Distributed, Parallel and Dynamic Graph Algorithms

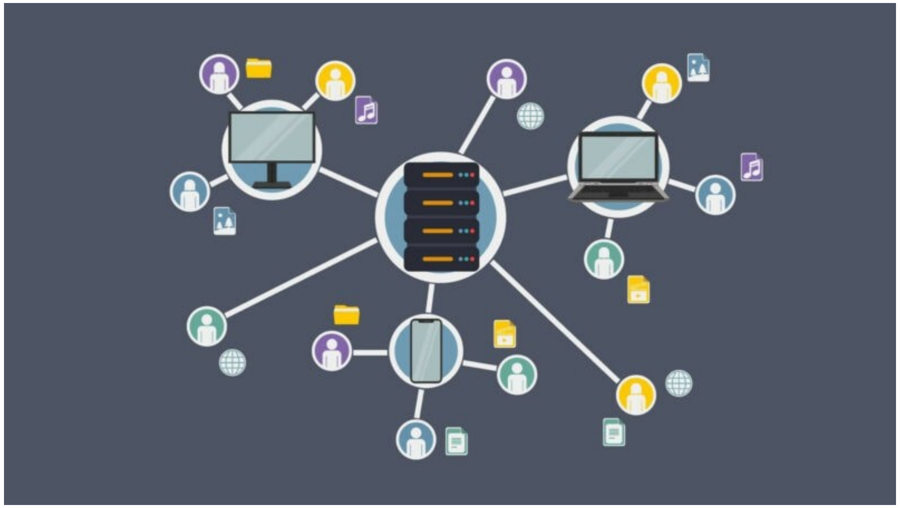
Yasamin Nazari

VU Amsterdam

Dutch Optimization Seminar
May 2023

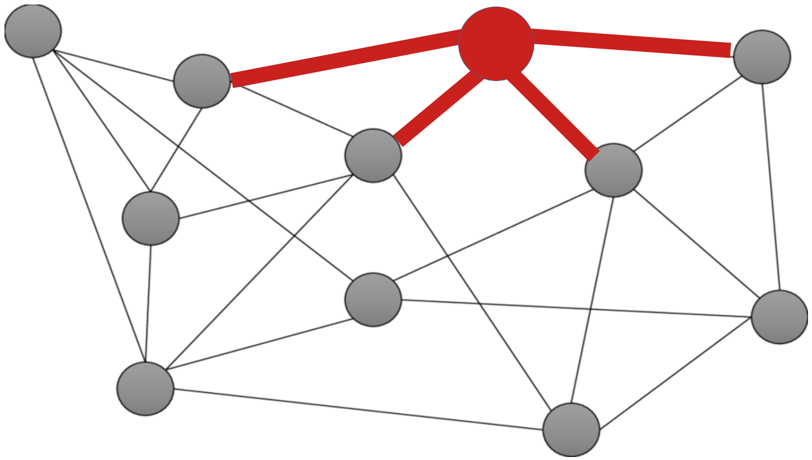
Computational Models

- **Theoretical** **models inspired by** **big data** **and modern computing systems:**
  - Distributed models (computation over networks)
  - Massively parallel computation
  - Dynamic models (changing input)
  - Fault tolerance (infrastructure)



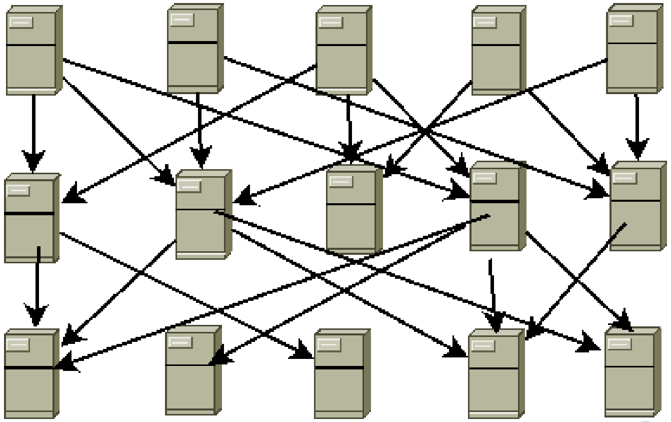
Distributed Models

- **Distributed Models**
  - Motivated e.g. by routing and broadcast on networks
  - Examples: LOCAL, CONGEST, Congested Clique,...
- **LOCAL model**
  - Given a graph G=(V,E), in each round each node sends a message to its neighbors.
  - Goal: minimize number of rounds communication.



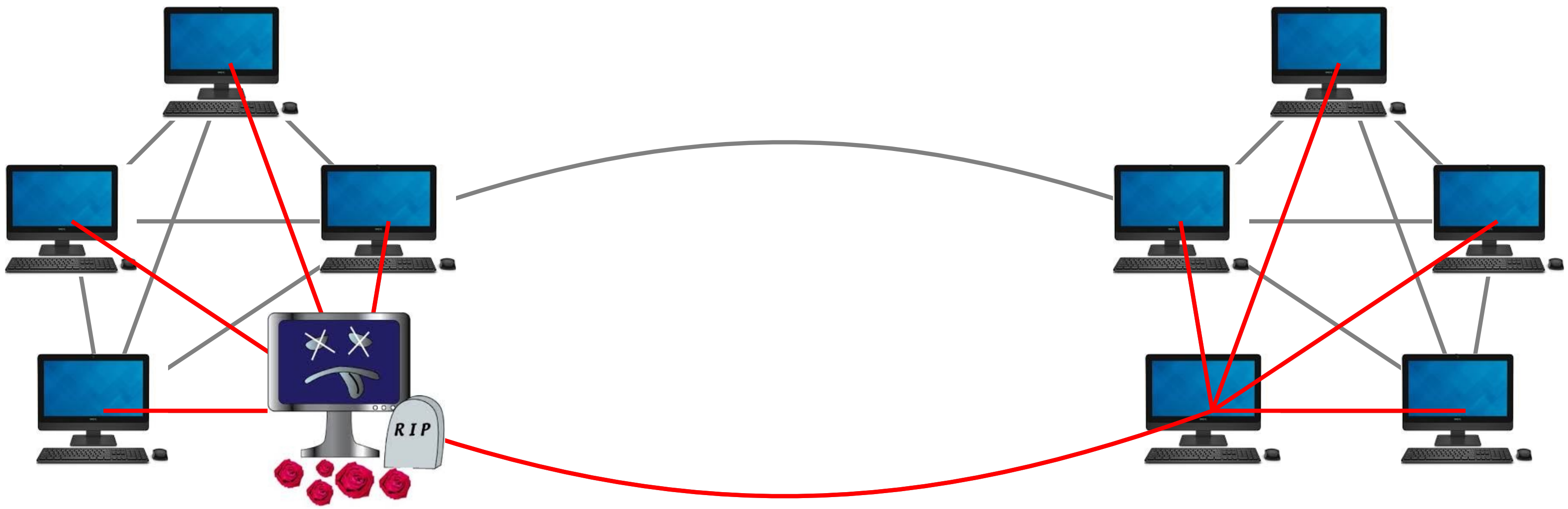
Massively Parallel Computation (MPC)

- Abstraction of modern platforms e.g. MapReduce, Spark, Hadoop
- **MPC Model:**
  - Input is distributed over a set of machines.
  - Each machine memory/communication: strictly sublinear in input size.
- Connections to both classical parallel models (PRAM) and distributed models



Fault-tolerance

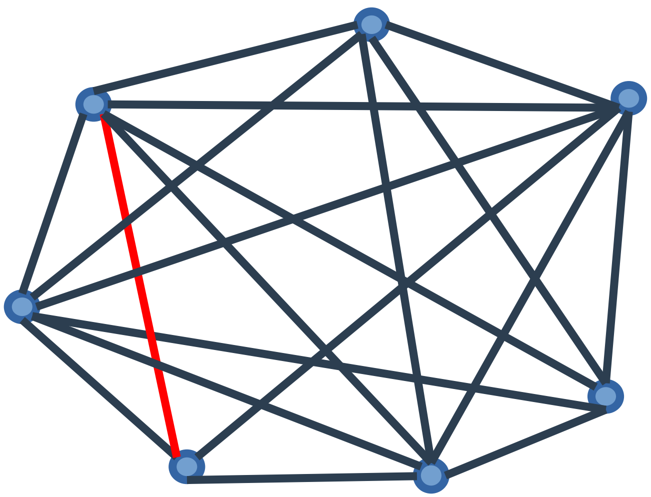
- **Fault-tolerant graph algorithms**
  - Valid solution after up to f (edge or vertex) faults



Dynamic Model

- **Dynamic Graphs**
  - Updates to input:

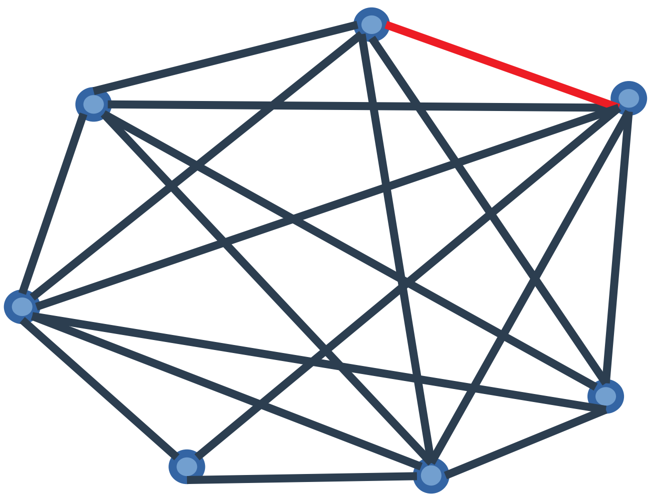
    (edge insertions, deletions)



Dynamic Model

- **Dynamic Graphs**
  - Updates to input:

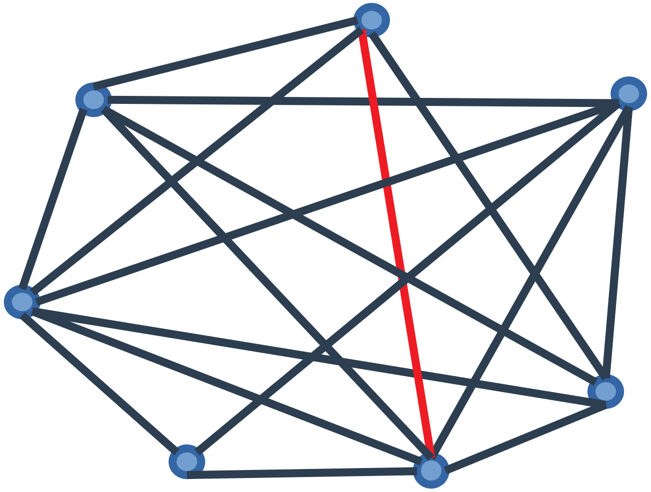
    (edge insertions, deletions)



Dynamic Model

- **Dynamic Graphs**
  - Updates to input:

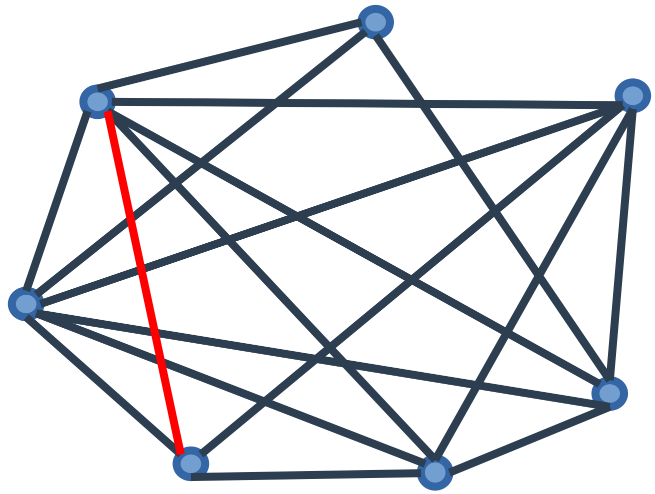
    (edge insertions, deletions)



Dynamic Model

- **Dynamic Graphs**
  - Updates to input:
    (edge insertions, deletions)

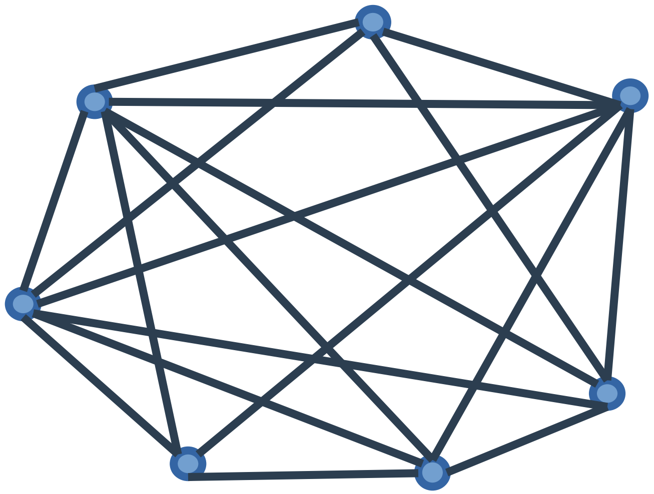
- **Goal**
  - Fast queries
  - Small update time



Dynamic Model

- **Dynamic Graphs**
  - Updates to input:

    (edge insertions, deletions)



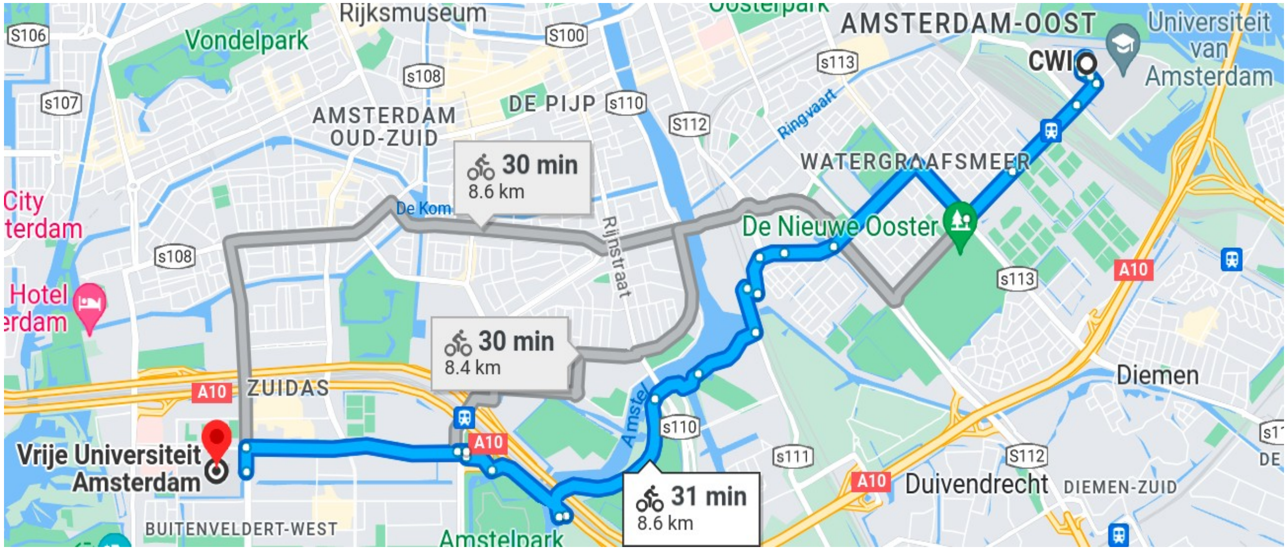
- **Goal**
  - Fast queries
  - Small update time

- **Partially or fully dynamic**
  - Insert-only (incremental), delete-only (decremental)

- **Previous:**
  - Models
- **Next:**
  - Distane Computation and Structures

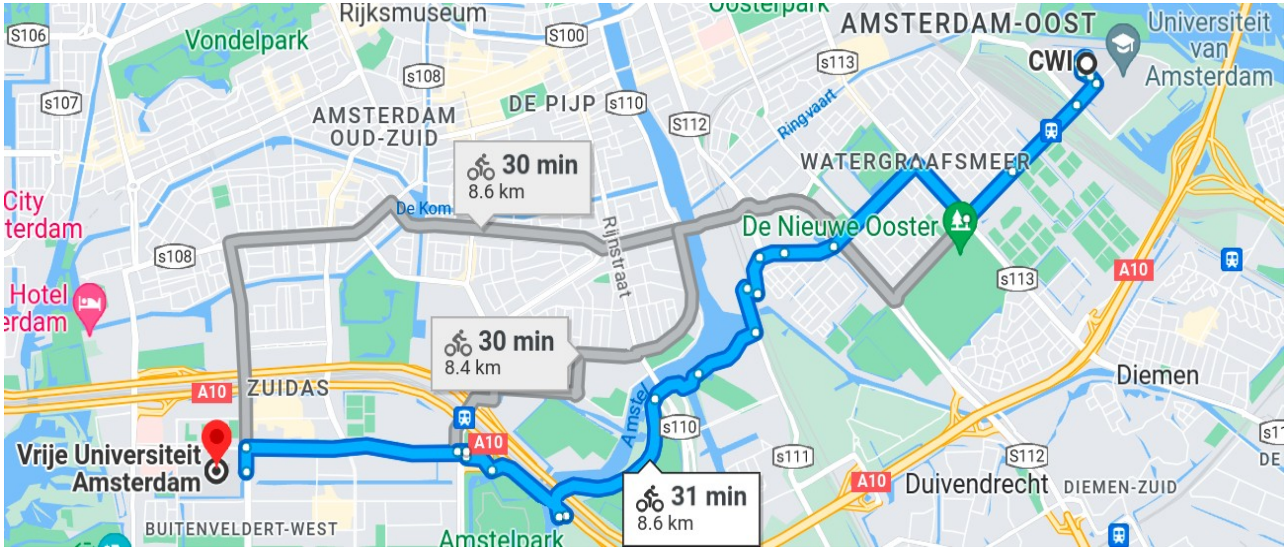
Distances in Graphs

- **Distance computation:** Given a graph **`G=(V,E)`**, compute (approx) distances between a set of sources and destinations.
  - Sequential: Dijkstra's (single source)



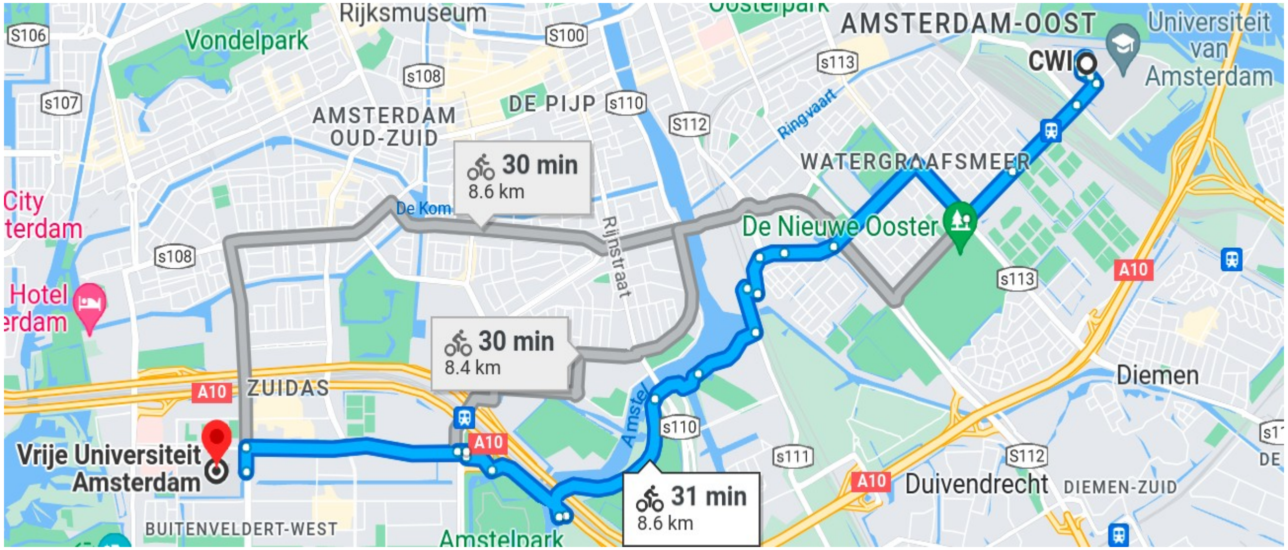
Distances in Graphs

- **Distance computation:** Given a graph `G=(V,E)`, compute (approx) distances between a set of sources and destinations.
  - Sequential: Dijkstra's (single source)
  - Distributed: BFS/Bellman-Ford
    - Slow if diameter is large



Distances in Graphs

- **Distance computation:** Given a graph `G=(V,E)`, compute (approx) distances between a set of sources and destinations.
  - Sequential: Dijkstra's (single source)
  - Distributed: BFS/Bellman-Ford
    - Slow if diameter is large
  - Dynamic:
    - Slow to recompute from scratch



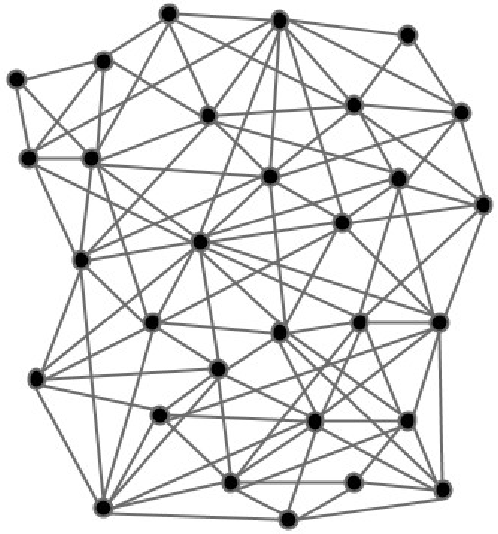
Distance Structures

- **Approximate distance sparsification**
  - Spanners
  - Emulators
  - Distance sketches/oracles
- **Hopsets**
  - Shortcut edges reducing number of hops in shortest paths

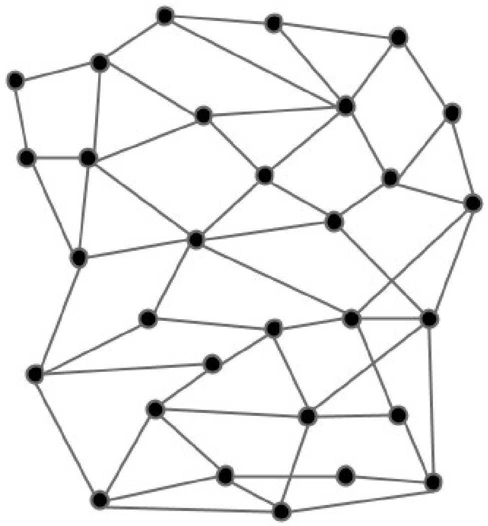
Spanners/ Emulators

Spanners: sparse subgraphs that approximately preserve distances

Emulators: no need to be a subgraph



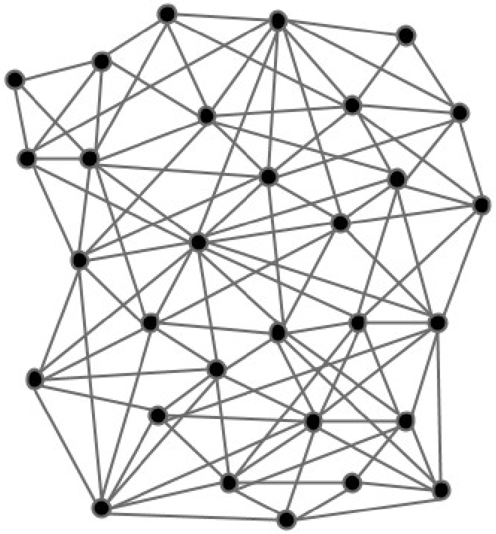
G



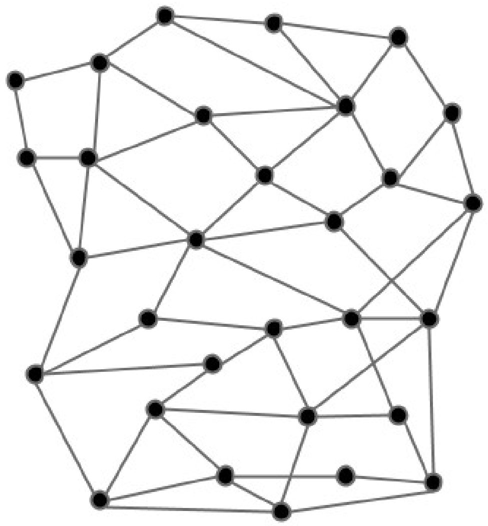
3-Spanner

Spanners

[Althöfer et al.,93]: Every undirected graph has a $(2k-1)$-spanner of size $O(n^{1+1/k})$ for all $k \geq 2$.



G

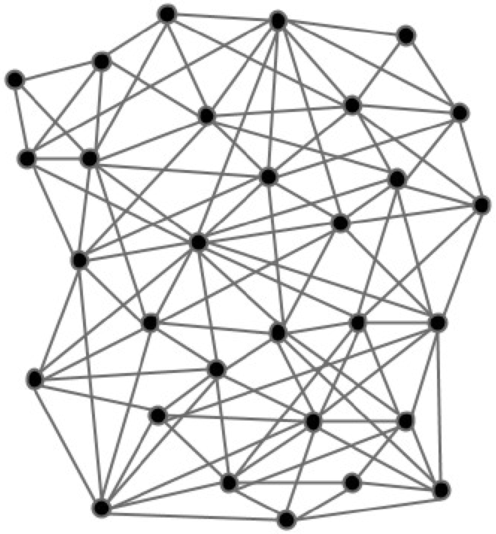


3-Spanner

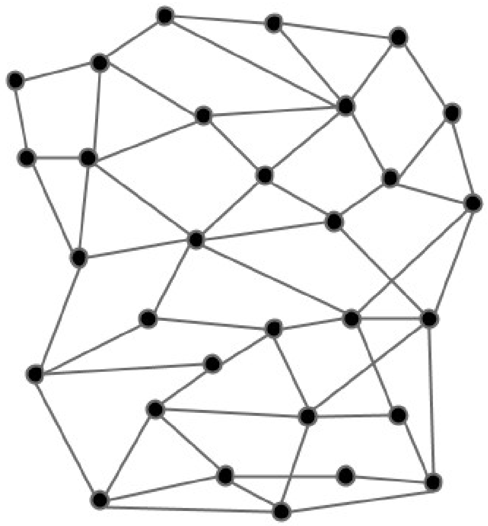
Spanners

[Althöfer et al.,93]: Every undirected graph has a $(2k-1)$-spanner of size $O(n^{1+1/k})$ for all $k \geq 2$.

Optimal (conditional on Erdős girth conjecture)



G



3-Spanner

Spanners

Greedy

Process edges in non-decreasing order of weight:

- Add edge (u, v) if there is no path of length $\leq k.w(u, v)$ so far

Existentially optimal!

Spanners

Greedy

Process edges in non-decreasing order of weight:

- Add edge (u, v) if there is no path of length $\leq k.w(u, v)$ so far

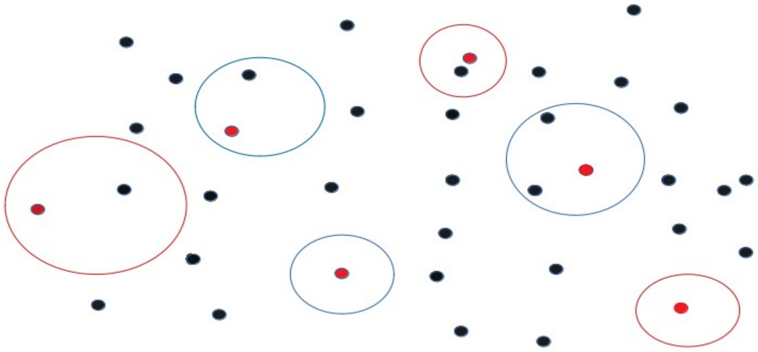
Existentially optimal!

Sparsity: No cycles of length less than k

Spanner Algorithms

Clustering based

- [BS07] distributed/parallel and later dynamic
- More efficient than greedy
- Subsequent sampling and growing clusters



Applications

- **Massively Parallel Distances**
  - Fast approximate APSP via spanners [BDGM**N**, SPAA 21]
- **Incremental (insert-only) shortest paths**
  - Polylog approx and amortized update time [F**N**P, STOC 23] via emulators
- **Fully-dynamic model**
  - Emulators + algebraic data structures [BF**N**, FOCS 22]
- **Fault tolerant emulators**
  - Motivated by routing in overlay networks [BD**N**, ITCS 22; BD**N**, ITCS 23]

- **Previous:**
  - Spanners and emulators
- **Next:**
  - Parallel shortest paths via hopsets

Motivation: Bellman-Ford

- **Single-source shortest path via Bellman-Ford:**
  - Nodes update their distance estimate from the source s by

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

- **Each iteration single distributed/parallel round.**
  - How many iterations do we need?

Motivation: Bellman-Ford

- **Single-source shortest path via Bellman-Ford:**
  - Nodes update their distance estimate from the source s by

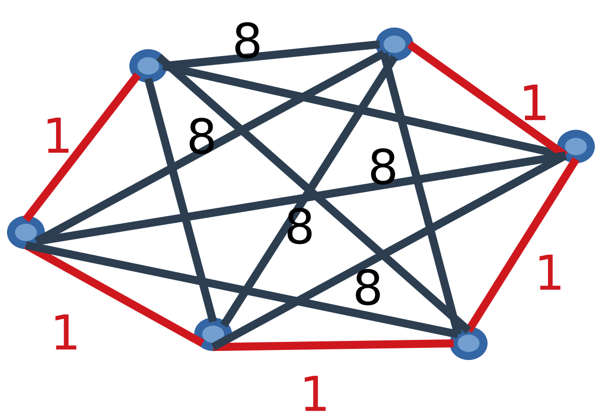
$$\tilde{d}(v, s) = \min_{u \in N(v)} \tilde{d}(u, s) + w(u, v)$$

- **Each iteration single distributed/parallel round.**
  - How many iterations do we need?
  - h iterations to compute $d_G^{(h)}(s, v)$ for all $v \in V$ (shortest distance using *h*-hop paths only).

Motivation: Bellman-Ford

Bellman-Ford from single source s:

- h iterations to compute $d_G^{(h)}(s,v)$ for all $v \in V$
- Requires $O(\mathrm{diam})$ iterations
  - diam is maximum number of hops in the shortest paths. Could be as large as $\Omega(n)$.

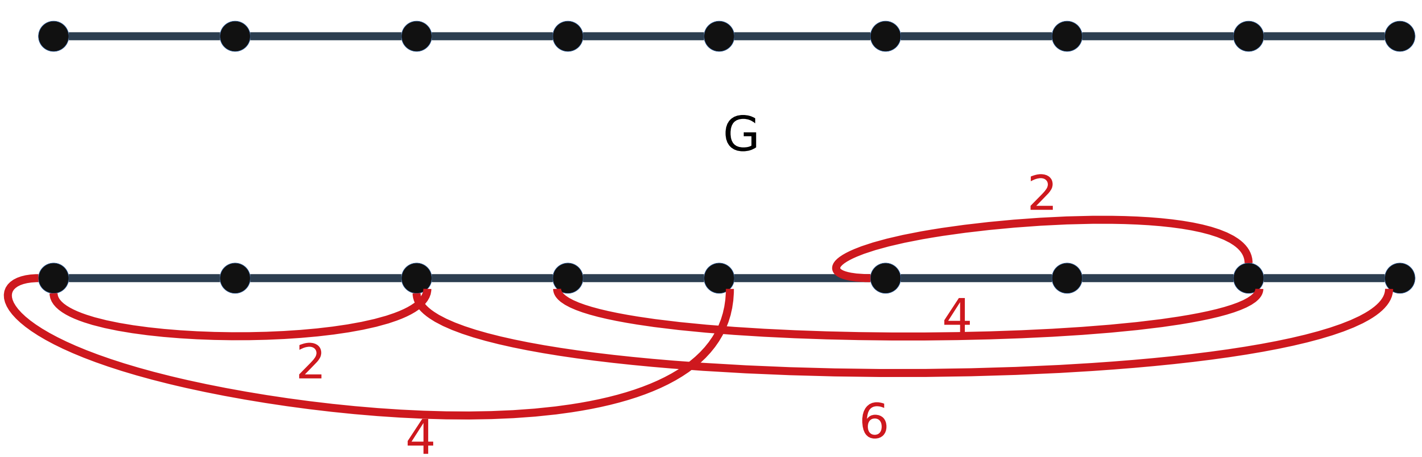


Hopsets

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

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

  – Intuition: adding shortcut edges for reducing the diameter.



(3,0)-Hopset

Hopsets: Parallel Shortest paths

- **Distributed/parallel SSSP** Given a (β, ϵ)-hopset H for G, approximate distances take β rounds.
  - Run Bellman-Ford for β rounds to obtain $(1+\epsilon)$-approx distances ($\beta \ll \mathrm{diam}$ e.g. polylogarithmic).

Hopsets: Parallel Shortest paths

- **Distributed/parallel SSSP** Given a (β, ϵ)-hopset H for G, approximate distances take β rounds.
  - Run Bellman-Ford for β rounds to obtain $(1+\epsilon)$-approx distances ($\beta \ll \text{diam}$ e.g. polylogarithmic).

- **Dynamic (delete-only) SSSP**
  - h-hop-bounded $(1+\epsilon)$-single source distances in $O(h)$ amortized time [ES98, B11].
  - Less immediate: (β, ϵ)-hopset for $(1+\epsilon)$-SSSP in $O(\beta)$ amortized time.

Hopsets

Goal: Fast construction of sparse hopset with small hopbound.

Existential bounds

- Upper bound [EN19-HP19]: any undirected graph has a (β, ϵ)-hopset of size $\tilde{O}(n^{1+1/k})$ with $\beta = O(1/\epsilon)^k$
- Lower bound [ABP19]: Cannot have both linear size and polylogarithmic hopbound

Hopset Algorithms

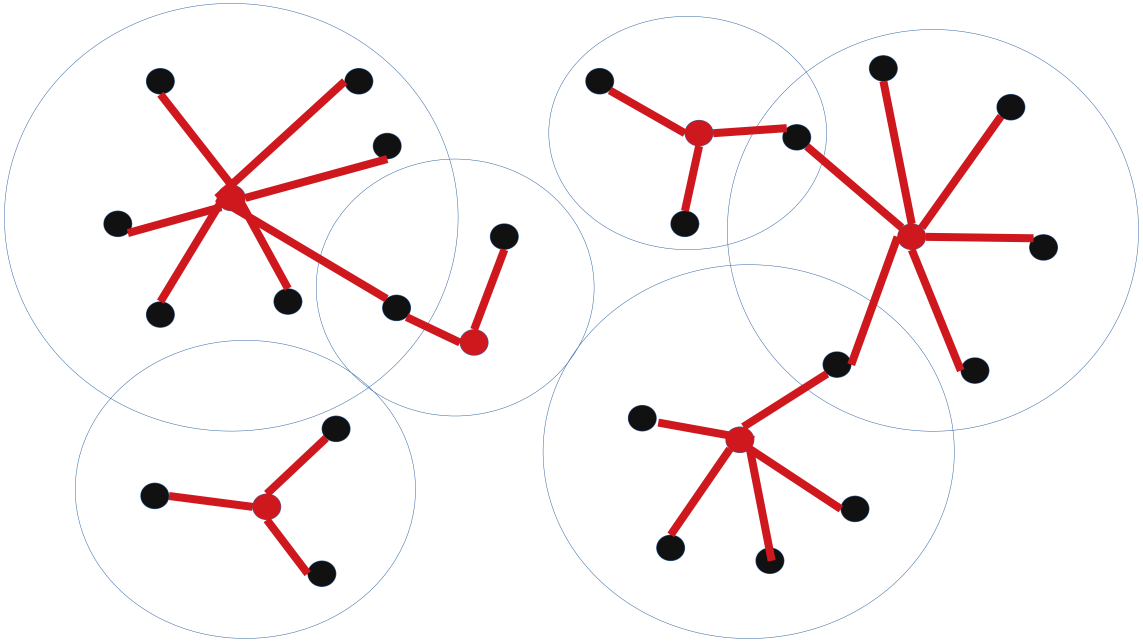
Algorithm structure of [Coh00, HKN14, EN16, EN19]

General structure:

- Covering graph with low overlapping clusters with known centers
- Add edges (weight corresponding to dist of endpoints)
  - From each center to all nodes in the cluster
  - Inter-cluster edges between some centers

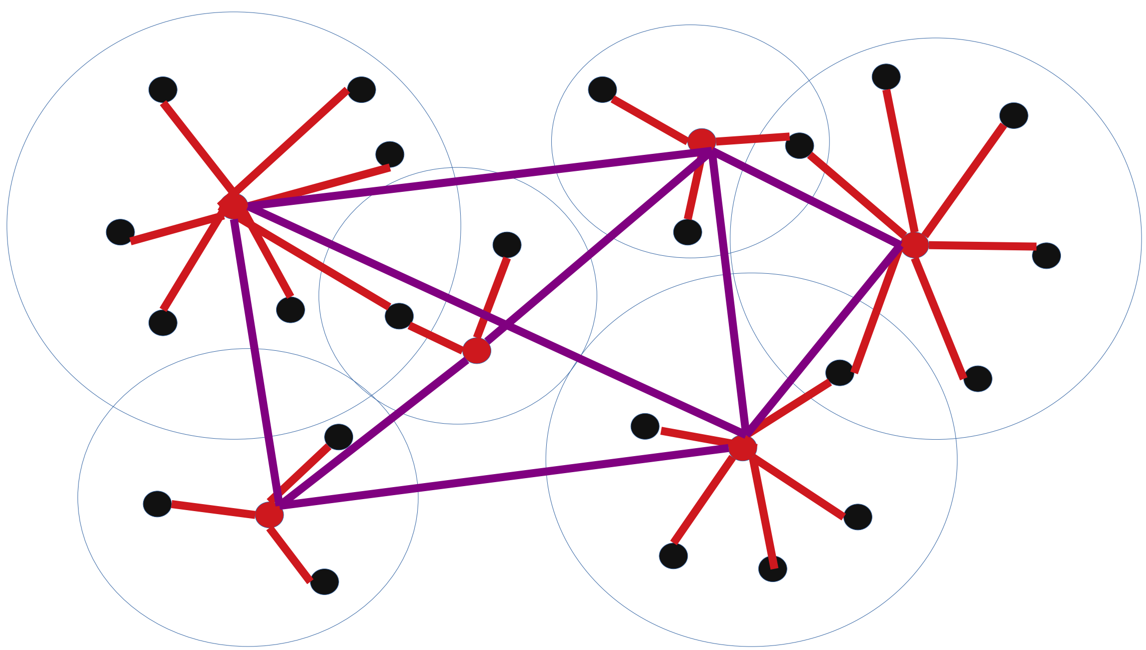
Hopset Structure

- Set of clusters with centers
- Edges (weighted by distance) are added inside each cluster



Hopset Structure

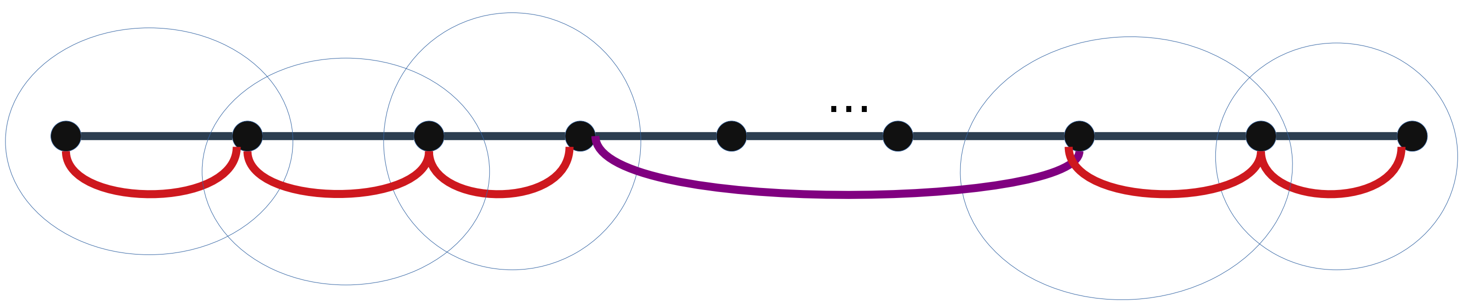
- Set of clusters with centers
- Edges (weighted by distance) are added inside each cluster
- Inter-cluster edges between centers



Intuition

Clustering property:

- Path segments are either covered by edges inside clusters or
- Inter-cluster edges shortcut the segments not covered.



Computational challenges

- **Dynamic**
  - Nodes keep on changing clusters
  - Even in delete-only settings there are insertions

Computational challenges

- **Dynamic**
  - Nodes keep on changing clusters
  - Even in delete-only settings there are insertions
- **Distributed/parallel**
  - Low congestion cluster growing

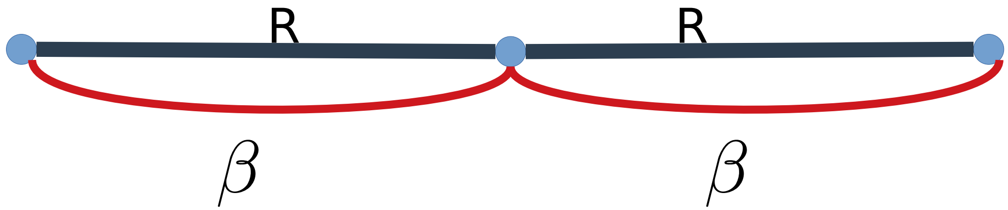
Computational challenges

- **Dynamic**
  - Nodes keep on changing clusters
  - Even in delete-only settings there are insertions
- **Distributed/parallel**
  - Low congestion cluster growing
- **General challenge**
  - Distances (for weights) for distances
  - Chicken and egg problem?

Computational challenges

Algorithmic idea

- First in PRAM [Coh00], distributed [EN16, EN19], dynamic [**ŁN**22]
- Assume (β, ϵ)-hopset edges are added up to distance R. We can look at 2β hops for distances up to 2R.



Hopset Applications

- **Parallel and distributed shortest paths**
  - $(1+\epsilon)$-SSSP in polylog rounds via hopsets with polylog hopbound
  - Fast computation of distance sketches supporting constant round approx all pair queries [D**N**19]

- **Dynamic (delete-only) shortest paths**
  - $(1+\epsilon)$-SSSP in $n^{o(1)}$ amortized update time [HKN16, Che19, Ł**N**22]
  - $O(k)$-APSP in $\tilde{O}(n^{1/k})$ amortized update time [Ł**N**22]

- **Previous:**
  - Hopsets and applications
- **Next:**
  - Fully dynamic approximate distances

Ideal Guarantees

- **Limitations of previous work**
  - Support only deletions or only insertions
  - Amortized guarantees

Ideal Guarantees

- **Limitations of previous work**
  - Support only deletions or only insertions
  - Amortized guarantees
  - Power of adversary
    - Assume adversary is oblivious to random choices

Ideal Guarantees

- **Limitations of previous work**
  - Support only deletions or only insertions
  - Amortized guarantees
  - Power of adversary
    - Assume adversary is oblivious to random choices

Goal: Fully-dynamic deterministic algorithms with optimal worst-case running time.

Deterministic Fully-Dynamic Distances

Goal: Fully-dynamic algorithms that have worst-case guarantees are deterministic.

Theorem [B**N**F, FOCS 22]:

Fully-dynamic deterministic $(1+\epsilon)$-approximate single-source and *st*-distances with conditionally optimal worst-case bounds in unweighted undirected graphs.

Deterministic Fully-Dynamic Distances

Approx.	Type	Update Time
$1+\epsilon$	single pair	$O(n^{1.407})$
$1+\epsilon$	single source	$O(n^{1.529})$
$1+\epsilon$	k sources	$O(n^{1.529} + kn^{1+o(1)})$
$1+\epsilon$	all pairs	$O(n^{2+o(1)})$

Conditional optimality: Based on an OMV-based hardness assumption by [BNS, FOCS 19]

Deterministic Fully-Dynamic Distances

- **Technical idea:** combination of two type of fully-dynamic data structures
  - Combinatorial structures (sparse emulators)
  - Algebraic data structures

Deterministic Fully-Dynamic Distances

- **Technical idea:** combination of two type of fully-dynamic data structures
  - Combinatorial structures (sparse emulators)
  - Algebraic data structures
- **Speed up:**
  - Static distance queries on a sparse graph
  - Faster bounded distance using algebraic structures

Algebraic Data Structures

Graph distances via matrix inverse

Adjacency matrix A

$A^k_{i,j}$:= walks from i to j of length k

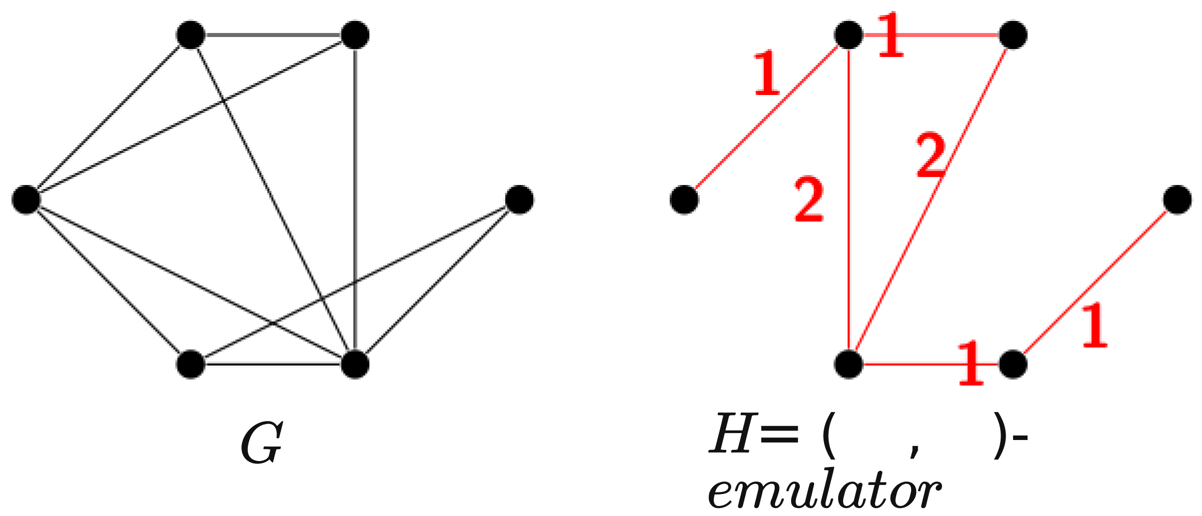
$$(I - A \cdot X)^{-1} = \sum_{k=1}^{n-1} A^k X^k$$

- **Algebraic data structures**
  - Matrix inverse used for distances previously [San05, BN19]
  - Faster algorithm based on properties of our emulators

Sparse Emulators

Emulators: Given a graph G=(V,E), an (α, β)-emulator is a sparse graph H such that:

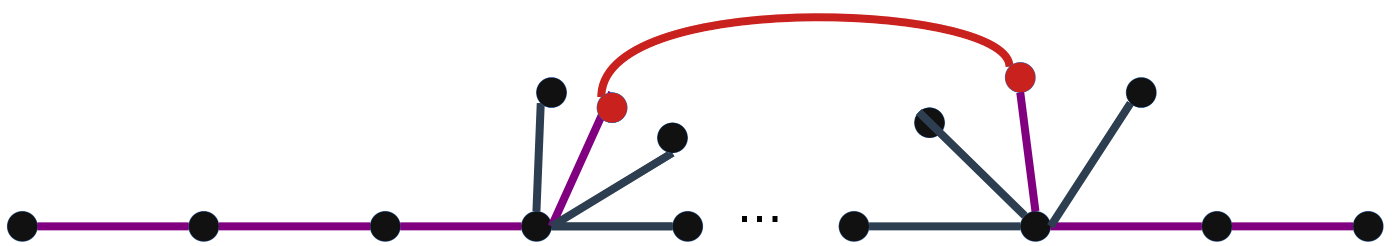
$$\forall u, v \in V : d_G(u,v) \leq d_H(u,v) \leq \alpha d_G(u,v) + \beta$$



Sparse Emulators

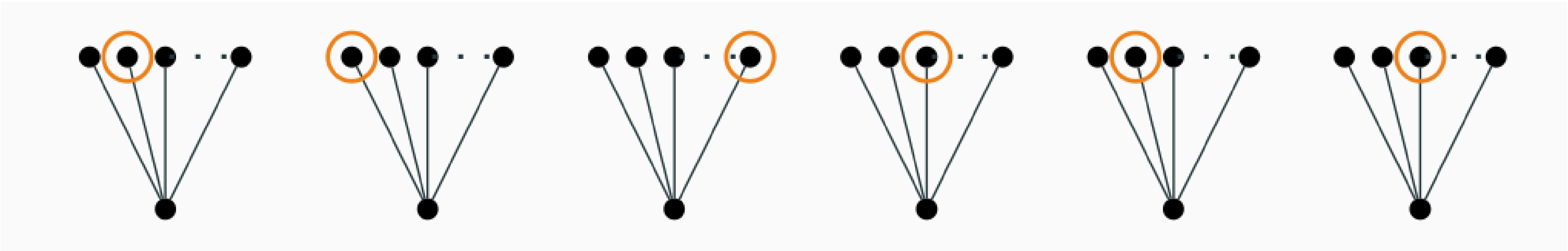
Simple $(1+\epsilon, 4)$ -emulator of size $O(n^{4/3})$:

- Low degree $\leq n^{1/3}$ nodes: all incident edges
- High degree $> n^{1/3}$ nodes: an edge corresponding to one neighbor in a hitting set **S**
- Weighted edges between nodes in **S** bounded by $O(1/\epsilon)$



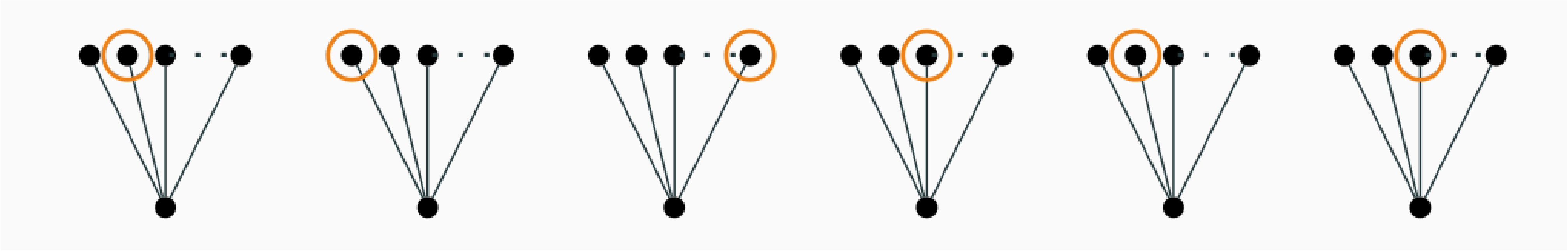
Hitting set

- **Hitting set:**
  - If randomness allowed: fixed set of sampled nodes



Hitting set

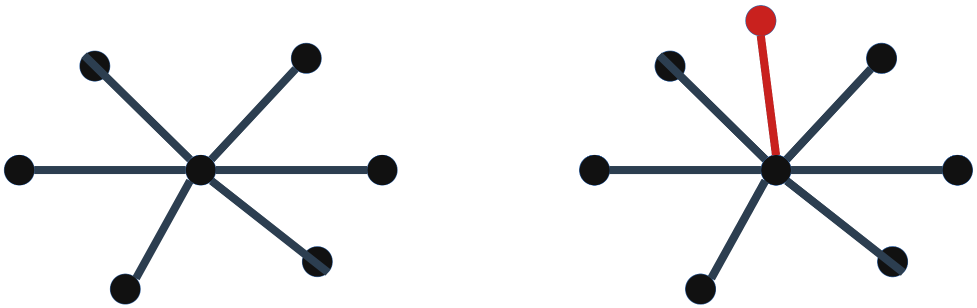
- **Hitting set:**
  - If randomness allowed: fixed set of sampled nodes
  - Deterministic challenge: set of sources change, but slowly



Deterministic Hitting sets

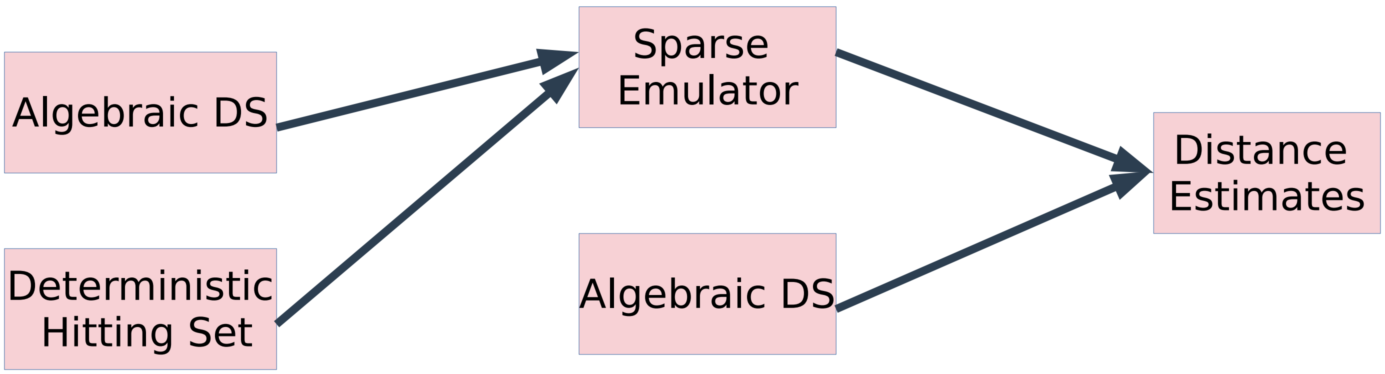
Deterministic low recourse approximate hitting set

- Edge update: one node added or deleted
- When size doubles recompute (done slowly for worst-case bound)



Deterministic Fully-Dynamic Distances

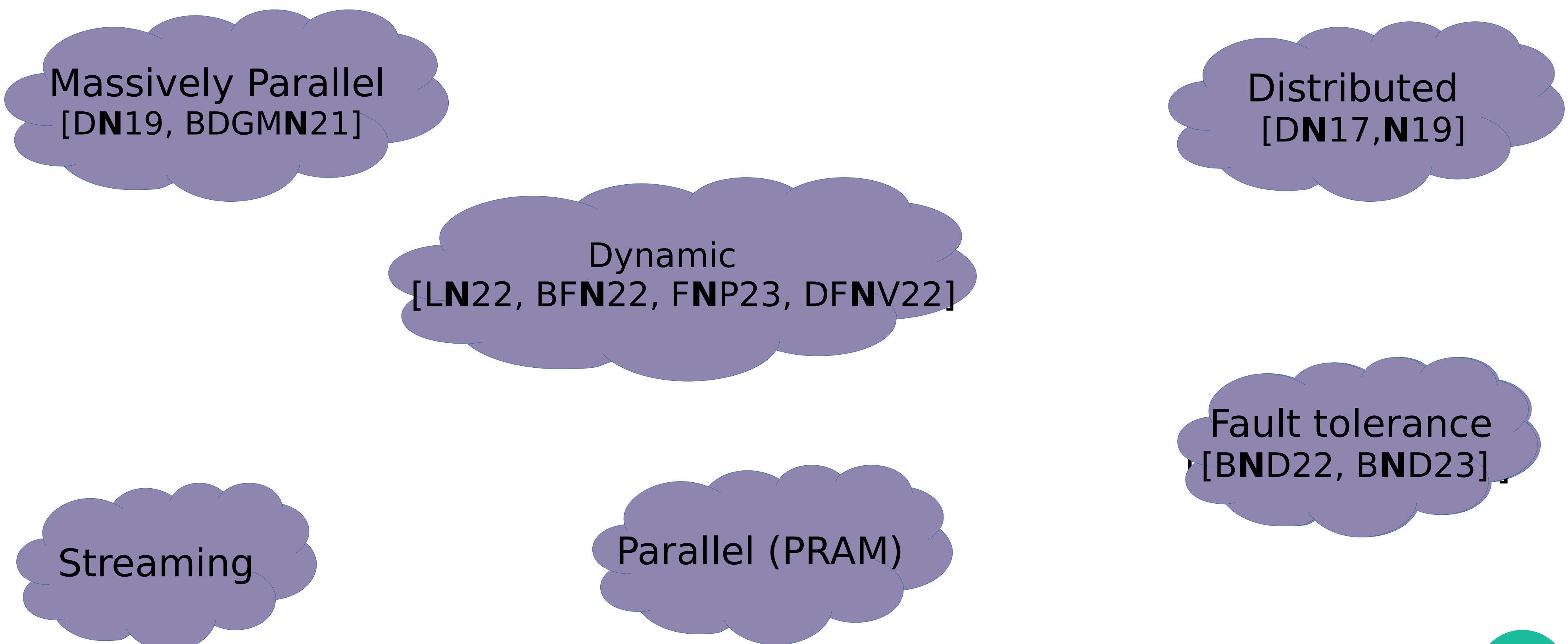
- **Maintain a $(1+\epsilon, 4)$-Emulator**
- **Distance queries:**
  - Bounded distances using algebraic data structures (dealing with the constant additive term)
  - Static shortest path on the sparse emulator
  - $(1+\epsilon)$-distance based on minimum of two estimates



- **Previous:**
  - Distance structures and theoretical applications
- **Next:**
  - Model connections and future directions

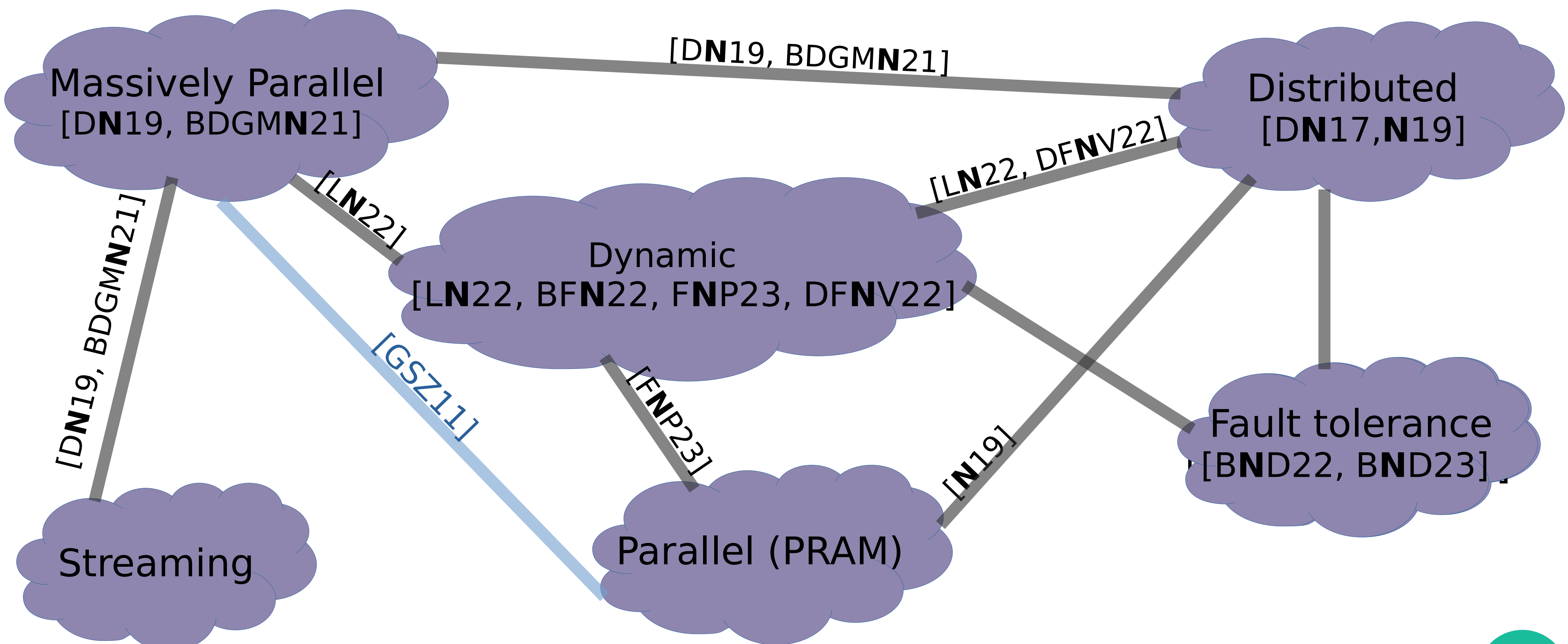
Unification

- Graph tools apply to different models



Unification

- Graph tools apply to different models
- Ideas transfer between models



- **Previous:**
  - Model connections
- **Next:**
  - Future directions

Dynamic Algorithms and Optimization

- **Recent breakthrough in near linear time max flow**
  - Uses adaptive decremental shortest path distance oracle

Dynamic Algorithms and Optimization

- **Recent breakthrough in near linear time max flow**
  - Uses adaptive decremental shortest path distance oracle
- **Static approximation algorithms for cuts/flows and clustering problems**
  - Iterative algorithms may utilize dynamic subroutines

Dynamic Algorithms and Optimization

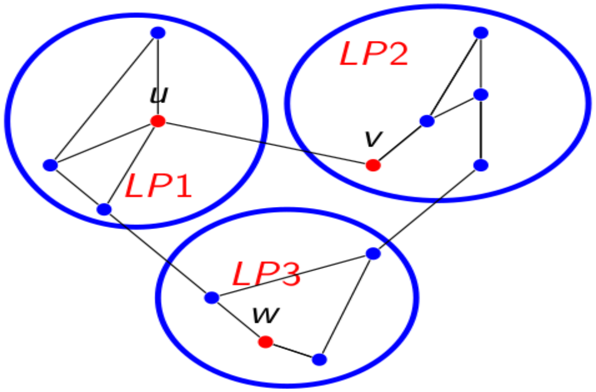
- **Recent breakthrough in near linear time max flow**
  - Uses adaptive decremental shortest path distance oracle
- **Static approximation algorithms for cuts/flows and clustering problems**
  - Iterative algorithms may utilize dynamic subroutines
- **Dynamic approximation algorithms**
  - Connection to low-recourse online algorithms

Clustering/Cut Problems

- **Graph Clustering**
  - Computationally more challenging than metrics
- **Dynamic distance computation**
  - Graph clustering algorithms requires repeated distance estimation
  - Approximation algorithms for clustering/cuts
    - Rounding based on LP solution as distances

Distributed Optimization

- **Distributed combinatorial optimization and approximation algorithms**
  - General linear programs/convex programs are challenging
  - Only special linear/convex programs can be solved (e.g. positive LPs , Local LPs, ...)
- **Combining tools from different models?**
  - LOCAL/CONGEST: network decompositions
  - Other models:
    - Algebraic tools, interior point methods



Conclusion

- **Take away**
  - Well-structured algorithmic tools will be adaptable to model changes

Conclusion

- **Take away**
  - Well-structured algorithmic tools will be adaptable to model changes
- **Future directions**
  - Distributed optimization with application in network design
  - Further application of dynamic algorithms in faster combinatorial optimization algorithms

Conclusion

- **Take away**
  - Well-structured algorithmic tools will be adaptable to model changes
- **Future directions**
  - Distributed optimization with application in network design
  - Further application of dynamic algorithms in faster combinatorial optimization algorithms

Thank you! Questions?