Deterministic Incremental APSP with Polylogarithmic Update Time and Stretch

Sebastian Forster, né Krinninger

University of Salzburg

Dagstuhl Seminar 22461 “Dynamic Graph Algorithms” (Nov. 2022)

Joint work with Yasamin Nazari and Maximilian Probst Gutenberg

This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 947702). Supported by the Austrian Science Fund (FWF): P 32863-N. The research leading to these results has received funding from the grant “Algorithms and complexity for high-accuracy flows and convex optimization” (no. 200021 204787) of the Swiss National Science Foundation.

Our Result

Theorem

There is a deterministic algorithm that, given an undirected graph with real edge weights in $[1, W]$ *undergoing edge insertions, maintains in total time* $O(m \log n \log\log n + n \log^6(nW) \log\log n)$ *over all updates a distance oracle with polylogarithmic stretch and query time* $O(\log\log n)$*, where* n *denotes the number of vertices and* m *denotes the final number of edges of the graph.*

Our Result

Theorem

There is a deterministic algorithm that, given an undirected graph with real edge weights in $[1, W]$ *undergoing edge insertions, maintains in total time* $O(m \log n \log\log n + n \log^6(nW) \log\log n)$ *over all updates a distance oracle with polylogarithmic stretch and query time* $O(\log\log n)$*, where* n *denotes the number of vertices and* m *denotes the final number of edges of the graph.*

Conditional lower bound

Constant stretch likely needs polynomial update time [Abboud, Bringmann, Khoury, Zamir '22, Abboud, Bringmann, Fischer '22]

Our Result

Theorem

There is a deterministic algorithm that, given an undirected graph with real edge weights in $[1, W]$ *undergoing edge insertions, maintains in total time* $O(m \log n \log\log n + n \log^6(nW) \log\log n)$ *over all updates a distance oracle with polylogarithmic stretch and query time* $O(\log\log n)$*, where* n *denotes the number of vertices and* m *denotes the final number of edges of the graph.*

Conditional lower bound

Constant stretch likely needs polynomial update time [Abboud, Bringmann, Khoury, Zamir '22, Abboud, Bringmann, Fischer '22]

Notable:

- No Even-Shiloach tree
- No expander decomposition
- No Thorup-Zwick based construction

$n^{o(1)}$ Barrier

Decremental:

- [Chechik '18] (rand.) stretch $O(\log n)$, total time $mn^{o(1)}$
- [Łącki, Nazari '22] (rand.) stretch $O(\log n)$, total time $\tilde{O}(m + n^{1+o(1)})$
- [Chuzhoy '21] (det.) polylog. stretch, total time $O(m^{1+\delta})$
- [Bernstein, Chechik] (det.) $(1+\epsilon)$-approx. SSSP, total time $\tilde{O}(n^2)$.

Incremental:

- [Chen, Goranci, Henzinger, Peng, Saranurak '20] (det.) stretch $O(1)$, total time $O(m^{1+o(1)})$

A Case for Partially Dynamic Algorithms

Decremental:

- Hope: extend to fully dynamic by reductions
- Useful for static algorithms
  - multicommodity flow [Mądry ’10,Chuzhoy ’21]
  - (approximate) min-cost flow [Bernstein, Probst Gutenberg, Saranurak ’21, Chen, Kyng, Liu, Peng, Probst Gutenberg, Sachdeva ’22]
  - expander routing

A Case for Partially Dynamic Algorithms

Decremental:

- Hope: extend to fully dynamic by reductions
- Useful for static algorithms
  - multicommodity flow [Mądry ’10,Chuzhoy ’21]
  - (approximate) min-cost flow [Bernstein, Probst Gutenberg, Saranurak ’21, Chen, Kyng, Liu, Peng, Probst Gutenberg, Sachdeva ’22]
  - expander routing

Incremental:

- Natural growth processes (co-authors, Wikipedia links, ...)
- Search for implementable algorithms

Static Construction: Hierarchy

[Andoni, Stein, Zhong '20]

Overall setup:

- Hierarchy of $k = \Theta(\log\log n)$ sparsifiers: $G = H_1, H_2, \ldots, H_k$
- $|V(H_{i+1})| = |V(H_i)|/b_i$ 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
  $\alpha^k = \operatorname{polylog} n$

Static Construction: One Level

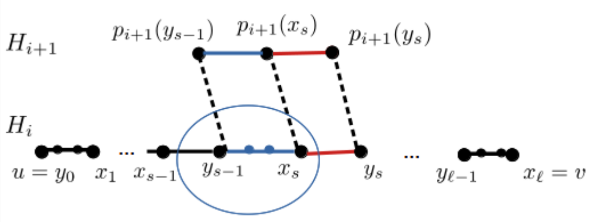
- Nodes of H_{i+1}: Randomized hitting set of size $\tilde{O}(n/b_i)$
- Compute b_i-ball around each node (b_i closest nodes)
  b_i-ball of u contains sampled node $p_i(u)$ ("pivot")

Static Construction: One Level

- 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

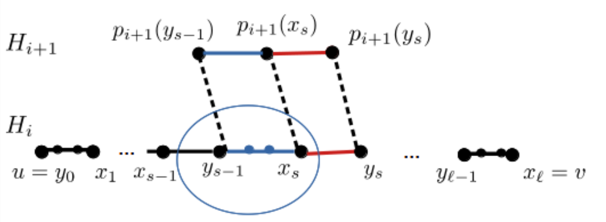
Static Construction: One Level

- 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
- $|E(H_{i+1})| \leq |E(H_i)| + |V(H_i)| \cdot b_i$



Static Construction: One Level

- 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 $d_{H_i}(y_s, p_i(y_s))$

Dynamic Algorithm: One Level

Approximate Pivots

Edge weights in H_i based on distances to pivots of endpoints

Lazy approach: Keep pivot until significantly closer pivot exists

Dynamic Algorithm: One Level

Approximate Pivots

Edge weights in H_i based on distances to pivots of endpoints

Lazy approach: Keep pivot until significantly closer pivot exists

Truncated Dijkstra primitive:

- Static computation of b_i-ball takes time $\tilde{O}(b_i^2)$

Dynamic Algorithm: One Level

Approximate Pivots

Edge weights in H_i based on distances to pivots of endpoints

Lazy approach: Keep pivot until significantly closer pivot exists

Truncated Dijkstra primitive:

- Static computation of b_i-ball takes time $\tilde{O}(b_i^2)$
- Let R be distance to current pivot and reassign pivot when b_i nodes at distance $R/2$

Dynamic Algorithm: One Level

Approximate Pivots

Edge weights in H_i based on distances to pivots of endpoints

Lazy approach: Keep pivot until significantly closer pivot exists

Truncated Dijkstra primitive:

- Static computation of b_i-ball takes time $\tilde{O}(b_i^2)$
- Let R be distance to current pivot and reassign pivot when b_i nodes at distance $R/2$ Happens after at most b_i^2 edge insertions to $R/2$-ball

Dynamic Algorithm: One Level

Approximate Pivots

Edge weights in H_i based on distances to pivots of endpoints

Lazy approach: Keep pivot until significantly closer pivot exists

Truncated Dijkstra primitive:

- Static computation of b_i-ball takes time $\tilde{O}(b_i^2)$
- Let R be distance to current pivot and reassign pivot when b_i nodes at distance $R/2$ Happens after at most b_i^2 edge insertions to $R/2$-ball
- Total time $\tilde{O}(|V(H_i)|b_i^4)$ for maintaining shrinking b_i-balls

Dynamic Algorithm: One Level

Approximate Pivots

Edge weights in H_i based on distances to pivots of endpoints

Lazy approach: Keep pivot until significantly closer pivot exists

Truncated Dijkstra primitive:

- Static computation of b_i-ball takes time $\tilde{O}(b_i^2)$
- Let R be distance to current pivot and reassign pivot when b_i nodes at distance $R/2$ Happens after at most b_i^2 edge insertions to $R/2$-ball
- Total time $\tilde{O}(|V(H_i)|b_i^4)$ for maintaining shrinking b_i-balls

Deterministic maintenance of pivots:

- Dynamic version of Greedy

Dynamic Algorithm: One Level

Approximate Pivots

Edge weights in H_i based on distances to pivots of endpoints

Lazy approach: Keep pivot until significantly closer pivot exists

Truncated Dijkstra primitive:

- Static computation of b_i-ball takes time $\tilde{O}(b_i^2)$
- Let R be distance to current pivot and reassign pivot when b_i nodes at distance $R/2$ Happens after at most b_i^2 edge insertions to $R/2$-ball
- Total time $\tilde{O}(|V(H_i)|b_i^4)$ for maintaining shrinking b_i-balls

Deterministic maintenance of pivots:

- Dynamic version of Greedy
- Charging scheme: each pivot reduces approximate pivot distance of b_i nodes by a constant factor
  In total: $\tilde{O}(|V(H_i)|/b_i)$ pivots $(= |V(H_{i+1})|)$


The
End


GAME
OVER

The problem:

- Incremental algorithm $\mathscr{A}_i$ maintaining H_{i+1} based on H_i
- Edges added to H_{i+1} appear as **insertions** to $\mathscr{A}_{i+1}$

Dynamic Algorithm: Hierarchy

The problem:

- Incremental algorithm $\mathcal{A}_i$ maintaining H_{i+1} based on H_i
- Edges added to H_{i+1} appear as **insertions** to $\mathcal{A}_{i+1}$
- Number of edges added to H_{i+1}: $O(|V(H_i)|b_i^2 \log n + |E(H_i)| \log n)$

Dynamic Algorithm: Hierarchy

The problem:

- Incremental algorithm $\mathscr{A}_i$ maintaining H_{i+1} based on H_i
- Edges added to H_{i+1} appear as **insertions** to $\mathscr{A}_{i+1}$
- Number of edges added to H_{i+1}: $O(|V(H_i)|b_i^2 \log n + |E(H_i)| \log n)$
- For $k = \Theta(\log\log n)$ levels: $m \cdot O(\log n)^{\log\log n}$ insertions (at best)
- Will not give $O(m \operatorname{polylog} n)$ total update time

Solutions and More Challenges

Idea: Avoid "projections of projections"

Solutions and More Challenges

Idea: Avoid "projections of projections"

- Directly project to *all* higher levels

Solutions and More Challenges

Idea: Avoid “projections of projections”

- Directly project to *all* higher levels
- projected edge weight = $\sum$ (approximate) distances to pivots + original edge weight

Solutions and More Challenges

Idea: Avoid "projections of projections"

- Directly project to *all* higher levels
- projected edge weight = $\sum$ (approximate) distances to pivots + original edge weight
- Project in a lazy fashion whenever this sum changes significantly

Solutions and More Challenges

Idea: Avoid "projections of projections"

- Directly project to *all* higher levels
- projected edge weight = $\sum$ (approximate) distances to pivots + original edge weight
- Project in a lazy fashion whenever this sum changes significantly
- "Pivot chains" form a tree structure

Solutions and More Challenges

Idea: Avoid "projections of projections"

- Directly project to *all* higher levels
- projected edge weight = $\sum$ (approximate) distances to pivots + original edge weight
- Project in a lazy fashion whenever this sum changes significantly
- "Pivot chains" form a tree structure
- Maintain sums with dynamic tree data structure

Solutions and More Challenges

Idea: Avoid "projections of projections"

- Directly project to *all* higher levels
- projected edge weight = $\sum$ (approximate) distances to pivots + original edge weight
- Project in a lazy fashion whenever this sum changes significantly
- "Pivot chains" form a tree structure
- Maintain sums with dynamic tree data structure

Challenge:

- We've opened up the "black box"
- But: level-by-level analysis was very convenient for correctness proof

Solutions and More Challenges

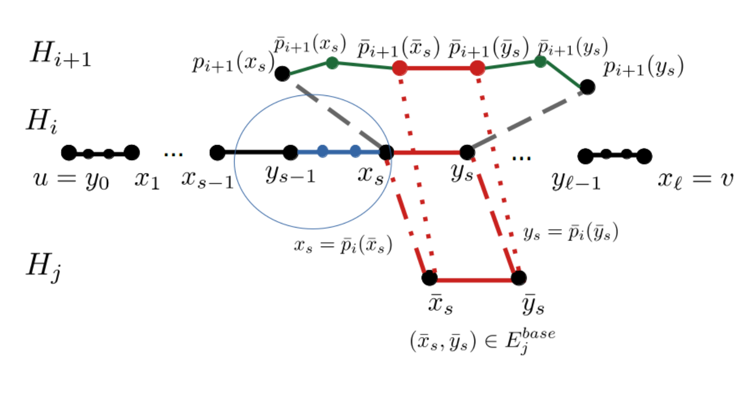
Idea: Avoid "projections of projections"

- Directly project to *all* higher levels
- projected edge weight = $\sum$ (approximate) distances to pivots + original edge weight
- Project in a lazy fashion whenever this sum changes significantly
- "Pivot chains" form a tree structure
- Maintain sums with dynamic tree data structure

Challenge:

- We've opened up the "black box"
- But: level-by-level analysis was very convenient for correctness proof
- Problem: pivots might be out of sync with actual projections

The Full Picture



Can make it work by maintaining additional types of edges

Challenges for Decremental Setting

Key challenge:

How to avoid too many changes of a node's pivot?

Challenges for Decremental Setting

Key challenge:

How to avoid too many changes of a node's pivot?

Non-working approach:

- Keep track of pivots at distance $[R, 2R]$

Challenges for Decremental Setting

Key challenge:

How to avoid too many changes of a node's pivot?

Non-working approach:

- Keep track of pivots at distance $[R, 2R]$
- Connect to one of them uniformly at random
- Would expect $O(\log n)$ changes of pivot in that range

Challenges for Decremental Setting

Key challenge:

How to avoid too many changes of a node's pivot?

Non-working approach:

- Keep track of pivots at distance $[R, 2R]$
- Connect to one of them uniformly at random
- Would expect $O(\log n)$ changes of pivot in that range

But: cannot simply maintain SSSP from set of pivots

(standard approach in Thorup-Zwick based constructions)

Questions

Decremental?

Worst-case update time?