A Simple Dynamic Spanner via APSP

Rasmus Kyng*
ETH Zurich
kyng@inf.ethz.ch

Simon Meierhans*
ETH Zurich
mesimon@inf.ethz.ch

Gernot Zöcklein
ETH Zurich
gzoecklein@ethz.ch

September 14, 2025

Abstract

We give a simple algorithm for maintaining a $n^{o(1)}$-approximate spanner H of a graph G with n vertices as G receives edge updates by reduction to the dynamic All-Pairs Shortest Paths (APSP) problem. Given an initially empty graph G, our algorithm processes m insertions and n deletions in total time $m^{1+o(1)}$ and maintains an initially empty spanner H with total recourse $n^{1+o(1)}$. When the number of insertions is much larger than the number of deletions, this notably yields recourse sub-linear in the total number of updates.

Our algorithm only has a single $O(\log n)$ factor overhead in runtime and approximation compared to the underlying APSP data structure. Therefore, future improvements for APSP will directly yield an improved dynamic spanner.

1 Introduction

Given a graph $G = (V, E)$, a δ-approximate spanner H is a subgraph of G with the same vertex set such that $\text{dist}_H(u, v) \leq \delta \cdot \text{dist}_G(u, v)$ for every pair of vertices $u, v \in V$. Althöfer, Das, Dobkin, Joseph and Soares developed the now-classical greedy spanner [ADD+93].

Given an integer δ, their algorithm incrementally constructs a 2δ-approximate spanner H of $G = (V, E)$ as follows.

1) Initialize $H = (V, \emptyset)$.

2) Consider the edges $(u, v) \in E$ in arbitrary order. If $\text{dist}_H(u, v) > 2\delta$, add (u, v) to H.

When this process finishes, every edge on the shortest path between two arbitrary vertices in G can be replaced by a detour of length at most 2δ in H. The claimed approximation $\text{dist}_H(u, v) \leq 2\delta \cdot \text{dist}_G(u, v)$ directly follows.

Crucially, the graph H is also sparse. This follows from the folklore observation that high-girth graphs are sparse. The girth of a graph is the length of the shortest cycle. By construction, the girth of H is $2\delta + 1$, which implies total edge count is bounded by $|V|^{1+\frac{1}{\delta}}$. In this article we consider the important setting $\delta \geq \log(|V|)$.

Observation 1.1. *Every graph $H = (V, E_H)$ with girth $2\delta + 1$ contains at most $2|V|^{1+\frac{1}{\delta}}$ edges.*

Proof. Repeatedly remove vertices of degree at most $d = |V|^{\frac{1}{\delta}} + 1$ from H. This process removes at most $d \cdot |V|$ edges. Assume for the sake of a contradiction that there is some remaining graph $\widetilde{H} \subseteq H$ after this process terminates. Fix an arbitrary remaining vertex r, and consider the breath-first search tree $T \subseteq \widetilde{H}$ to depth δ from r. Since the girth of $\widetilde{H}$ is at least $2\delta + 1$, no two vertices at depth $< \delta$ in T share an edge in $\widetilde{H} \setminus T$. Thus, the number of vertices in T is at least $\sum_{i=0}^{\delta} (d-1)^i > (d-1)^{\delta} = |V|$ which is a contradiction. $\qquad\square$

Despite the simplicity of this framework, previous deterministic *dynamic algorithms* for maintaining a spanner $H \subseteq G$ as G undergoes updates were based on expander decompositions [CKL+22, BCK+23], which are arguably more natural for capturing cuts than distances.

However, in [BSS22] the authors observed that the greedy spanner construction can be used to maintain a spanner $H \subseteq G$ as G receives edge deletions with low amortized recourse. As part of their almost-linear time incremental min-cost flow algorithm, [CKL+24] presented a computationally efficient algorithm for maintaining a spanner of a fully-dynamic graph G based on the greedy spanner.[1] Their algorithm is complicated by the fact that they need to process vertex splits in addition to edge updates. In this article, we present a simplified algorithm for maintaining a spanner of an edge-dynamic graph via an All-Pairs Shortest Paths (APSP) oracle.

Theorem 1.2. *Given an initially empty edge-dynamic graph $G = (V, E)$ on $n = |V|$ vertices receiving m edge insertions and up to n edge deletions where $m \geq n$, there is an algorithm maintaining a $n^{o(1)}$-approximate spanner H with total update time $m^{1+o(1)}$ and total recourse $n^{1+o(1)}$.*

We note in particular that the total recourse is almost-optimal and only depends on the number of deletions and vertices, and is therefore completely independent of the number of insertions. We refer the reader to Definition 3.1 and Theorem 4.7 for a more fine grained analysis of the dependency on the underlying APSP data structure. Our algorithm can use any path-reporting approximate APSP data structure and only adds an $O(\log n)$ factor overhead in runtime and approximation.

Prior Work. In [BKS12], the authors present a randomized dynamic spanner with poly-logarithmic stretch and amortized update time against an oblivious adversary.[2] Then [FG19] significantly reduced the poly-logarithmic overhead in both size and update time. In [BvdBPG+22], the authors gave the first non-trivial algorithm for maintaining a dynamic spanner against an adaptive adversary. They maintain a spanner of near linear size with poly-logarithmic amortized update time and stretch. Deterministic algorithms for maintaining spanners of dynamic graphs with bounded degree that additionally undergo vertex splits were developed in the context of algorithms for minimum cost flow [CKL+22, BCK+23, CKL+24]. These have sub-polynomial overhead.

2 Preliminaries

Static and Dynamic Graphs. We let $G = (V, E)$ denote unweighted and undirected graphs with vertex set V and edge set E. When we want to refer to the vertex/edge set of some graph G, we sometimes use $V(G)$ and $E(G)$ respectively. For a graph $G = (V, E)$ and a pair of vertices

---

[1]They crucially exploit that the recourse is sub-linear in the number of insertions. Expander decomposition based algorithms seem less suitable for achieving such a result.

[2]Oblivious adversaries generate the sequence of updates at the start, i.e. independently from the random output of the algorithm. Adaptive adversaries can use the previous output to fool a randomized algorithm.

$u, v \in V$ we let $\text{dist}_G(u, v)$ denote the length of a shortest uv-path in G. We call a graph edge-dynamic if it receives arbitrary updates to E (i.e. edge insertions and deletions). We refer to the total number of such edge updates a graph receives as the recourse.

Edge Embeddings. Given two graph H and G on the same vertex set such that $H \subseteq G$, we let $\Pi_{G \to H}$ be a function that maps edges in $E(G)$ to paths in H. Given an embedding $\Pi_{G \to H}$, we define the edge congestion $\text{econg}(\Pi_{G \to H}, e) \overset{\text{def}}{=} |\{f \in E(G) : e \in \Pi_{G \to H}(f)\}|$ for $e \in E(H)$.

3 Dynamic All-Pairs Shortest Paths

We first define approximate *All-Pairs Shortest Paths* (APSP) data structures for edge-dynamic graphs.

Definition 3.1 (APSP)**.** *For an initially empty edge-dynamic graph $G = (V, E)$ a γ-approximate α-update β-query APSP data structure supports the following operations.*

- ADDEDGE(u, v) / REMOVEEDGE(u, v): *Add/Remove edge (u, v) from/to G in (amortized) time α.*

- DIST(u, v): *Returns a distance estimate $\widetilde{\text{dist}}(u, v)$ such that*

$$\text{dist}(u, v) \le \widetilde{\text{dist}}(u, v) \le \gamma \cdot \text{dist}(u, v).$$

  in (amortized) time β.

- PATH(u, v): *Returns a simple uv-path P in G of length $|P| \le \text{DIST}(u, v)$ in time $\beta \cdot |P|$.*

We refer the reader to [CZ23, KMPG24, HLS24] for recent results on path-reporting dynamic APSP. The parameters γ, α and β should currently be thought of as subpolynomial factors $n^{o(1)}$ in the number of vertices n.

4 A Dynamic Greedy Spanner

A Simple Low-Recourse Algorithm. We first present a very simple dynamic variant of the greedy spanner that merely limits the number of changes to H, i.e. its recourse. Then we describe the necessary changes for obtaining an efficient version using a dynamic APSP datastructure.

Let $G = (V, E)$ be an initially empty graph on n vertices. We maintain a $2 \log n$ spanner $H \subseteq G$ as follows.

- When an edge (u, v) is inserted to G, we check if $\text{dist}(u, v) > 2 \log n$. If so, we add it to H.

- When an edge (u, v) is inserted to G, we check if $\text{dist}(u, v) > 2 \log n$. If so, we add it to H. When an edge (u, v) is deleted from G, we remove it from H and re-insert all edges in $E(G) \setminus E(H)$ using the procedure described above.

This algorithm still maintains that the girth of H is at least $2 \log n + 1$, and therefore the graph H contains at most $O(n)$ edges at any point. Since at most n edges leave H due to deletions a total recourse bound of $O(n)$ follows directly.

A Simple Efficient Algorithm. To turn the above framework into an efficient algorithm, we maintain an embedding $\Pi_{G \to H}$ that maps each edge in G to a short path in H and thus certifies that such a path still exists. We then only have to re-insert all edges whose embedding paths use the deleted edge. To bound the number of re-insertions, we carefully manage the edge congestion of $\Pi_{G \to H}$. Finally, we use an APSP data structure (See Definition 3.1) to obtain distances and paths. In the following, we assume there are at most m insertions, n deletions and that $|V| = n$.

Our algorithm internally maintains two graphs: The current spanner H and a not yet congested sub-graph $\widehat{H} \subseteq H$. We maintain that the edge congestion is strictly less than m/n for every edge in $\widehat{H}$, and maintain access to distances and paths in $\widehat{H}$ via a γ-approximate α-update β-query APSP data structure $\mathcal{D}_{\widehat{H}}$.

- When an edge e is inserted, we check if $\mathcal{D}_{\widehat{H}}$ returns distance at most $2\gamma \log n$ between the endpoints. If this is the case, we query it for a witness path P and store P as the embedding path of e. Then we check if any of the edges on the path now have m/n embedding paths using them, and if so we remove them from $\widehat{H}$. Otherwise, we add the edge e to H and $\widehat{H}$.

- When an edge e is deleted, we remove it from $\widehat{H}$ and H and re-insert all edges that now have a broken embedding path using the procedure above.

See Algorithm 1 for detailed pseudocode.

Algorithm 1: DYNAMICSPANNER()

1 **procedure** INITIALIZE(n, m)
2     $G \leftarrow (V, \emptyset); H \leftarrow (V, \emptyset); \widehat{H} \leftarrow (V, \emptyset); \Pi_{G \to H} \leftarrow \emptyset$;   // Let $V = \{1, \ldots, n\}$.
3   Let $\mathcal{D}_{\widehat{H}}$ be a γ-approximate α-update β-query APSP datastructure (See Definition 3.1)
4 **procedure** INSERTEDGE$(e = (u, v))$
5     $G \leftarrow G \cup (e)$
6   **if** $\mathcal{D}_{\widehat{H}}.\text{DIST}(u, v) \leq \gamma \cdot 2 \cdot \log n$ **then**
7        $P \leftarrow \mathcal{D}_{\widehat{H}}.\text{PATH}(e); \Pi_{G \to H}(e) \leftarrow P$
8   **foreach** $f \in P$ **do**
9   **if** $\text{econg}(\Pi_{G \to H}, f) \geq \frac{m}{n}$ **then**
10              $\widehat{H} \leftarrow \widehat{H} \setminus f; \mathcal{D}_{\widehat{H}}.\text{REMOVEEDGE}(f)$
11   **else**
12        $H \leftarrow H \cup (e); \widehat{H} \leftarrow H \cup (e); \Pi_{G \to H}(e) \leftarrow (e); \mathcal{D}_{\widehat{H}}.\text{ADDEDGE}(e)$
13 **procedure** DELETEEDGE$(e = (u, v))$
14     $G \leftarrow G \setminus e$
15   **if** $e \in H$ **then**
16        $F \leftarrow \{f \in E(G) : e \in \Pi_{G \to h}(f)\}$;   // Edges with broken embedding paths.
17        $G \leftarrow G \setminus F$;   // Temporarily remove edges in F from G
18   **foreach** $f \in F$ **do**
19           $\Pi_{G \to H}(f) \leftarrow \emptyset$;   // Delete broken embeddings.
20   **foreach** $f \in F$ **do**
21   INSERTEDGE(f) ;   // Re-insert edges in F.

Analysis.   We first show that H is a $2 \cdot \gamma \cdot \log n$-spanner.

Lemma 4.1. *The graph H maintained by* DYNAMICSPANNER() *(Algorithm 1) is a $2 \cdot \gamma \cdot \log n$-spanner throughout.*

Proof. When an edge $e \in E(G)$ is inserted, it afterwards has a valid embedding path of length at most $2 \cdot \gamma \cdot \log n$ by the description of INSERTEDGE(e). Therefore the edge has a valid embedding path at this point. However, if this any edge on this path is deleted, the edge is re-inserted by the description of DELETEEDGE(e). This establishes the claim.   □

We then prove that the total number of times the routine INSERTEDGE(e) is called is bounded by $2m$. In the following, we assume without loss of generality that m/n is an integer.

Claim 4.2. *For all $f \in H$, $\mathrm{econg}(\Pi_{G \to H}, f) \leq m/n$ throughout.*

Proof. Whenever the edge congestion of an edge in $\widehat{H}$ reaches m/n it is removed from $\widehat{H}$ and $\mathcal{D}_{\widehat{H}}$. By the description of INSERTEDGE() such an edge is never used in embedding paths again. Therefore the edge congestion of $\mathrm{econg}(\Pi_{G \to H}, f)$ is bounded by m/n for every edge.   □

Corollary 4.3. *A sequence of at most m external insertions and n deletions causes at most $2m$ calls to* INSERTEDGE() *in total.*

Proof. Directly follows from Claim 4.2 and the description of DELETEEDGE().   □

Given the previous corollary, we are ready bound the total recourse of H.

Claim 4.4. $|E(H)| \leq O(\gamma \cdot n \log n)$ *throughout.*

Proof. We bound the edges in $E(H) \backslash E(\widehat{H})$ and in $E(\widehat{H})$ separately, starting with the former. These edges were congested by m/n paths when they were removed from $\widehat{H}$. Since the total number of edges ever added is $2m$, the total cumulative congestion that all the edges ever experience is at most $4 \cdot m \cdot \gamma \cdot \log n$ by the description of INSERTEDGE(). Therefore, the total number of edges in $E(H) \setminus E(\widehat{H})$ is at most $4 \cdot n \cdot \gamma \cdot \log n$. Finally, by Definition 3.1 the graph $\widehat{H}$ has girth at least $2 \log n + 1$, and therefore at most $O(n)$ edges by Observation 1.1.   □

Corollary 4.5. *The recourse of H is $O(\gamma \cdot n \log n)$.*

Proof. Follows from Claim 4.4 since at most n edges get removed from H.   □

It remains to bound the total time spent processing updates.

Lemma 4.6. *The algorithm processes m insertions and n deletions in total time $O(\beta \cdot \gamma \cdot m \cdot \log n + \alpha \cdot \gamma \cdot n \cdot \log n)$.*

Proof. There are at most $2m$ calls to INSERTEDGE(). Each such call causes a distance and possibly a path reporting query to $\mathcal{D}_{\widehat{H}}$. Whenever a path reporting query is called, the resulting path P has length at most $2\gamma \cdot \log n$. This causes a total runtime of $O(\beta \cdot \gamma \cdot m \cdot \log n)$.

Then, by Corollary 4.5, at most $O(\gamma \cdot n \log n)$ get added and therefore also removed from $\widehat{H}$. These cause $O(\gamma \cdot n \log n)$ update calls to $\mathcal{D}_{\widehat{H}}$. This causes total runtime $O(\alpha \cdot \gamma \cdot n \cdot \log n)$.

All other operations can be subsumed in the times above.   □

Our main theorem then follows directly.

Theorem 4.7. *Given a γ-approximate α-update β-query APSP data structure, there is an algorithm that maintains a $2\gamma \log n$-approximate spanner H of an initially empty graph G on n vertices throughout a sequence up to m insertions and n deletions in total time $O(\beta \cdot \gamma \cdot m \cdot \log n + \alpha \cdot \gamma \cdot n \cdot \log n)$. The total recourse of H is $O(\alpha \cdot \gamma \cdot n \log n)$.*

Proof. Follows from Corollary 4.5 and Lemma 4.6. □

Finally, Theorem 1.2 follows from Theorem 4.7 in conjunction with any of the APSP data structures of [CZ23, KMPG24, HLS24].

Acknowledgments

5 References

[ADD+93]   Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993. 1

[BCK+23]   J. Brand, L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. Gutenberg, S. Sachdeva, and A. Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514, Los Alamitos, CA, USA, nov 2023. IEEE Computer Society. 2

[BKS12]   Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4), oct 2012. 2

[BSS22]   Sayan Bhattacharya, Thatchaphol Saranurak, and Pattara Sukprasert. Simple Dynamic Spanners with Near-Optimal Recourse Against an Adaptive Adversary. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 2

[BvdBPG+22] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-Dynamic Graph Sparsifiers Against an Adaptive Adversary. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 2

[CKL+22]   Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022. 2

[CKL+24]   Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, s-t shortest path, and minimum-cost flow. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, page 1165–1173, New York, NY, USA, 2024. Association for Computing Machinery. 2

[CZ23]   Julia Chuzhoy and Ruimin Zhang. A new deterministic algorithm for fully dynamic all-pairs shortest paths. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 1159–1172, New York, NY, USA, 2023. Association for Computing Machinery. 3, 6

[FG19]   Sebastian Forster and Gramoz Goranci. Dynamic low-stretch trees via dynamic low-diameter decompositions. In *Proceedings of the 51st Annual ACM SIGACT*

Symposium on Theory of Computing, STOC 2019, page 377–388, New York, NY, USA, 2019. Association for Computing Machinery. 2

[HLS24]   Bernhard Haeupler, Yaowei Long, and Thatchaphol Saranurak. Dynamic deterministic constant-approximate distance oracles with n^ϵ worst-case update time, 2024. 3, 6

[KMPG24]   Rasmus Kyng, Simon Meierhans, and Maximilian Probst Gutenberg. A dynamic shortest paths toolbox: Low-congestion vertex sparsifiers and their applications. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, page 1174–1183, New York, NY, USA, 2024. Association for Computing Machinery. 3, 6