Connectivity and Spanning Forest Problems in MPC

Sam Coy
University of Warwick, UK

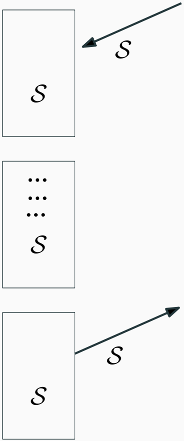
AMG
28th October 2022

In This Talk

- Recent developments in connectivity in sublinear MPC
- Techniques and challenges
- Lower bounds
- What about MST?
- Open problems

MPC: How the Model Works

- $\mathcal{M}$ machines
- Each has $\mathcal{S}$ local storage
- Input initially distributed arbitrarily
- Synchronous rounds:
  - Each machine does arbitrary local computation
  - Machines send messages to each other
- A machine may only send and receive $\mathcal{S}$ words per round



MPC: Local Space

Superlinear	Linear	Sublinear
$\mathcal{S} = O(n^{1+\delta})$	$\mathcal{S} = O(n)$	$\mathcal{S} = O(n^{\delta})$

MPC: Local Space

Superlinear	Linear	Sublinear
$\mathcal{S} = O(n^{1+\delta})$	$\mathcal{S} = O(n)$	$\mathcal{S} = O(n^{\delta})$

MPC: Sublinear Local Space

Let N be the total input size[1]. In $O(1)$ rounds on MPC with sublinear local space we can:

- Sort N values
- Prefix sum of N values
- "Colored summation" (given N values, each with an associated color, sum the values of each color)
- Broadcast values to all machines
- Simulate $O(1)$ rounds of PRAM

[1] When considering graphs, $N = m + n$.

MPC: Global Space

Total space available to the MPC, $\mathcal{T} = \mathcal{S} \times \mathcal{M}$

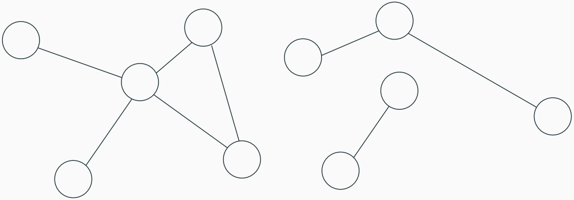
Often assumed that $\mathcal{T} = O(\text{poly}(m+n))$

Must have that $\mathcal{T} = \Omega(m+n)$

Ideally want to get $\mathcal{T} = \Theta(m+n)$, but this is challenging

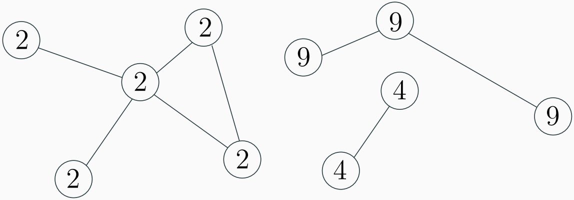
Connectivity

Connectivity



- Compute labelling $\ell : V \to V$ such that:
  - If u and v are in the same connected component then $\ell(u) = \ell(v)$
  - Otherwise, $\ell(u) \neq \ell(v)$
- Same as "picking a representative" for each component
- Fundamental subroutine in graph algorithms

Connectivity

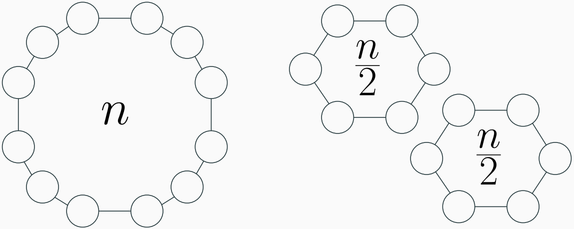


- Compute labelling $\ell : V \to V$ such that:
  - If u and v are in the same connected component then $\ell(u) = \ell(v)$
  - Otherwise, $\ell(u) \neq \ell(v)$
- Same as "picking a representative" for each component
- Fundamental subroutine in graph algorithms

Connectivity in MPC

$\mathcal{S}$	Connectivity	Source
Superlinear: $O(n^{1+\delta})$	$O(1)$	[LMSV '11]
Linear: $O(n)$	$O(1)$ rand.	[JN '18]
	$O(1)$ det.	[Now '21]
Sublinear: $O(n^{\delta})$	$O(\log n)$	PRAM algorithm

The problem seems to be hard when $\mathcal{S} = O(n^{\delta})$...



1-vs-2-Cycles Conjecture

Distinguishing one cycle from two cycles requires $\Omega(\log n)$ rounds in MPC with $\mathcal{S} = O(n^\delta)$.

This conjecture is widely believed!

Sublinear MPC: Lower Bounds

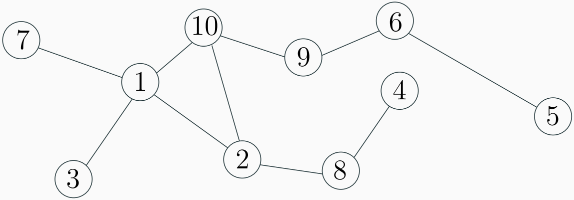
Difficulty of MPC **Lower Bounds [RVW '16] (informal)**

Any non-trivial lower bound in sublinear MPC implies $NC^1 \subsetneq P$

Conditionally Faster Connectivity

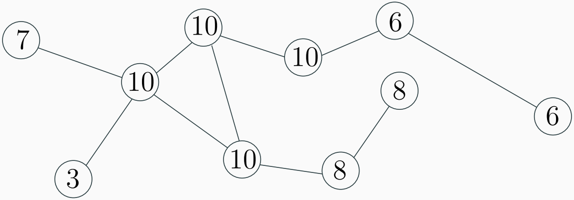
- Maybe long cycles are just difficult?
- Can we do better than $\Omega(\log n)$ *conditionally*?

Conditionally Faster Connectivity



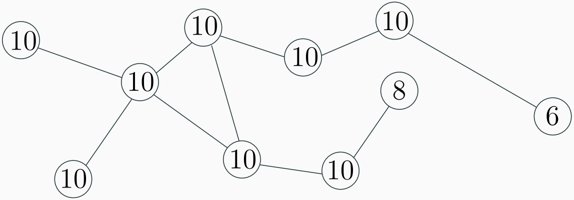
- Maybe long cycles are just difficult?
- Can we do better than $\Omega(\log n)$ *conditionally*?
- Trivial $O(D)$ algorithm where D is diameter. Each phase:
  - Broadcast highest ID you know to all neighbors
  - Check if any edge has different “highest ID”s at endpoints
- Stop when no edge has different IDs at endpoints

Conditionally Faster Connectivity



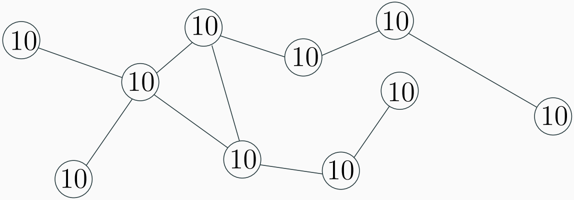
- Maybe long cycles are just difficult?
- Can we do better than $\Omega(\log n)$ *conditionally*?
- Trivial $O(D)$ algorithm where D is diameter. Each phase:
  - Broadcast highest ID you know to all neighbors
  - Check if any edge has different "highest ID"s at endpoints
- Stop when no edge has different IDs at endpoints

Conditionally Faster Connectivity



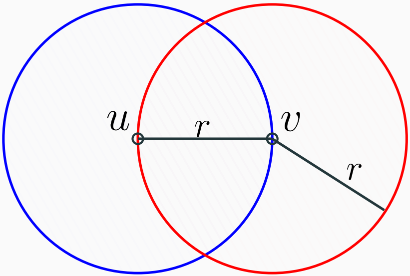
- Maybe long cycles are just difficult?
- Can we do better than $\Omega(\log n)$ *conditionally*?
- Trivial $O(D)$ algorithm where D is diameter. Each phase:
  - Broadcast highest ID you know to all neighbors
  - Check if any edge has different "highest ID"s at endpoints
- Stop when no edge has different IDs at endpoints

Conditionally Faster Connectivity



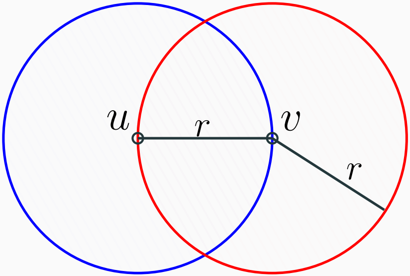
- Maybe long cycles are just difficult?
- Can we do better than $\Omega(\log n)$ *conditionally*?
- Trivial $O(D)$ algorithm where D is diameter. Each phase:
  - Broadcast highest ID you know to all neighbors
  - Check if any edge has different "highest ID"s at endpoints
- Stop when no edge has different IDs at endpoints

Conditionally Faster Connectivity



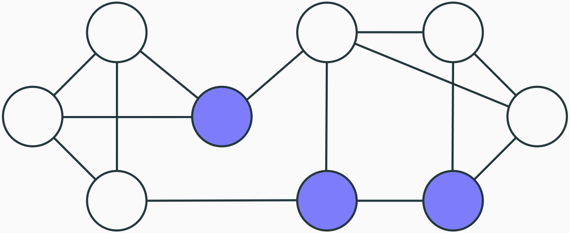
- What about graph exponentiation?
  - Create edges to nodes at distance 2, repeatedly
  - Halves diameter in $O(1)$ rounds; takes $O(\log D)$ rounds in total
  - ...but needs $\mathcal{T} = \Omega(n^{\omega})$
- Can we solve connectivity in $O(\log D)$ rounds when $\mathcal{T} = \Theta(m + n)$?

Connectivity: First Breakthrough



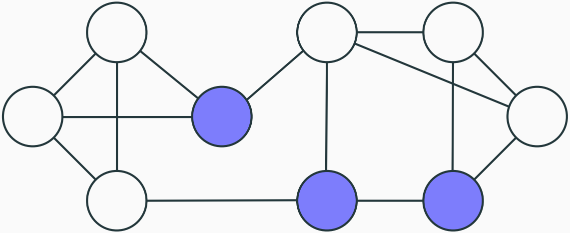
- Idea: graph exponentiation, but stop before we exceed $\mathcal{T}$ [ASSWZ '18]
- Increase the degree of all nodes to $\left[\sqrt{\frac{m}{n}}, \frac{m}{n}\right]$
  - (we stop if we find the whole component)
- Takes $O(\log D)$ rounds

Connectivity: First Breakthrough



- A graph with min-degree d has a dominating set of size $\widetilde{O}(\frac{n}{d})$
- Can easily find it using sampling in $O(1)$ rounds
- Idea: find such a set (of "leaders") and contract non-leaders to them

Connectivity: First Breakthrough



- A graph with min-degree d has a dominating set of size $\widetilde{O}(\frac{n}{d})$
- Can easily find it using sampling in $O(1)$ rounds
- Idea: find such a set (of “leaders”) and contract non-leaders to them
- **Insight: we have much more space now!**

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to $b = \sqrt{\frac{m}{n}}$

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to $b = \sqrt{\frac{m}{n}}$
- Find $O(\frac{n}{b})$ "leaders", contract non-leaders into leaders

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to $b = \sqrt{\frac{m}{n}}$
- Find $O(\frac{n}{b})$ "leaders", contract non-leaders into leaders
- Now have $O(\frac{n}{b})$ vertices, $\Omega(b^2)$ space per vertex

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to b^2
- Find $O(\frac{n}{b})$ "leaders", contract non-leaders into leaders
- Now have $O(\frac{n}{b})$ vertices, $\Omega(b^2)$ space per vertex

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to b^2
- Find $O(\frac{n}{b^3})$ "leaders", contract non-leaders into leaders
- Now have $O(\frac{n}{b})$ vertices, $\Omega(b^2)$ space per vertex

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to b^2
- Find $O(\frac{n}{b^3})$ "leaders", contract non-leaders into leaders
- Now have $O(\frac{n}{b^3})$ vertices, $\Omega(b^4)$ space per vertex

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to b^4
- Find $O(\frac{n}{b^3})$ "leaders", contract non-leaders into leaders
- Now have $O(\frac{n}{b^3})$ vertices, $\Omega(b^4)$ space per vertex

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to b^4
- Find $O(\frac{n}{b^7})$ "leaders", contract non-leaders into leaders
- Now have $O(\frac{n}{b^3})$ vertices, $\Omega(b^4)$ space per vertex

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to b^4
- Find $O(\frac{n}{b^7})$ "leaders", contract non-leaders into leaders
- Now have $O(\frac{n}{b^7})$ vertices, $\Omega(b^8)$ space per vertex

etc...

Connectivity: First Breakthrough

Their algorithm is as follows:

- Increase minimum degree to b^4
- Find $O(\frac{n}{b^7})$ "leaders", contract non-leaders into leaders
- Now have vertices, $\Omega(b^8)$ space per vertex

etc...

Start with $b = \sqrt{\frac{m}{n}}$, make double-exponential progress on b...

$O(\log \log_{m/n} n)$ phases overall!

Connectivity: First Breakthrough

Conditionally Sublogarithmic Connectivity [ASSWZ '18]

Solves connectivity on MPC with $\mathcal{S} = O(n^{\delta})$ and $\mathcal{T} = \Theta(m + n)$ in $O(\log D \cdot \log \log_{m/n} n)$ rounds, with good probability.

They obtain $O(\log D)$ rounds and high success probability if, for some arbitrary constant $\epsilon > 0$, either:

- $\mathcal{T} = \Omega((m + n)^{1+\epsilon})$; or
- $m = \Omega(n^{1+\epsilon})$

Spanning Forest in ASSWZ '18

- Authors of [ASSWZ '18] extended the idea to spanning forest
- Not too difficult, because of the "phase" structure:
- Idea is to (while doing expansion at node v) maintain a "local shortest path" tree rooted at v of all the nodes which v knows. Need to take care:
  - When performing expansion, need to "merge" the local shortest path trees
  - When performing contraction, need to show that the nodes contracted into some leader are a subtree of the local shortest path tree of that leader
  - Preserve information about edges post-contraction

Other Results from [ASSWZ '18]

Using connectivity and spanning forest as black boxes:

- Diameter estimate
  - Gives estimate D' s.t. $D \leq D' \leq D^{O(\log \log_{m/n} n)}$
- MST (we'll talk about this later!)
  - Approximate MST
  - Bottleneck Spanning Tree (BST)

Connectivity: Second Breakthrough

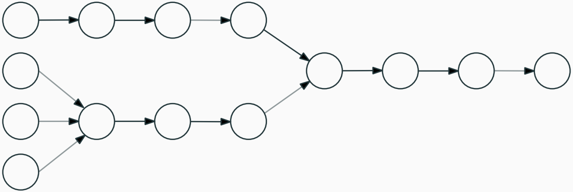
- Previous result has low success probability
- Average degree (b^2) can initially be constant
- For concentration bounds, need $\frac{m}{n} = \Omega(\text{polylog}\, n)$

Connectivity: Second Breakthrough

- Previous result has low success probability
- Average degree (b^2) can initially be constant
- For concentration bounds, need $\frac{m}{n} = \Omega(\text{polylog}\, n)$

- **Idea 1:** Perform random contractions to reduce vertices by a constant factor in $O(1)$ rounds [BDEŁM '19]
  - If repeated $O(\log\log n)$ times, reduces $n \to \frac{n}{\text{polylog}(n)}$
  - Requires a subroutine to find a linear matching on a line

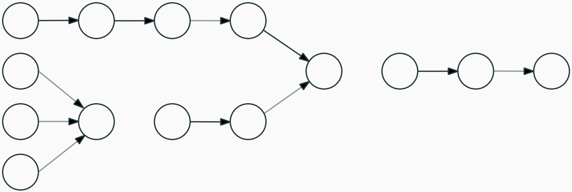
Connectivity: Second Breakthrough



Each vertex selects an outgoing edge to its highest ID neighbor.

- **Idea 1:** Perform random contractions to reduce vertices by a constant factor in $O(1)$ rounds [BDEŁM '19]
  - If repeated $O(\log\log n)$ times, reduces $n \to \frac{n}{\text{polylog}(n)}$
  - Requires a subroutine to find a linear matching on a line

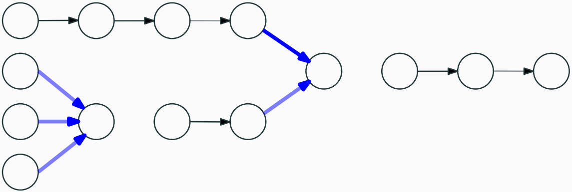
Connectivity: Second Breakthrough



Remove outgoing edges of nodes with in-degree ≥ 2.

- **Idea 1:** Perform random contractions to reduce vertices by a constant factor in $O(1)$ rounds [BDEŁM '19]
  - If repeated $O(\log \log n)$ times, reduces $n \to \frac{n}{\text{polylog}(n)}$
  - Requires a subroutine to find a linear matching on a line

Connectivity: Second Breakthrough



Contract all in-edges of nodes with in-degree ≥ 2.

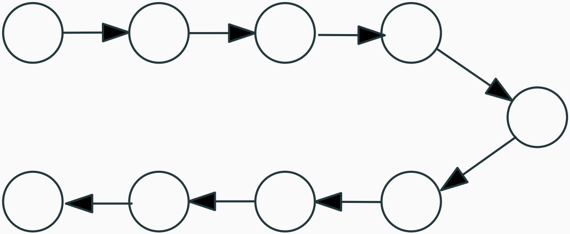
- **Idea 1:** Perform random contractions to reduce vertices by a constant factor in $O(1)$ rounds [BDEŁM '19]
  - If repeated $O(\log \log n)$ times, reduces $n \to \frac{n}{\text{polylog}(n)}$
  - Requires a subroutine to find a linear matching on a line

Connectivity: Second Breakthrough

We're left with a series of directed paths; we need to contract a constant fraction of edges on those paths.

- **Idea 1:** Perform random contractions to reduce vertices by a constant factor in $O(1)$ rounds [BDEŁM '19]
  - If repeated $O(\log \log n)$ times, reduces $n \to \frac{n}{\text{polylog}(n)}$
  - Requires a subroutine to find a linear matching on a line

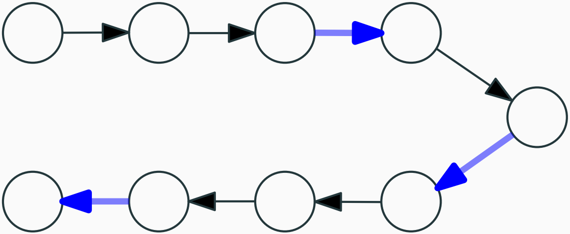
Connectivity: Second Breakthrough



We're left with a series of directed paths; we need to contract a constant fraction of edges on those paths.

- **Idea 1:** Perform random contractions to reduce vertices by a constant factor in $O(1)$ rounds [BDEŁM '19]
  - If repeated $O(\log \log n)$ times, reduces $n \to \frac{n}{\text{polylog}(n)}$
  - Requires a subroutine to find a linear matching on a line

Connectivity: Second Breakthrough



We use coin-tossing to find a linear-size matching on this path, and contract these edges.

- **Idea 1:** Perform random contractions to reduce vertices by a constant factor in $O(1)$ rounds [BDEŁM '19]
  - If repeated $O(\log \log n)$ times, reduces $n \to \frac{n}{\text{polylog}(n)}$
  - Requires a subroutine to find a linear matching on a line

Connectivity: Second Breakthrough

- **Idea 2:** Can improve the running time by interleaving expansion/contraction per-vertex [BDEŁM '19]
- Each vertex has a *level* which controls its space budget
- Guarantee that after $O(1)$ rounds, each node either:
  - Learns its 2-hop neighborhood (expansion), or;
  - Participates in leader selection with other nodes (contraction), the leaders' levels are increased by 1
- Since maximum level is $O(\log \log_{m/n} n)$, this gives a $O(\log D + \log \log_{m/n})$-round algorithm

Connectivity: Second Breakthrough

Faster Connectivity [BDEŁM '19]

Connectivity can be solved on MPC with $\mathcal{S} = O(n^\delta)$ and $\mathcal{T} = \Theta(m + n)$ in $O(\log D + \log\log_{m/n} n)$ rounds, **with high probability**.

Again, significant graph density or significantly superlinear global space give an $O(\log D)$-round algorithm.

Connectivity: Second Breakthrough

Faster Connectivity [BDEŁM '19]

Connectivity can be solved on MPC with $\mathcal{S} = O(n^{\delta})$ and $\mathcal{T} = \Theta(m+n)$ in $O(\log D + \log\log_{m/n} n)$ rounds, **with high probability**.

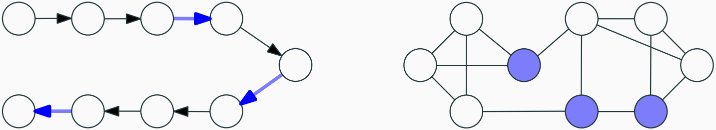
Again, significant graph density or significantly superlinear global space give an $O(\log D)$-round algorithm.

Extension to spanning forest seems much harder here, because of the decoupling of the expansion/contraction process ...

Deterministic Connectivity

Randomization

Two randomized subroutines:



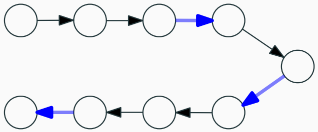
(a) Finding a constant-fraction sized matching on a path

(b) Finding a dominating set of size $O(n/b)$ in a graph with min. degree b

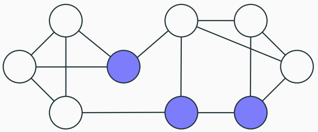
Can we derandomise them?

Randomization

Two randomized subroutines:



(a) Finding a constant-fraction sized matching on a path



(b) Finding a dominating set of size $O(n/b)$ in a graph with min. degree b

Can we derandomise them?

[CMT '21] showed that the randomness can be reduced *slightly*: $(\log n)^{O(\log D + \log\log_{m/n} n)}$ bits suffice if D is not too large.

Method of Conditional Expectations

- Method for derandomizing algorithms
- Idea: “fix the seed” of randomized algorithms
- If a set of seeds meet an objective in expectation when you pick one uniformly, at least one seed meets the objective

k-wise Independence

k-wise Independence

Let $k, n, \ell \in \mathbb{N}$ with $k \leq n$. A family of hash functions $\mathcal{H} = \{h : \{1, \ldots, n\} \to \{0,1\}^\ell\}$ is called *k-wise independent* if for all $I \subseteq \{1, \ldots, n\}$ with $|I| \leq k$, the random variables $h(i)$ with $i \in I$ are independent and uniformly distributed in $\{0,1\}^\ell$ when h is chosen randomly from $\mathcal{H}$.

Small Families of k-wise independent Hash Functions

For every $n, \ell \in \mathbb{N}$, one can construct a family of pairwise independent hash functions $\mathcal{H} = \{h : \{1, \ldots, n\} \to \{0,1\}^\ell\}$ such that choosing a uniformly random function h from $\mathcal{H}$ takes $O(\ell + \log n)$ random bits.

Method of Conditional Expectations

- Given an algorithm which solves the target problem using k-wise independent random variables:
- Construct a family of k-wise approximate hash functions
  - Each function can be specified with $O(\log n)$ bits
- Define some objective function h and argue that in expectation its value is sufficient to solve the problem

Method of Conditional Expectations

- Given an algorithm which solves the target problem using k-wise independent random variables:
- Construct a family of k-wise approximate hash functions
  - Each function can be specified with $O(\log n)$ bits
- Define some objective function h and argue that in expectation its value is sufficient to solve the problem

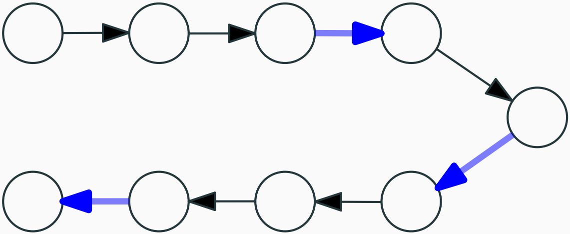
- Fix the seed $\log n$ bits at a time
- Iteratively set prefix until entire seed is specified
- $O(\log n)$ length seeds, fix $\log n$ bits at a time: $O(1)$ stages

Method of Conditional Expectations in MPC

$$b_1, b_2 \dots b_k \underbrace{b_{k+1} \dots b_{k+\log n}}_{\beta} \dots b_{O(\log n)}$$

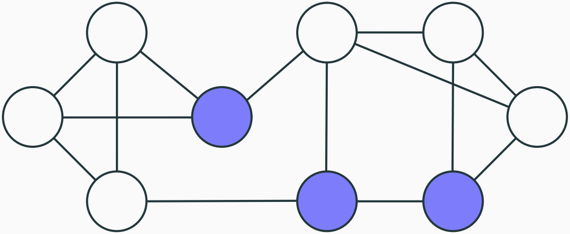
- We've fixed the first k bits, $b_1 \dots b_k$
- Consider a possible setting β of the next $\log n$ bits ($b_{k+1} \dots b_{k+\log n}$)
- Machines compute the expected value of h at the hash functions prefixed by $b_1 \dots b_k \cdot \beta$
- Aggregate using colored summation
- Pick the best possible β; broadcast result to all machines

k-wise, ϵ-approximate Hash Functions



- For finding a large matching on a line, pairwise (2-wise) independence is enough

k-wise, ϵ-approximate Hash Functions



- For the dominating set problem, wanted $O(\log n)$-wise independence
  - ...but this family of hash functions is too large!
- Solution: k-wise, ϵ-*approximately* independence
- Weaker tail bounds, but hash functions can be specified by $O(\log n)$ bits again!

Deterministic Connectivity

Deterministic Connectivity [CC '21]

Connectivity can be solved on MPC with $\mathcal{S} = O(n^\delta)$ and $\mathcal{T} = \Theta(m + n)$ in $O(\log D + \log\log_{m/n} n)$ rounds, **deterministically**.

As before, superlinear global space or some polynomial density gives an $O(\log D)$-round algorithm.

Improved Deterministic Connectivity

- Method of conditional expectations requires locally evaluating poly(n) seeds
- We often don't care about "local work" in MPC
- But local work bounded by poly(n), when n could be billions...

Improved Deterministic Connectivity

- Method of conditional expectations requires locally evaluating poly(n) seeds
- We often don't care about "local work" in MPC
- But local work bounded by poly(n), when n could be billions...

- Algorithm of [FGG '22] reduces total work by reducing the number of seeds which need to be searched through.
- Number of seeds so low that they can be brute-forced.
- Idea: color the graph first!
  - Nodes of the same color make the same choice
  - Reduce domain of hash functions from $|V|$ to $|C|$
- Also analysis of dominating set using only pairwise independence

Improved Deterministic Connectivity

Deterministic Connectivity [FGG '22]

Connectivity can be solved on MPC with $\mathcal{S} = O(n^{\delta})$ and $\mathcal{T} = \Theta(m + n)$ in $O(\log D + \log\log_{m/n} n)$ rounds, deterministically, **with $\widetilde{O}(m)$ local computation in total**.

Connectivity in Forests

Brand new result! (Will appear at SODA '23)

Connectivity in Forests [BLMOU '23]

Connectivity can be solved on MPC with $\mathcal{S} = O(n^\delta)$ and $\mathcal{T} = \Theta(m + n)$ in $O(\log D)$ rounds, deterministically, when the input graph is a forest.

- Achieve this by rooting the tree
- "Balanced exponentiation" approach
- Can perform $O(\log D)$ rounds of vertex contraction "for free" to get $O(\text{poly}(D))$ factor extra space.

Connectivity: State of the Art

$$D = \Omega(\mathrm{polylog}(n)) \implies O(\log D)$$

$$\mathcal{T} = \Omega((m+n)^{1+c}) \implies O(\log D)$$

$$m = \Omega(n^{1+c}) \implies O(\log D)$$

$$G \text{ is a forest} \implies O(\log D)$$

$$\text{otherwise} \implies O(D)$$

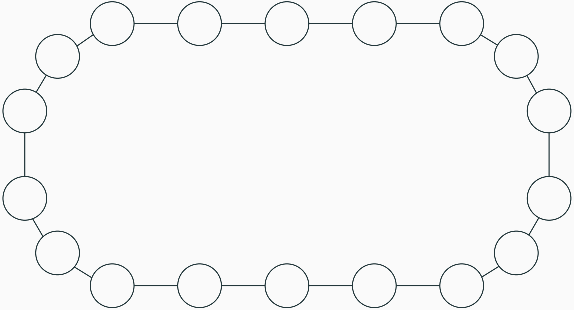
Connectivity: Lower Bounds

Connectivity: Lower Bounds

Conditional Lower Bound [BDEŁM '19]

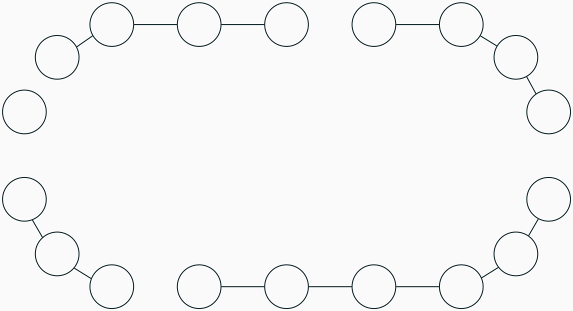
Connectivity on MPC with $\mathcal{S} = O(n^\delta)$ requires $\Omega(\log D)$ rounds, where $D \geq \log^{1+\rho} n$ is the diameter of the graph, unless the 1-vs-2-cycles conjecture is false.

Connectivity: Lower Bounds



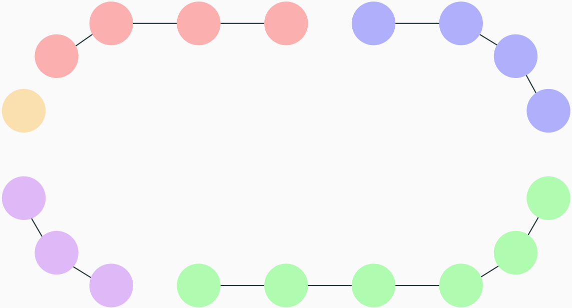
Start with an instance of 1-v-2 cycles

Connectivity: Lower Bounds

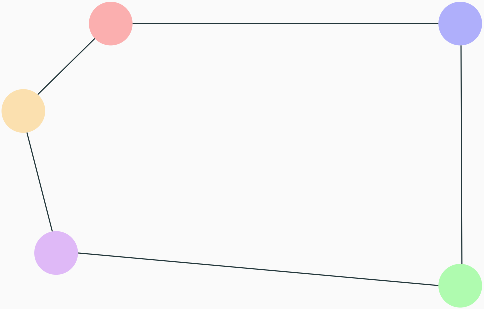


Temporarily remove edges with probability $O(\log n/D')$, breaking cycle into paths

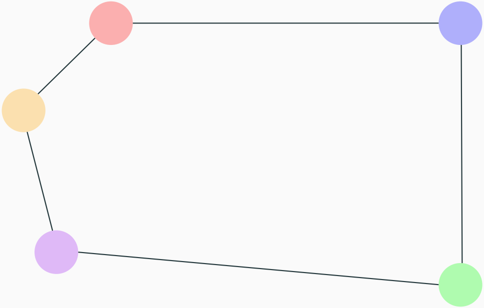
Connectivity: Lower Bounds



Use fast algorithm to find connected components in $o(\log D')$ rounds



Contract components and re-add edges; repeat



Takes $o(\log n)$ rounds overall, which contradicts 1-v-2 cycle

Connectivity: Lower Bounds

Conditional Lower Bound [CC '22]

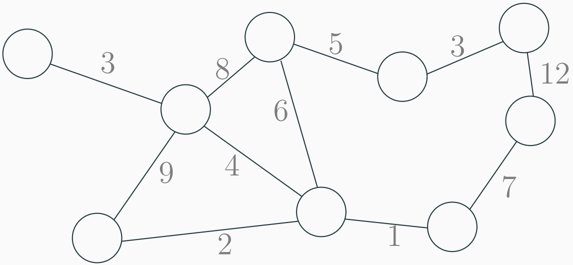
Connectivity on MPC with $\mathcal{S} = O(n^{\delta})$ requires $\Omega(\log D)$ rounds, where D is the diameter of the graph, unless the 1-vs-2-cycles conjecture is false.

The bound holds for any value of D.

- Similar argument to [BDEŁM '19]
- Remove dependence on n in the sampling probability
- No longer need to worry so much about bounding the diameter of all components: just ignore high-diameter components!

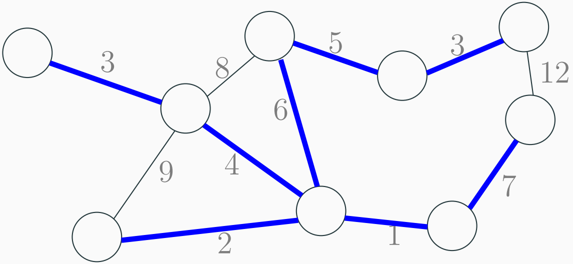
MST

MST



- Given a weighted graph as input, compute a minimum spanning forest
- Clearly no easier than connectivity; a minimum spanning forest *is a spanning forest*
- But is it *harder* than connectivity?

MST



- Given a weighted graph as input, compute a minimum spanning forest
- Clearly no easier than connectivity; a minimum spanning forest *is a spanning forest*
- But is it *harder* than connectivity?

MST in MPC

$\mathcal{S}$	MST	Source
Superlinear: $O(n^{1+\delta})$	$O(1)$	[LMSV '11]
Linear: $O(n)$	$O(1)$ rand.	[JN '18]
	$O(1)$ det.	[Now '21]
Sublinear: $O(n^{\delta})$	$O(\log n)$	PRAM algorithm

MST in MPC

$\mathcal{S}$	MST	Source
Superlinear: $O(n^{1+\delta})$	$O(1)$	[LMSV '11]
Linear: $O(n)$	$O(1)$ rand.	[JN '18]
	$O(1)$ det.	[Now '21]
Sublinear: $O(n^{\delta})$	$O(\log n)$	PRAM algorithm

This is the same table as before!

MST in Sublinear MPC

Algorithm for MST given in [ASSWZ '18]; runs in[2]:

$$O\left(\min\left\{\left(\log D_{MST} + \log\left(\frac{\log n}{2+\gamma\log n}\right)\right)\cdot\frac{\log n}{2+\gamma\log n}, \log n\right\}\right)$$

with $\mathcal{T} = O((m+n)^{1+\gamma})$.

[2] This is better than the bound in [ASSWZ '18]: it incorporates the improved connectivity algorithm of [BDEŁM '19]

MST in Sublinear MPC

Algorithm for MST given in [ASSWZ '18]; runs in[2]:

$$O\left(\min\left\{\left(\log D_{MST} + \log\left(\frac{\log n}{2+\gamma\log n}\right)\right)\cdot\frac{\log n}{2+\gamma\log n}, \log n\right\}\right)$$

with $\mathcal{T} = O((m+n)^{1+\gamma})$. Simplifying a bit:

$$\gamma = 0 \implies O(\log n) \text{ rounds}$$

$$\gamma > c \implies O(\log D_{MST}) \text{ rounds}$$

[2] This is better than the bound in [ASSWZ '18]: it incorporates the improved connectivity algorithm of [BDEŁM '19]

MST in Sublinear MPC

Algorithm for MST given in [ASSWZ '18]; runs in[2]:

$$O\left(\min\left\{\left(\log D_{MST} + \log\left(\frac{\log n}{2+\gamma\log n}\right)\right)\cdot\frac{\log n}{2+\gamma\log n}, \log n\right\}\right)$$

with $\mathcal{T} = O((m+n)^{1+\gamma})$. Simplifying a bit:

$$\gamma = 0 \Longrightarrow O(\log n) \text{ rounds}$$

$$\gamma > c \Longrightarrow O(\log D_{MST}) \text{ rounds}$$

This result was derandomized by [**C**C '22]

---

[2] This is better than the bound in [ASSWZ '18]: it incorporates the improved connectivity algorithm of [BDEŁM '19]

MST in Sublinear MPC

- Algorithm of [ASSWZ '18] use repeated applications of connectivity
- Given an instance of MST with m edges, n vertices, and a factor of k extra space, create sub-instances. For instance $i \in [1, k]$:
  - Contract lightest $(i-1) \cdot m/k$ edges *using a connectivity algorithm*
  - Discard edges heavier than the (im/k)th lightest
  - Solve MSF on the remaining graph (recursively)

MST in Sublinear MPC

- Algorithm of [ASSWZ '18] use repeated applications of connectivity
- Given an instance of MST with m edges, n vertices, and a factor of k extra space, create sub-instances. For instance $i \in [1, k]$:
    - Contract lightest $(i-1) \cdot m/k$ edges *using a connectivity algorithm*
    - Discard edges heavier than the (im/k)th lightest
    - Solve MSF on the remaining graph (recursively)
- If $\gamma = 0$, the recursion depth is dominant;
  If $\gamma > c$, the connectivity complexity dominates
- Dependence on D_{MST} rather than D because we contract edges in size order

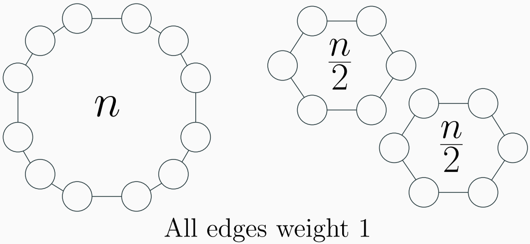
MST in $O(\log D)$ rounds?

MST in $O(\log D)$ rounds?

No.

MST: Lower Bound

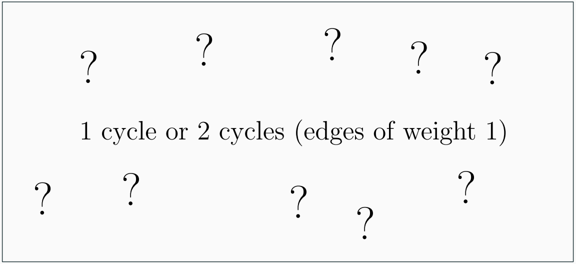
MST can't be calculated in $o(\log n)$ rounds, parameterized on D, n [**C**C '22]:



We start with an instance of either 1 cycle or two cycles: all edges have weight 1

MST: Lower Bound

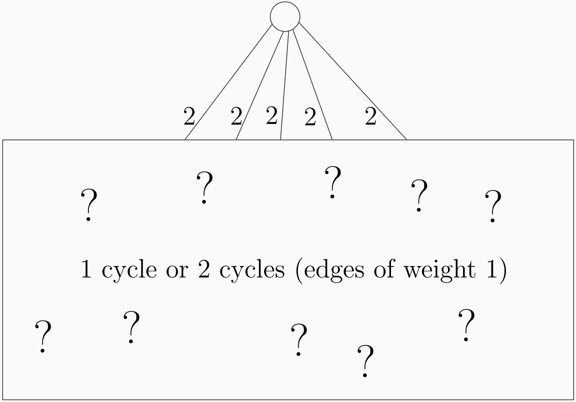
MST can't be calculated in $o(\log n)$ rounds, parameterized on D, n [**C**C '22]:



We start with an instance of either 1 cycle or two cycles: all edges have weight 1

MST: Lower Bound

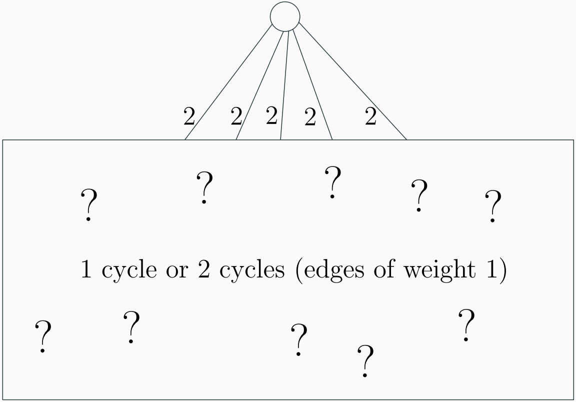
MST can't be calculated in $o(\log n)$ rounds, parameterized on D, n [**C**C '22]:



We add a universal vertex with edges to all existing nodes; these edges have weight 2.

MST: Lower Bound

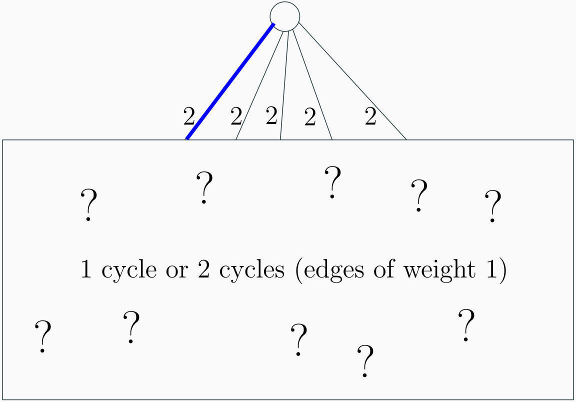
MST can't be calculated in $o(\log n)$ rounds, parameterized on D, n [**C**C '22]:



We then run our MST algorithm.

MST: Lower Bound

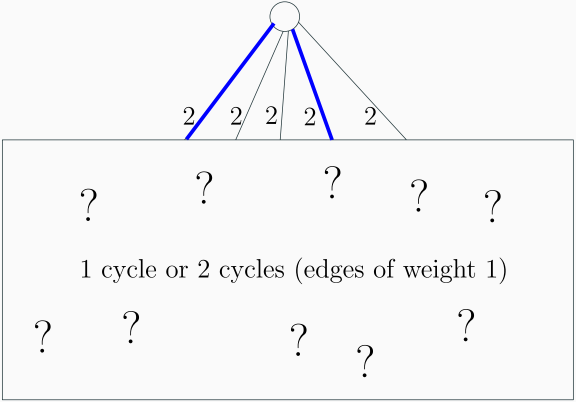
MST can't be calculated in $o(\log n)$ rounds, parameterized on D, n [**C**C '22]:



If one weight-2 edge is used, then we have one cycle...

MST: Lower Bound

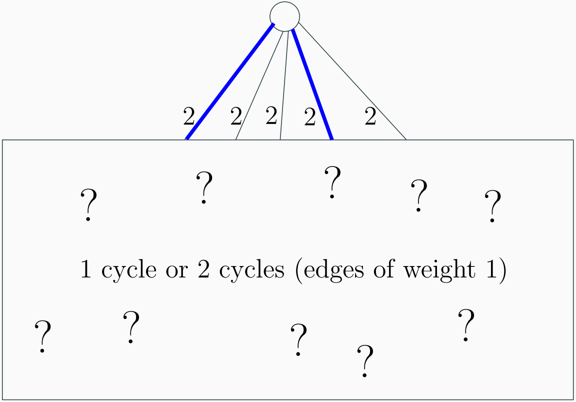
MST can't be calculated in $o(\log n)$ rounds, parameterized on D, n [**C**C '22]:



If two are used, we have 2 cycles!

MST: Lower Bound

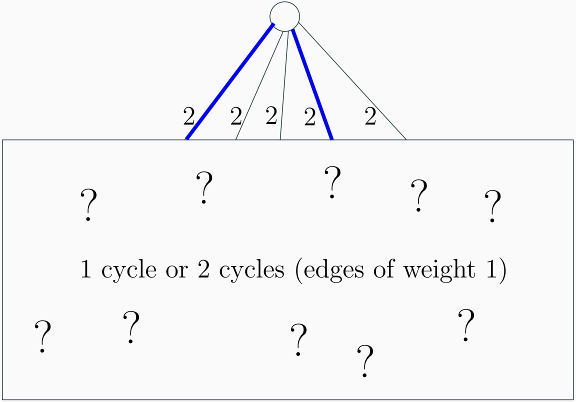
MST can't be calculated in $o(\log n)$ rounds, parameterized on D, n [**C**C '22]:



Note that $D = 2$ but $D_{MST} = O(n)$.

MST: Lower Bound

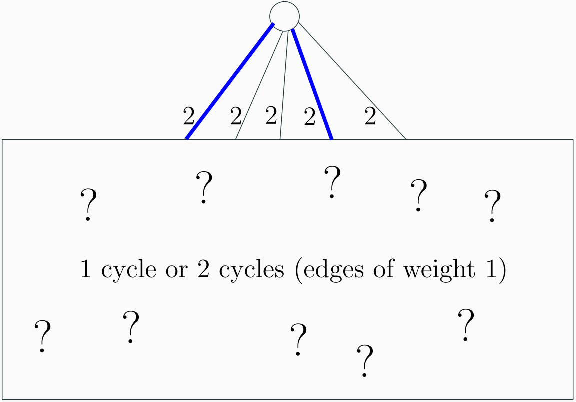
MST can't be calculated in $o(\log n)$ rounds, parameterized on D, n [**C**C '22]:



Even *approximating* an MST is difficult!

MST: Lower Bound

MST can't be calculated in $o(\log n)$ rounds, parameterized on D, n [**C**C '22]:



Even *approximately calculating the weight of an* MST is difficult!

Conclusion

Open Problems

- **Can connectivity be solved in sublinear** MPC **with optimal global space in** $O(\log D)$ **rounds?**
  - ...deterministically?
  - ...using $\Theta(m+n)$ local computation?
  - ...while maintaining a spanning forest?
- If not, what about a lower bound? Impossibility result only holds for polynomial memory...
- Can we compute an MST in sublinear MPC with optimal global space in $o(\log n)$ rounds (conditionally)?
- Computation-efficient derandomization in MPC—what more can be done?

Thank you for listening!