Designing Algorithms for the Congested Clique

Sriram V. Pemmaraju

The University of Iowa, USA
sriram-pemmaraju@uiowa.edu

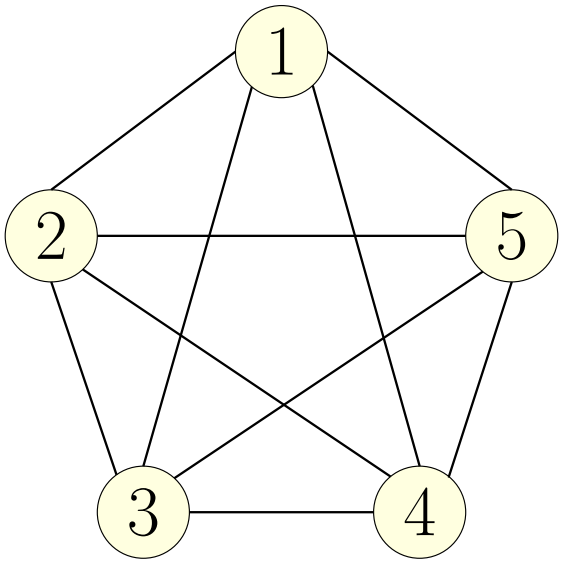
AGDA 2015, Tokyo, Oct 2015

Acknowledgements

- Students James Hegeman and Vivek Sardeshmukh
- Collaborators Gopal Pandurangan and Michele Scquizzato

Outline

The Congested Clique Model



- **CONGEST**: Synchronous, message passing; $O(\log n)$ bits per round per edge
- **Clique**: Communication network is fully connected

The Congested Clique Model

Compute-Communicate Cycle

At each clock tick:

- Each node performs local computation — using local knowledge, including messages received at previous clock tick.
- Each node sends a possibly different $O(\log n)$-size message to each of the other $n - 1$ nodes.

An alternate model:

Broadcast Congested Clique

Each node is required to send the same message to all $n - 1$ nodes.

The Congested Clique Model

Other assumptions:

- Each node has a unique $O(\log n)$ bit `ID`.
- All nodes know all IDs initially; given an `ID`, a node knows how to send a message to a node with that `ID`.

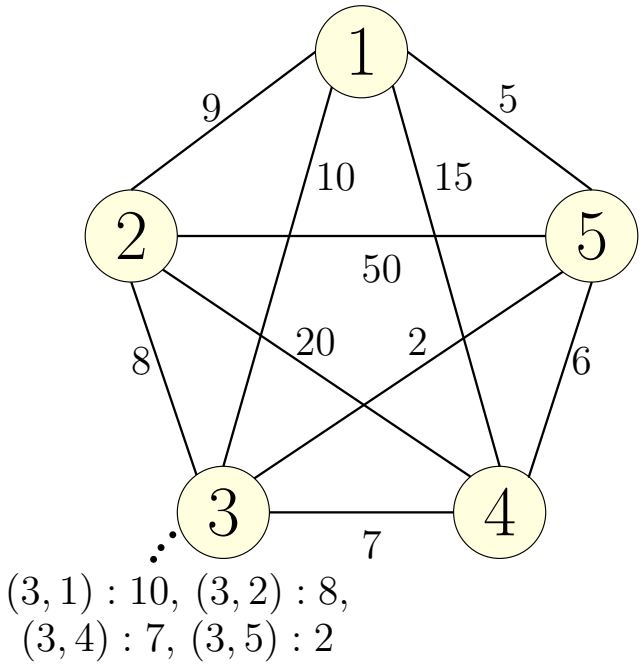
An alternate model:

Port model (aka *KT*0 model)

Each node has a "port" labeling (on its $n-1$ ports) but does not know the `ID`s of the nodes at the other end of its ports.

A Problem in the Congested Clique Model

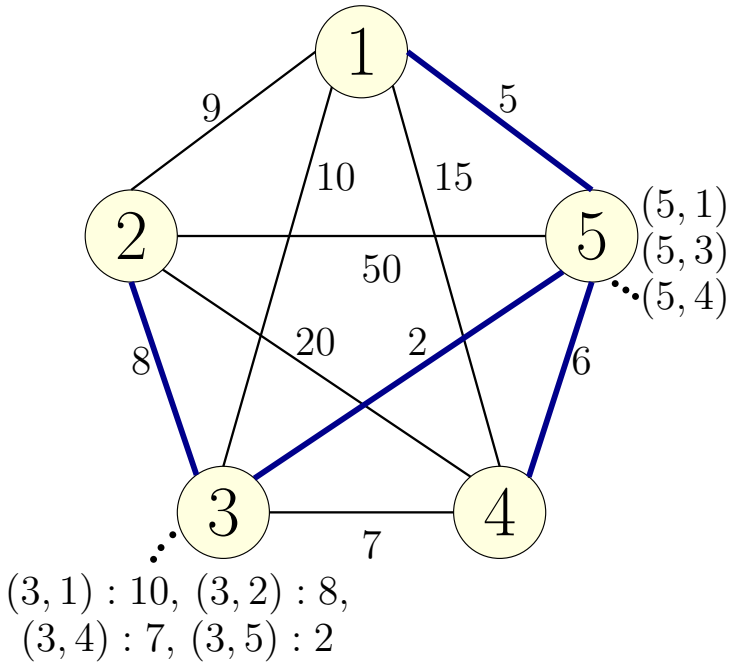
Minimum Spanning Tree (MST)



- **Initially**: each node knows weights of $n - 1$ incident edges
- **Finally**: each node knows incident MST edges

A Problem in the Congested Clique Model

Minimum Spanning Tree (MST)



- **Initially**: each node knows weights of $n - 1$ incident edges
- **Finally**: each node knows incident MST edges

An Early Congested Clique Result

Lotker, Patt-Shamir, Pavlov, Peleg *SICOMP* 2005

Deterministic $O(\log \log n)$-round algorithm.

Possible Motivation: (Lotker et al PODC 2001)

- Lower bound on round complexity of MST in the **CONGEST** model for diameter-3 graphs: $\Omega\left(\left(\frac{n}{\log n}\right)^{1/4}\right)$.
- MST can be solved in $O(\log n)$ rounds in the **CONGEST** model on diameter-2 graphs.

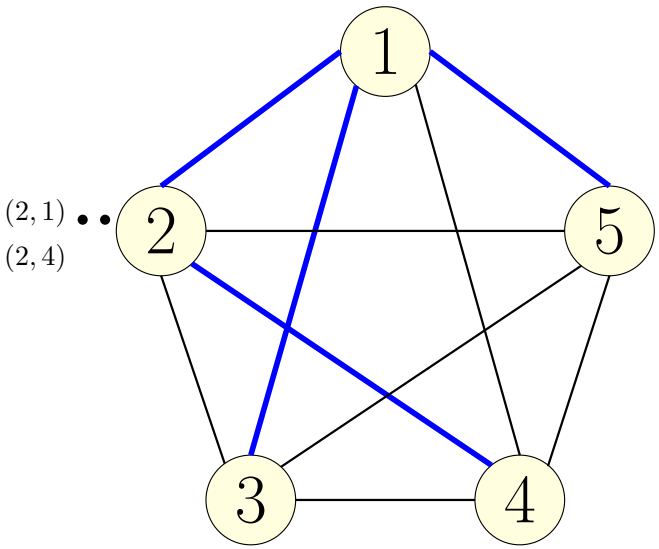
Floodgates have Opened...

Example Problems:

- All Pairs Shortest Paths (APSP), Triangle Counting
  (Dolev et al DISC 2012, Censor-Hillel et al PODC 2015)
- Graph Coloring
  (Hegeman et al SIROCCO 2014)
- Ruling Sets, Metric Facility Location
  (Berns et al ICALP 2012, Hegeman et al DISC 2014)
- Sorting, Routing
  (Lenzen PODC 2013, Patt-Shamir et al PODC 2011)

But, aren't many of these problems (e.g., graph coloring) trivial on a clique?

Input Graph is a Spanning Subgraph of Clique



- **Initially:** Each node knows all incident edges in input graph.
- Input graph is (slightly) decoupled from communication network.

Further decoupling is easy to imagine, e.g., input graph can be much larger than communication network — wait for the next talk to hear more about this!.

Why design algorithms for the Congested Clique?

- A simple abstraction for modern parallel computing infrastructure. (Similar to BSP, but easier to reason about.)
- For a Congested Clique algorithm to be fast, it needs to heavily exploit parallelism. So the Congested Clique model is a nice test bed for developing new techniques for the design of parallel algorithms.
- Fast Congested Clique algorithms can lead to fast algorithms in other (more realistic?) settings:
  - MapReduce (Hegeman et al SIROCCO 2014),
  - k-machine model (Klauck et al SODA 2015).

So how hard are problems in this model?

Example Problems:

- Counting Triangles: $O(n^{0.158})$ rounds
  (Censor-Hillel et al PODC 2015)
- Maximal Independent Set $O(\log n)$ rounds
  (Luby's algorithm fastest known?)
- Minimum Spanning Tree $O(\log \log \log n)$ rounds
  (Hegeman et al PODC 2015)
- Sorting $O(1)$ rounds
  (Lenzen PODC 2013)

What about lower bounds?

Simulation Thm (Drucker, Kuhn, Oshman PODC 2014)

The Congested Clique can simulate "powerful classes of bounded depth circuits."

Implication: Any non-trivial lower bound for the Congested Clique will imply new circuit complexity lower bounds (and solve decades-old open problems!).

Bad news: Current techniques may be hopeless for Congested Clique lower bounds!

Time is not the only resource

Consider recent results on Congested Clique MST algorithms (Hegeman et al PODC 2015)

Rounds	Message Complexity
$O(\log \log \log n)$	$\Omega(n^2)$
$O(\text{poly} \log n)$	$\Omega(n \cdot \text{poly} \log n)$

Focus on Round *and* Message Complexity

- **More relevant:** Congested Clique results that are most relevant to other models (e.g., MapReduce, k-machine model) have both low round complexity *and* message complexity.
- **Richer structure:** Problems have a much richer structure: yes, a node can talk to everyone, but it needs to be careful in figuring out who to talk to.
- **Lower bounds may be possible:** Round complexity lower bounds may be possible when constraints are placed on message complexity.

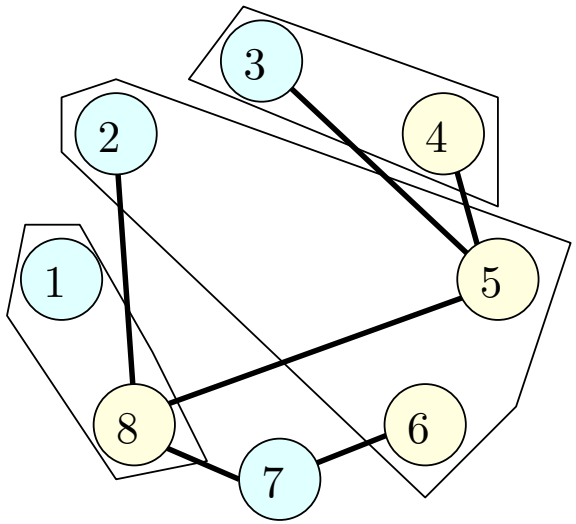
Roadmap

Congested Clique Algorithms for MST

- **Warm-up Example:**
  $O(\log n)$ rounds,$O(m)$ messages.
- **Reducing Message Complexity:**
  $O(\text{poly} \log n)$ rounds, $O(n \cdot \text{poly} \log n)$ messages.
  (**Idea:** Graph Sketches)
- **Reducing Round Complexity:**
  $O(\log \log n)$ rounds, $O(m)$ messages.
  (**Idea:** Quadratic growth rate of fragments)
- **Reducing Round Complexity Even More:**
  $O(\log \log \log n)$ rounds, $O(m)$ messages.
  (**Ideas:** Graph Sketches, Quadratic growth rate, Karger-Klein-Tarjan sampling, etc.)

Outline

MST in $O(\log n)$ rounds



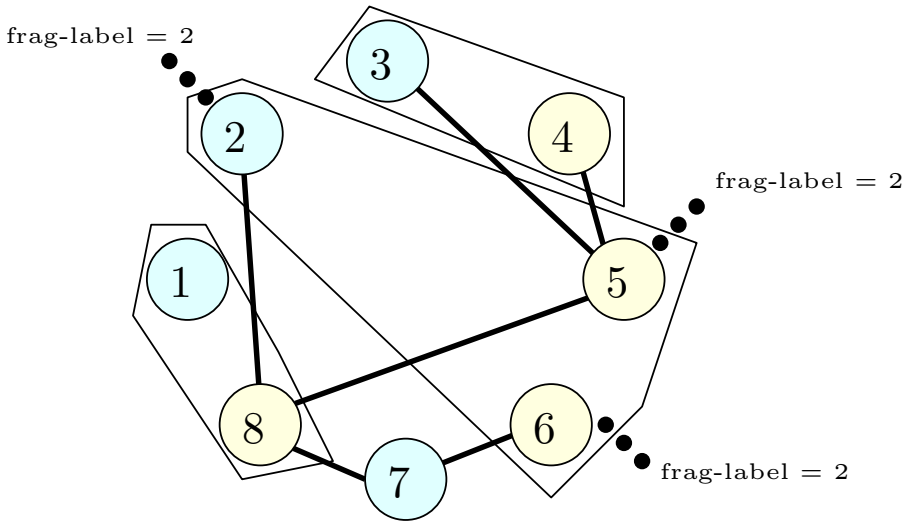
Invariant: Prior to each iteration:

(a) Each node knows fragment leader (label)

(b) Each node knows fragment labels of neighbors

Invariant (b) is costly to maintain.

MST in $O(\log n)$ rounds



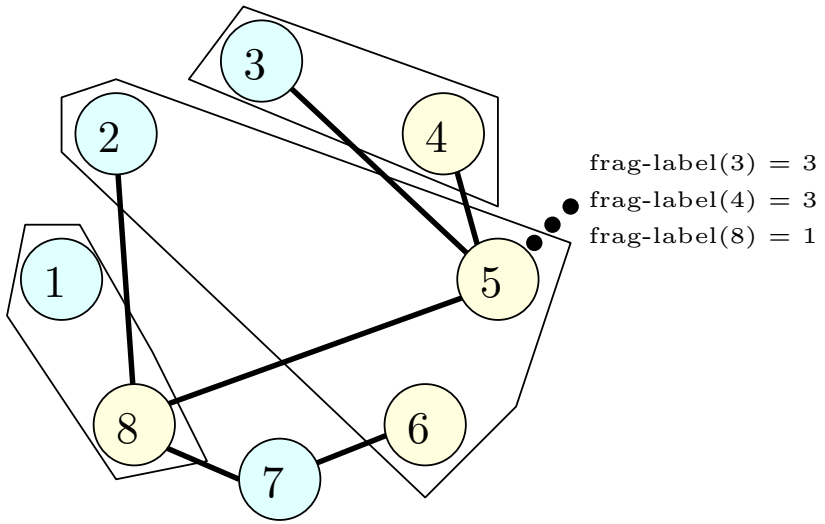
Invariant: Prior to each iteration:

(a) Each node knows fragment leader (label)

(b) Each node knows fragment labels of neighbors

Invariant (b) is costly to maintain.

MST in $O(\log n)$ rounds



Invariant: Prior to each iteration:

(a) Each node knows fragment leader (label)

(b) Each node knows fragment labels of neighbors

Invariant (b) is costly to maintain.

MST in $O(\log n)$ rounds

Round 1: Each node sends lightest incident inter-fragment edge to fragment leader ($O(n)$ messages).

Round 2: Each fragment leader picks lightest edge it receives and sends it to the *boss* ($O(n)$ messages).

Round 3: The boss merges fragments and informs every node of its new fragment leader ($O(n)$ messages).

Round 4: Each node sends its fragment label to all neighbors ($O(m)$ messages).

MST in $O(\log n)$ rounds

- **Invariant** prior to next iteration is satisfied.
- Size of smallest component has doubled.

Result

This is a Congested Clique MST algorithm that runs in $O(\log n)$ rounds using $O((m+n)\log n)$ messages.

A Digression: Broadcast Congested Clique

This algorithm works in the Broadcast Congested Clique model!

Open Question?

Can we design an $o(\log n)$-round in the Broadcast Congested Clique model (even for *Connectivity*)?

A Digression: Broadcast Congested Clique

A few lower bounds do exist in the Broadcast Congested Clique model.

Example (Drucker, Kuhn, Oshman PODC 2014)

Any deterministic algorithm for triangle detection in the Broadcast Congested Clique model requires $\Omega(n/e^{O(\sqrt{\log n})})$ rounds.

Open Question?

Can we prove a non-trivial lower bound on MST in the Broadcast Congested Clique model?

Back to Congested Clique: Questions

- Can we improve message complexity? (To $o(m)$?)
- Can we improve round complexity? (To $o(\log n)$?)
- Can we simultaneously improve both?

Outline

How do we reduce message complexity?

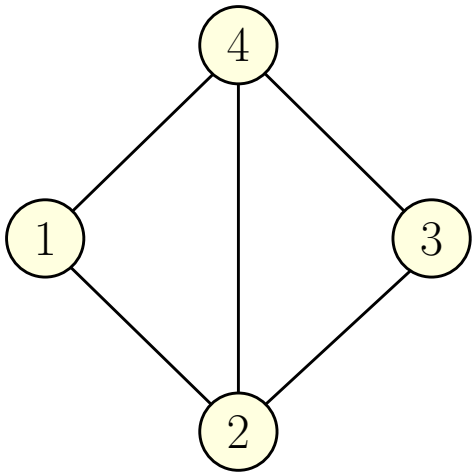
Graph sketching is a technique for computing graph properties (e.g., connected components, maximal matchings) in the streaming model

(Ahn, Guha, McGregor *SODA* 2012, *PODS* 2012)

Graph Sketches: Representation

For each vertex v in an n-vertex graph,

$$\mathbf{a}_v \in \{-1, 0, 1\}^{\binom{n}{2}}$$



$$\begin{array}{lcccccccc} & & 1\text{-}2 & 1\text{-}3 & 1\text{-}4 & 2\text{-}3 & 2\text{-}4 & 3\text{-}4 & \\ a_1 = (& & 1, & 0, & 1, & 0, & 0, & 0 &) \\ a_2 = (& & -1, & 0, & 0, & 1, & 1, & 0 &) \end{array}$$

Key Property: For neighbors u and v, $\mathbf{a}_u + \mathbf{a}_v$ "cancels" out edge $\{u, v\}$ and leaves only edges incident on component $u + v$

Graph Sketches: Properties

Project the $\mathbf{a}_v$'s into lower dimensional "sketch" space.

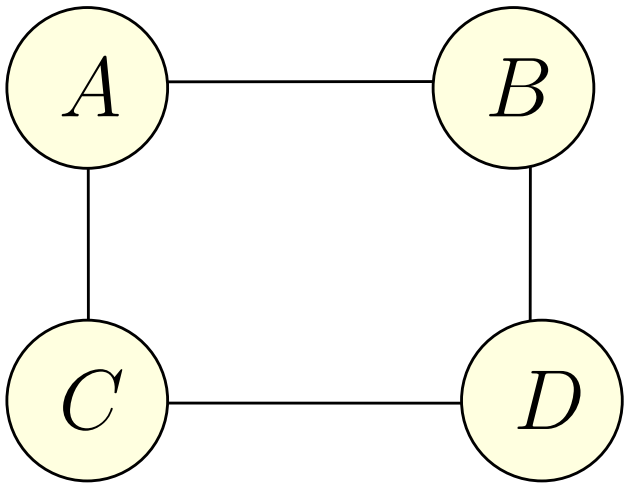
- Construct an $O(\text{poly} \log n) \times \binom{n}{2}$ random matrix L
- Use L to construct a linear projection $\mathbf{s}_v = L \cdot \mathbf{a}_v$ with two properties:

  Linearity: $\mathbf{s}_{u+v} = L \cdot (\mathbf{a}_u + \mathbf{a}_v) = L \cdot \mathbf{a}_u + L \cdot \mathbf{a}_v = \mathbf{s}_u + \mathbf{s}_v$

  ℓ_0**-sampling:** With constant probability, sampling from $\mathbf{s}_v$ yields a non-zero entry from $\mathbf{a}_v$ uniformly at random

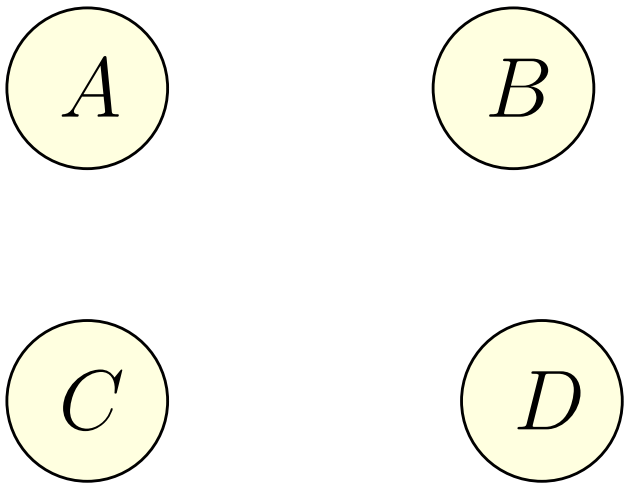
Example Construction: Jowhari, Sağlam, Tardos PODS 2011.

Example with Sketches



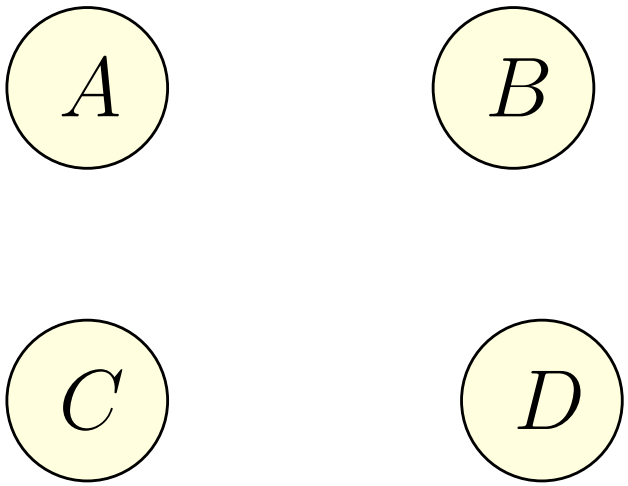
Input Graph

Example with Sketches



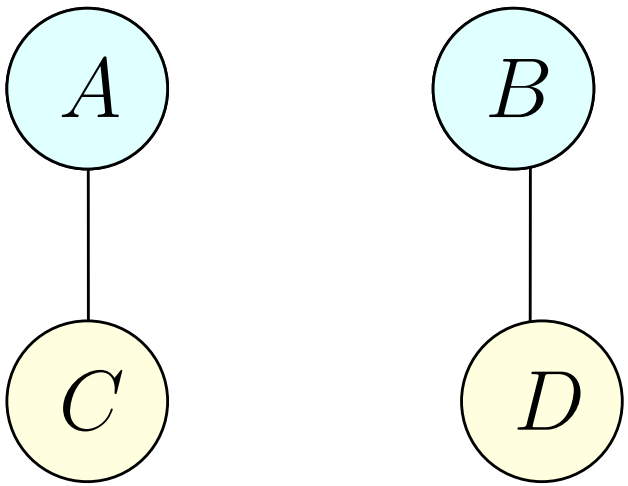
- Initial graph; each node v computes an $O(\log n)$-sized collection $\{s_v\}$ of neighborhood sketches
- $O(\log n)$ independently computed sketches provide high probability sampling

Example with Sketches



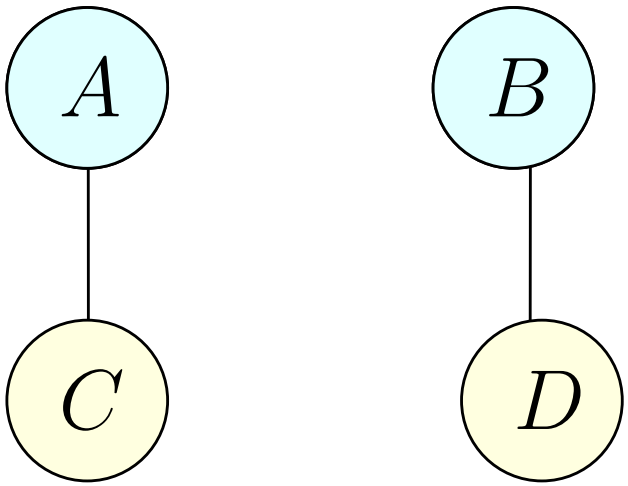
Sampling: Suppose
sampling $\{s_A\}$ yields (A, C)
sampling $\{s_B\}$ yields (B, D)
sampling $\{s_C\}$ yields (A, C)
sampling $\{s_D\}$ yields (B, D)

Example with Sketches



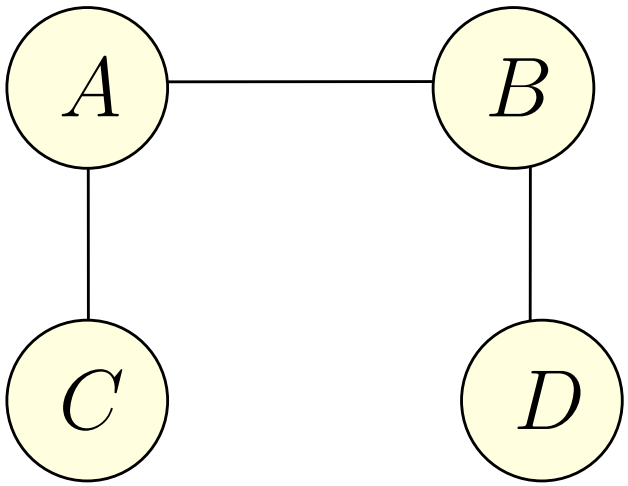
- Fragments are merged using sampled edges
- C sends s_C to leader A; A computes $s_A + s_C = s_{AC}$
- D sends s_D to leader B; B computes $s_B + s_D = s_{BD}$

Example with Sketches



Sampling: Suppose sampling s_{AC} yields (A, B)
sampling s_{BD} fails

Example with Sketches



No inter-component edges are left; we're done.

Connectivity in the Congested Clique

Invariant:

- Each fragment leader knows a sketch collection for its fragment.
- Each node knows who its fragment leader is.

Round 1: Each fragment leader samples its sketch collection for an outgoing edge; sends this edge to the boss ($O(n)$ messages).

Round 2: The boss merges components and informs each node of its new fragment label ($O(n)$ messages).

Round 3: Each node recomputes a sketch collection and sends to its new fragment leader and fragment leader computes sum of the sketches ($O(n \cdot \text{poly} \log n)$ messages).

Connectivity in the Congested Clique

Result

This is an algorithm for Connectivity that runs in $O(\log n)$ rounds using $O(n \cdot \operatorname{poly} \log n)$ messages.

What about MST?

How to pick lightest edge using sketches?

Sampling for a lightest outgoing edge

- Each fragment leader samples its sketches to produce an outgoing edge and sends this edge weight to all its followers.
- Each node recomputes sketches considering only those incident edges with weight less than the received edge weight.

Lightest edge can be sampled in $O(\log n)$ rounds with high probability.

Low Message Complexity MST

Result (Hegeman et al PODC 2014)

This algorithm computes an MST in $O(\text{poly} \log n)$ rounds using $O(n \cdot \text{poly} \log n)$ messages.

Digression: Port Model

A standard sequence in the above algorithm is:

- The boss tells each node their new fragment leaders' `ID`
- Then each node communicates directly with new fragment leader.

This step is impossible in the port model!

Lower Bound (Korach, Moran, Zaks SICOMP 1987)

Any algorithm (even randomized Monte Carlo) for Connectivity in the Congested Clique uses $\Omega(m)$ messages.

Outline

MST in $O(\log \log n)$ rounds

Lotker, Patt-Shamir, Pavlov, Peleg SICOMP 2005

How to accelerate the rate of growth of MST fragments?

Goal: Suppose each MST fragment has at least μ vertices. In one iteration ($O(1)$ rounds) we want every fragment to grow to size at least μ^2.

"Quadratic growth rate" (if achieved) will lead to an $O(\log \log n)$ round algorithm.

Achieving Quadratic Growth Rate

Suppose

- Each MST fragment has at least μ vertices
- For each MST fragment F, we find $L(F)$, a set of μ lightest edges connecting F to μ *distinct* components.

Theorem

It is possible to use the edges in $\cup_F L(F)$ to merge the MST fragments such that each super-fragment contains at least μ fragments.

Algorithm Sketch

Step 1: For each MST fragment F, the leader of F computes $L(F)$ and sends to the boss.

Step 2: The boss uses these edges to merge fragments and informs all nodes of their new fragment leaders.

Step 1 Questions:

- How does the leader of fragment F compute $L(F)$?
- How does the leader send all these edges to the boss?

Computing $L(F)$?

- Each node (say $v \in F'$) finds the lightest edge from it to a node in F. Node v sends this edge to the leader of F.
- The leader of F now nows lightest edges from every node $v \notin F$ to F. Using these, it picks μ lightest edges connecting F to distinct components.

Sending $L(F)$ to the boss

Note that $|L(F)| \leq |F|$ and so the following simple *Scatter-and-Gather* step will work.

- The leader of F "scatters" edges in $L(F)$ to the nodes in F so that each node in F receives at most one edge.
- Each node in F sends its edge to the boss.

MST in $O(\log \log n)$ rounds

Result

This is a Congested Clique MST algorithm running in $O(\log \log n)$ rounds and using $O(m)$ messages.

We don't know how to sustain this rate of growth while using sketches (so as to reduce message complexity).

Open Question

Can we solve MST in $O(\log \log n)$ rounds using $o(m)$ messages?

Outline

Reducing the Round Complexity Further

Main Result

Randomized (Monte Carlo) MST algorithm running in $O(\log \log \log n)$ rounds w.h.p.

Additionally: Algorithm runs in $O(1)$ rounds, if per edge bandwidth is expanded to $\Theta(\text{poly} \log n)$ (from $O(\log n)$).

Outline

MST Algorithm: Version 1

MST

1. Sort edges by weight
2. Partition and distribute edges such that Node 1 receives lightest n edges (E_1), Node 2 receives next lightest n edges (E_2), and so on.
3. *In parallel*, for each Node i:
   (a) Node i computes connected components of G_i (the graph induced by edges $\cup_{j=1}^{i-1} E_j$)
   (b) Node i determines, for each edge $e \in E_i$, if e belongs to MST

Sorting and Routing in a Congested Clique

Lenzen *PODC* 2013

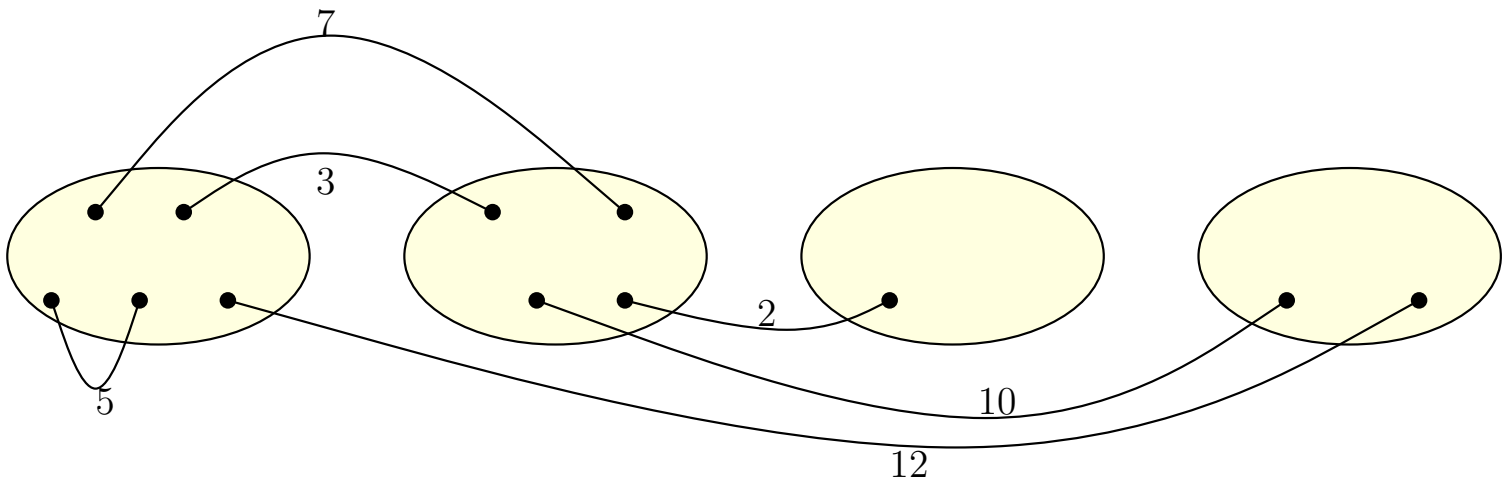
Step 1 (Sorting) and Step 2 (Routing) can be (deterministically) completed in $O(1)$ rounds each

Sorting: Each node has at most n items (from a totally ordered universe). Each node needs to learn the *rank* of each item it has in the global total ordering of all items possessed by all nodes.

Routing: Each node has at most n messages it wants to send to various destinations such that every node is the destination of at most n messages.

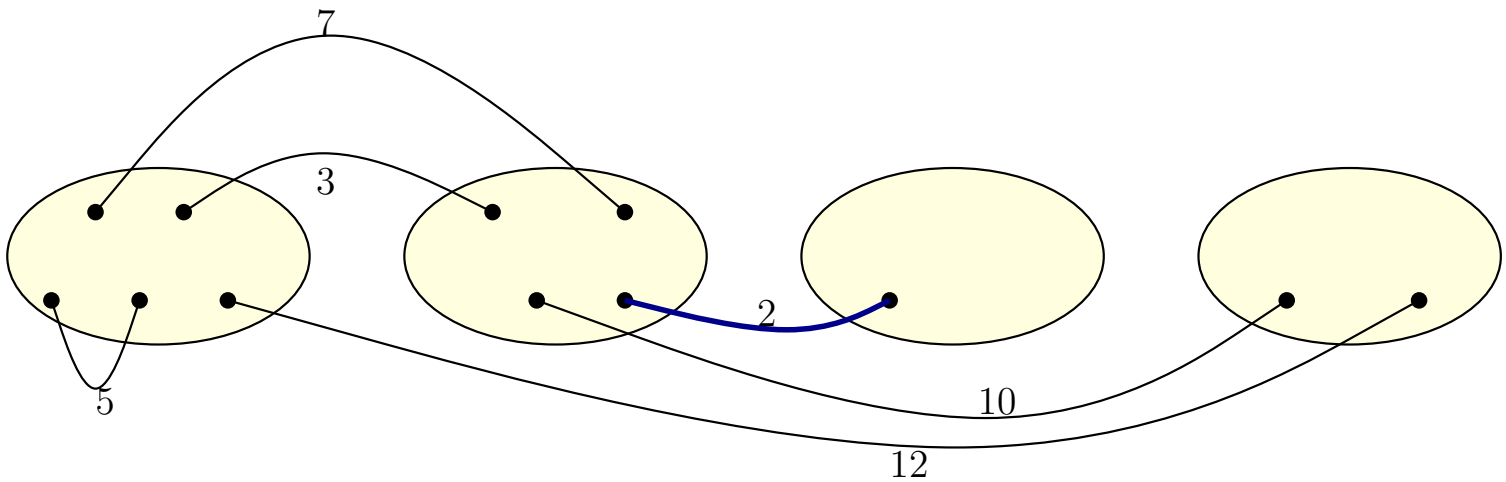
Testing Membership in MST

- Suppose Node i knows the connected components in G_i (graph induced by $\cup_{j=1}^{i-1} E_i$)
- Node i can determine which edges in E_i are in MST by locally executing Kruskal's algorithm (**Step 3(b)**)



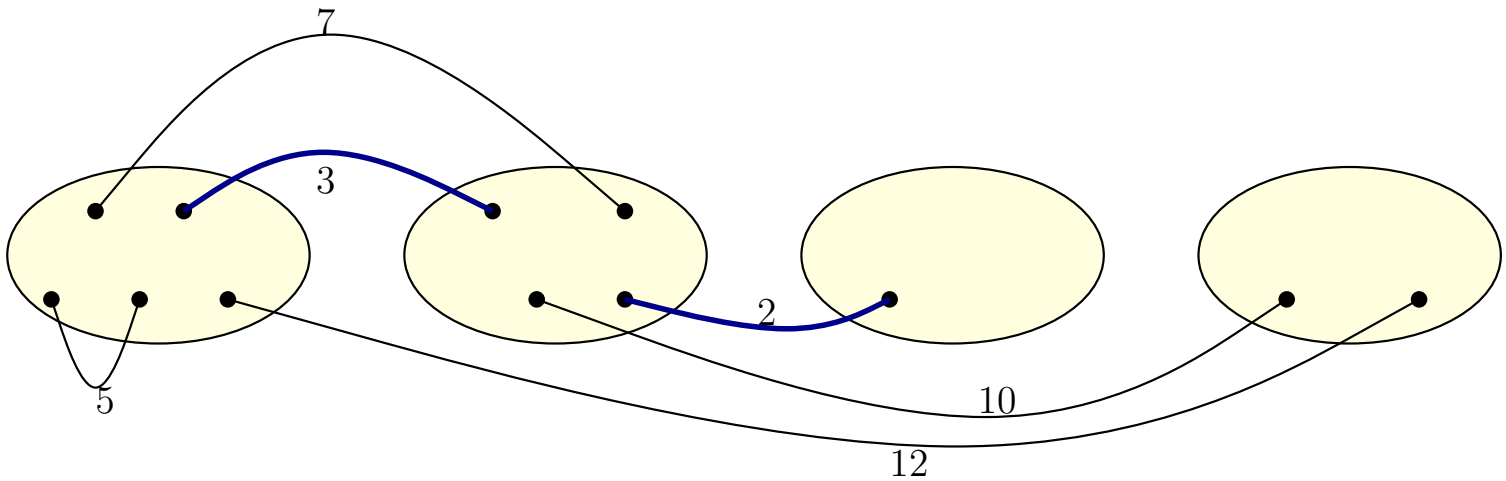
Testing Membership in MST

- Suppose Node i knows the connected components in G_i (graph induced by $\cup_{j=1}^{i-1} E_i$)
- Node i can determine which edges in E_i are in MST by locally executing Kruskal's algorithm (**Step 3(b)**)



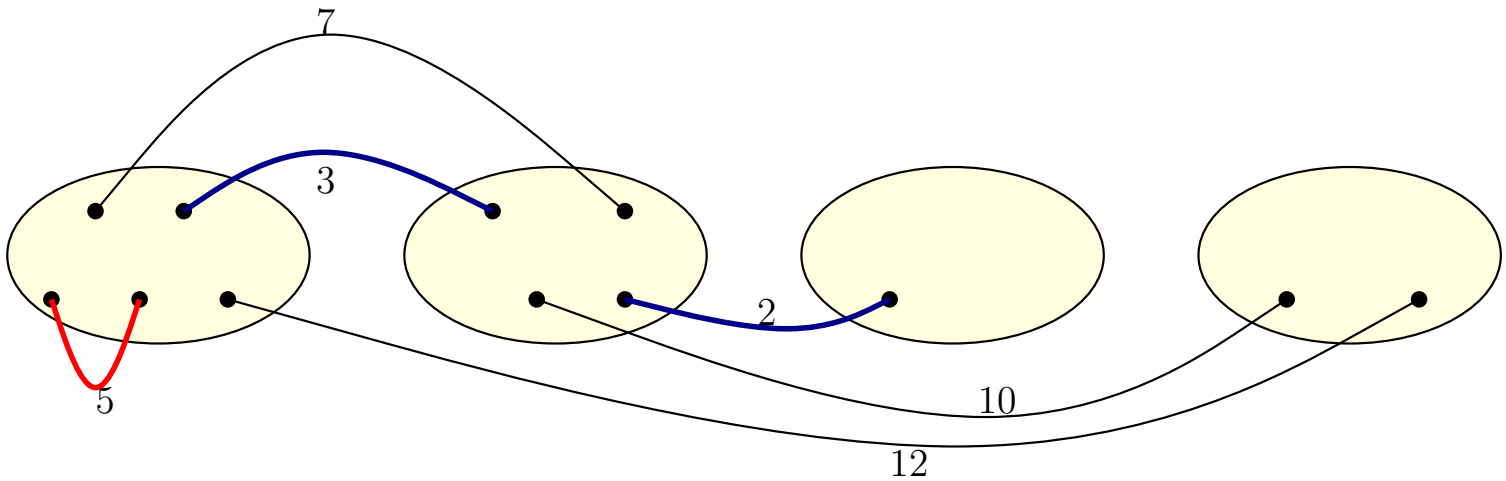
Testing Membership in MST

- Suppose Node i knows the connected components in G_i (graph induced by $\cup_{j=1}^{i-1} E_i$)
- Node i can determine which edges in E_i are in MST by locally executing Kruskal's algorithm (**Step 3(b)**)



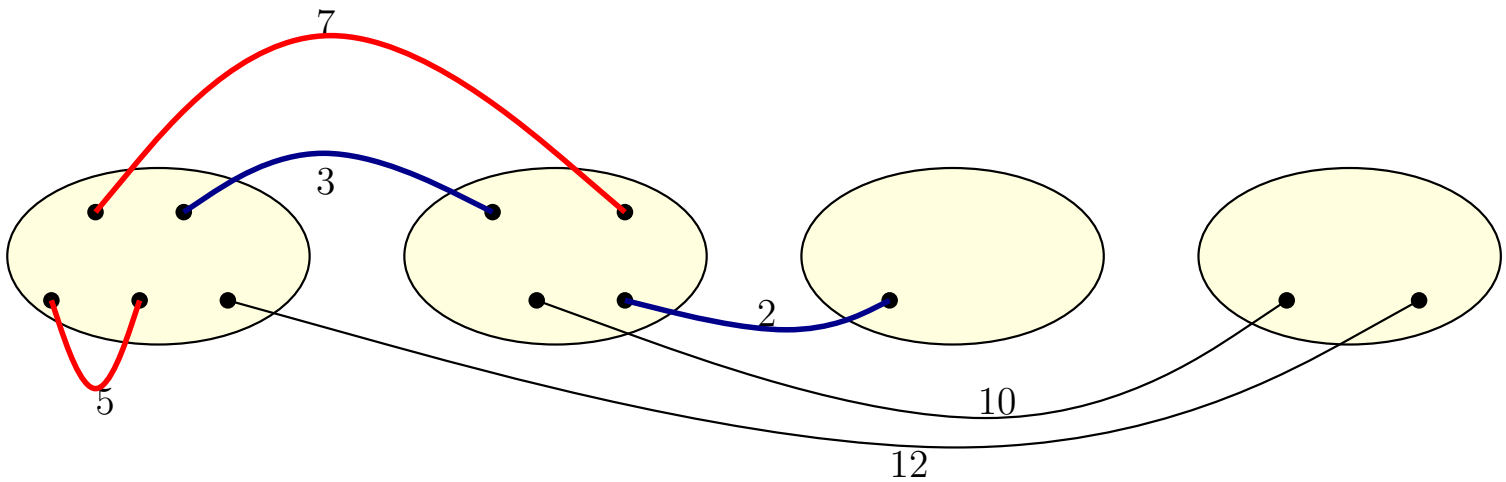
Testing Membership in MST

- Suppose Node i knows the connected components in G_i (graph induced by $\cup_{j=1}^{i-1} E_i$)
- Node i can determine which edges in E_i are in MST by locally executing Kruskal's algorithm (**Step 3(b)**)



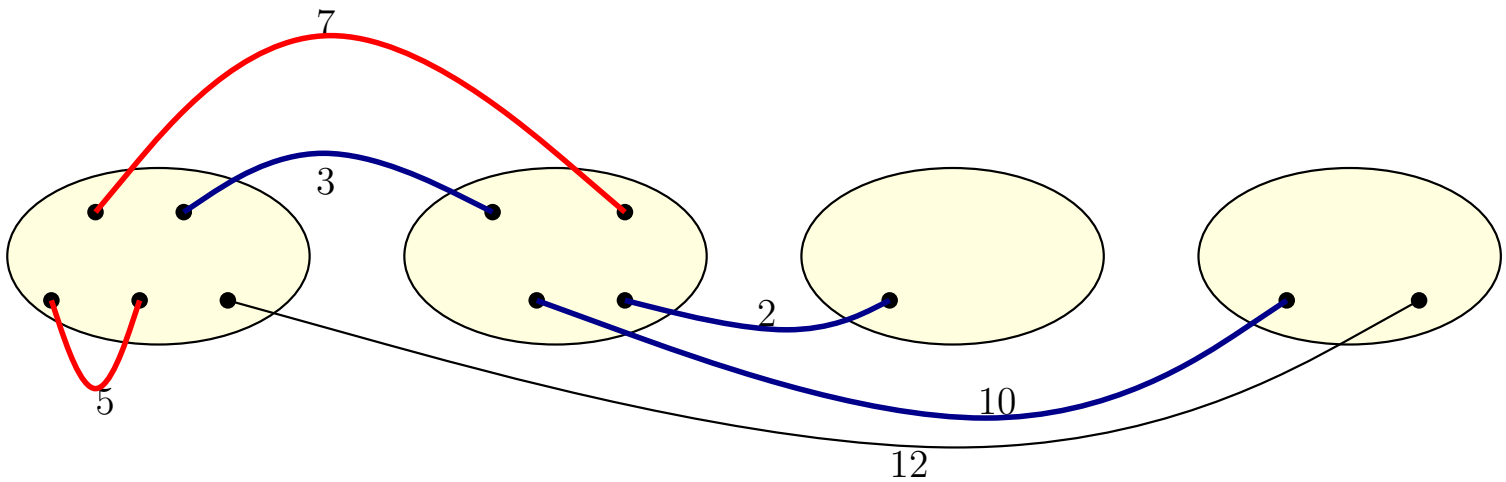
Testing Membership in MST

- Suppose Node i knows the connected components in G_i (graph induced by $\cup_{j=1}^{i-1} E_i$)
- Node i can determine which edges in E_i are in MST by locally executing Kruskal's algorithm (**Step 3(b)**)



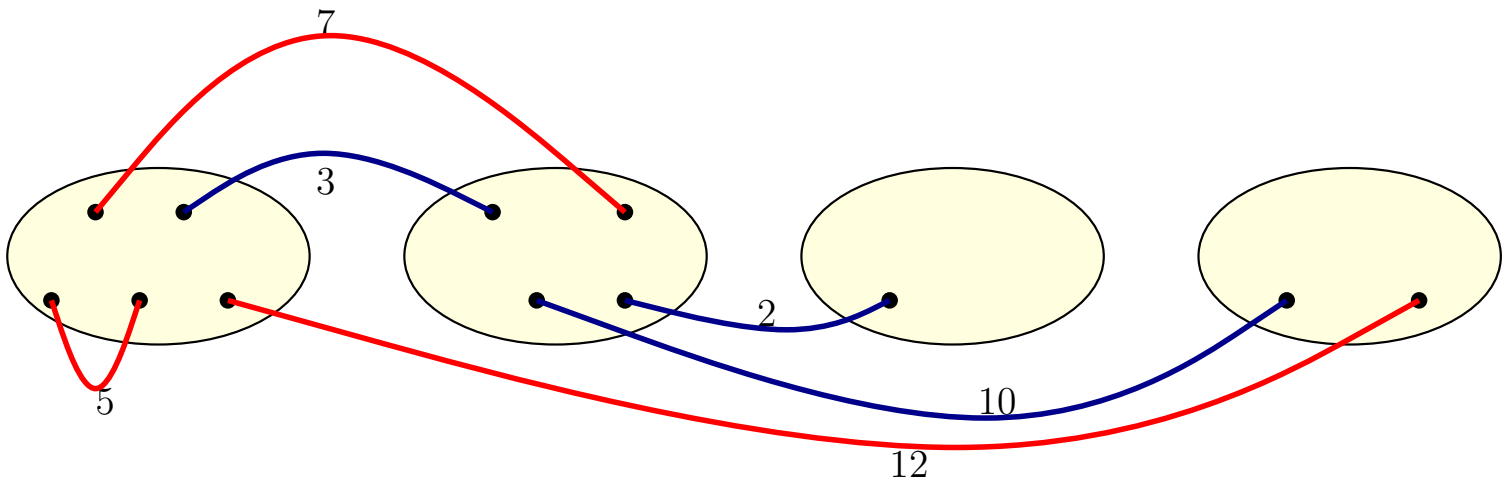
Testing Membership in MST

- Suppose Node i knows the connected components in G_i (graph induced by $\cup_{j=1}^{i-1} E_i$)
- Node i can determine which edges in E_i are in MST by locally executing Kruskal's algorithm (**Step 3(b)**)



Testing Membership in MST

- Suppose Node i knows the connected components in G_i (graph induced by $\cup_{j=1}^{i-1} E_i$)
- Node i can determine which edges in E_i are in MST by locally executing Kruskal's algorithm (**Step 3(b)**)



MST Algorithm: Version 1

MST

1. Sort edges by weight
2. Partition and distribute edges such that Node 1 receives smallest n edges (E_1), Node 2 receives next smallest n edges (E_2), and so on.
3. *In parallel*, for each Node i:
   (a) Node i computes connected components of G_i (the graph induced by edges $\cup_{j=1}^{i-1} E_j$)
   (b) Node i determines, for each edge $e \in E_i$, if e belongs to MST

Outline

Connected Components in Parallel

We now consider two questions:

(A) How to compute connected components of G_i fast? (*Graph Sketches*)

(B) How to compute connected components of all G_i's in parallel? (*Preprocessing to Reduce Number of Nodes and Edges*)

SAMPLE-AND-MERGE Algorithm

Sketches lead to the following simple, *sequential* algorithm for computing connected components:

SAMPLE-AND-MERGE

① **repeat** until no inter-component edge
  ② **for** each component C **do**
    ③ sample an outgoing edge from the sketches of C
  ④ merge components using sampled edges
  ⑤ build sketches of new components by bitwise addition of sub-component sketches

How many sketches are needed?

- To sample an outgoing edge *with high probability* we need $\Theta(\log n)$ sketches for a component per iteration.
- To ensure independence, we need different sketch-collections in each of the $O(\log n)$ iterations.

In Summary

Every node v needs to compute $O(\text{poly} \log n)$ different sketches.

Congested Clique Algorithm for Connectivity

Attempt 1

CONNECTED-COMPONENTS

1. Each node v computes $O(\text{poly} \log n)$ sketches $\{\mathbf{s}_v\}$ of neighborhood
2. Each node v sends sketch-collection $\{\mathbf{s}_v\}$ to the boss
3. The boss runs SAMPLE-AND-MERGE algorithm
4. The boss communicates connected components back to nodes

MST Algorithm: Version 1

MST

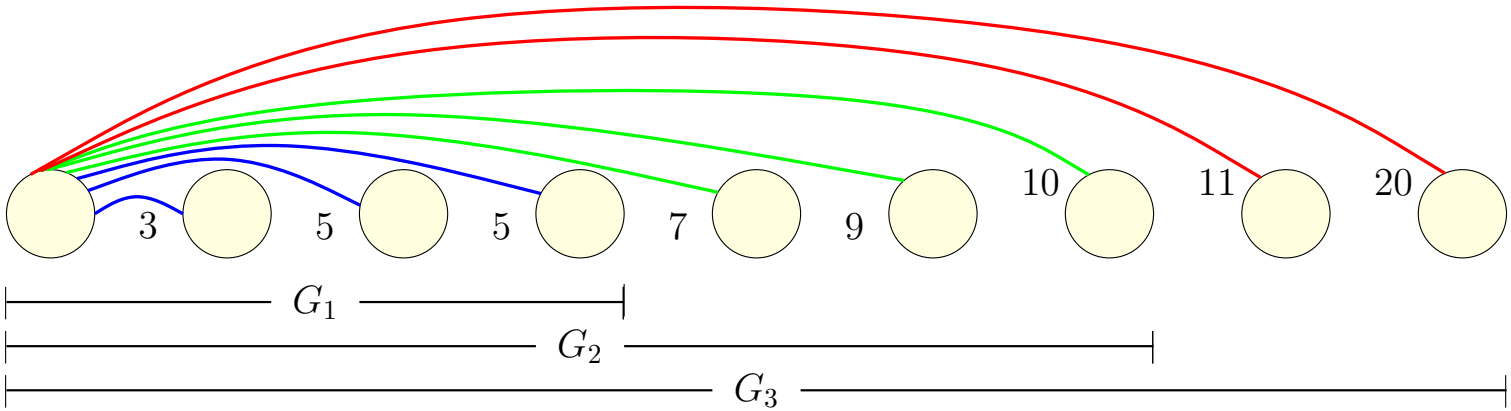
1. Sort edges by weight
2. Partition and distribute edges such that Node 1 receives smallest n edges (E_1), Node 2 receives next smallest n edges (E_2), and so on.
3. *In parallel*, for each Node i:
   (a) Node i computes connected components of G_i (the graph induced by edges $\cup_{j=1}^{i-1} E_j$)
   (b) Node i determines, for each edge $e \in E_i$, if e belongs to MST

Outline

Communication Bottlenecks

Receiver-side bottleneck: Each node needs to send $\Theta(\text{poly} \log n)$ bits to the boss, which needs to *receive* a total of $\Theta(n \cdot \text{poly} \log n)$ bits.

Sender-side bottleck: Multiple connected component computations – on graphs $G_2, G_3, \ldots$ – need to occur in parallel. This means each node may need to *send* $\Theta(n \cdot \text{poly} \log n)$ bits



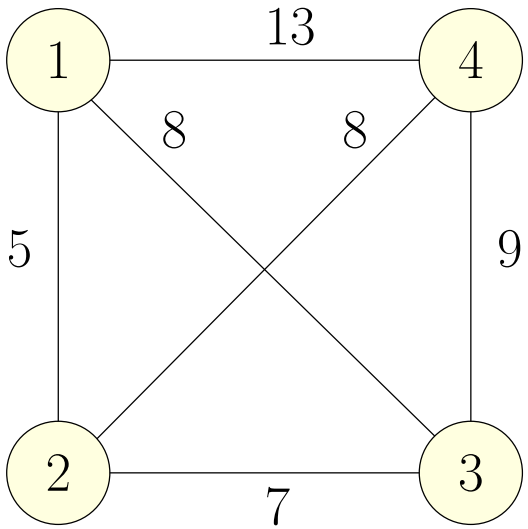
Shrink Number of Nodes

- We run the Lotker et al. algorithm for $O(\log \log \log n)$ rounds.
- Every connected component has size $\Omega(\text{poly} \log n)$ and hence there are at most $O(n/\text{poly} \log n)$ components.
- We can treat these components as (super) nodes and now the boss has to receive $O(\text{poly} \log n)$ sketches from $O(n/\text{poly} \log n)$ nodes.
- This means the boss has to receive n messages, resolving the Receiver-side bottleneck.

Shrink Number of Edges

Karger-Klein-Tarjan (KKT) Sampling

- Let H be the graph obtained by independently sampling each edge (of the clique) with probability $p = 1/\sqrt{n}$. With high probability H has $\Theta(n^{3/2})$ edges.
- Find an MST F of H (**Note**: H need not be connected).

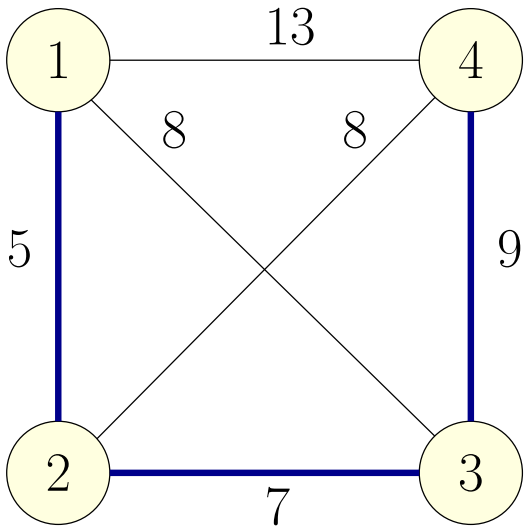


- The edges of the clique are partitioned into *F-light* and *F-heavy* edges.
- F-heavy edges will not be in MST.

Shrink Number of Edges

Karger-Klein-Tarjan (KKT) Sampling

- Let H be the graph obtained by independently sampling each edge (of the clique) with probability $p = 1/\sqrt{n}$. With high probability H has $\Theta(n^{3/2})$ edges.
- Find an MST F of H (**Note**: H need not be connected).

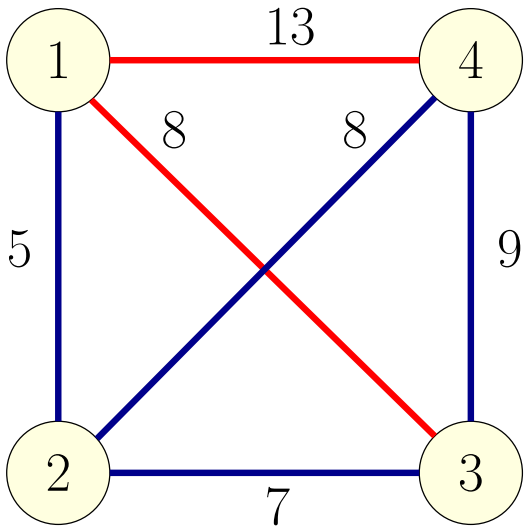


- The edges of the clique are partitioned into *F-light* and *F-heavy* edges.
- F-heavy edges will not be in MST.

Shrink Number of Edges

Karger-Klein-Tarjan (KKT) Sampling

- Let H be the graph obtained by independently sampling each edge (of the clique) with probability $p = 1/\sqrt{n}$. With high probability H has $\Theta(n^{3/2})$ edges.
- Find an MST F of H (**Note**: H need not be connected).



- The edges of the clique are partitioned into *F-light* and *F-heavy* edges.
- F-heavy edges will not be in MST.

KKT Sampling Theorem

KKT Sampling Theorem

Number of F-light edges is $O(n/p) = O(n^{3/2})$ with high probability.

- Find F-light edges.
- Return MST of F-light edges.

In Summary

Via KKT Sampling, MST computation on a clique (with $\Theta(n^2)$ edges) can be reduced to two MST computations on graphs with $O(n^{3/2})$ edges, each.

MST Algorithm: Version 2

MST Algorithm: Version 2

1. Run $O(\log \log \log n)$ rounds of the Lotker et al. Congested Clique MST algorithm.
2. Construct "super graph" with MST fragments as "super vertices."
3. Independently sample edges of the "super graph" with probability $1/\sqrt{n}$ to construct graph H.
4. Compute MST F of H using MST Version 1.
5. Compute L, the set of F-light edges.
6. Compute MST of L using MST Version 1.

Missing Details

- Construction of graph sketches (as described) requires shared randomness. This can be avoided using alternate, *limited dependence* linear graph sketch constructions.

  (Cormode and Firmani, *Distributed and Parallel Databases* 2014)

- "Super graph" construction (Step 2) and computing the set of F-light edges (Step 5) can be done in $O(1)$ rounds each.

Outline

Open Questions

Open Questions

(1) Is there an $O(1)$-round MST algorithm in the Congested Clique?

(2) Is there an $o(\log n)$-round MST algorithm that uses $O(n \cdot \text{poly} \log n)$ messages?

Advertisement: Visit our poster (with Vivek Sardeshmukh) on progress on second question.

Thank you!

- Thank you for listening to my talk.
- Thanks to the organizers for giving me ths opportunity to talk about stuff I am excited about.