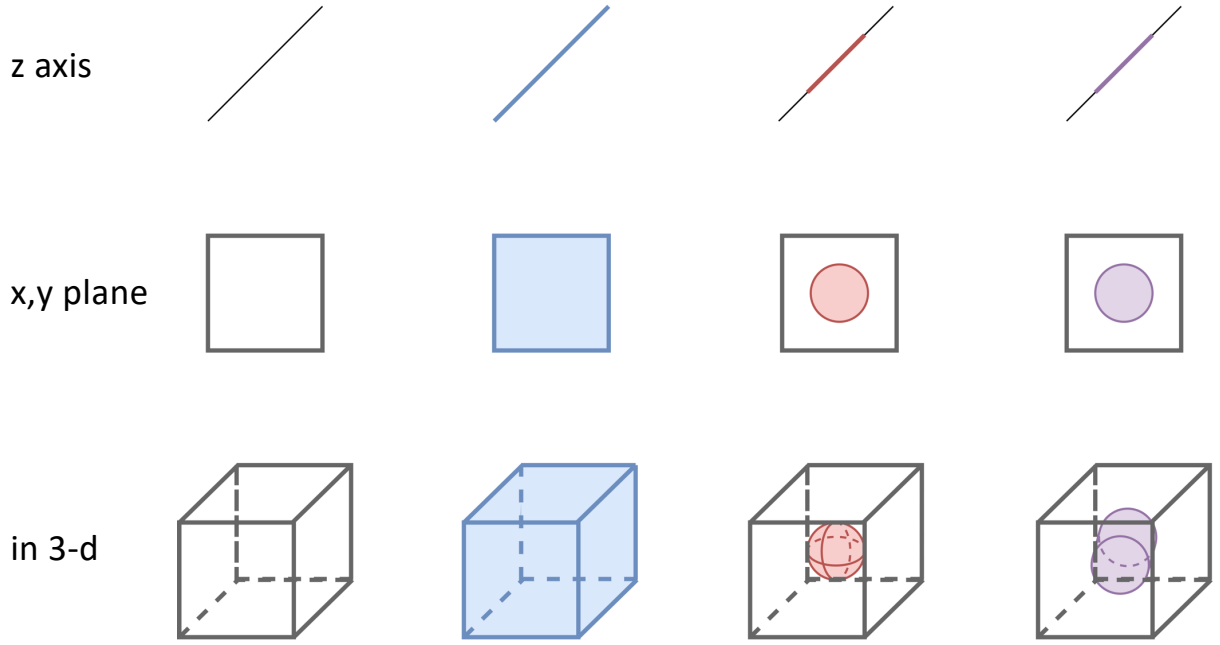
…

$x^{(r)} = (x_{(r-1)d/r}, \ldots x_d)$ = the projection of x into B_r

For each bucket, do a ball partition. If at least two points x, y are in the same ball for *all buckets*, then create a "cluster" with these two points (i.e., a node in the tree – we will recurse on only these points).

Bucketing Dimension

Dimensions: x, y, z, let r = 2. Say we bucket x and y together. This is what the “cluster” shapes will look like.



A Sequential Algorithm

Let the tree root be a “cluster” of all data.

Bucket our dimensions.

For each bucket, run a ball partitioning.

For each maximal set of vertices contained in the same ball for all buckets, create a “cluster” for that set (and a corresponding node in the tree).

Recurse on each child.

Translating to MPC

Embed the data in O(log(n)) dimensions via MPC Fast Johnson-Lindenstrauss **that we provide as byproduct in this paper**

Construct random shifted grids for each run of ball partitioning through the entire algorithm and send a copy to *all* machines

- Proving this can be done in the space requirements is a key point of the paper and is why we cannot run ball partitioning as is

Assign a subset of points to each machine. In each machine:

- Run the hybrid partioning
- For each vertex in the partitioning, construct its path to the root of the tree embedding
- Join these paths to make a tree (a partial embedding)

Take a union of all trees to find the final embedding

Our results are optimized when r = loglog(n).

Summary

Algorithm	Distortion	Rounds	Space
Grid Partitioning	$O(\log^2(n))$	O(1)	OK
Ball Partitioning	$O(\log^{1.5}(n))$	O(1)	Too high
Hybrid Partitioning	**$O(\log^{1.5}(n))$**	**O(1)**	**OK**

MPC Orthogonal Vector Conjecture

Orthogonal Vector Conjecture

- **Orthogonal Vector (OV) Problem** - Given a set of n binary vectors, is there a pair of vectors a, b such that $a \cdot b = 0$.
- **OV Conjecture** - No truly subquadratic time algorithm exists for the Orthogonal Vector problem unless the Strong Exponential Time Hypothesis is false.
  - Well-studied conjecture in fine-grained complexity
  - Provides nearly-quadratic lower bounds for many problems such as Longest Common Substring, Regular Expression Matching, Edit Distance, Largest Common Subtree, Approximating Graph Diameter, and more.

MPC Lower Bounds

Few lower bounds in MPC:

- Recently, Ghaffari, Kuhn, and Uitto (FOCS 2019) showed hardness conditioned on 1 versus 2-Cycle problem for Maximum Matching, Vertex Cover, Maximal Independent Set, and Coloring problems in MPC.
- Roughgarden, Vassilvitskii, and Wang (SPAA 2016) showed that proving a super-logarithmic communication round lower bound would imply $P \neq NC^1$

It would still be useful to build a class of problems with conditional super-logarithmic communication rounds.

MPC Orthogonal Vector Conjecture

MPC Orthogonal Vector Conjecture - For every constant $\varepsilon \in (0, 1)$, there is no MPC algorithm that solves OV using $n^{\varepsilon-\Omega(1)}$ communication rounds, $\Theta(n^{\varepsilon})$ processors, and $n^{1+o(1)}$ total memory.

Intuition:

- Each processor has $n^{1-\varepsilon+o(1)}$ local memory and can compute the dot product of $n^{2-2\varepsilon+o(1)}$ pairs of vectors in a single communication round.
- A direct solution requires checking $O(n^2)$ pairs of vectors, and to compute the dot product necessitates the vectors be co-located on the same processor
- Even if the vectors are constant-length, this does not improve the round complexity of any algorithm that must check $O(n^2)$ pairs.

MPC OV Conjecture Implications

For **any** problem X, if there exists a reduction f from OV to X such that:

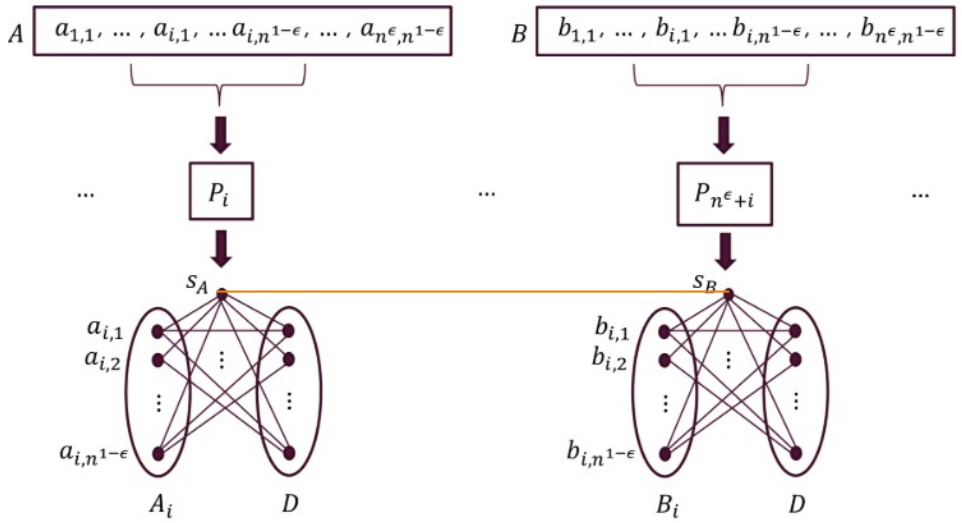
1. Each vector v has gadget size $|f(v)| = O(1)$.
2. Each gadget f(v) can be computed locally in MPC.

then assuming the MPC OV Conjecture, no MPC algorithm can solve X with $n^{\varepsilon-\Omega(1)}$ communication rounds for $\varepsilon \in (0, 1)$, $O(n^{\varepsilon})$ processors, and $n^{1+o(1)}$ total memory.

Most of the previously mentioned problems, e.g., Longest Common Substring, Edit Distance, Approximate Graph Diameter, etc., can be shown to support the above type of reductions.

Example: 1.5-Approximate Graph Diameter

- Each processor P_i is responsible for constructing the vector gadgets for $S = O(n^{1-\varepsilon})$ vectors $a_{i,1}, a_{i,2}, \dots, a_{i,S}$
- After distributing the vectors to processors, no additional communication rounds necessary



l: Example of reduction from O.V. to Graph Diameter problem in the MPC medel.


Thank you