Distributed and Parallel Graph Distance Approximation

Yasamin Nazari

University of Salzburg
Fall 2021

Publications

- **Relevant papers:**
  - M. Dinitz and **N**, *Massively Parallel Distance Sketches*, **Conference on Principles of Distributed Systems (OPODIS) 2019** (*best student paper)*.
  - **N**, *Sparse Hopsets in Congested Clique*, **Conference on Principles of Distributed Systems (OPODIS) 2019**.

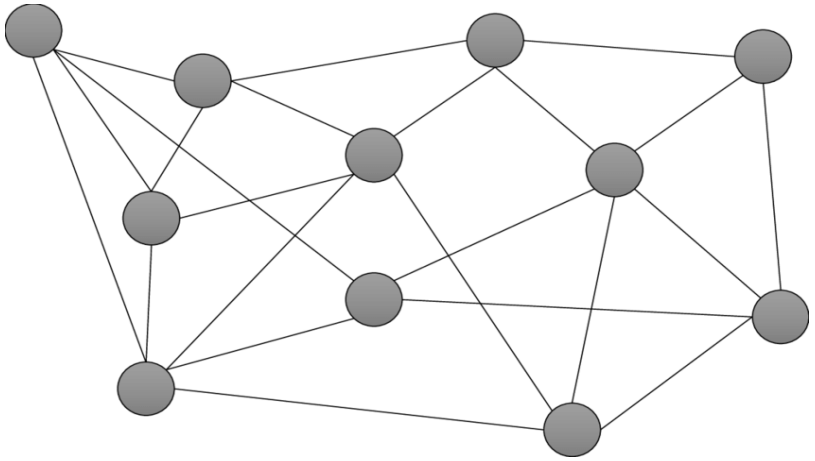
- **Also related:**
  - J. Łącki and **N**, *Faster Decremental Approximate Shortest Paths via Hopsets with Low Hopbound* (to be submitted).

Outline

- **Models**
  - Background on distributed/parallel/big data models
- **Introduction to hopsets**
  - Application in distributed distance computation
  - Massively parallel distance sketches
- **Distributed algorithm for constructing hopsets**

Evolution of Distributed Models

- **LOCAL Model**
  - Given an input graph G=(V,E). Communication in synchronous rounds on G.
  - In each round each node can send a message of unlimited size to each neighbor.
  - Goal: Minimize rounds of communication until nodes know their portion of output.

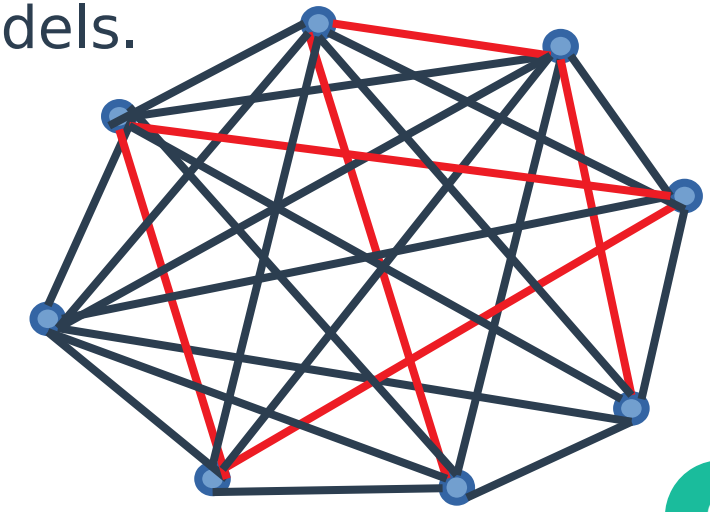


Distributed and Parallel Models

- **LOCAL Model**
  - Given an input graph G=(V,E). Communication in synchronous rounds on G.
  - In each round each node can send a message of unlimited size to each neighbor.
  - Goal: Minimize rounds of communication until nodes know their portion of output.
- **CONGEST Model**
  - Similiar to LOCAL but messages can have size at most O(log n).
- **Parallel (PRAM) Models**
  - Processers read/write on registers. Goal is to minimize parallel rounds/depth.

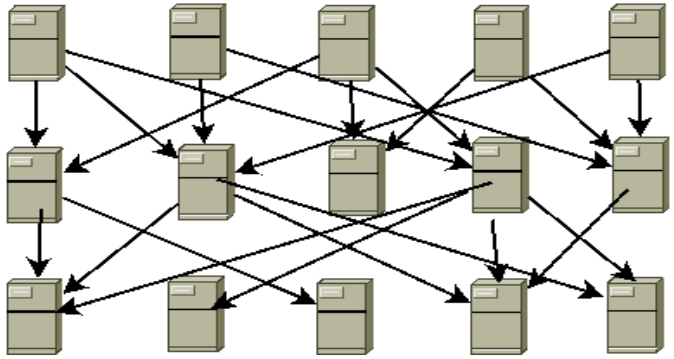
Congested Clique Model

- **Input graph G=(V,E). Communication over a clique (all-to-all).**
  - Each node sends a message of size O(log n) (congestion) to any other node.
  - Similiar to CONGEST except communication graph is different from input graph
  - Closer to modern models, e.g. SDNs, MapReduce and Massively Parallel Computation models.



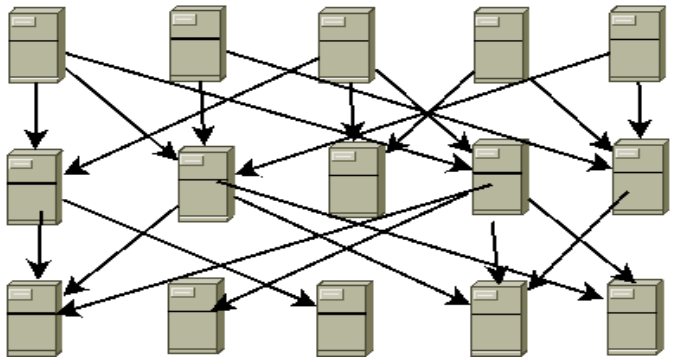
Massively Parallel Computation (MPC)

- **MPC model: An input of size N, distributed over P = N/S machines, O(S) memory per machine.**
  - Each machine has memory strictly sublinear in N. IO/Communication bounded by memory.
  - Abstraction of big data platforms such as MapReduce, Spark, Hadoop, etc.



Massively Parallel Computation (MPC)

- **MPC model: An input of size N, distributed over P = N/S machines, O(S) memory per machine.**
  - Each machine has memory strictly sublinear in N. IO/Communication bounded by memory.
- **Low memory MPC for a graph G (m edges and n nodes):**
  - Memory per machine is $O(n^{\epsilon}), \epsilon < 1$.
  - Overall memory is $O(m)$.



MPC vs distributed and parallel models

- **Power of MPC depends**
  - Primarily: memory per machine
  - Secondary: number of machines
- **For graphs MPC with linear in number of nodes memory (per machine) close to Congested Clique**
  - In each round each machine can send a message to any other machine since memory per machine is O(n)
- **Closer to PRAM when memory per machine is very small**

Massively Parallel Computation

- **Intuition: MapReduce**
  - Input: <key, value> pairs
  - Map and shuffle: pairs go to machine based on their key
  - Reduce: Sequential computation on one key (local computation on a machine)
- **Example: Given a graph G=(V,E), compute an aggregate function over edges of a node (degree, lightest edge)**
  - Sort the input (pairs (u,v)) based on ID of the first vertex
  - Pairs incident to u are in a contigious set of machines
  - Find the minimum over these machines based on an aggregate tree

- **Previous:**
  - Distributed/Parallel/Big Data Models

- **Next:**
  - Distributed Shortest Paths

Shortest Path Computation

- **Single-source shortest paths:**
  - Given an undirected weighted graph $G=(V,E)$, and a source node s, compute (approximate) distances from s to all nodes in V.
- **Variants:**
  - Single-source (SSSP), Multi-source (MSSP), all-pairs (APSP)

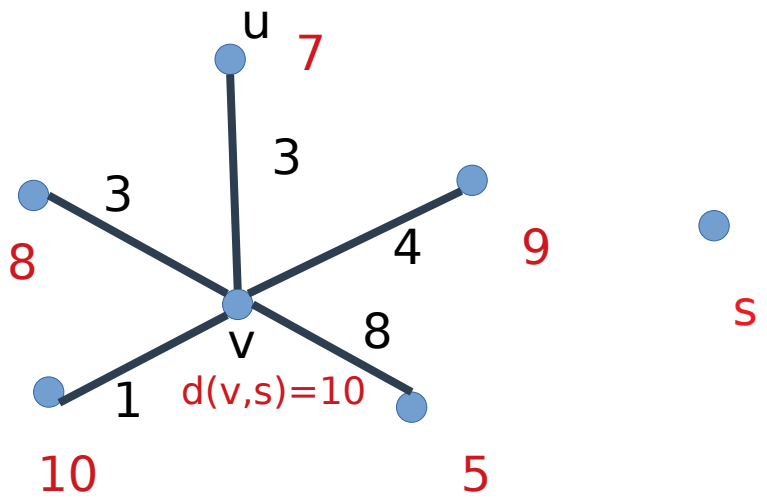
Shortest Path Computation

- **Shortest path algorithms**
  - Dijksta's: very sequential, not parallelizable
- **Simple parallel/distributed algorithm:**
  - For unweighted graphs run BFS.
  - For weighted graphs, run Bellman-Ford.
  - Slow for graphs with large diameter, so we need a new tool.

Bellman-Ford

- **Single-source shortest path via Bellman-Ford:**
  - Each iteration: each node updates their distance estimate from the source s by computing

$$\tilde{d}(v,s) = \min_{u \in N(v)} \tilde{d}(u,s) + w(u,v)$$



Distributed Bellman-Ford

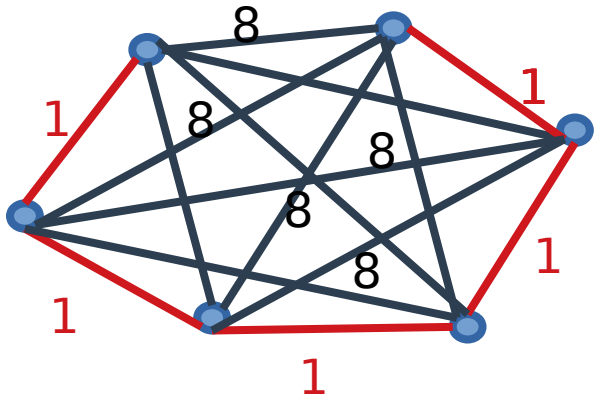
- **Distributed Bellman-Ford:**
  - Each iteration: each node updates their distance estimate from the source s by computing

$$\tilde{d}(v,s) = \min_{u \in N(v)} \tilde{d}(u,s) + w(u,v)$$

  - Each iteration can be performed in one round of Congested Clique or MPC
    - Congested Clique: Need to send only one message of O(log n) bits for each edge
    - MPC: Aggregate tree idea

Bellman-Ford

- **Bellman-Ford from single source s:**
  - h iterations to compute $d_G^{(h)}(s, v)$ (distance using paths of at most h hops) for all $v \in V$.
  - Require $O(\text{diam})$ iterations diam is the shortest path diameter:
    - Maximum number of hops in the shortest paths. Could be as large as $\Omega(n)$.



Hopsets

- **Given a weighted undirected graph $G = (V, E, w)$, for any $u, v \in V$:**
  - $d_G(u, v)$: shortest path distance between u and v in G.
  - $d_G^{(\beta)}(u, v)$: shortest path with at most β hops (number of edges) between u and v.
- **Given a $G = (V, E, w)$, a (β, ϵ)-hopset H is a set of edges, s.t. between every pair of nodes u, v :**

$$d_G(u, v) \leq d_{G \cup H}^{(\beta)}(u, v) \leq (1 + \epsilon) d_G(u, v)$$

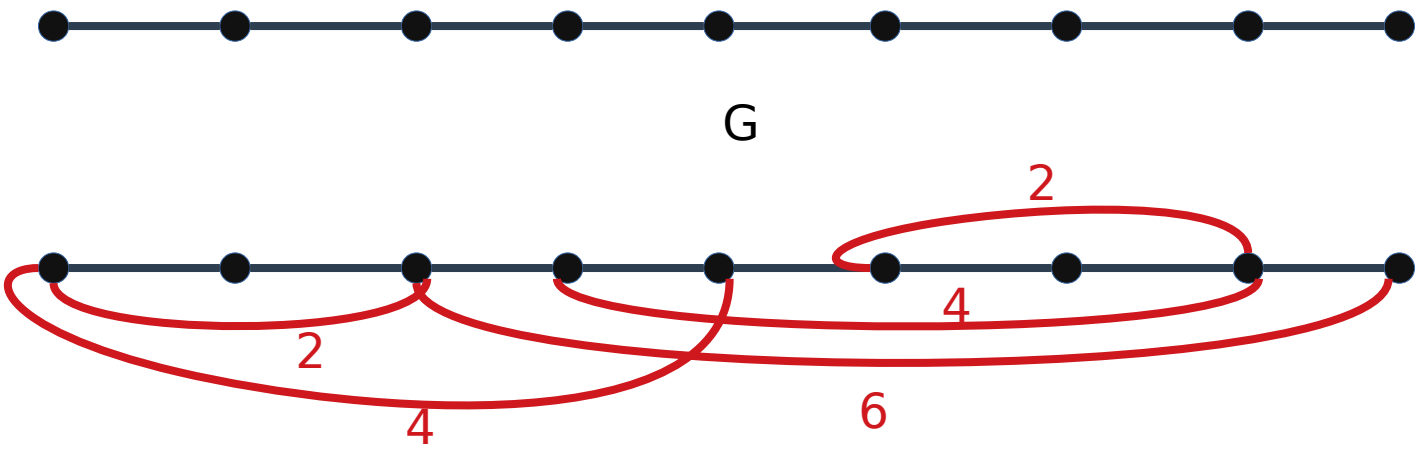
  - hopbound: β
  - Approx factor: $1 + \epsilon$

Hopsets

- **Given $G = (V, E, w)$, a (β, ϵ)-hopset H is a set of edges, s.t. between every pair of nodes u, v :**

$$d_G(u,v) \leq d^{(\beta)}_{G \cup H}(u,v) \leq (1+\epsilon) d_G(u,v)$$

  - Intuition: adding hopset edges is like adding shortcuts for reducing the diameter.



(3,0)-Hopset

Hopsets

- **Application: Given a (β, ϵ)-hopset H for G, we can compute approximate distances in β dist rounds.**
  - Run Bellman-Ford for β rounds to obtain approximate distances ($\beta \ll \mathrm{diam}$).
- **Goal:**
  - a sparse hopset, with small hopbound, often polylogarithmic in n, and fast construction.
  - Tradeoffs based on existential lower bounds

- **Previous:**
  - Intro to hopsets, and application in shortest path computation via Bellman-Ford.

- **Next:**
  - Using hopsets for computing distributed/parallel *distance sketches*

Distance Sketches

- **Distance sketches for a graph:**
  - Small information stored for each node, such that approximate distance of a pair u, v of nodes can be queried only using sketches of u and v.
- **Existing Distance Sketches** (Thorup-Zwick 05)**:**
  - Size: $\tilde{O}(n^{1/k})$ per node.
  - Stretch: $2k-1$ (approximation factor)
  - Query time (sequential): $O(k)$
  - Distributed/MPC query time: only 2 rounds!

Distance Sketches in MPC

- **Distance sketches in MPC:**
  - Distributed algorithm by Das Sarma et al (2015) will take $O(\mathrm{diam} \cdot n^{1/k})$ rounds.
  - Using hopsets we can compute distance sketches in $\tilde{O}(n^{1/k})$ rounds of MPC.
- **Can we construct distance sketches in polylogarithmic rounds?**
  - Yes, at the cost of a weaker stretch using spanners.

MPC Distance Sketches

Results	Size per node	Stretch	Time (rounds)
Das Sarma et al. (2015)	$\tilde{O}(n^{1/k}),$ $k \geq 2$	$(2k-1)$	$O(\text{diam} \cdot n^{1/k})$
DN 2019	$\tilde{O}(n^{1/k}),$ $k \geq 2$	$(2k-1)(1+\epsilon)$	$\tilde{O}(n^{1/k})$
DN 2019	$\tilde{O}(n^{1/k}),$ $k \geq 2$	$O(k^2)$	Polylogarithmic

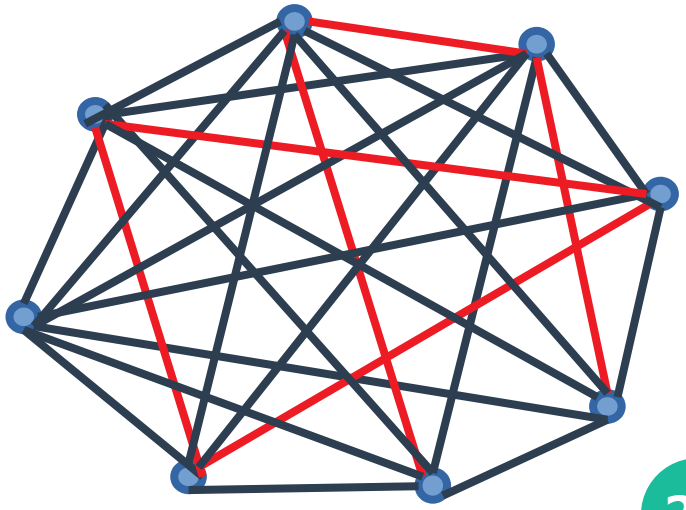
- **Previous:**
  - Using hopsets for computing massively parallel *distance sketches*
- **Next:**
  - Sparse hopsets in Congested Clique

Sparse Hopsets in Congested Clique

- **Goal: Distributed algorithm for constructing a sparse hopset**
  - Relevant paper: **N**, Sparse Hopsets in Congested Clique (OPODIS 2019).
  - Polylogarithmic round algorithm for sparse hopsets with polylogairthmic hopbound in the Congested Clique model

Congested Clique Model

- **Given a graph G=(V,E), each node can send a message of size O(log n) (congestion) to any other node.**
  - Initially each node knows the incident portion of the input, and should know incident part of the output.
  - **Goal:** minimize rounds of communication.

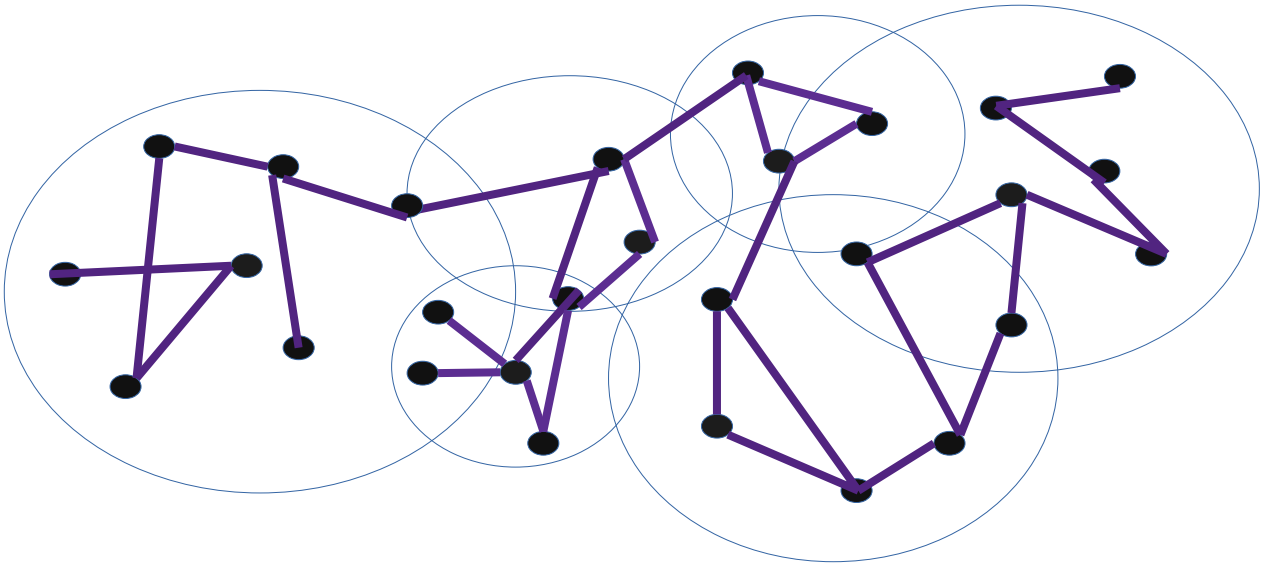


Hopsets in Congested Clique

Results	Size	Hopbound	Time (rounds)
Censor-Hillel et al. (2019)	$\tilde{O}(n^{3/2})$	$O(\log^2(n)/\epsilon)$	Polylogarithmic
Elkin-Neiman (2017, 2019)	$\tilde{O}(n^{1+1/k})$, $k \geq 2$	Polylogarithmic (func of k, ϵ)	Polynomial
N 2019	$\tilde{O}(n^{1+1/k})$, $k \geq 2$	Polylogarithmic (func of k, ϵ)	Polylogarithmic

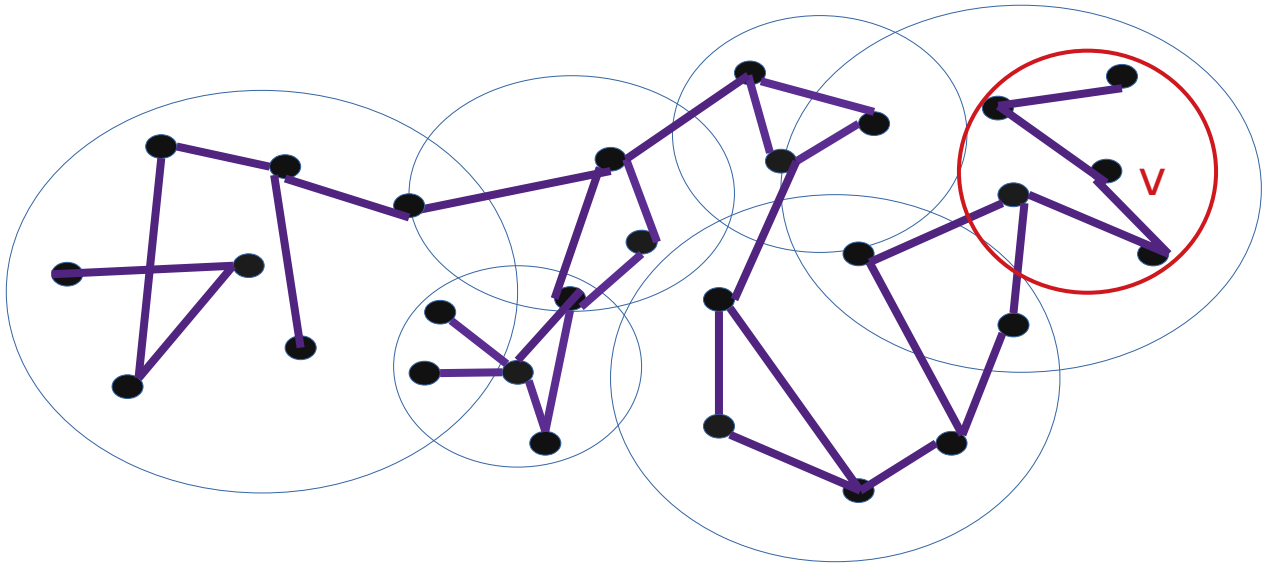
Neighborhood Covers

- **W-neighborhood cover: a clustering of nodes, s.t.**
  - Low diameter: Each cluster has diameter O(W log(n)).



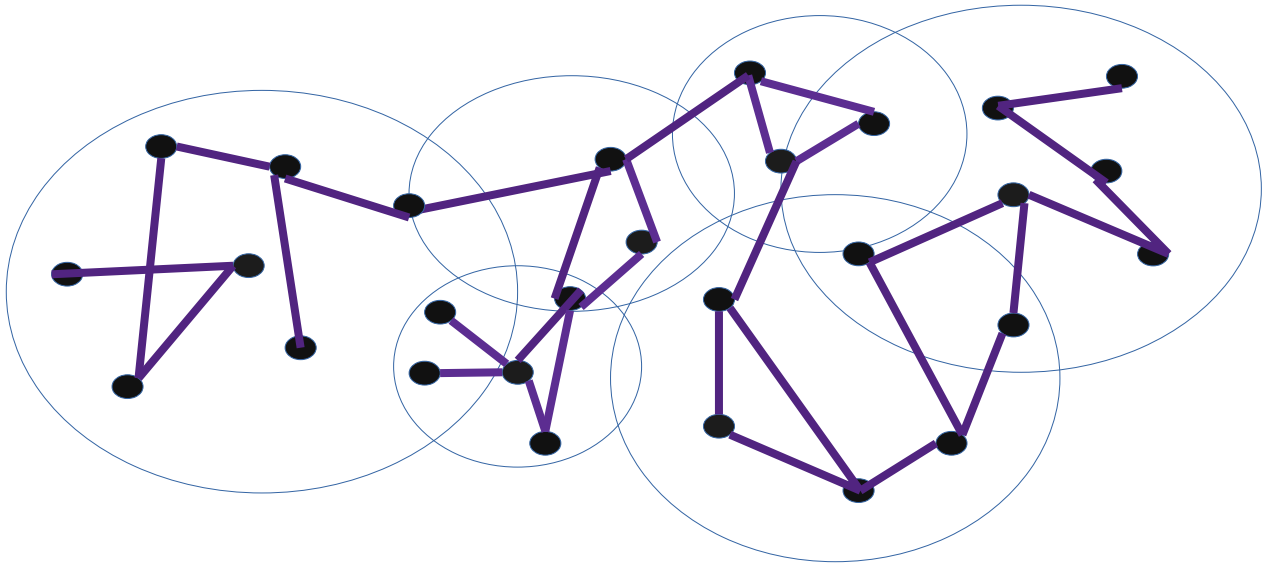
Neighborhood Covers

- **W-neighborhood cover: a clustering of nodes, s.t.**
  - Low diameter: Each cluster has diameter O(W log(n)).
  - Ball preservance: For each node v, W-neighborhood (ball of radius W) around v is contained in a cluster.



Neighborhood Covers

- **W-neighborhood cover: a clustering of nodes, s.t.**
  - Low diameter: Each cluster has diameter O(W log(n)).
  - Ball preservance: For each node v, W-neighborhood (ball of radius W) around v is contained in a cluster.
  - Low congestion/overlap: Each node overlaps with O(log(n)) clusters.

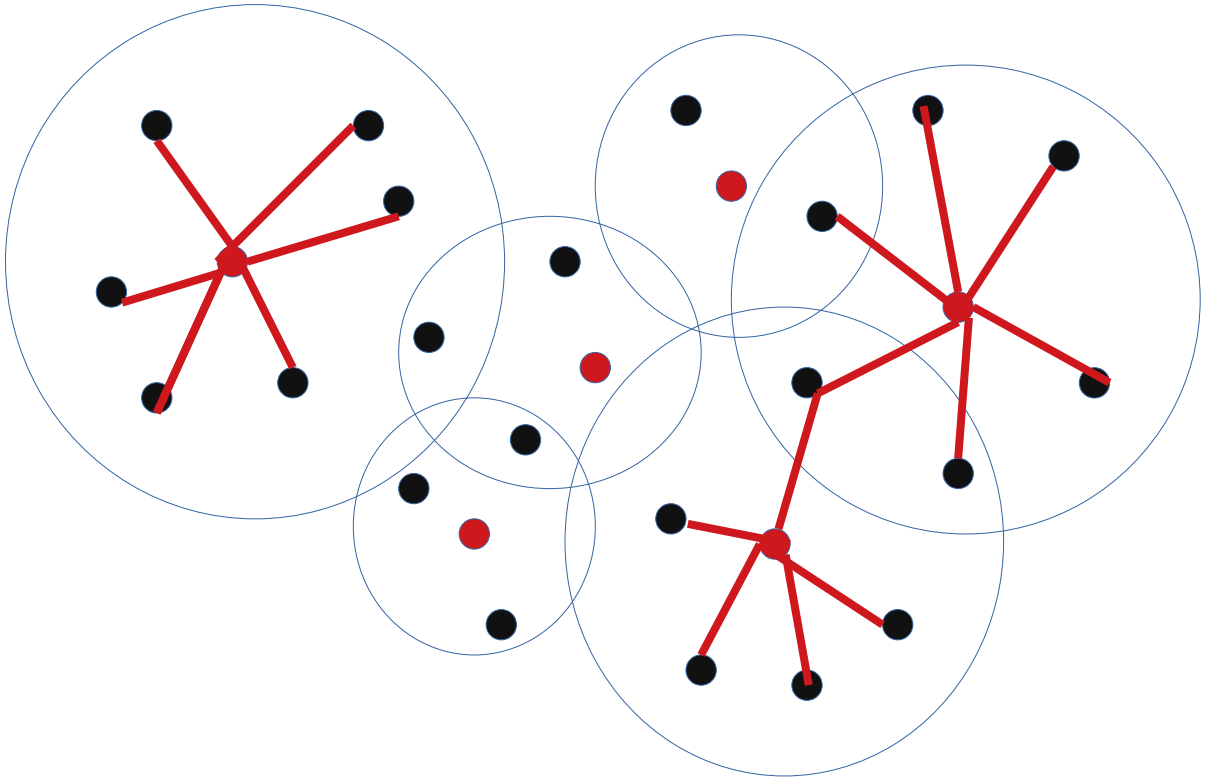


Hopset construction

- **Centralized algorithm inspired by (Cohen 2000).**
- **Each iteration handles pairs of nodes u,v with distances $(R, 2R]$:**
  - Compute W-neighborhood covers for $W = \epsilon R/4 \log(n)$
  - Clusters are small if their size is less than $\sqrt{n}$, and big otherwise.
  - Add a star rooted at the center of big clusters.
  - Add all pairwise edges (clique) between all big cluster centers.
  - A clique for each small cluster (too dense).

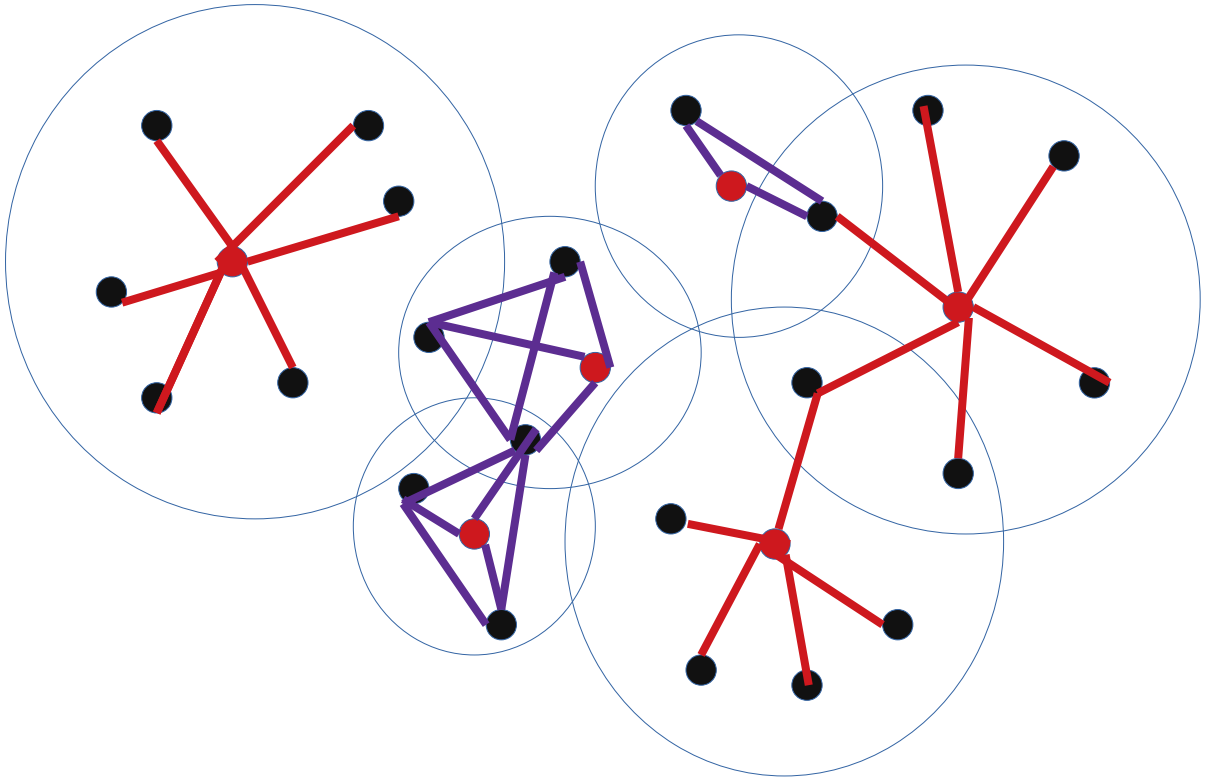
Hopset construction

- Add a star rooted at the cluster center of big clusters.
  - Set weight of an edge (u,v) in H to distance between u and v in G .



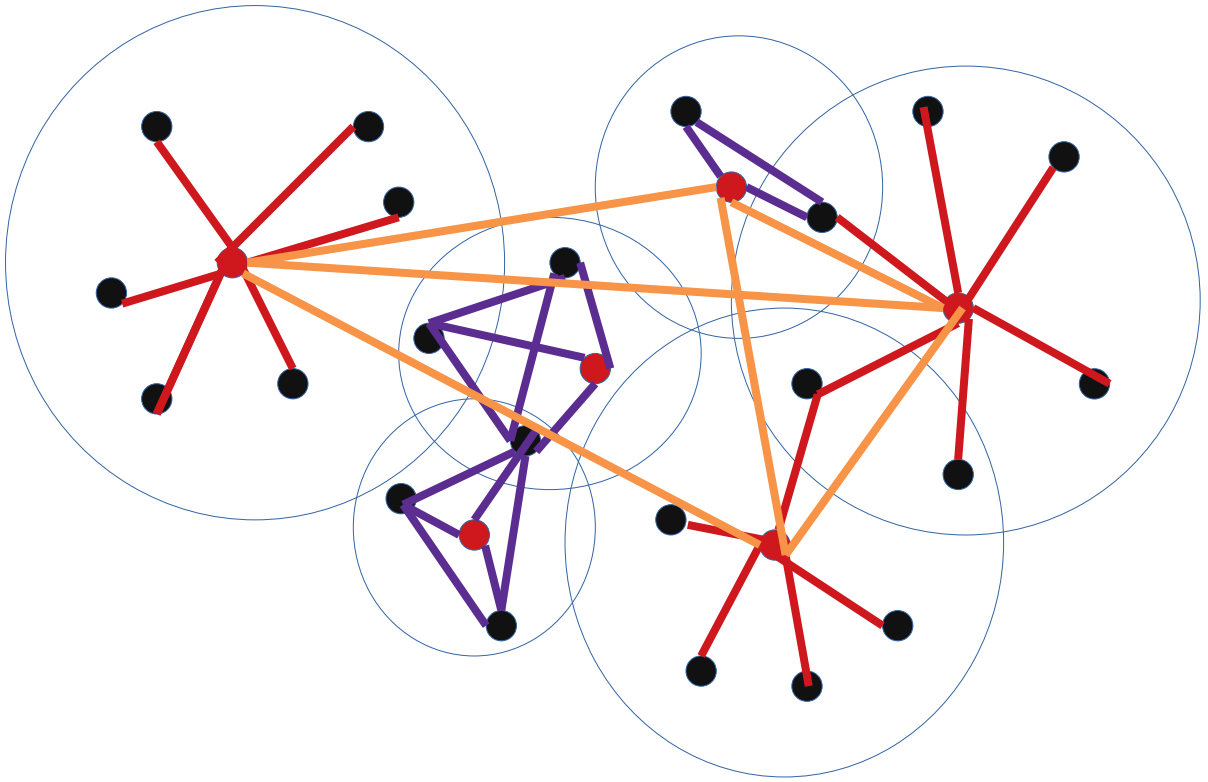
Hopset construction

- Centralized: a clique for each small clusters.
- Distributed: replaced with a hopset with constant hopbound.



Hopset construction

Add a clique between all big cluster centers.



New sparse distributed hopset

- **Goal: sparser hopset and faster construction**
- **For distance scale $(R, 2R]$:**
  - Compute W-neighborhood covers for $W = \epsilon R/4 \log(n)$
  - Clusters are small if their size is less than $\sqrt{n}$, and big otherwise.
  - Add a star rooted at the center of big clusters.
  - Add a clique between all big cluster centers.
  - Locally construct a sparse hopset for each small cluster (leads to improvements in Congested Clique).

Size analysis

- **For a small clusters of size in $[s, 2s)$:**
  - We added $O(s^{1+1/k})$ edges (size of local hopsets).
  - There are at most $O(n/s)$ such clusters.
  - Summing over all size buckets: $O(n^{1+1/2k})$

Size analysis

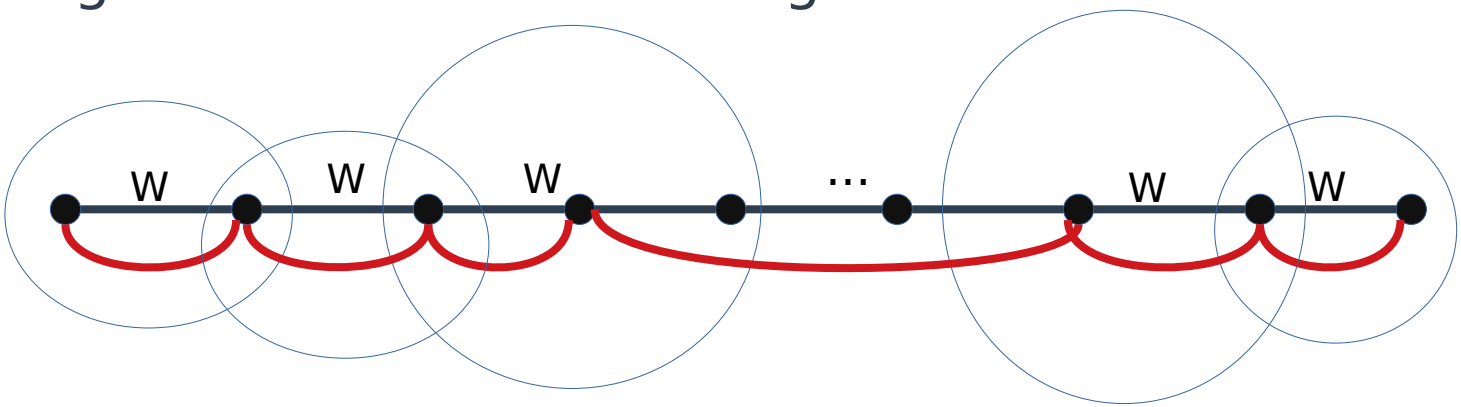
- **For a small clusters of size in $[s, 2s)$:**
  - We added $O(s^{1+1/k})$ edges (size of local hopsets).
  - There are at most $O(n/s)$ such clusters.
  - Summing over all size buckets: $O(n^{1+1/2k})$
- **Star edges for each big cluster:**
  - adds at most $O(\log(n))$ forests for each scale.

Size analysis

- **For a small clusters of size in $[s, 2s)$:**
  - We added $O(s^{1+1/k})$ edges (size of local hopsets).
  - There are at most $O(n/s)$ such clusters.
  - Summing over all size buckets: $O(n^{1+1/2k})$
- **Star edges for each big cluster:**
  - adds at most $O(\log(n))$ forests for each scale.
- **Clique edges between big cluster centers.**
  - At most $O(n)$ edges in total for each scale.
- **Log factor added to cover all scales.**

Hopbound and Stretch analysis

- **Path of length (R,2R] divided into $O(\log(n)/\epsilon)$ equal length segments.**
  - Each is contained in a cluster, (parameter of neighborhood covers is $W = \epsilon R/\log(n)$).
- **Hop bound:** $O(\log(n)/\epsilon)$
  - small clusters: constant hops (local hopset construction)
  - big clusters: one direct edge.



Congested Clique Implementation

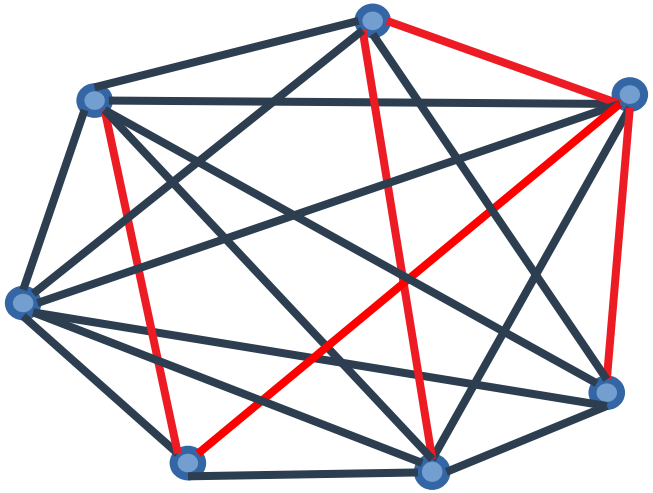
- **Constructing W-neighborhood covers**
  - Known constructions are too slow for large W.
  - Use a relaxed notion of limited neighborhood covers that can be implemented efficiently.
- **Local hopsets for small clusters**
  - Collecting local topology and local computation by cluster center.
  - Message routing algorithm by Lenzen (2013)
- **Adding a clique between big cluster centers**
  - $(1+\epsilon)$-MSSP algorithm by Censor-hillel et al. (2019).

- **Previous:**
  - Efficient construction of sparse hopsets in Congested Clique

- **Next:**
  - Dynamic hopsets with applications in near-optimal shortest path computation
  - J. Łącki and **N**, *Faster Decremental Approximate Shortest Paths via Hopsets with Low Hopbound*.

Dynamic Model

- **Dynamic graph algorithms**
  - Input changes over time (insertions or deletions)
- **Goal:**
  - Fast queries
  - small update time
- **Partially dynamic**
  - Insert only (incremental) or delete only (decremental)
  - This work: decremental (deletion and weight increase)



Applications of Dynamic Hopsets

- **Dynamic tools:**
  - Even-Schiloach’81: decremental maintainenace of distance up to distance d in O(md) time.
  - Can turn this to maintain hop bounded distance in O(mh) time
- **Dynamic All-Pairs Distance Oracles :**
  - Maintain a hopset with polylogarithmic hopbound
  - Use the hopset to maintain distance oracles more efficiently

Thank you!
Questions?