A Survey on Dynamic Algorithms

Sebastian Forster, né Krinninger

University of Salzburg

@HALG (June 2025)

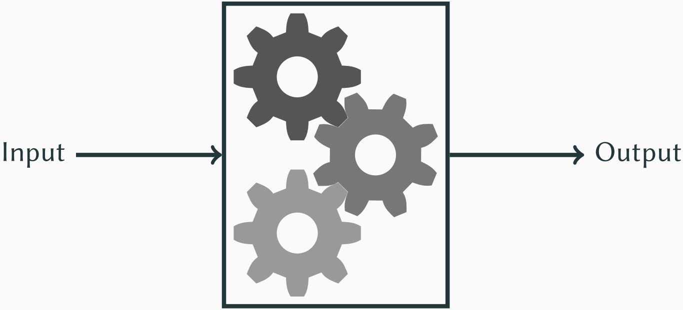
A Survey on Dynamic Distance Algorithms

Sebastian Forster, né Krinninger

University of Salzburg

@HALG (June 2025)

Static Approach



Dynamic Environments

Intra-Algorithmic Motivation

Idea

Use dynamic algorithms as powerful data structures inside of static algorithms

Intra-Algorithmic Motivation

Idea

Use dynamic algorithms as powerful data structures inside of static algorithms

Successful Research Program

Design efficient flow optimization algorithm by combining **iterative methods** with **dynamic algorithms**

Intra-Algorithmic Motivation

Idea

Use dynamic algorithms as powerful data structures inside of static algorithms

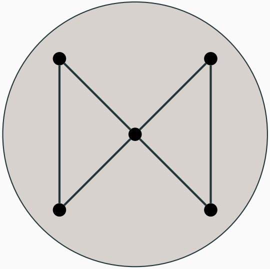
Successful Research Program

Design efficient flow optimization algorithm by combining **iterative methods** with **dynamic algorithms**

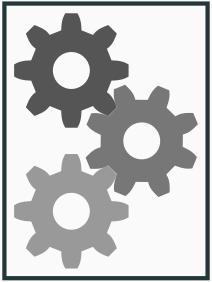
Many highlights ranging from [Mądry '10] to [Chen, Kyng, Liu, Peng Probst Gutenberg, Sachdeva '22]

Dynamic Distance Maintenance

Input graph G



Algorithm

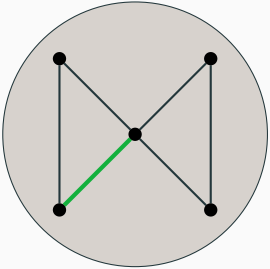


Distance Matrix

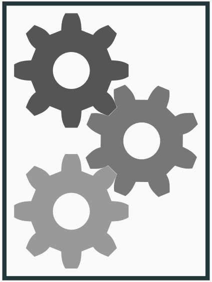
$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 2 & 2 \\ 1 & 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 & 0 \end{pmatrix}$$

Dynamic Distance Maintenance

Input graph G



Algorithm



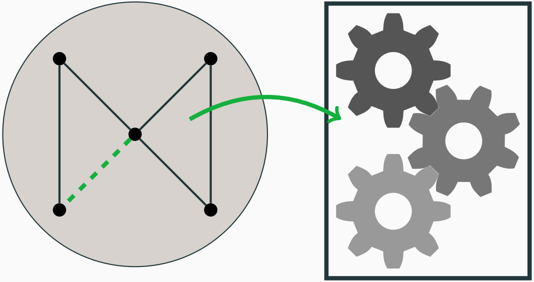
Distance Matrix

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 2 & 2 \\ 1 & 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 & 0 \end{pmatrix}$$

Adversary inserts and deletes edges

Dynamic Distance Maintenance

Input graph G | Algorithm | Distance Matrix

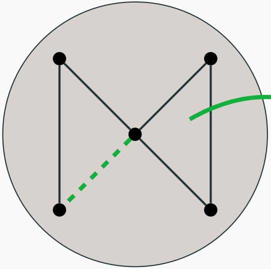


$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 2 & 2 \\ 1 & 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 & 0 \end{pmatrix}$$

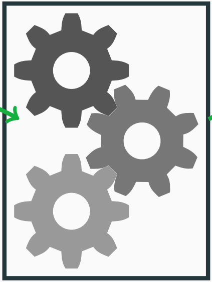
Adversary inserts
and deletes edges

Dynamic Distance Maintenance

Input graph G



Algorithm



Distance Matrix

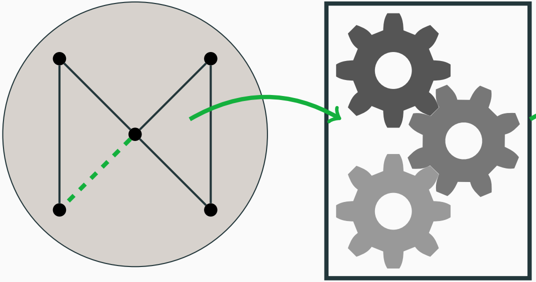
$$\begin{pmatrix} 0 & \mathbf{2} & 1 & 1 & 1 \\ \mathbf{2} & 0 & 1 & \mathbf{3} & \mathbf{3} \\ 1 & 1 & 0 & 2 & 2 \\ \mathbf{3} & 2 & 2 & 0 & 1 \\ \mathbf{3} & 2 & 2 & 1 & 0 \end{pmatrix}$$

Adversary inserts and deletes edges

Algorithm updates distance matrix

Dynamic Distance Maintenance

Input graph G — Algorithm — Distance Matrix



$$\begin{pmatrix} 0 & \mathbf{2} & 1 & 1 & 1 \\ \mathbf{2} & 0 & 1 & \mathbf{3} & \mathbf{3} \\ 1 & 1 & 0 & 2 & 2 \\ \mathbf{3} & 2 & 2 & 0 & 1 \\ \mathbf{3} & 2 & 2 & 1 & 0 \end{pmatrix}$$

Adversary inserts and deletes edges

Algorithm updates distance matrix

State of the Art

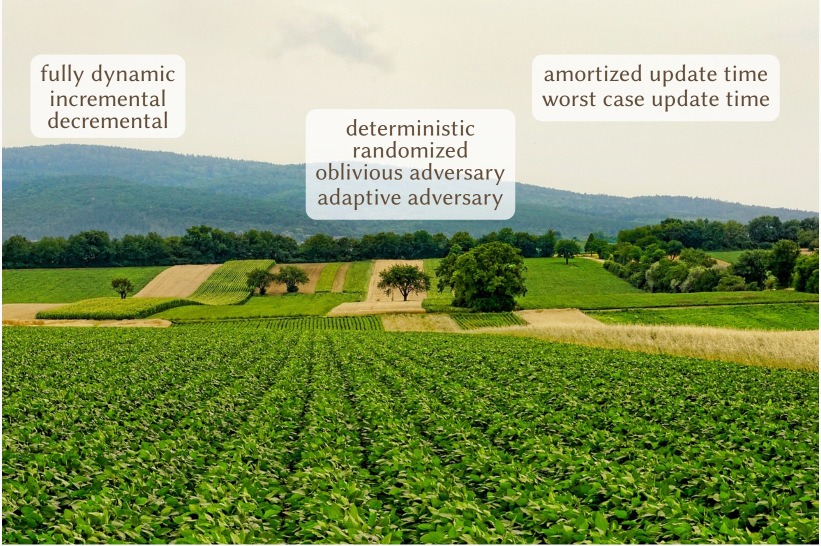
Amortized update time $\tilde{O}(n^2)$ [Demetrescu, Italiano ’03]

The Landscape of Dynamic Graph Algorithms

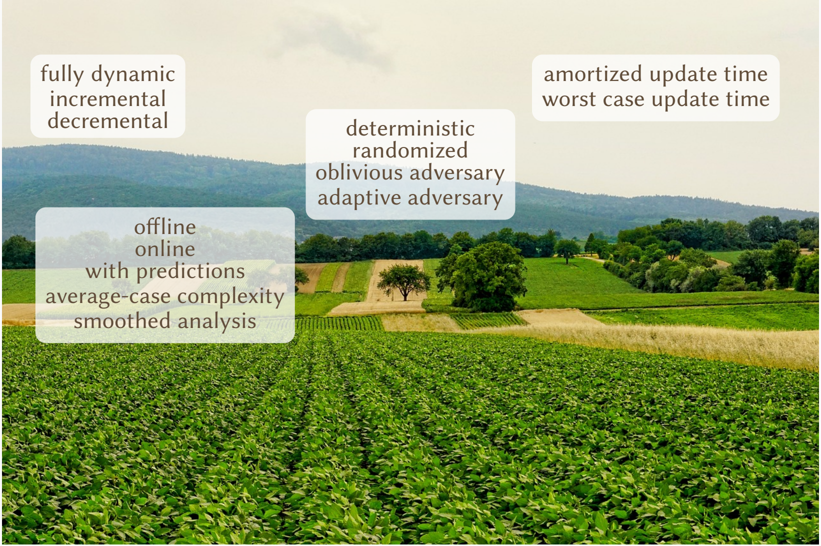
The Landscape of Dynamic Graph Algorithms

The Landscape of Dynamic Graph Algorithms

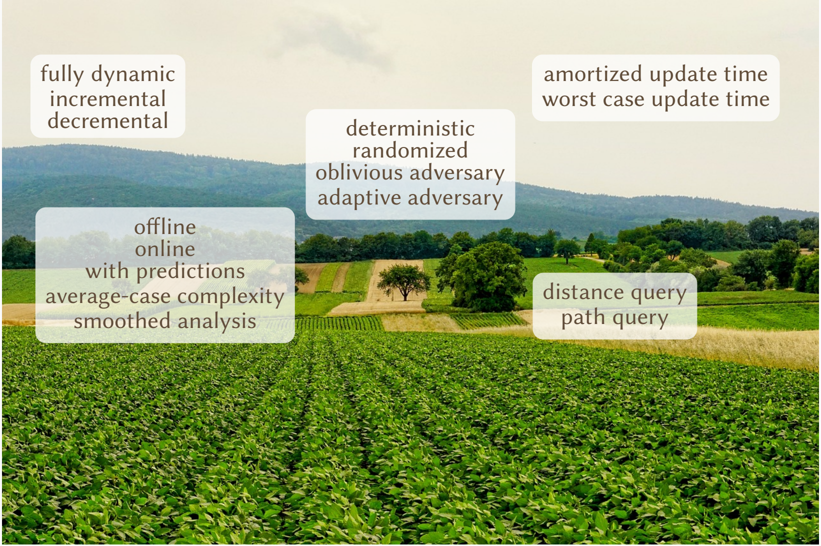
The Landscape of Dynamic Graph Algorithms



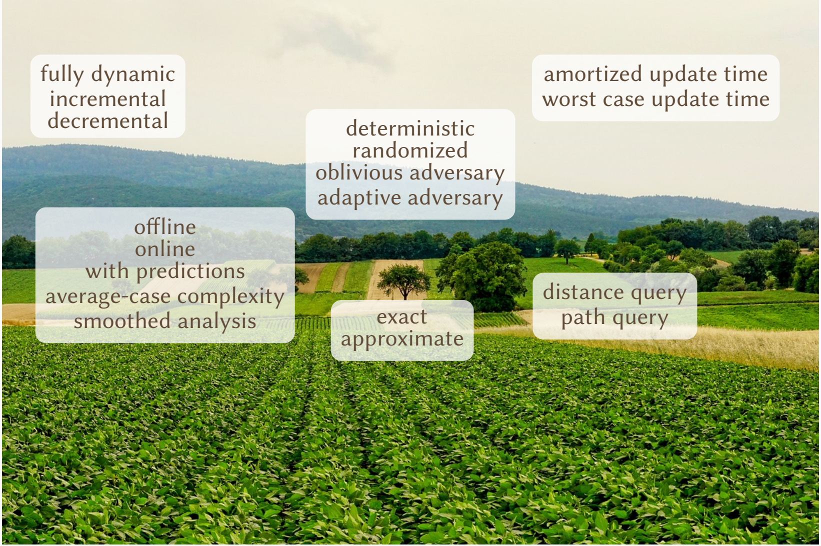
The Landscape of Dynamic Graph Algorithms



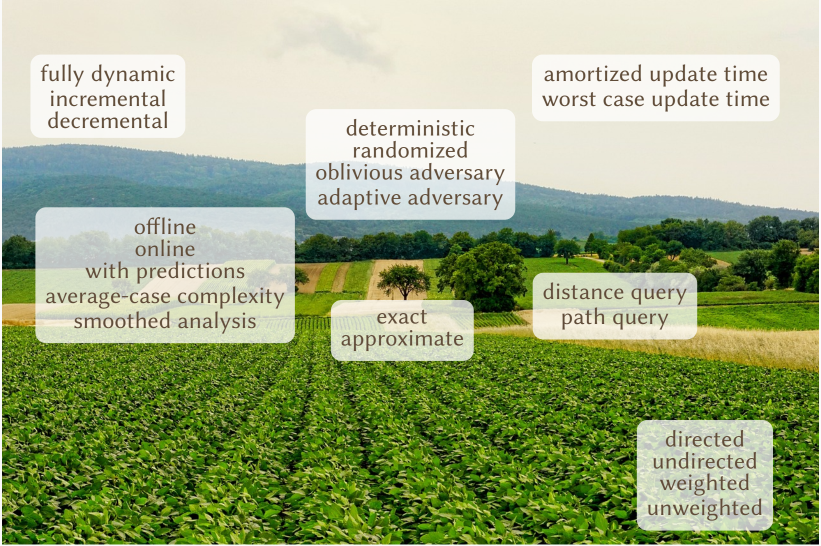
The Landscape of Dynamic Graph Algorithms



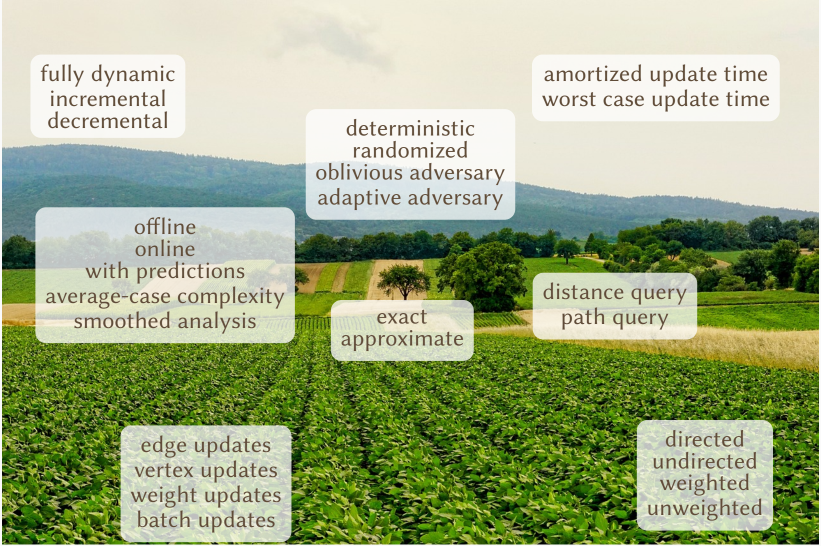
The Landscape of Dynamic Graph Algorithms



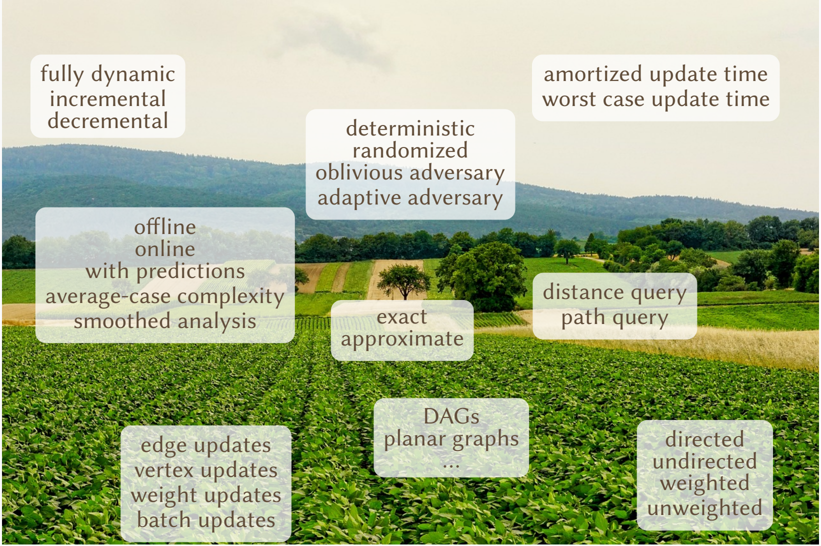
The Landscape of Dynamic Graph Algorithms



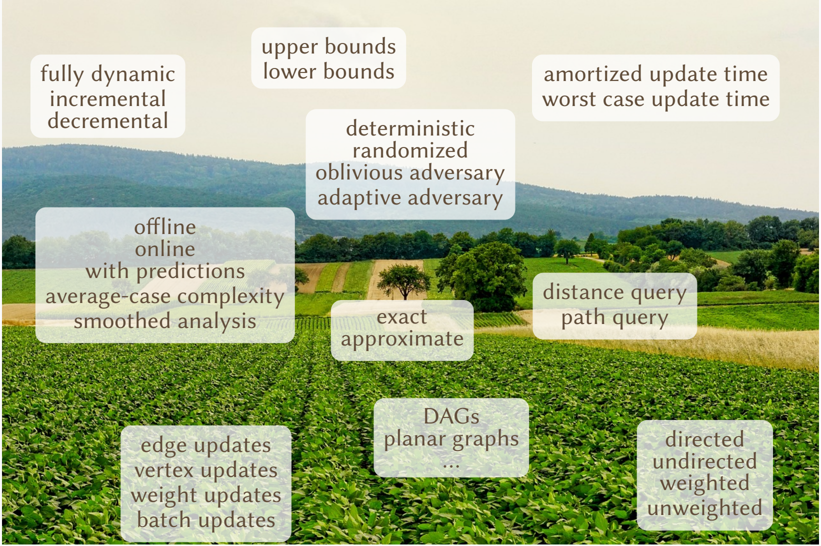
The Landscape of Dynamic Graph Algorithms



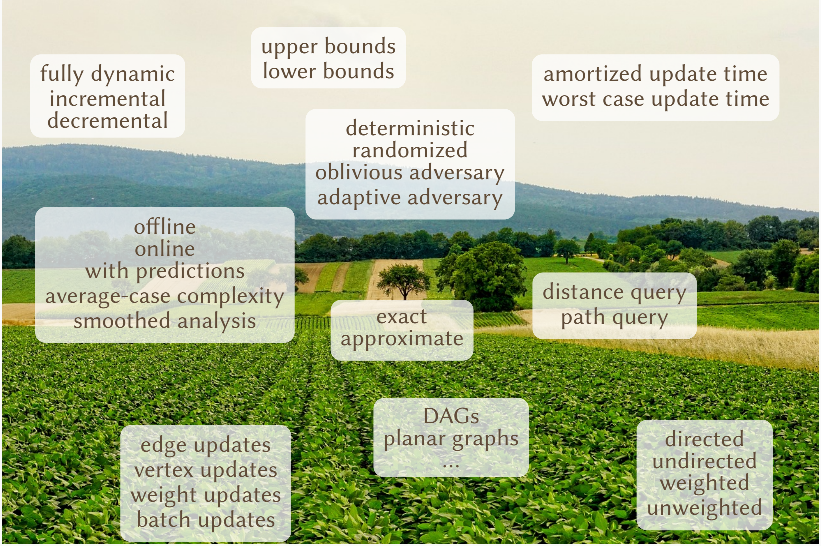
The Landscape of Dynamic Graph Algorithms



The Landscape of Dynamic Graph Algorithms



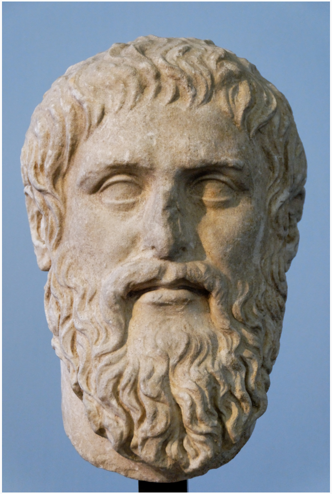
The Landscape of Dynamic Graph Algorithms



n: #nodes, m: #edges, edge weights polynomially bounded, constant ϵ

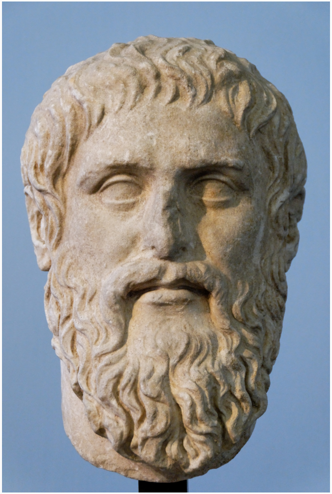
Five Regimes

1. ± 0 (exact)
2. $1 + \epsilon$ (almost exact)
3. $2 + \epsilon$ (small constant)
4. $2k - 1$ (large constant)
5. $\omega(1)$ (superconstant)



Five Regimes

1. ± 0 (exact)
2. $1 + \epsilon$ (almost exact)
3. $2 + \epsilon$ (small constant)
4. $2k - 1$ (large constant)
5. $\omega(1)$ (superconstant)



Stretch α:

$d(u, v) \leq \tilde{d}(u, v) \leq \alpha \cdot d(u, v)$

± 0 (exact)

Regime 1: ± 0 (exact)

Constant query time:

- Amortized update time $\tilde{O}(n^2)$ (det.) [Demetrescu, Italiano '03] (log-factor improvement by [Thorup '04])

Regime 1: ± 0 (exact)

Constant query time:

- Amortized update time $\tilde{O}(n^2)$ (det.) [Demetrescu, Italiano '03] (log-factor improvement by [Thorup '04])
- Worst-case update time $\tilde{O}(n^{2.5})$ (rand.) [Mao '24]
  $\rightarrow$ Framework of [Abraham, Chechik, K '16]

Regime 1: ± 0 (exact)

Constant query time:

- Amortized update time $\tilde{O}(n^2)$ (det.) [Demetrescu, Italiano '03]
  (log-factor improvement by [Thorup '04])
- Worst-case update time $\tilde{O}(n^{2.5})$ (rand.) [Mao '24]
  $\rightarrow$ Framework of [Abraham, Chechik, K '16]
- Deterministic worst-case update time:
  - $\tilde{O}(n^{2+41/61})$ [Chechik, Zhang '23]
  - $\tilde{O}(n^{2.6})$ for unweighted [Probst Gutenberg, Wulff-Nilsen '20]

Regime 1: ± 0 (exact)

Constant query time:

- Amortized update time $\tilde{O}(n^2)$ (det.) [Demetrescu, Italiano '03] (log-factor improvement by [Thorup '04])
- Worst-case update time $\tilde{O}(n^{2.5})$ (rand.) [Mao '24] **Open:** tight?
  → Framework of [Abraham, Chechik, K '16]
- Deterministic worst-case update time:
  - $\tilde{O}(n^{2+41/61})$ [Chechik, Zhang '23]
  - $\tilde{O}(n^{2.6})$ for unweighted [Probst Gutenberg, Wulff-Nilsen '20]

Regime 1: ± 0 (exact)

Constant query time:

- Amortized update time $\tilde{O}(n^2)$ (det.) [Demetrescu, Italiano '03] (log-factor improvement by [Thorup '04])
- Worst-case update time $\tilde{O}(n^{2.5})$ (rand.) [Mao '24] **Open:** tight?
  → Framework of [Abraham, Chechik, K '16]
- Deterministic worst-case update time:
  - $\tilde{O}(n^{2+41/61})$ [Chechik, Zhang '23]
  - $\tilde{O}(n^{2.6})$ for unweighted [Probst Gutenberg, Wulff-Nilsen '20]

Trade-offs:

- Worst-case update/query time $O(n^{1.724})$ in unweighted graphs [Sankowsi '05; v.d. Brand, Nanongkai, Saranurak '19]
- Amortized update time $\tilde{O}(mn^2/t^2)$, query time $O(t)$ (for $t \approx [n^{1/2}, n^{3/4}]$) in unweighted graphs [Roditty, Zwick '11]
- Worst-case update time $\tilde{O}(mn^{4/5})$, query time $\tilde{O}(n^{4/5})$ [Karczmarz, Sankowski '23]

Regime 1: ± 0 (exact)

Constant query time:

- Amortized update time $\tilde{O}(n^2)$ (det.) [Demetrescu, Italiano '03] (log-factor improvement by [Thorup '04])
- Worst-case update time $\tilde{O}(n^{2.5})$ (rand.) [Mao '24] **Open:** tight?
  $\rightarrow$ Framework of [Abraham, Chechik, K '16]
- Deterministic worst-case update time:
  - $\tilde{O}(n^{2+41/61})$ [Chechik, Zhang '23]
  - $\tilde{O}(n^{2.6})$ for unweighted [Probst Gutenberg, Wulff-Nilsen '20]

Trade-offs:

- Worst-case update/query time $O(n^{1.724})$ in unweighted graphs [Sankowsi '05; v.d. Brand, Nanongkai, Saranurak '19]
- Amortized update time $\tilde{O}(mn^2/t^2)$, query time $O(t)$ (for $t \approx [n^{1/2}, n^{3/4}]$) in unweighted graphs [Roditty, Zwick '11]
- Worst-case update time $\tilde{O}(mn^{4/5})$, query time $\tilde{O}(n^{4/5})$ [Karczmarz, Sankowski '23] **Open:** improved trade-offs?

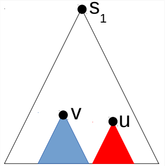
ACK Framework: Preprocessing

Construct shortest path tree up to h edges for all sources one by one **counting** total size of subtrees for every node (idea of [Thorup '05]):

ACK Framework: Preprocessing

Construct shortest path tree up to h edges for all sources one by one **counting** total size of subtrees for every node (idea of [Thorup '05]):

G

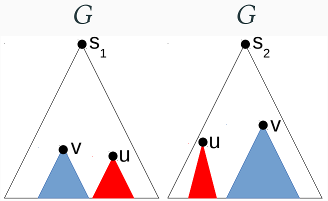


Rule: If number of nodes in subtrees of v exceeds λ:

- v is added to set of **heavy** nodes H
- v is removed from graph, i.e., not considered in future trees

ACK Framework: Preprocessing

Construct shortest path tree up to h edges for all sources one by one **counting** total size of subtrees for every node (idea of [Thorup '05]):

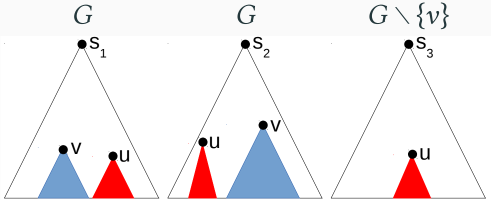


Rule: If number of nodes in subtrees of v exceeds λ:

- v is added to set of **heavy** nodes H
- v is removed from graph, i.e., not considered in future trees

ACK Framework: Preprocessing

Construct shortest path tree up to h edges for all sources one by one **counting** total size of subtrees for every node (idea of [Thorup '05]):

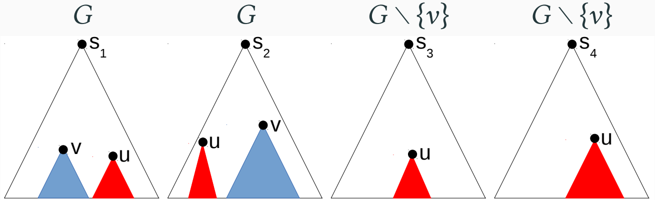


Rule: If number of nodes in subtrees of v exceeds λ:

- v is added to set of **heavy** nodes H
- v is removed from graph, i.e., not considered in future trees

ACK Framework: Preprocessing

Construct shortest path tree up to h edges for all sources one by one **counting** total size of subtrees for every node (idea of [Thorup '05]):

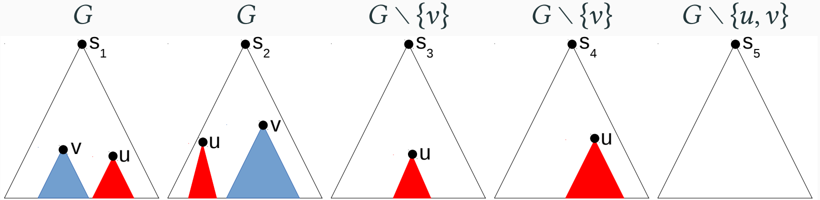


Rule: If number of nodes in subtrees of v exceeds λ:

- v is added to set of **heavy** nodes H
- v is removed from graph, i.e., not considered in future trees

ACK Framework: Preprocessing

Construct shortest path tree up to h edges for all sources one by one **counting** total size of subtrees for every node (idea of [Thorup '05]):

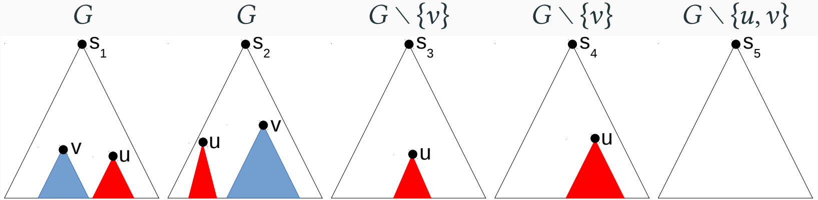


Rule: If number of nodes in subtrees of v exceeds λ:

- v is added to set of **heavy** nodes H
- v is removed from graph, i.e., not considered in future trees

ACK Framework: Preprocessing

Construct shortest path tree up to h edges for all sources one by one **counting** total size of subtrees for every node (idea of [Thorup '05]):



Rule: If number of nodes in subtrees of v exceeds λ:

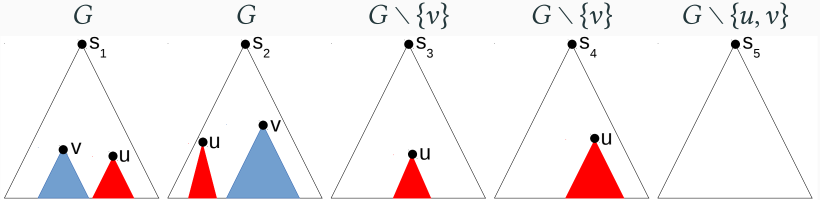
- v is added to set of **heavy** nodes H
- v is removed from graph, i.e., not considered in future trees

Observations:

- All shortest paths not using heavy nodes included in trees

ACK Framework: Preprocessing

Construct shortest path tree up to h edges for all sources one by one **counting** total size of subtrees for every node (idea of [Thorup '05]):



Rule: If number of nodes in subtrees of v exceeds λ:

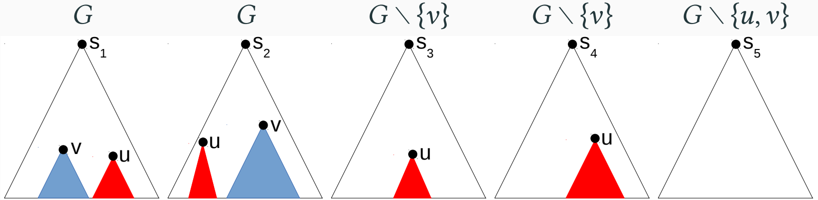
- v is added to set of **heavy** nodes H
- v is removed from graph, i.e., not considered in future trees

Observations:

- All shortest paths not using heavy nodes included in trees
- Number of heavy nodes: $|H| \leq O(\frac{|S|nh}{\lambda}) \leq O(\frac{n^2h}{\lambda})$

ACK Framework: Preprocessing

Construct shortest path tree up to h edges for all sources one by one **counting** total size of subtrees for every node (idea of [Thorup '05]):



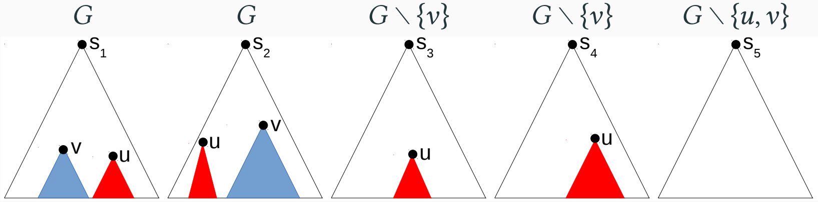
Rule: If number of nodes in subtrees of v exceeds λ:

- v is added to set of **heavy** nodes H
- v is removed from graph, i.e., not considered in future trees

Observations:

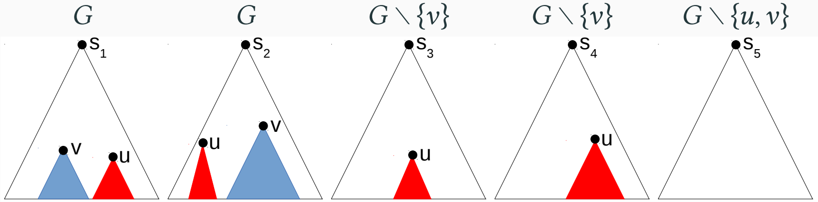
- All shortest paths not using heavy nodes included in trees
- Number of heavy nodes: $|H| \leq O(\frac{|S|nh}{\lambda}) \leq O(\frac{n^2h}{\lambda})$
- Preprocessing time: $O(|S|n^2) \leq O(n^3)$

ACK Framework: Batch of Δ Vertex Deletions



1. For all deleted nodes: Reattach successors to tree → Dijkstra

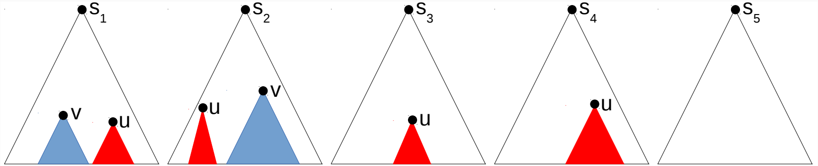
ACK Framework: Batch of Δ Vertex Deletions



1. For all deleted nodes: Reattach successors to tree → Dijkstra
   Running time: $\tilde{O}(\Delta\lambda n)$ per deletion
   - Subtree size at most λ per node
   - Number of deleted nodes at most Δ

ACK Framework: Batch of Δ Vertex Deletions

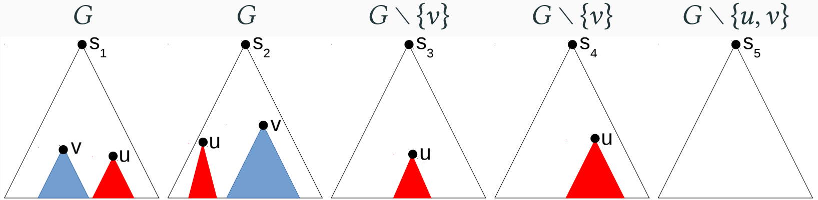
G | G | $G \setminus \{v\}$ | $G \setminus \{v\}$ | $G \setminus \{u, v\}$



1. For all deleted nodes: Reattach successors to tree → Dijkstra
   Running time: $\tilde{O}(\Delta\lambda n)$ per deletion
   - Subtree size at most λ per node
   - Number of deleted nodes at most Δ

   Correct for all shortest paths not containing heavy nodes

ACK Framework: Batch of Δ Vertex Deletions



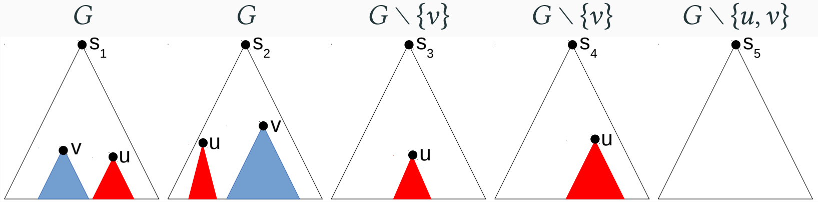
1. For all deleted nodes: Reattach successors to tree → Dijkstra
   Running time: $\tilde{O}(\Delta\lambda n)$ per deletion
   - Subtree size at most λ per node
   - Number of deleted nodes at most Δ

   Correct for all shortest paths not containing heavy nodes
2. Additionally: shortest paths via heavy nodes

   $$\text{Compute } \min_{v \in H}(dist(s, v) + dist(v, t)) \text{ for all } s \text{ and } t$$

   Time per deletion: $O(|H|n^2) = O(\frac{n^4 h}{\lambda})$

ACK Framework: Batch of Δ Vertex Deletions



1. For all deleted nodes: Reattach successors to tree → Dijkstra
   Running time: $\tilde{O}(\Delta\lambda n)$ per deletion
   - Subtree size at most λ per node
   - Number of deleted nodes at most Δ

   Correct for all shortest paths not containing heavy nodes
2. Additionally: shortest paths via heavy nodes

   Compute $\min_{v \in H}(dist(s,v) + dist(v,t))$ for all s and t

   Time per deletion: $O(|H|n^2) = O(\frac{n^4 h}{\lambda})$

Fully dynamic can be reduced to bounded-hop batch-deletion

Regime 1: ± 0 (exact)

Unweighted graphs:

- **Incremental:** total update time $\tilde{O}(n^3)$ [Ausiello et al. 92]
- **Decremental:** total update time $\tilde{O}(n^3)$ [Demetrescu, Italiano '01; Baswana, Hariharan, Sen '02]

  $\rightarrow$ Deterministic version: [Evald, Fredslund-Hansen, Probst Gutenberg, Wulff-Nilsen '21]

Regime 1: ± 0 (exact)

Unweighted graphs:

- **Incremental:** total update time $\tilde{O}(n^3)$ [Ausiello et al. 92]
- **Decremental:** total update time $\tilde{O}(n^3)$ [Demetrescu, Italiano '01; Baswana, Hariharan, Sen '02]
  $\rightarrow$ Deterministic version: [Evald, Fredslund-Hansen, Probst Gutenberg, Wulff-Nilsen '21]

Lower bounds:

- Static time: $O(n^{2.575})$ [Zwick '02]

Regime 1: ± 0 (exact)

Unweighted graphs:

- **Incremental:** total update time $\tilde{O}(n^3)$ [Ausiello et al. 92]
- **Decremental:** total update time $\tilde{O}(n^3)$ [Demetrescu, Italiano '01; Baswana, Hariharan, Sen '02]
  $\rightarrow$ Deterministic version: [Evald, Fredslund-Hansen, Probst Gutenberg, Wulff-Nilsen '21]

Lower bounds:

- Static time: $O(n^{2.575})$ [Zwick '02]
- $\Omega(n^3)$ changes to pairwise distance even in sparse graphs

Regime 1: ± 0 (exact)

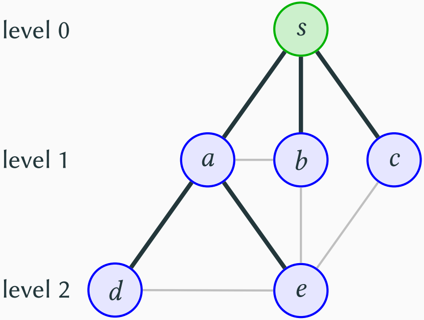
Unweighted graphs:

- **Incremental:** total update time $\tilde{O}(n^3)$ [Ausiello et al. 92]
- **Decremental:** total update time $\tilde{O}(n^3)$ [Demetrescu, Italiano '01; Baswana, Hariharan, Sen '02]
  $\rightarrow$ Deterministic version: [Evald, Fredslund-Hansen, Probst Gutenberg, Wulff-Nilsen '21]

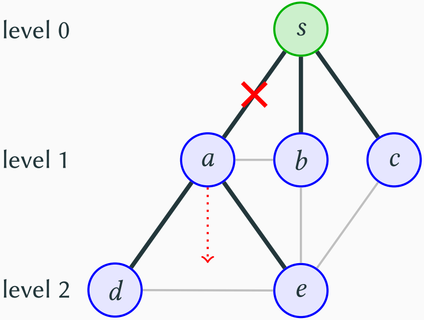
Lower bounds:

- Static time: $O(n^{2.575})$ [Zwick '02]
- $\Omega(n^3)$ changes to pairwise distance even in sparse graphs
- No $O(n^{3-\delta})$ total update time with small query time based on OMv conjecture [Henzinger, K, Nanongkai, Saranurak '15]

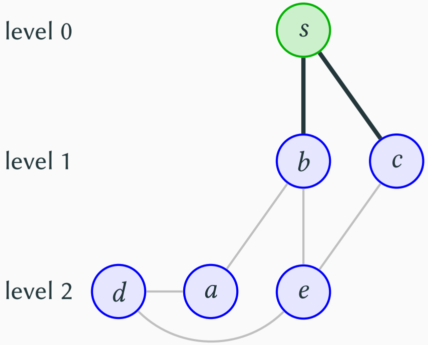
Decremental Shortest Path Tree [Even/Shiloach '81]



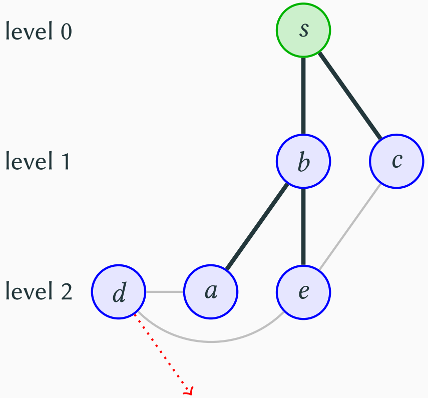
Decremental Shortest Path Tree [Even/Shiloach '81]



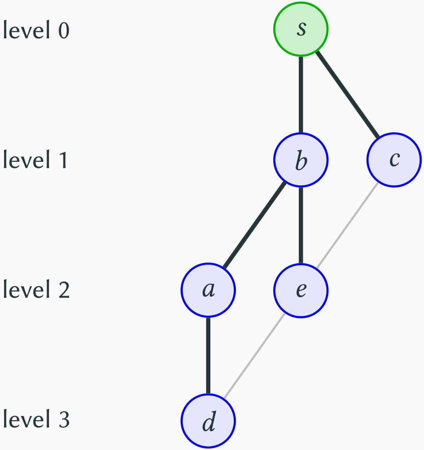
Decremental Shortest Path Tree [Even/Shiloach ’81]



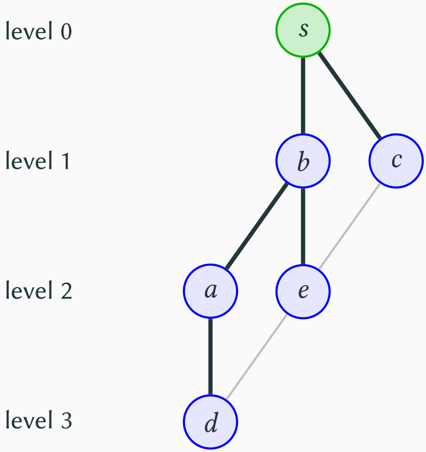
Decremental Shortest Path Tree [Even/Shiloach ’81]



Decremental Shortest Path Tree [Even/Shiloach '81]

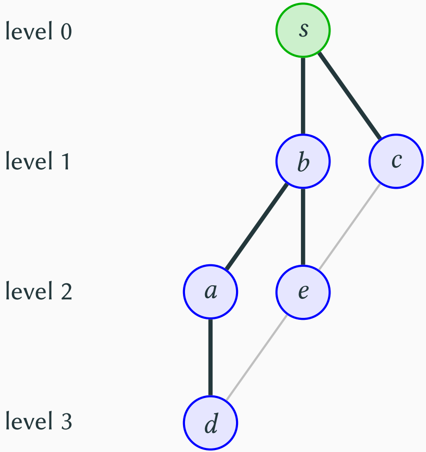


Decremental Shortest Path Tree [Even/Shiloach '81]



Running time: $O(degree(v))$ per level increase of each node v

Decremental Shortest Path Tree [Even/Shiloach ’81]



Running time: $O(degree(v))$ per level increase of each node v

Over all deletions for tree up to depth D:

$$O\left(\sum_v degree(v) \cdot D\right) = O(mD)$$

$1 + \epsilon$ (almost exact)

Regime 2: $1 + \epsilon$ (almost exact)

Partially dynamic:

- Decremental: randomized, total time $\tilde{O}(mn)$ in directed graphs [Bernstein '13]
- Decremental: determ., total time $O(mn^{1+o(1)})$ in undirected graphs via SSSP [Bernstein, Probst Gutenberg, Saranurak '21]
- Incremental: determ., total time $\tilde{O}(mn^{4/3})$ in directed graphs [Karczmarz, Łącki '19]

Regime 2: $1 + \epsilon$ (almost exact)

Partially dynamic:

- Decremental: randomized, total time $\tilde{O}(mn)$ in directed graphs [Bernstein ’13]
- Decremental: determ., total time $O(mn^{1+o(1)})$ in undirected graphs via SSSP [Bernstein, Probst Gutenberg, Saranurak ’21]
- Incremental: determ., total time $\tilde{O}(mn^{4/3})$ in directed graphs [Karczmarz, Łącki ’19]

Fully dynamic:

- Update time $\tilde{O}(n^{2.045})$ [v.d. Brand, Nanongkai ’19]
- Update time $O(n^{1.863})$, query time $O(n^{0.666})$ [v.d. Brand, Nanongkai ’19]
- Unweighted, undirected graphs
    - Update time $\tilde{O}(n^2)$ [v.d. Brand, Nanongkai ’19]
    - Update time $O(n^{1.788})$, query time $O(n^{0.45})$ [v.d. Brand, F, Nazari ’22]

Regime 2: $1 + \epsilon$

Trade-offs:

- Based on improved *decremental* APSP or SSSP algorithms (reduction)
- General form: Amortized update time mn/t, query time t
  - Randomized, unweighted, undirected graphs: [Roditty, Zwick '04]
  - Deterministic, unweighted, undirected graphs: [Henzinger, K, Nanongkai '13]
  - Randomized, weighted, directed graphs: [Bernstein '13]
  - Deterministic, weighted, undirected graphs: [Bernstein, Gutenberg, Saranurak '21]

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0, 1\}^{n \times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0, 1\}^{n \times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k
- Consider field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo large enough prime p

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0, 1\}^{n \times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k
- Consider field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo large enough prime p
- Consider $\mathbb{F}[X]/X^h$, polynomial ring modulo X^h for given h

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0,1\}^{n \times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k
- Consider field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo large enough prime p
- Consider $\mathbb{F}[X]/X^h$, polynomial ring modulo X^h for given h
- Observation: $(\mathbf{I} - X\mathbf{A})^{-1} = \sum_{k=0}^{h-1} X^k \mathbf{A}^k$

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0,1\}^{n \times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k
- Consider field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo large enough prime p
- Consider $\mathbb{F}[X]/X^h$, polynomial ring modulo X^h for given h
- Observation: $(\mathbf{I} - X\mathbf{A})^{-1} = \sum_{k=0}^{h-1} X^k \mathbf{A}^k$
  $\Rightarrow$ Inverse contains all pairwise distances smaller than h

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0,1\}^{n \times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k
- Consider field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo large enough prime p
- Consider $\mathbb{F}[X]/X^h$, polynomial ring modulo X^h for given h
- Observation: $(\mathbf{I} - X\mathbf{A})^{-1} = \sum_{k=0}^{h-1} X^k \mathbf{A}^k$
  $\Rightarrow$ Inverse contains all pairwise distances smaller than h
- Updating inverse after row/column update: $O(n^2)$ operations

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0,1\}^{n \times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k
- Consider field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo large enough prime p
- Consider $\mathbb{F}[X]/X^h$, polynomial ring modulo X^h for given h
- Observation: $(\mathbf{I} - X\mathbf{A})^{-1} = \sum_{k=0}^{h-1} X^k \mathbf{A}^k$
  $\Rightarrow$ Inverse contains all pairwise distances smaller than h
- Updating inverse after row/column update: $O(n^2)$ operations
  Recall: Update solution to linear program after basis exchange

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0,1\}^{n\times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k
- Consider field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo large enough prime p
- Consider $\mathbb{F}[X]/X^h$, polynomial ring modulo X^h for given h
- Observation: $(\mathbf{I} - X\mathbf{A})^{-1} = \sum_{k=0}^{h-1} X^k \mathbf{A}^k$
  $\Rightarrow$ Inverse contains all pairwise distances smaller than h
- Updating inverse after row/column update: $O(n^2)$ operations
  Recall: Update solution to linear program after basis exchange
- Time per operation: $O(h \log p)$

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0,1\}^{n \times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k
- Consider field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo large enough prime p
- Consider $\mathbb{F}[X]/X^h$, polynomial ring modulo X^h for given h
- Observation: $(\mathbf{I} - X\mathbf{A})^{-1} = \sum_{k=0}^{h-1} X^k \mathbf{A}^k$
  $\Rightarrow$ Inverse contains all pairwise distances smaller than h
- Updating inverse after row/column update: $O(n^2)$ operations
  Recall: Update solution to linear program after basis exchange
- Time per operation: $O(h \log p)$
  - $p = \Theta(n^h)$: suffices for $\leq n^h$ paths of length h

Dynamic Inverse Maintenance: Idea

- $\mathbf{A} \in \{0,1\}^{n \times n}$ adjacency matrix of unweighted, directed graph
- Entry $\mathbf{A}^k_{s,t}$ contains the *number* of paths from s to t of length k
- Consider field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo large enough prime p
- Consider $\mathbb{F}[X]/X^h$, polynomial ring modulo X^h for given h
- Observation: $(\mathbf{I} - X\mathbf{A})^{-1} = \sum_{k=0}^{h-1} X^k \mathbf{A}^k$
  $\Rightarrow$ Inverse contains all pairwise distances smaller than h
- Updating inverse after row/column update: $O(n^2)$ operations
  Recall: Update solution to linear program after basis exchange
- Time per operation: $O(h \log p)$
  - $p = \Theta(n^h)$: suffices for $\leq n^h$ paths of length h
  - *Random* $p = \Theta(n^c)$: degree zero check correct with high probability (Schwartz-Zippel Lemma)

Dynamic Inverse Maintenance: Literature

Sophisticated lazy update schemes

- for maintaining submatrix (row, column, entry, ...)
- using fast matrix multiplication
- leading to update/query trade-offs

[King Sagert '99]

[Demetrescu, Italiano '00]

[Demetrescu, Italiano '01]

[Sankowski '04]

[Sankowski '05]

[v.d. Brand, Nangongkai, Saranurak '19]

[v.d. Brand, Forster, Nazari, Polak '24]

$2 + \epsilon$ (small constant)

Regime 3: $2 + \epsilon$

This regime is not very well explored!

Regime 3: $2 + \epsilon$

This regime is not very well explored!

- Fully dynamic: stretch $2 + \epsilon$, update time $m^{1+o(1)}$ weighted graphs [Bernstein '09]

Regime 3: $2 + \epsilon$

This regime is not very well explored!

- Fully dynamic: stretch $2 + \epsilon$, update time $m^{1+o(1)}$ weighted graphs [Bernstein '09]

  Nowadays: reduction to decremental SSSP [Henzinger, K, Nanongkai '14; Bernstein, Probst Gutenberg, Saranurak '21]

Regime 3: $2 + \epsilon$

This regime is not very well explored!

- Fully dynamic: stretch $2 + \epsilon$, update time $m^{1+o(1)}$ weighted graphs [Bernstein '09]

  Nowadays: reduction to decremental SSSP [Henzinger, K, Nanongkai '14; Bernstein, Probst Gutenberg, Saranurak '21]
- Decremental: stretch 2, total update time $\tilde{O}(n^{2.5})$ unweighted graphs [Henzinger, K, Nanongkai '13]
  - Follows from *near-additive* stretch guarantee: $(1 + \epsilon, 2)$

Regime 3: $2 + \epsilon$

This regime is not very well explored!

- Fully dynamic: stretch $2 + \epsilon$, update time $m^{1+o(1)}$ weighted graphs [Bernstein ’09]

  Nowadays: reduction to decremental SSSP [Henzinger, K, Nanongkai ’14; Bernstein, Probst Gutenberg, Saranurak ’21]
- Decremental: stretch 2, total update time $\tilde{O}(n^{2.5})$ unweighted graphs [Henzinger, K, Nanongkai ’13]
  - Follows from *near-additive* stretch guarantee: $(1 + \epsilon, 2)$
  - Follow-up: stretch $(1 + \epsilon, 2(k - 1))$, total update time $O(n^{2-1/k} + m^{1/k+o(1)})$ [Dory, F, Nazari, de Vos ’24]

Regime 3: $2 + \epsilon$

This regime is not very well explored!

- Fully dynamic: stretch $2 + \epsilon$, update time $m^{1+o(1)}$ weighted graphs [Bernstein '09]

  Nowadays: reduction to decremental SSSP [Henzinger, K, Nanongkai '14; Bernstein, Probst Gutenberg, Saranurak '21]
- Decremental: stretch 2, total update time $\tilde{O}(n^{2.5})$ unweighted graphs [Henzinger, K, Nanongkai '13]
  - Follows from *near-additive* stretch guarantee: $(1 + \epsilon, 2)$
  - Follow-up: stretch $(1 + \epsilon, 2(k - 1))$, total update time $O(n^{2-1/k} + m^{1/k+o(1)})$ [Dory, F, Nazari, de Vos '24]
- Decremental: stretch $2 + \epsilon$, total update time $\tilde{O}(m^{1/2} n^{3/2+o(1)})$ (weighted) or $\tilde{O}(m^{7/4})$ (unweighted) [Dory, F, Nazari, de Vos '24]

Enforcing Monotonicity

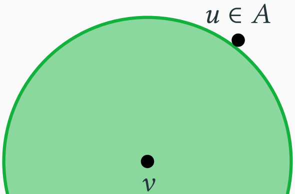
Typical problem: Large internal recourse leads to inefficiencies

Enforcing Monotonicity

Typical problem: Large internal recourse leads to inefficiencies

Example from decremental algorithm of [Dory et al. '24]

- Ball around every node v up to closest pivot in random set A of size pn

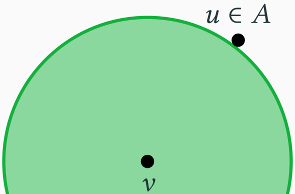


Enforcing Monotonicity

Typical problem: Large internal recourse leads to inefficiencies

Example from decremental algorithm of [Dory et al. ’24]

- Ball around every node v up to closest pivot in random set A of size pn
- Each such ball has expected size $\leq \frac{1}{p}$

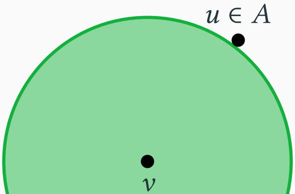


Enforcing Monotonicity

Typical problem: Large internal recourse leads to inefficiencies

Example from decremental algorithm of [Dory et al. '24]

- Ball around every node v up to closest pivot in random set A of size pn
- Each such ball has expected size $\leq \frac{1}{p}$
- By oblivious adversary: true in all versions of the graph

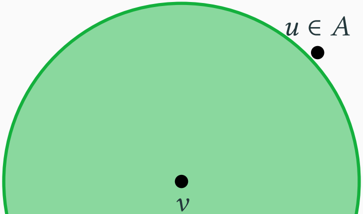


Enforcing Monotonicity

Typical problem: Large internal recourse leads to inefficiencies

Example from decremental algorithm of [Dory et al. '24]

- Ball around every node v up to closest pivot in random set A of size pn
- Each such ball has expected size $\leq \frac{1}{p}$
- By oblivious adversary: true in all versions of the graph
- But: as size of closest pivot increases, new nodes enter the ball
  Naive total recourse bound (#nodes ever contained in ball): $\Omega(n)$

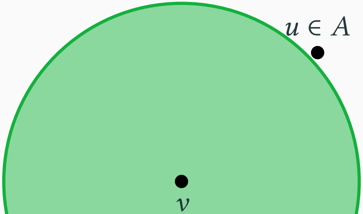


Enforcing Monotonicity

Typical problem: Large internal recourse leads to inefficiencies

Example from decremental algorithm of [Dory et al. '24]

- Ball around every node v up to closest pivot in random set A of size pn
- Each such ball has expected size $\leq \frac{1}{p}$
- By oblivious adversary: true in all versions of the graph
- But: as size of closest pivot increases, new nodes enter the ball
  Naive total recourse bound (#nodes ever contained in ball): $\Omega(n)$
- **Solution:** Only update ball when distance to closest pivot increases by $(1 + \epsilon)$-factor

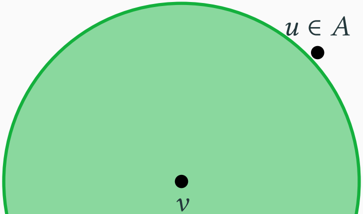


Enforcing Monotonicity

Typical problem: Large internal recourse leads to inefficiencies

Example from decremental algorithm of [Dory et al. '24]

- Ball around every node v up to closest pivot in random set A of size pn
- Each such ball has expected size $\leq \frac{1}{p}$
- By oblivious adversary: true in all versions of the graph
- But: as size of closest pivot increases, new nodes enter the ball
  Naive total recourse bound (#nodes ever contained in ball): $\Omega(n)$
- **Solution:** Only update ball when distance to closest pivot increases by $(1+\epsilon)$-factor
  $\Rightarrow$ Total recourse: $\tilde{O}(\frac{1}{p}\log_{1+\epsilon}(nW))$



$2k - 1$ (large constant)

Regime 4: $2k - 1$ (large constant)

Static [Thorup, Zwick '05]

Stretch $2k - 1$, preprocessing time $O(mn^{1/k})$, query time $O(k)$

Regime 4: $2k-1$ (large constant)

Static **[Thorup, Zwick ’05]**

Stretch $2k-1$, preprocessing time $O(mn^{1/k})$, query time $O(k)$

Conditional lower bound **[Abboud, Bringmann, Fischer ’23]**

Stretch $2k-1$ requires update or query time of $\approx n^{1/k}$

Regime 4: $2k-1$ (large constant)

Static **[Thorup, Zwick '05]**

Stretch $2k-1$, preprocessing time $O(mn^{1/k})$, query time $O(k)$

Conditional lower bound **[Abboud, Bringmann, Fischer '23]**

Stretch $2k-1$ requires update or query time of $\approx n^{1/k}$

Partially dynamic:

- Decremental with stretch $(2k-1)(1+\epsilon)$, total update time $\tilde{O}((m+n^{1+o(1)})n^{1/k})$ [Chechik '18; Łącki, Nazari '22]

Regime 4: $2k-1$ (large constant)

Static **[Thorup, Zwick '05]**

Stretch $2k-1$, preprocessing time $O(mn^{1/k})$, query time $O(k)$

Conditional lower bound **[Abboud, Bringmann, Fischer '23]**

Stretch $2k-1$ requires update or query time of $\approx n^{1/k}$

Partially dynamic:

- Decremental with stretch $(2k-1)(1+\epsilon)$, total update time $\tilde{O}((m+n^{1+o(1)})n^{1/k})$ [Chechik '18; Łącki, Nazari '22]
- Decremental with stretch $(\log n)^{2^{1/\rho}}$, total update time $O(m^{1+O(\rho)}(\log n)^{O(1/\rho^2)})$ **deterministic** [Chuzhoy '21]

Regime 4: $2k-1$ (large constant)

Static [Thorup, Zwick '05]

Stretch $2k-1$, preprocessing time $O(mn^{1/k})$, query time $O(k)$

Conditional lower bound [Abboud, Bringmann, Fischer '23]

Stretch $2k-1$ requires update or query time of $\approx n^{1/k}$

Partially dynamic:

- Decremental with stretch $(2k-1)(1+\epsilon)$, total update time $\tilde{O}((m+n^{1+o(1)})n^{1/k})$ [Chechik '18; Łącki, Nazari '22]
- Decremental with stretch $(\log n)^{2^{1/\rho}}$, total update time $O(m^{1+O(\rho)}(\log n)^{O(1/\rho^2)})$ **deterministic** [Chuzhoy '21]
- Decremental with stretch $n^{o(1)}$, total update time $m^{1+o(1)}$ **deterministic** [Bernstein, Probst Gutenberg, Saranurak '21]

Regime 4: $2k-1$ (large constant)

Static [Thorup, Zwick '05]

Stretch $2k-1$, preprocessing time $O(mn^{1/k})$, query time $O(k)$

Conditional lower bound [Abboud, Bringmann, Fischer '23]

Stretch $2k-1$ requires update or query time of $\approx n^{1/k}$

Partially dynamic:

- Decremental with stretch $(2k-1)(1+\epsilon)$, total update time $\tilde{O}((m+n^{1+o(1)})n^{1/k})$ [Chechik '18; Łącki, Nazari '22]
- Decremental with stretch $(\log n)^{2^{1/\rho}}$, total update time $O(m^{1+O(\rho)}(\log n)^{O(1/\rho^2)})$ **deterministic** [Chuzhoy '21]
- Decremental with stretch $n^{o(1)}$, total update time $m^{1+o(1)}$ **deterministic** [Bernstein, Probst Gutenberg, Saranurak '21]
- Incremental with stretch $(2k-1)^t$ and worst case update/query time $\tilde{O}(m^{1/(t+1)}n^{t/k})$ **deterministic** [Chen, Goranci, Henzinger, Peng, Saranurak '20]

Regime 4: $2k-1$ (large constant)

Fully dynamic:

- Stretch $2^{O(\rho k)}$, update time $\tilde{O}(m^{1/2}n^{1/k})$, and query time $O(k^2\rho^2)$ (for $\rho = 1 + \lceil \log n^{1-1/k} / \log(m/n^{1-1/k}) \rceil$) [Abraham, Chechik, Talwar '14]

Regime 4: $2k-1$ (large constant)

Fully dynamic:

- Stretch $2^{O(\rho k)}$, update time $\tilde{O}(m^{1/2}n^{1/k})$, and query time $O(k^2\rho^2)$ (for $\rho = 1 + \lceil \log n^{1-1/k} / \log(m/n^{1-1/k}) \rceil$) [Abraham, Chechik, Talwar '14]
- Stretch $\tilde{O}(\log n)$, update/query time $O(m^{2/3+o(1)})$ [Chen, Goranci, Henzinger, Peng, Saranurak '20]

Regime 4: $2k-1$ (large constant)

Fully dynamic:

- Stretch $2^{O(\rho k)}$, update time $\tilde{O}(m^{1/2}n^{1/k})$, and query time $O(k^2\rho^2)$ (for $\rho = 1 + \lceil \log n^{1-1/k} / \log(m/n^{1-1/k}) \rceil$) [Abraham, Chechik, Talwar '14]
- Stretch $\tilde{O}(\log n)$, update/query time $O(m^{2/3+o(1)})$ [Chen, Goranci, Henzinger, Peng, Saranurak '20]
- Stretch $(256/\rho^2)^{4/\rho}$, update time n^ρ, query time $n^{\rho/8}$ (for $0 < \rho < 1$) [F, Goranci, Nazari, Skarlatos '23]

Regime 4: $2k-1$ (large constant)

Fully dynamic:

- Stretch $2^{O(\rho k)}$, update time $\tilde{O}(m^{1/2}n^{1/k})$, and query time $O(k^2\rho^2)$ (for $\rho = 1 + \lceil \log n^{1-1/k} / \log(m/n^{1-1/k}) \rceil$) [Abraham, Chechik, Talwar '14]
- Stretch $\tilde{O}(\log n)$, update/query time $O(m^{2/3+o(1)})$ [Chen, Goranci, Henzinger, Peng, Saranurak '20]
- Stretch $(256/\rho^2)^{4/\rho}$, update time n^ρ, query time $n^{\rho/8}$ (for $0 < \rho < 1$) [F, Goranci, Nazari, Skarlatos '23]
- Stretch $(\log\log n)^{2^{1/\rho^3}}$, update time $\tilde{O}(n^{O(\rho)})$, query time $\tilde{O}(2^{O(1/\rho)})$ **deterministic** (for $\frac{2}{(\log n)^{1/200}} < \rho < \frac{1}{400}$) [Chuzhoy, Zhang '23]

Regime 4: $2k-1$ (large constant)

Fully dynamic:

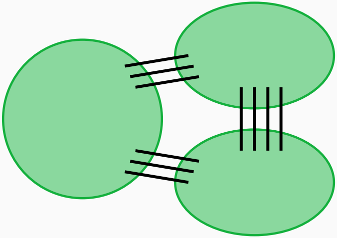
- Stretch $2^{O(\rho k)}$, update time $\tilde{O}(m^{1/2} n^{1/k})$, and query time $O(k^2 \rho^2)$ (for $\rho = 1 + \lceil \log n^{1-1/k} / \log(m/n^{1-1/k}) \rceil$) [Abraham, Chechik, Talwar '14]
- Stretch $\tilde{O}(\log n)$, update/query time $O(m^{2/3+o(1)})$ [Chen, Goranci, Henzinger, Peng, Saranurak '20]
- Stretch $(256/\rho^2)^{4/\rho}$, update time n^ρ, query time $n^{\rho/8}$ (for $0 < \rho < 1$) [F, Goranci, Nazari, Skarlatos '23]
- Stretch $(\log \log n)^{2^{1/\rho^3}}$, update time $\tilde{O}(n^{O(\rho)})$, query time $\tilde{O}(2^{O(1/\rho)})$ **deterministic** (for $\frac{2}{(\log n)^{1/200}} < \rho < \frac{1}{400}$) [Chuzhoy, Zhang '23]
- Stretch $2^{\mathrm{poly}(1/\rho)}$, update time $O(n^\rho)$, query time $O(\log \log n / \rho^4)$ (for $\frac{1}{\log^c n} < \rho < 1$) **deterministic** + **worst case** [Haeupler, Long, Saranurak '24]

Expander Decompositions

[Spielman, Teng '04] [Nanongkai, Saranurak, Wulff-Nilsen '17]

Partition of nodes into clusters $C_1, \ldots, C_k$ such that

- each $G[C_i]$ is a ϕ-expander
- #inter-cluster edges = $\phi m \log n$

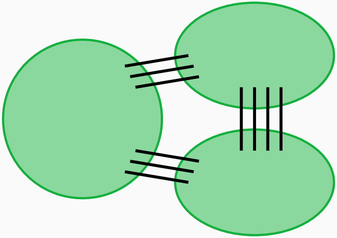


Expander Decompositions

[Spielman, Teng ’04] [Nanongkai, Saranurak, Wulff-Nilsen ’17]

Partition of nodes into clusters $C_1, \ldots, C_k$ such that

- each $G[C_i]$ is a ϕ-expander
  $\rightarrow \mathrm{Diam}(G[C_i]) = O(\log(m)/\phi)$
- #inter-cluster edges = $\phi m \log n$

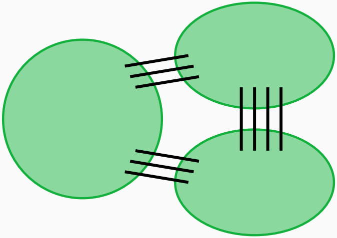


Expander Decompositions

[Spielman, Teng '04] [Nanongkai, Saranurak, Wulff-Nilsen '17]

Yes:

Partition of nodes into clusters $C_1, \ldots, C_k$ such that



- each $G[C_i]$ is a ϕ-expander
  $\rightarrow \mathrm{Diam}(G[C_i]) = O(\log(m)/\phi)$
- #inter-cluster edges = $\phi m \log n$

But:

- Inherent logarithmic blowup
  $\rightarrow$ Prohibitive for vertex sparsifier hierarchies

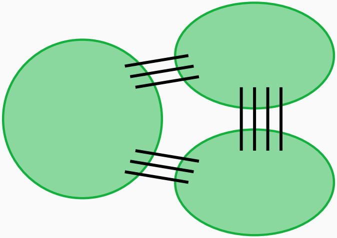
Expander Decompositions

[Spielman, Teng '04] [Nanongkai, Saranurak, Wulff-Nilsen '17]

Yes:

Partition of nodes into clusters $C_1, \ldots, C_k$ such that

- each $G[C_i]$ is a ϕ-expander
  $\rightarrow \mathrm{Diam}(G[C_i]) = O(\log(m)/\phi)$
- #inter-cluster edges = $\phi m \log n$



But:

- Inherent logarithmic blowup
  $\rightarrow$ Prohibitive for vertex sparsifier hierarchies
- Development of new types of decompositions

Detour: Dynamic Spanner Algorithms

Definition ([Peleg, Schäffer ’89])

A **spanner** of **stretch** t of $G = (V, E)$ is a subgraph $H = (V, E')$ such that

$$dist_H(u, v) \leq t \cdot dist_G(u, v)$$

for all pairs of nodes $u, v \in V$.

Detour: Dynamic Spanner Algorithms

Definition ([Peleg, Schäffer '89])

A **spanner** of **stretch** t of $G = (V, E)$ is a subgraph $H = (V, E')$ such that

$$dist_H(u, v) \leq t \cdot dist_G(u, v)$$

for all pairs of nodes $u, v \in V$.

- Typical trade-off: stretch $(2k - 1)$, size $O(n^{1+1/k})$

Detour: Dynamic Spanner Algorithms

Definition ([Peleg, Schäffer '89])

A **spanner** of **stretch** t of $G = (V, E)$ is a subgraph $H = (V, E')$ such that

$$dist_H(u, v) \leq t \cdot dist_G(u, v)$$

for all pairs of nodes $u, v \in V$.

- Typical trade-off: stretch $(2k - 1)$, size $O(n^{1+1/k})$
- Can run any sparsity-sensitive dynamic APSP algorithm on low-recourse spanner

Detour: Dynamic Spanner Algorithms

Definition ([Peleg, Schäffer '89])

A **spanner** of **stretch** t of $G = (V, E)$ is a subgraph $H = (V, E')$ such that

$$dist_H(u, v) \leq t \cdot dist_G(u, v)$$

for all pairs of nodes $u, v \in V$.

- Typical trade-off: stretch $(2k - 1)$, size $O(n^{1+1/k})$
- Can run any sparsity-sensitive dynamic APSP algorithm on low-recourse spanner
- Many papers on dynamic spanners ranging from [Ausiello, Franciosa, Italiano '05] to [Chuzhoy, Parter '25]

Detour: Dynamic Spanner Algorithms

Definition ([Peleg, Schäffer '89])

A **spanner** of **stretch** t of $G = (V, E)$ is a subgraph $H = (V, E')$ such that

$$dist_H(u, v) \leq t \cdot dist_G(u, v)$$

for all pairs of nodes $u, v \in V$.

- Typical trade-off: stretch $(2k - 1)$, size $O(n^{1+1/k})$
- Can run any sparsity-sensitive dynamic APSP algorithm on low-recourse spanner
- Many papers on dynamic spanners ranging from [Ausiello, Franciosa, Italiano '05] to [Chuzhoy, Parter '25]
- Several open problems
- Techniques overlap with dynamic shortest paths
- Additionally, the problem has a "local" flavor

$\omega(1)$ (superconstant)

Regime 5: Superconstant

Motivation:

- Subpolynomial update and query time
- Large stretch ok if it goes into running time of another algorithm

Regime 5: Superconstant

Motivation:

- Subpolynomial update and query time
- Large stretch ok if it goes into running time of another algorithm

Partially dynamic:

- Decremental: Stretch $n^{o(1)}$, total update time $m^{1+o(1)}$ [Chuzhoy '21; Bernstein, Probst Gutenberg, Saranurak '21]
- Incremental: Stretch $\tilde{O}(1)$, total update time $\tilde{O}(m)$, query time $O(\log \log n)$ [F, Nazari, Probst Gutenberg '23]

Regime 5: $\omega(1)$ (superconstant)

Fully dynamic:

- Stretch $O(\log n)^{3k-2}$, update time time $m^{1/k+o(1)} \cdot O(\log n)^{4k-2}$, query time $O(k(\log n)^2)$ [F, Goranci, Henzinger '21]

Regime 5: $\omega(1)$ (superconstant)

Fully dynamic:

- Stretch $O(\log n)^{3k-2}$, update time time $m^{1/k+o(1)} \cdot O(\log n)^{4k-2}$, query time $O(k(\log n)^2)$ [F, Goranci, Henzinger '21]
  $n^{o(1)}, n^{o(1)}, n^{o(1)}$

Regime 5: $\omega(1)$ (superconstant)

Fully dynamic:

- Stretch $O(\log n)^{3k-2}$, update time time $m^{1/k+o(1)} \cdot O(\log n)^{4k-2}$, query time $O(k(\log n)^2)$ [F, Goranci, Henzinger '21]
  $n^{o(1)}, n^{o(1)}, n^{o(1)}$
- Stretch $n^{o(1)}$, update time $n^{o(1)}$, query $n^{o(1)}$ **deterministic** + **worst case** [Kyng, Meierhans, Probst Gutenberg '24]

Regime 5: $\omega(1)$ (superconstant)

Fully dynamic:

- Stretch $O(\log n)^{3k-2}$, update time time $m^{1/k+o(1)} \cdot O(\log n)^{4k-2}$, query time $O(k(\log n)^2)$ [F, Goranci, Henzinger '21]
  $n^{o(1)}, n^{o(1)}, n^{o(1)}$
- Stretch $n^{o(1)}$, update time $n^{o(1)}$, query $n^{o(1)}$ **deterministic** + **worst case** [Kyng, Meierhans, Probst Gutenberg '24]

Open Problem

Stretch $\tilde{O}(1)$, update time $\tilde{O}(1)$, query time $\tilde{O}(1)$

ASZ Construction

Initially developed for the PRAM model [Andoni, Stein, Zhong '20]

Overall setup:

- Hierarchy of $k = \Theta(\log \log n)$ sparsifiers: $G = H_1, H_2, \dots, H_k$
- $|V(H_{i+1})| = |V(H_i)|/b_i$ for double exponentially increasing b_i's

$$|V(H_i)| = O\left(\frac{n}{b_1 \cdot b_2 \cdot \cdots \cdot b_{i-1}}\right)$$

$$|E(H_i)| \leq m + O\left(\frac{n}{b_1 \cdot b_2 \cdot \cdots \cdot b_{i-1}} \cdot b_i\right)$$

- H_i is an α-approximation of H_{i-1} for some constant α
  $\alpha^k = \operatorname{polylog} n$

One Level of ASZ

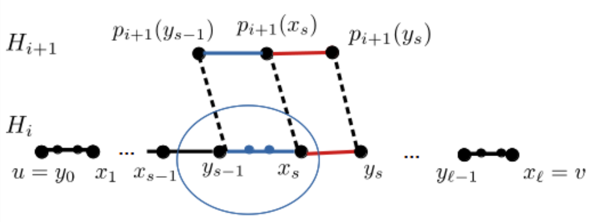
- Nodes of H_{i+1}: Randomized hitting set of size $\tilde{O}(n/b_i)$
- Compute b_i-ball around each node (b_i closest nodes)
  b_i-ball of u contains sampled node $p_i(u)$ ("pivot")

One Level of ASZ

- Nodes of H_{i+1}: Randomized hitting set of size $\tilde{O}(n/b_i)$
- Compute b_i-ball around each node (b_i closest nodes)
  b_i-ball of u contains sampled node $p_i(u)$ ("pivot")
- "Ball edges": $(p_i(u), p_i(v))$ for every u and v in b_i-ball of u
- "Projected edges": $(p_i(u), p_i(v))$ for every edge (u, v) of H_i

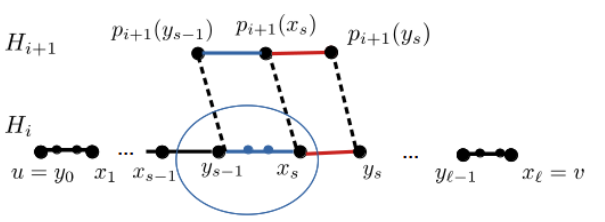
One Level of ASZ

- Nodes of H_{i+1}: Randomized hitting set of size $\tilde{O}(n/b_i)$
- Compute b_i-ball around each node (b_i closest nodes)
  b_i-ball of u contains sampled node $p_i(u)$ ("pivot")
- "Ball edges": $(p_i(u), p_i(v))$ for every u and v in b_i-ball of u
- "Projected edges": $(p_i(u), p_i(v))$ for every edge (u, v) of H_i
- $|E(H_{i+1})| \leq |E(H_i)| + |V(H_i)| \cdot b_i$



One Level of ASZ

- Nodes of H_{i+1}: Randomized hitting set of size $\tilde{O}(n/b_i)$
- Compute b_i-ball around each node (b_i closest nodes)
  b_i-ball of u contains sampled node $p_i(u)$ ("pivot")
- "Ball edges": $(p_i(u), p_i(v))$ for every u and v in b_i-ball of u
- "Projected edges": $(p_i(u), p_i(v))$ for every edge (u, v) of H_i
- $|E(H_{i+1})| \leq |E(H_i)| + |V(H_i)| \cdot b_i$



segment $y_{s-1} \to y_s$ approximated in H_{i+1} with multiplicative stretch α and additive stretch $d_{H_i}(y_s, p_i(y_s))$

Summary

- Each "regime" needs slightly different treatment
- Old problems, many new techniques
- Similar stories for dynamic matching, dynamic connectivity, dynamic spanners, ...

Summary

- Each “regime” needs slightly different treatment
- Old problems, many new techniques
- Similar stories for dynamic matching, dynamic connectivity, dynamic spanners, ...

Thank you!