Brief Announcement:$(1+\epsilon)$-Approximate Shortest Paths in Dynamic Streams

Michael Elkin
elkinm@cs.bgu.ac.il
Ben-Gurion University of the Negev
Beer-Sheva, Israel

Chhaya Trehan
c.trehan@lse.ac.uk
London School of Economics & Political Science
London, United Kingdom

ABSTRACT

Computing approximate shortest paths in the dynamic streaming setting is a fundamental challenge that has been intensively studied. Currently existing solutions for this problem either build a sparse multiplicative spanner of the input graph and compute shortest paths in the spanner offline, or compute an exact single source BFS tree. Solutions of the first type are doomed to incur a stretch-space tradeoff of $2\kappa - 1$ versus $n^{1+1/\kappa}$, for an integer parameter κ. (In fact, existing solutions also incur an extra factor of $1+\epsilon$ in the stretch for weighted graphs, and an additional factor of $\log^{O(1)} n$ in the space.) The only existing solution of the second type uses $n^{1/2-O(1/\kappa)}$ passes over the stream (for space $O(n^{1+1/\kappa})$), and applies only to unweighted graphs.

We show that $(1+\epsilon)$-approximate single-source shortest paths can be computed with $\tilde{O}(n^{1+1/\kappa})$ space using just *constantly* many passes in unweighted graphs, and polylogarithmically many passes in weighted graphs. Moreover, the same result applies for multi-source shortest paths, as long as the number of sources is $O(n^{1/\kappa})$. We achieve these results by devising efficient dynamic streaming constructions of $(1+\epsilon,\beta)$-spanners and hopsets. We believe that these constructions are of independent interest.

CCS CONCEPTS

• **Theory of computation → Streaming models**; **Streaming, sublinear and near linear time algorithms**; **Shortest paths**; **Sparsification and spanners**.

KEYWORDS

Shortest Paths, Dynamic Streams , Approximate Distances, Spanners, Hopsets

ACM Reference Format:
Michael Elkin and Chhaya Trehan. 2022. Brief Announcement:$(1+\epsilon)$-Approximate Shortest Paths in Dynamic Streams. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC '22), July 25–29, 2022, Salerno, Italy.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3519270.3538469

1 INTRODUCTION

In this report we give a brief overview of our work on approximate shortest paths in dynamic streams. We refer the reader to the full version [15] for a detailed exposition. One of the most common theoretical models for addressing the challenge of processing massive graphs is the *semi-streaming* model of computation [1, 17, 27]. In this model, edges of an input n-vertex graph $G=(V,E)$ arrive one after another, while the storage capacity of the algorithm is limited. One usually allows space of $\tilde{O}(n)$, though it is often relaxed to $n^{1+o(1)}$, sometimes to $O(n^{1+\rho})$, for an arbitrarily small constant parameter $\rho > 0$, or even to $O(n^{1+\eta_0})$, for some fixed constant η_0, $0 < \eta_0 < 1$. Generally, the model allows several passes over the stream, and the objective is to keep both the number of passes and the space complexity of the algorithm in check. The model comes in two main variations. In the first one, called *static* or *insertion-only* model [17], the edges can only arrive, and never get deleted. In the more general *dynamic* (also known as *turnstile*) streaming setting [1], edges may either arrive or get deleted.

In this work, we address the problem of computing computing approximate *shortest paths* in the dynamic streaming model by constructing *spanners* and *hopsets*. For a pair of parameters $\alpha \geq 1$, $\beta \geq 0$, given an undirected graph $G=(V,E)$, a subgraph $G'=(V,H)$ of G is said to be an *(α,β)-spanner* of G, if for every pair $u,v \in V$, it holds that $d_{G'}(u,v) \leq \alpha \cdot d_G(u.v) + \beta$, where d_G and $d_{G'}$ are the distance functions of G and G', respectively. A spanner with $\beta = 0$ is called a *multiplicative* spanner and one with $\alpha = 1$ is called an *additive* spanner. There is another important variety of spanners called *near-additive* spanners for which $\beta \geq 0$ and $\alpha = 1+\epsilon$, for an arbitrarily small $\epsilon > 0$. Given an n-vertex weighted undirected graph $G=(V,E,\omega)$, consider another graph $G'=(V,H,\omega')$ on the same vertex set V. Define the union graph $G'' = GUG' = (V, E \cup H, \omega'')$, where $\omega''(e) = \omega'(e)$ for $e \in H$ and $\omega''(e) = \omega(e)$ for $e \in E \setminus H$. For a positive integer parameter β and a pair $u,v \in V$, a *β-bounded distance*, denoted $d_G^{(\beta)}(u,v)$,between u and v in G is the length of a shortest u-v path in G that contains at most β edges. For a parameter $\epsilon > 0$ and a positive integer β, a graph $G'=(V,H,\omega')$ on the same vertex set as G is called a *$(1+\epsilon,\beta)$-hopset* of G, if for every pair of vertices $u,v \in V$, we have $d_G(u,v) \leq d^{(\beta)}_{G \cup G'}(u,v) \leq (1+\epsilon) \cdot d_G(u,v)$.

Both spanners and hopsets are fundamental graph-algorithmic constructs for approximating distances and shortest paths. Hopsets are particularly useful in various computational settings in which computing shortest paths with a limited number of hops is significantly easier than computing them with no limitation on the number of hops. A partial list of these settings includes streaming, distributed, parallel and centralized dynamic models. Recently, hopsets were also shown to be useful for computing approximate shortest paths in the standard centralized model of computation as well [12].

Most of the algorithms for computing (approximate) distances and shortest paths in the streaming setting compute a sparse spanner, and then employ it for computing exact shortest paths and distances in it offline, i.e., in the post-processing, after the stream is over [2, 4, 8, 10, 16, 18–20, 24]. The algorithms of [4, 8, 18] apply to unweighted graphs, but they can be extended to weighted graphs by running many copies of them in parallel, one for each weight scale. Let $\Lambda = \Lambda(G)$ denote the *aspect ratio* of the graph, i.e., $\Lambda = \frac{max_{u,v \in V} d_G(u,v)}{min_{u,v \in V} d_G(u,v)}$. Also, let $\epsilon \geq 0$ be a slack parameter. Then by running $O(\frac{\log \Lambda}{\epsilon})$ copies of the algorithm for unweighted graphs and taking the union of their outputs as the ultimate spanner, one obtains a one-pass static streaming algorithm for $2(1+\epsilon)\kappa$-spanner with $\tilde{O}(n^{1+\frac{1}{\kappa}} \cdot (\log \Lambda)/\epsilon)$ edges. [14] Elkin and Zhang [16] and Elkin and Neiman [10] devised static streaming algorithms for building $(1+\epsilon, \beta_{EN})$-spanners with $\tilde{O}(n^{1+1/\kappa})$ edges using β_{EN} passes over the stream and space $\tilde{O}(n^{1+\rho})$, where, for any parameters $\epsilon, \rho > 0$ and $\kappa = 1, 2, \ldots$. $\beta = \beta_{EN} = \left(\frac{\log \kappa\rho + 1/\rho}{\epsilon}\right)^{\log \kappa\rho + 1/\rho}$.

Recently [5] devised a *dynamic streaming* algorithm that uses $\tilde{O}(n/p)$ passes (for parameter $1 \leq p \leq n$ as above) and space $\tilde{O}(n + p^2)$ for the SSSP (*Single Source Shortest Paths*) problem, and space $\tilde{O}(|S|n + p^2)$ for the $S \times V$ approximate shortest path computation. Ahn, Guha and McGregor [2] devised the first *dynamic streaming* algorithm for computing approximate distances. Their algorithm computes a $(2\kappa - 1)$-spanner (for any $\kappa = 1, 2, \ldots$) with $\tilde{O}(n^{1+1/\kappa})$ edges (and the same space complexity) in κ passes over the stream. A number of dynamic streaming algorithms for spanner construction were devised thereafter. [2, 19, 20, 24]

All known dynamic streaming algorithms for computing approximately shortest paths (with space $\tilde{O}(n^{1+1/\kappa})$, for a parameter $\kappa = 1, 2, \ldots$), can be divided into two categories. The algorithms in the first category build a sparse multiplicative $(2\kappa-1)$-spanner, and they provide a *multiplicative* stretch of at least $2\kappa - 1$ [2, 19, 20, 24]. Moreover, due to existential lower bounds for spanners, this approach is doomed to provide stretch of at least $\frac{4}{3}\kappa$ [26]. The algorithms in the second category compute *exact single source* shortest paths in *unweighted* graphs, but they employ $n^{1/2 - O(1/\kappa)}$ passes [5, 9].

Our Results: We present the first dynamic streaming algorithm for SSSP with stretch $1 + \epsilon$, space $\tilde{O}(n^{1+1/\kappa})$, and *constant* (as long as ϵ and κ are constant) number of passes for unweighted graphs. For weighted graphs, our number of passes is *polylogarithmic* in n. Specifically, the number of passes of our SSSP algorithm is $(\frac{\kappa}{\epsilon})^{\kappa(1+o(1))}$ for unweighted graphs, and $\left(\frac{(\log n)\kappa}{\epsilon}\right)^{\kappa(1+o(1))}$ for weighted ones. Moreover, within the same complexity bounds, our algorithm can compute $(1+\epsilon)$-approximate $S \times V$ shortest paths from $|S| = n^{1/\kappa}$ designated sources. Moreover, in *unweighted* graphs, *all* pairs almost shortest paths with stretch $(1 + \epsilon, \left(\frac{\kappa}{\epsilon}\right)^{\kappa})$ can also be computed within the same space and number of passes. Note that our multiplicative stretch $(1 + \epsilon)$ is dramatically better than $(2\kappa - 1)$, exhibited by algorithms based on multiplicative spanners [2, 19, 20, 24]. Our number of passes is *independent of* n, for unweighted graphs, and depends only polylogarithmically on n for weighted ones.

2 TECHNICAL OVERVIEW

We devise two algorithms. One of them builds a near-additive spanner and the other builds a hopset. The following two theorems summarize the results of our spanner and hopset constructions.

Theorem 2.1. *For any unweighted graph $G(V, E)$ on n vertices, parameters $0 < \epsilon < 1$, $\kappa \geq 2$, and $\rho > 0$, our dynamic streaming algorithm computes a $(1 + \epsilon, \beta)$-spanner with $O_{\epsilon,\kappa,\rho}(n^{1+1/\kappa})$ edges, in $O(\beta)$ passes using $O(n^{1+\rho} \log^4 n)$ space whp, where β is given by:*

$$\beta = \left(\frac{\log \kappa\rho + 1/\rho}{\epsilon}\right)^{\log \kappa\rho + 1/\rho}.$$

We then use the spanner thus constructed to compute $(1 + \epsilon)$-approximate $S \times V$ shortest paths from up to $n^{1/\kappa}$ designated sources. (See Section 6 of [15] for more details.)

Theorem 2.2. *For any n-vertex graph $G(V, E, \omega)$ with aspect ratio Λ, $2 \leq \kappa \leq (\log n)/4$, $1/\kappa \leq \rho \leq 1/2$ and $0 < \epsilon' < 1$, our dynamic streaming algorithm computes whp a $(1 + \epsilon', \beta')$ hopset H with expected size $O(n^{1+1/\kappa} \cdot \log \Lambda)$ and the hopbound β' given by*

$$\beta' = O\left(\frac{\log \Lambda}{\epsilon'}(\log \kappa\rho + 1/\rho)\right)^{\log \kappa\rho + 1/\rho}.$$

It does so by making $O(\beta' \cdot \log \Lambda \cdot (\log \kappa\rho + 1/\rho))$ passes through the stream and using $O(\frac{\beta'}{\epsilon'} \cdot n^{1+\rho} \cdot \log \Lambda \cdot \log^2 n \cdot (\log^2 n + \log \Lambda))$ bits of space.

We then use the hopset thus constructed to compute $(1 + \epsilon)$-approximate $S \times V$ shortest paths from up to $n^{1/\kappa}$ designated sources. (See Section 8 of [15] for more details.) These algorithms extend the results of [10, 11] from the static streaming setting to dynamic streaming one. The algorithms of [10, 11], like their predecessor, the algorithm of [13], are based on the *superclustering-and-interconnection* (henceforth, SAI) approach. Our algorithms in the current paper also fall into this framework. Algorithms that follow the SAI approach proceed in phases, and in each phase they maintain a partial partition of the vertex set V of the graph. Some of the clusters of G are selected to create superclusters around them. This is the superclustering step. Clusters that are not superclustered into these superclusters are then *interconnected* with their nearby clusters. The main challenge in implementing this scheme in the dynamic streaming setting is in the interconnection step. Indeed, the superclustering step requires a single and rather shallow BFS exploration, and implementing depth-d BFS in unweighted graphs in d passes over the dynamic stream can be done in near-linear space (See, e.g., [2, 5]). For the weighted graphs, we devise a routine for performing an approximate Bellman-Ford exploration (henceforth, BFE) up to a given hop-depth d, using d passes and $\tilde{O}(n)$ space. On the other hand, the interconnection step requires implementing simultaneous BFS (BFE for the weighted case) distance explorations originated at multiple sources. A crucial property that enabled [10, 11] to implement it in the static streaming setting is that one can argue that with high probability, not too many distance explorations traverse any particular vertex. Let us denote by N, an upper bound on the number of explorations (traversing any particular vertex). In the dynamic streaming setting, however, at any point of the stream, there may well be much more than N explorations that traverse a specific vertex $v \in V$, based on the stream of updates observed so far. Storing data about all these explorations would make the space requirement of the algorithm prohibitively large.

To resolve this issue (and a number of related similar issues), we incorporate a *sparse recovery* routine into our algorithms. Sparse recovery is a fundamental and well-studied primitive in the dynamic streaming setting [3, 7, 21, 23]. It is defined for an input which is a stream of (positive and negative) updates to an n-dimensional vector $\overrightarrow{a} = (a_1, a_2, \ldots, a_n)$. In the *strict* turnstile setting, which is sufficient for our application, ultimately each coordinate a_i (i.e., at the end of the stream) is non-negative, even though negative updates are allowed and intermediate values of coordinates may be negative. In the *general* turnstile model coordinates of the vector $\overrightarrow{a}$ may be negative at the end of the stream as well. For a parameter s, an *s-sparse recovery* routine returns the vector $\overrightarrow{a}$, if $|supp(\overrightarrow{a})| \leq s$, and returns failure otherwise, where $supp(\overrightarrow{a})$ denotes the size of the support of $\overrightarrow{a}$. Most of sparse recovery routines are based on 1-sparse recovery, i.e., the case $s = 1$. The first 1-sparse recovery algorithm was devised by [21], and it applies to the strict turnstile setting. The space requirement of the algorithm of [21] is $O(\log n)$. The result was later extended to the general turnstile setting by [7] (See also, [28]). We devise an alternative streaming algorithm for this basic task in the strict turnstile setting. Our 1-sparse recovery technique is inspired by ideas from [22, 25] and [6]. The space complexity of our algorithm is $O(\log n)$, like that of [21]. The processing time-per-item of Ganguly's algorithm [21] is however $O(1)$, instead of $polylog(n)$ of our algorithm.[1] Nevertheless, we believe that our new algorithm for this task is of independent interest.

REFERENCES

[1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Analyzing Graph Structure via Linear Measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms* (Kyoto, Japan) *(SODA '12)*. Society for Industrial and Applied Mathematics, USA, 459–467.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Graph Sketches: Sparsification, Spanners, and Subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Scottsdale, Arizona, USA) *(PODS'12)*. Association for Computing Machinery, New York, NY, USA, 5–14. https://doi.org/10.1145/2213556.2213560

[3] Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. 2010. Lower Bounds for Sparse Recovery. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, Moses Charikar (Ed.). SIAM, 1190–1197. https://doi.org/10.1137/1.9781611973075.95

[4] Surender Baswana. 2008. Streaming algorithm for graph spanners - single pass and constant processing time per edge. *Inform. Process. Lett.* 106, 3 (2008), 110 – 114. https://doi.org/10.1016/j.ipl.2007.11.001

[5] Yi-Jun Chang, Martin Farach-Colton, Tsan-sheng Hsu, and Meng-Tsung Tsai. 2020. Streaming Complexity of Spanning Tree Computation. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France (LIPIcs, Vol. 154)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 34:1–34:19. https://doi.org/10.4230/LIPIcs.STACS.2020.34

[6] Don Coppersmith and Michael Elkin. 2006. Sparse Sourcewise and Pairwise Distance Preservers. *SIAM Journal on Discrete Mathematics* 20, 2 (2006), 463–501. https://doi.org/10.1137/050630696

[7] Graham Cormode and D. Firmani. 2013. A unifying framework for ℓ_0–sampling algorithms. *Distributed and Parallel Databases* 32 (2013), 315–335.

[8] Michael Elkin. 2011. Streaming and Fully Dynamic Centralized Algorithms for Constructing and Maintaining Sparse Spanners. *ACM Trans. Algorithms* 7, 2, Article 20 (March 2011), 17 pages. https://doi.org/10.1145/1921659.1921666

[9] Michael Elkin. 2017. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. ACM, 757–770. https://doi.org/10.1145/3055399.3055452

[10] Michael Elkin and Ofer Neiman. 2018. Efficient Algorithms for Constructing Very Sparse Spanners and Emulators. *ACM Trans. Algorithms* 15, 1, Article 4 (Nov. 2018), 29 pages. https://doi.org/10.1145/3274651

[11] Michael Elkin and Ofer Neiman. 2019. Hopsets with Constant Hopbound, and Applications to Approximate Shortest Paths. *SIAM J. Comput.* 48, 4 (2019), 1436–1480. https://doi.org/10.1137/18M1166791

[12] Michael Elkin and Ofer Neiman. 2020. Centralized and Parallel Multi-Source Shortest Paths via Hopsets and Fast Matrix Multiplication. *CoRR* abs/2004.07572 (2020). arXiv:2004.07572 https://arxiv.org/abs/2004.07572

[13] Michael Elkin and David Peleg. 2001. $(1+\epsilon, \beta)$-Spanner Constructions for General Graphs. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing* (Hersonissos, Greece) *(STOC '01)*. Association for Computing Machinery, New York, NY, USA, 173–182. https://doi.org/10.1145/380752.380797

[14] Michael Elkin and Shay Solomon. 2013. Fast Constructions of Light-Weight Spanners for General Graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*. SIAM, 513–525. https://doi.org/10.1137/1.9781611973105.37

[15] Michael Elkin and Chhaya Trehan. 2021. \$(1+\epsilon)\$-Approximate Shortest Paths in Dynamic Streams. *CoRR* abs/2107.13309 (2021). arXiv:2107.13309 https://arxiv.org/abs/2107.13309

[16] Michael Elkin and Jian Zhang. 2006. Efficient algorithms for constructing $(1+\epsilon, \beta)$-spanners in the distributed and streaming models. *Distributed Computing* 18, 5 (2006), 375–385. https://doi.org/10.1007/s00446-005-0147-2

[17] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. 2004. On Graph Problems in a Semi-streaming Model. In *Automata, Languages and Programming*, Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 531–543.

[18] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. 2009. Graph Distances in the Data-Stream Model. *SIAM J. Comput.* 38, 5 (2009), 1709–1727. https://doi.org/10.1137/070683155

[19] Manuel Fernandez, David P. Woodruff, and Taisuke Yasuda. 2020. Graph Spanners in the Message-Passing Model. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA (LIPIcs, Vol. 151)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 77:1–77:18. https://doi.org/10.4230/LIPIcs.ITCS.2020.77

[20] Arnold Filtser, Michael Kapralov, and Navid Nouri. 2021. *Graph Spanners by Sketching in Dynamic Streams and the Simultaneous Communication Model*. 1894–1913.

[21] S. Ganguly. 2007. Counting distinct items over update streams. *Theor. Comput. Sci.* 378 (2007), 211–222.

[22] David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. 2015. Dynamic graph connectivity with improved worst case update time and sublinear space. *CoRR* abs/1509.06464 (2015). arXiv:1509.06464 http://arxiv.org/abs/1509.06464

[23] Piotr Indyk, Eric Price, and David P. Woodruff. 2011. On the Power of Adaptivity in Sparse Recovery. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, Rafail Ostrovsky (Ed.). IEEE Computer Society, 285–294. https://doi.org/10.1109/FOCS.2011.83

[24] Michael Kapralov and David Woodruff. 2014. Spanners and Sparsifiers in Dynamic Streams. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing* (Paris, France) *(PODC '14)*. Association for Computing Machinery, New York, NY, USA, 272–281. https://doi.org/10.1145/2611462.2611497

[25] Valerie King, Shay Kutten, and Mikkel Thorup. 2015. Construction and impromptu repair of an MST in a distributed network with $o(m)$ communication. *CoRR* abs/1502.03320 (2015). arXiv:1502.03320 http://arxiv.org/abs/1502.03320

[26] Felix Lazebnik and Vasiliy A. Ustimenko. 1992. Some Algebraic Constructions of Dense Graphs of Large Girth and of Large Size. In *Expanding Graphs, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, May 11-14, 1992*, Vol. 10. 75–93. https://doi.org/10.1090/dimacs/010/07

[27] Andrew McGregor. 2014. Graph Stream Algorithms: A Survey. *SIGMOD Rec.* 43, 1 (May 2014), 9–20. https://doi.org/10.1145/2627692.2627694

[28] Morteza Monemizadeh and David P. Woodruff. 2010. 1-Pass Relative-Error L_p-Sampling with Applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, Moses Charikar (Ed.). SIAM, 1143–1160. https://doi.org/10.1137/1.9781611973075.92

[1] If the algorithm knows in advance the dimension n of the vector $\overrightarrow{a}$ and is allowed to compute during preprocessing, a table of size n, then our algorithm can also have $O(1)$ processing time per update. This scenario occurs in dynamic streaming graph algorithms.