Graph Distances in the Data Stream Model*

Joan Feigenbaum† Sampath Kannan‡ Andrew McGregor§ Siddharth Suri¶
Jian Zhang‖

Abstract

We explore problems related to computing graph distances in the data stream model. The goal is to design algorithms that can process the edges of a graph in an arbitrary order given only a limited amount of working memory. We are motivated by both the practical challenge of processing massive graphs such as the web-graph and the desire for a better theoretical understanding of the data stream model. In particular, we are interested in the trade-offs between model parameters such as per-data-item processing time, space-use, and the number of passes that may be taken over the stream. These trade-offs are more apparent when considering graph problems than they were in previous streaming work that solved problems of a statistical nature. Our results include the following:

1. *Spanner Construction:* There exists a single-pass $O(tn^{1+1/t}\log^2 n)$-space, $O(t^2n^{1/t}\log n)$-time-per-edge algorithm that constructs a $(2t+1)$-spanner. For $t=\Omega(\log n/\log\log n)$, the algorithm satisfies the semi-streaming space restriction of $O(n\,\mathrm{polylog}\,n)$, and has per-edge processing time $O(\mathrm{polylog}\,n)$. This resolves an open question from [20].
2. *BFS-Tree Construction:* For constant t, any algorithm that computes the first t layers of a BFS tree from a prescribed node with probability at least $2/3$ requires either $(t-1)/2$ passes or $\Omega(n^{1+1/t})$ space. Since constructing BFS-trees is an important subroutine in many traditional graph algorithms, this demonstrates the need for new algorithmic techniques when processing graphs in the data stream model.
3. *Graph Distance Lower-Bounds:* Testing if a graph is connected requires $\Omega(n)$ space. Our proof generalizes to a wide range of graph properties which we call "balanced properties." Any t-approximation of the distance between two nodes requires $O(n^{1+1/t})$ space and therefore, approximating a distance using the above spanner construction is only about a factor two from optimal. Any p-pass algorithm that determines whether the length of the shortest cycle is longer than g, requires $\Omega\left(p^{-1}(n/g)^{1+4/(3g-4)}\right)$-space.

We conclude with two general techniques for speeding up the per-edge computation time of streaming algorithms while only increasing the space by a small factor.

*This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795. It first appeared in the Symposium on Discrete Algorithms 2005 [19].

†Dept. CS, Yale University, P.O. Box 208285, New Haven, CT 06520-8285. Email: feigenbaum-joan@cs.yale.edu. Supported in part by ONR grant N00014-01-1-0795 and NSF grant ITR-0331548.

‡Dept. CIS, University of Pennsylvania, Philadelphia, PA 19104. Email: kannan@cis.upenn.edu. Supported in part by ARO grant DAAD 19-01-1-0473 and NSF grant CCR-0105337.

§Information Theory & Applications Center, UCSD, La Jolla, CA 92093. Email: andrewm@ucsd.edu. Supported in part by NSF grant ITR-0205456. This work was done while the author was at the University of Pennsylvania.

¶Dept. CIS, University of Pennsylvania, Philadelphia, PA 19104. Email: ssuri@cis.upenn.edu. Supported in part by NIH grant T32HG000046-05.

‖Dept. CS, Stanford University, 353 Serra Mall, MC: 9045, Stanford, CA 94305. Email: jz@cs.stanford.edu. This work was done while the author was at Yale University. Supported by NSF grant ITR-0331548.

1 Introduction

In recent years, streaming has become an active area of research and an important paradigm for processing massive data sets [4, 25, 21]. Much of the existing work has focused on computing statistics of a stream of data elements, e.g., frequency moments [4, 27], ℓ_p distances [21, 26], histograms [24, 22], and quantiles [23]. More recently, there have been extensions of the streaming research to the study of graph problems [6, 20, 25]. Solving graph problems in this model has raised new challenges, because many existing approaches to the design of graph algorithms are rendered useless by the sequential-access limitation and the space limitation of the streaming model.

Processing Massive Graphs: Massive graphs arise naturally in many real world scenarios. Two examples are the *call-graph* and the *web-graph*. In the call-graph, nodes represent telephone numbers and edges correspond to calls placed during some time interval. In the web-graph, nodes represent web pages, and the edges correspond to hyper-links between pages. Also, massive graphs appear in structured data mining, where the relationships among the data items in the data set are represented as graphs. When processing these graphs it is often appropriate to use the streaming model. For example, the graph may be revealed by a web-crawler or the graph may be stored on external memory devices and being able to process the edges in an arbitrary order improves I/O efficiency. Indeed, the authors of [31] argue that one of the major drawbacks of standard graph algorithms, when applied to massive graphs such as the web, is their need to have random access to the edge set.

In general it seems that most graph algorithms need to access the data in a very adaptive fashion. Since we can not store the entire graph, emulating a traditional algorithm may necessitate an excessive number of passes over the data. There has been some success in estimating quantities that are of a statistical nature, e.g. counting triangles [10, 6, 29] or estimating frequency and entropy moments of the degrees in a multi-graph [14, 11]. However, it seemed for a while that more "complicated" computation was not possible in this model. For example, Buchsbaum et al. [9] demonstrated the intrinsic difficultly of computing common neighborhoods in the streaming model with small space. One possible way to ameliorate the situation is to consider algorithms that use $\tilde{O}(n)$ space, i.e., space proportional to the number of nodes rather than the number of edges. This space-restriction was identified as an apparent "sweet-spot" for graph streaming in a survey article by Muthukrishnan [35] and dubbed the *semi-streaming* space restriction in Feigenbaum et al. [20]. This spurred further research on designing algorithm for solving graph problems in the streaming model such as distance estimation [16, 20], matchings [20, 34] and connectivity [19, 39] including the work described here. We will provide further discussion on the results on distance estimation in the next section.

A related model is the *semi-external model*. This was introduced by Abello et al. [1] for computations on massive graphs. In this model the vertex set can be stored in memory, but the edge set cannot. However unlike our model, random access to the edges, although expensive, was allowed. Lastly, graph problems have been considered in a model that extends the stream model by allowing the algorithm to write to the stream during each pass [2, 15]. These *annotations* can then be utilized by the algorithm during successive passes. [2] goes further and suggests a model in which *sorting passes* are permitted in which the data stream is sorted according to a key encoded by the annotations.

Graph Distances: Designing algorithms for computing graph distances is a very well studied problem and graph distances are a natural quantity when trying to understand properties of massive graphs such as the diameter of the world-wide-web [3]. We start with a formal definition of the relevant terms.

Definition 1 (Graph Distance, Diameter, and Girth)**.** *For an undirected, unweighted graph $G = (V, E)$ we define a distance function $d_G : V \times V \to \{0, \ldots, n-1\}$ where $d_G(u, v)$ is the length of the shortest path in G between u and v. The diameter of G is the length of the longest shortest path, i.e.,*

$$\mathrm{Diam}(G) = \max_{u,v \in V} d_G(u, v) \ .$$

The girth of G is the length of the shortest cycle in G, i.e.,

$$\mathrm{Girth}(G) = 1 + \min_{(u,v) \in E} d_{G \setminus (u,v)}(u, v) \ .$$

Classic algorithms such as Dijkstra's algorithm, the Bellman-Ford algorithm and the Floyd-Warshall algorithm are taught widely [13]. Recent research has focused on computing approximate graph distances [5, 17, 38, 7]. Unfortunately these algorithms seem to be inherently unsuitable for computing distances in the streaming model; an important sub-routine of many of the existing algorithms is the construction of Breadth-First-Search (BFS) trees and one of main results is a lower-bound on the computational resources required to compute a BFS-tree. For example, Thorup and Zwick provide a construction of distance oracles for approximating distances in graphs [38]. Although *all-pairs*-shortest-path distances can be approximated using this oracle, their oracle construction requires the computation of shortest-path trees for certain vertices. Indeed, many constructions of distance oracles have this requirement, i.e., they need to compute *some* distances between certain pairs of vertices in order to build a data structure from which the all-pairs-shortest-path distances can be approximated.

A common method for approximating graph distances is via the construction of *spanners*.

Definition 2 (Spanners)**.** *A subgraph $H = (V, E')$ is a* (α, β)-spanner *of $G = (V, E)$ if, for any vertices $x, y \in V$,*

$$d_G(x, y) \leq d_H(x, y) \leq \alpha d_G(x, y) + \beta \ .$$

When $\beta = 0$, we call the spanner a *multiplicative-spanner* and refer to α as the *stretch factor* of the spanner. In [20], a simple semi-streaming spanner construction is presented. That algorithm constructs a $(\log n)$-spanner in one pass using $O(n \operatorname{polylog} n)$ space. However, this algorithm needs $O(n)$ time to process each edge in the input stream. Such a per-edge processing time is prohibitive, especially in the streaming model when edges may be arriving in quick succession. The work of [16] studies the construction of additive spanners in the streaming model. However, the algorithm of [16] requires multiple passes over the input stream, while our construction only needs one pass.

1.1 Our Results

Our results include the following:

1. *Spanner Construction:* There exists a single-pass $O(tn^{1+1/t} \log^2 n)$-space, $O(t^2 n^{1/t} \log n)$-time-per-edge algorithm that constructs a $(2t+1)$-spanner. For $t = \Omega(\log n / \log \log n)$, the

algorithm satisfies the semi-streaming space restriction of $O(n \operatorname{polylog} n)$, and has per-edge processing time $O(\operatorname{polylog} n)$. The algorithm is presented in Section 3. This result resolves an open question from [20].

2. *BFS-Tree Construction:* For constant t, any algorithm that computes the first t layers of a BFS tree from a prescribed node with probability at least 2/3 requires either $(t-1)/2$ passes or $\Omega(n^{1+1/t})$ space. This result is proved in Section 4. Since constructing BFS-trees is an important subroutine in many traditional graph algorithms, this demonstrates the need for new algorithmic techniques when processing graphs in the data stream model.

3. *Lower Bounds:* In Section 5, we present lower bounds for the following problems:

   (a) *Connectivity and Other Balanced Properties:* We show that testing any of a large class of graph properties, which we refer to as *balanced properties*, in one pass requires $\Omega(n)$ space. This class includes properties such as connectivity and bipartite-ness. This result provides a formal motivation for the *semi-streaming* space restriction where algorithms are permitted $O(n \operatorname{polylog} n)$ space.

   (b) *Graph Distances and Graph Diameter:* We show that any single-pass algorithm that returns a t-approximation of the graph distance between two given nodes with probability at least 3/4 requires $\Omega(n^{1+1/t})$ bits of space. Furthermore, this bound also applies to estimating the diameter of the graph. Therefore, approximating a distance using the above spanner construction is only about a factor two from optimal.

   (c) *Girth:* Any p-pass algorithm that ascertains whether the length of the shortest cycle is longer than g, requires $\Omega\left(p^{-1}(n/g)^{1+4/(3g-4)}\right)$ bits of space.

Finally, in Section 6, we present a method for local amortization of per-data item complexity. We also present a technique for adapting existing partially dynamic graph algorithms to the semi-streaming model.

Trade-offs: The above results indicate various trade-offs between model parameters and accuracy. These include the smooth trade-off between the space a single-pass algorithm is permitted and the accuracy achievable when estimating graph distances. For multiple-pass algorithms, a smooth trade-off between passes and space is evident when trying to compute the girth of a graph. This trade-off is, in a sense, fundamental as it indicates that the only way to get away with using half the amount of space is essentially to make half as much progress in each pass. The trade-off between space and passes when computing BFS-trees indicates that as we restrict the space, no algorithm can do much better than emulating a trivial traditional graph algorithm and will consequently require an excessive number of passes.

2 Preliminaries

In this section we give a formal definition of a graph stream and introduce some notation.

Definition 3 (Graph Stream)**.** *For a data stream $A = \langle a_1, a_2, \ldots, a_m \rangle$, with each data item $a_j \in [n] \times [n]$, we define a graph G on n vertices $V = \{v_1, \ldots, v_n\}$ with edges $E = \{(v_i, v_k) : a_j = (i,k)$ for some $j \in [m]\}$.*

We normally assume that each a_j is distinct although this assumption is often not necessary. When the data items are not distinct, the model can naturally be extended to consider multi-graphs, i.e. an edge (v_i, v_k) has multiplicity equal to $|\{j : a_j = (i,k)\}|$. Similarly, we consider undirected graphs but the definition can be generalized to define directed graphs. Sometimes we will consider weighted graphs and in this case $a_j \in [n] \times [n] \times \mathbb{N}$ where the third component of the data item indicates a weight associated with the edge. Note that some authors have also considered a special case of the model, the adjacency-list model, in which all incident edges are grouped together in the stream [6]. We will be interested in the fully general model.

Notation and Terminology: We refer to an event occurring "with high probability" if the probability of the event is at least $1 - 1/n^{\Omega(1)}$. We denote $[t] := \{1, \ldots, t\}$.

3 Spanner-Construction

In this section we present a single-pass streaming algorithm that constructs a $(2t+1)$-spanner for an unweighted, undirected graph. The algorithm uses some of the ideas from [7] and adapts them for use in the data stream model. We then extend this algorithm to construct $(1+\epsilon)(2t+1)$-spanners for weighted undirected graphs using a geometric grouping technique.

Overview of the Algorithm: Intuitively a dense graph will contain a small number of subgraphs that are dense and have small diameters. We can group the vertices in each of the subgraphs into a subset, and treat each subset as a super-node. If we remove, from the original graph, those subgraphs that have been grouped into super-nodes, the remaining graph, whose diameter may be large, will not have too many edges. We call the remaining graph the *sparse part* of the original graph. The induced graph on the super-nodes, and the sparse part of the original graph then form a spanner.

Note that the vertices in a super-node form a cluster of small diameter. We devise a randomized labeling scheme to identify these clusters. That is, the set of vertices labeled with the same label form a cluster. Our clusters are similar to those used in [5, 17, 7]. However, the constructions of the clusters in [5, 17, 7] all employ an approach similar to BFS, i.e., the clusters are constructed layer by layer. Such a layer-by-layer process is important to ensure that the clusters constructed in [5, 17, 7] have small diameters. In the streaming model, this would necessitate multiple passes over the input stream. Our labeling scheme employs a different strategy to control the clusters' diameters, therefore bypasses the BFS. This enables our algorithm to run in one pass through the stream.

The Algorithm: A label l used in our construction is a positive integer. Given two parameters n and t, the set of labels L used by our algorithm is generated in the following way. Initially, we have labels $[n]$. We denote by L^0 this set of labels and call them the *level* 0 labels. Independently, and with probability $n^{-1/t}$, each label $l \in L^0$ will be selected into the set S^0 and l will be marked as *selected*. From each label l in S^0, we generate a new label $l' = l + n$. We denote by L^1 the set of newly generated labels and call them level 1 labels. We then apply the above selection and new label generation procedure on L^1 to get the set of level 2 labels L^2. We continue this until the level $\lfloor t/2 \rfloor$ labels $L^{\lfloor t/2 \rfloor}$ are generated. If a level $i+1$ label l is generated from a level i label l', we call l the *successor* of l' and denote by $Succ(l') = l$. The set of labels we will use in our algorithm is

the union of labels of level $1, 2, \ldots, \lfloor t/2 \rfloor$, i.e., $L = \bigcup_0^t L^i$. Note that L can be generated before the algorithm sees the edges in the stream. But, in order to generate the labels in L, except in the case $t = O(\log n)$, the algorithm needs to know n, the number of vertices in the graph, before seeing the edges in the input stream. For $t = O(\log n)$, a simple modification of the above method can be used to generate L without knowing n, because the probability for a label to be selected is constant.

At first glance it might appear that our labeling scheme resembles the sequences of sets initially constructed by the algorithm Thorup and Zwick [38]. In our construction the generation of the clusters of vertices depends on both the labels and the edge set. Thorup and Zwick, however, generate their sequence of sets independent of the edges. But, this is just the first step in their algorithm. Subsequently, their algorithm computes the exact distance between a fixed vertex and each of the sets of vertices in the sequence. Computing the exact distances between a pair of vertices is difficult in the streaming model. Our algorithm takes a very different approach from [38] to avoid this difficulty.

While going through the stream, our algorithm will label each vertex with labels chosen from L. The algorithm may label a vertex v with multiple labels, however, v will be labeled by at most one label from L^i, for $i \in [\lfloor t/2 \rfloor]$. Moreover, if v is labeled by a label l, and l is selected, the algorithm will also label v with the label $Succ(l)$.

Denote by l^i a label of level i, i.e., $l^i \in L^i$. Let $L(v) = \{l^0, l^{k_1}, l^{k_2}, \ldots l^{k_j}\}$ be the collection of labels that has been assigned to the vertex v where $0 < k_1 < \ldots < k_j < t$. Let

$$Height(v) = max\{j | l^j \in L(v)\}$$

and $Top(v) = l^k \in L(v)$ s.t. $k = Height(v)$. Let $C(l)$ be the collection of vertices that are labeled with the label l.

The sets $L(v)$ and $C(l)$ will grow while the algorithm goes through the stream and labels the vertices. For each $C(l)$, our algorithm stores a BFS tree, $Tree(l)$, on the vertices of $C(l)$. We say an edge (u, v) connects $C(l)$ and $C(l')$ if u is labeled with l and v is labeled with l'. For some pairs of labels $l, l' \in L^{\lfloor t/2 \rfloor}$, our algorithm will store edges that connects $C(l)$ and $C(l')$. We denote by H the set of such edges store by our algorithm. In addition, for each vertex v, we denote by $M(v)$ the other edges incident to v that are stored by our algorithm. Intuitively, the subgraph induced by $\cup_{v \in V} M(v)$ is the sparse part of the graph G. The spanner constructed by the algorithm is the union of the BFS-trees for all the labels, $M(v)$ for all the vertices and the set H. The detailed algorithm is given in Figure 1.

Analysis: We start with two preliminary lemmas that will substantiate that the spanner construction requires only a small amount of working space.

Lemma 4. *With high probability, for all $v \in V$, $|M(v)| \leq O(tn^{1/t} \log n)$.*

Proof. Let $M^{(i)}(v) \subseteq M(v)$ be the set of edges added to $M(v)$ during the period when $Height(v) = i$. Let $L(M(v)) = \cup_{(u,v) \in M(v)} L(u)$ be the set of labels that have been assigned to the vertices in $M(v)$. An edge (u, v) is added to $M(v)$ only in step 2(b)(ii). Note that in this case, $\lfloor \frac{t}{2} \rfloor \geq Height(u) \geq Height(v)$. Hence, the set $L_v(u)$ is not empty. Also by the condition in step 2(b)(ii), none of the labels in $L_v(u)$ appears in $L(M(v))$. Thus, by adding the edge (u, v) to $M(v)$, we introduce at least one new label to $L(M(v))$. $M^{(i)}(v)$ will then introduce a set B of distinct labels to $L(M(v))$. Furthermore, the size of B is at least $|M^{(i)}(v)|$. Note that the labels in B are not

Algorithm 1 (Efficient One Pass Spanner Construction).
The input to the algorithm is a unweighted, undirected graph $G = (V, E)$, presented as a stream of edges, and two positive integer parameters n and t. (See the descriptions before Lemma 4 for the definition of H, $M(v)$, $L(v)$, $C(l)$, $Tree(l)$, $Succ(l)$, $Top(v)$ and $Height(v)$.)

1. Generate the set L of labels as described. $\forall\ v_i \in V$, label vertex v_i with label $i \in L^0$, and set $M(v) \leftarrow \emptyset$, $H \leftarrow \emptyset$.

2. Upon seeing an edge (u, v) in the stream, if $L(v) \cap L(u) = \emptyset$, consider the following cases:

   (a) If $Height(v) = Height(u) = \lfloor t/2 \rfloor$, and there is no edge in H that connects $C(Top(v))$ and $C(Top(u))$, set $H \leftarrow H \cup \{(u, v)\}$.

   (b) Otherwise, assume, without loss of generality, $\lfloor t/2 \rfloor \geq Height(u) \geq Height(v)$. Consider the collection of labels

   $$L_v(u) = \{l^{k_1}, l^{k_2}, \ldots, l^{Height(u)}\} \subseteq L(u)\ ,$$

   where $Height(v) \leq k_1 < k_2 < \ldots < Height(u)$. Let $l = l^i \in L_v(u)$ such that l^i is marked as selected and there is no $l^j \in L_v(u)$ with $j < i$ that is marked as selected.

      i. If such a label l exists, label the vertex v with the successor $l' = Succ(l)$ of l, i.e., $L(v) \leftarrow L(v) \cup \{l'\}$. Incorporate the edge in the BFS-tree $Tree(l')$. If l' is selected, label v with $l'' = Succ(l')$ and incorporate the edge in the tree $Tree(l'')$. Continue this until we see an label that is not marked as selected.

      ii. If no such label l exists and there is no edge (u', v) in $M(v)$ such that u, u' are labeled with the same label $l \in L_v(u)$, add (u, v) to $M(v)$.

3. After seeing all the edges in the stream, output the union of the BFS-trees for all the labels, $M(v)$ for all the vertices and the set H as the spanner.

Figure 1: An Efficient One Pass Algorithm for Computing Sparse Spanners

marked as selected. Otherwise, the algorithm would have taken step 2(b)(i) instead of step 2(b)(ii). Hence, the size of B follows a geometric distribution,$\Pr(|B| = k) \leq (1 - 1/n^{1/t})^k$. Thus, with high probability, $|M^{(i)}(v)| \leq O(n^{1/t} \log n)$. Because i can take at most $O(t)$ values, with high probability,

$$|M(v)| = \sum_i |M^{(i)}(v)| = O(tn^{1/t} \log n) \ .$$

□

Lemma 5. *With high probability, the algorithm stores $O(tn^{1+1/t} \log n)$ edges.*

Proof. The algorithm stores edges in the set H, in the BFS trees for each cluster $C(l)$, and in the sets $M(v)$, $\forall v \in V$. By the Chernoff bound and the union bound, with probability $1 - \frac{1}{n^{\Omega(1)}}$, the number of clusters at level $t/2$ is $O(\sqrt{n})$ and the size of the set H is $O(n)$.

For each label l, the algorithm stores a BFS tree for the set of vertices $C(l)$. Note that for $i \in [\lfloor t/2 \rfloor]$, a vertex is labeled with at most one label in L^i. Hence, $\cup_{l \in L^i} C(l) \subseteq V$. Thus, the total number of edges in the BFS trees is at most $O(tn)$. Finally, by Lemma 4, with high probability, $|M(v)| \leq O(tn^{1/t} \log n)$. By the union bound, with high probability $\sum_{v \in V} |M(v)| \leq O(tn^{1+1/t} \log n)$. □

Theorem 6. *Let G be an unweighted graph. There exists a single-pass $O(tn^{1+1/t} \log^2 n)$-space algorithm that constructs a $(2t+1)$-spanner of G with high probability and processes each edge in $O(t^2 n^{1/t} \log n)$ time.*

Proof. Consider Algorithm 1 in Figure 1. At the beginning of the algorithm, for all the labels $l \in L^0$, $C(l)$ is a singleton set and the depth of the BFS tree for $C(l)$ is zero. We now bound, for label l^i, where $i > 0$, the depth of the BFS tree T^i on the vertices in $C(l^i)$. A tree grows when an edge (u, v) is incorporated into the tree in step 2(b)(i). In this case, l^i is a successor of some label l^{i-1} of level $i - 1$. Assume that the depth $d_{T^i}(v)$ of the vertex v in the tree is one more than the depth $d_{T^i}(u)$ of the vertex u. Then $u \in C(l^{i-1})$, and the depth $d_{T^{i-1}}(u)$ of u in the BFS tree T^{i-1} of $C(l^{i-1})$ is the same as $d_{T^i}(u)$. Hence, $d_{T^i}(v) = d_{T^{i-1}}(u) + 1$ where T^i is a tree of level i and T^{i-1} is a tree of level $i - 1$. Given that $d_{T^0}(x) = 0$ for all $x \in V$, the depth of a BFS tree for $C(l)$, where l is a label of level i, is at most i.

We proceed to show that for any edge which the algorithm does not store, there is a path of length at most $2t + 1$ that connects the two endpoints of the edge. The algorithm ignores three types of edges. Firstly, if $L(u) \cap L(v) \neq \emptyset$, the edge (u, v) is ignored. In this case, let l be one of the label(s) in $L(u) \cap L(v)$, u and v are both on the BFS tree for $C(l)$. Hence, there is a path of length at most t connecting u and v. Secondly, (u, v) will be ignored if $Height(v) = Height(u) = \lfloor \frac{t}{2} \rfloor$ and there is already an edge connecting $C(Top(u))$ and $C(Top(v))$. In this case, the path connecting u and v has a length at most $2t + 1$. Finally, in step 2(b)(ii), (u, v) will be ignored if there is already another edge in $M(v)$ that connects v to some $u' \in C(l)$ where $l \in L(u)$. Note that u and u' are both on the BFS tree of $C(l)$. Hence, there is a path of length at most $t + 1$ connecting u and v.

Hence, the stretch factor of the spanner constructed by Algorithm 1 is $2t+1$. By Lemma 5, with high probability, the algorithm stores $O(tn^{1+1/t} \log n)$ edges and requires $O(tn^{1+1/t} \log^2 n)$ bits of space. Also note that the bottleneck in the processing of each edge lies on step 2(b)(ii), where for each label in $L_v(u)$, we need to examine the whole set of $M(v)$. This takes at most $O(t^2 n^{1/t} \log n)$ time. □

Once the spanner is constructed, all-pairs shortest distances of the graph can be computed from the spanner. This computation does not need to access the input stream and thus can be viewed as post-processing.

Extensions: We first note the above algorithm can be used to construct a spanner of a weighted graph $G = (V, E)$ using a geometric grouping technique [12, 20]. Namely, we round each edge weight ω' up to $\min\{\omega(1+\epsilon)^i : i \in \mathbb{Z}, \omega(1+\epsilon)^i \geq \omega'\}$ where ω is the weight of the first edge. Let $G^i = (V, E^i)$ be the graph formed from G be removing all edges not of weight $\omega(1+\epsilon)^i$. For each G^i we construct a spanner in parallel and take the union of these spanners. This leads to the following theorem.

Theorem 7. *Let G be a weighted graph and W be the ratio between the maximum and minimum weights. There exists a single-pass $O(\epsilon^{-1}tn^{1+1/t}\log W \log^2 n)$-space algorithm that constructs a $(1+\epsilon)(2t+1)$-spanner of G with high probability and processes each edge in $O(t^2 n^{1/t}\log n)$ time.*

In the case where $t = \log n/\log\log n$, Algorithm 1 computes a $(2\log n/\log\log n + 1)$-spanner in one pass using $O(n\log^4 n)$ bits of space and processing each edge in $O(\log^4 n)$ time. This answers an open question in [20].

Finally, note that constructing a $(2t+1)$-spanner gives a $(2t+1)$-approximation for the diameter and, indirectly, a $(2t+2)/3$-approximation of the girth. The diameter result is immediate. For the girth approximation, note that if the constructed spanner is a strict subgraph of G then the girth of G must have been between 3 and $2t+2$.

4 Constructing BFS-Trees

In this section we prove a lower bound on the number of passes required to construct the first l layers of breadth first search tree in the streaming model. The result is proved using a reduction from the communication-complexity problem "multi-valued pointer chasing." This is a naturally defined generalization of the pointer-chasing problem considered by Nisan and Wigderson [36]. We prove the first results on the multi-round communication complexity of the problem.

Overview of Proof: Nisan and Wigderson [36] considered the problem where Alice and Bob have functions f_A and f_B respectively, mapping $[m]$ to $[m]$. The k-round pointer chasing problem is to output the result of starting from 1 and alternatively applying f_A and f_B a total of k times, starting with f_A. Nisan and Wigderson proved that if Bob speaks first the communication complexity of any k-round communication protocol to solve this problem is $\Omega(m/k^2 - k\log m)$. Jain, Radhakrishnan, and Sen [28] gave a direct sum extension showing that if there are t pairs of functions and the goal is to perform k-round pointer chasing as above on each pair, the communication complexity lower bound is approximately t times the bound of [36]. More precisely, they showed a lower bound of $\Omega(tm/k^3 - tk\log m - 2t)$ for the problem.

We show how the lower bound of [28] also implies a lower bound on the communication complexity of pointer chasing with t-valued functions. If f_A and f_B are such functions, then the result of pointer chasing starting from 1 produces a set of size at most t^k. The key difference between this problem and the problem of [28] is that in the problem of [28] we are only concerned with chasing "like" pointers. In other words, if we get to an element j using the function f_A^i, then we can only

continue with f_B^i. Nevertheless, we show by our reduction that the two problems have fairly similar communication complexity.

Finally, we create a layered graph with l layers in which alternate layers have edges corresponding to d-valued functions f_A and f_B. In order to construct the breadth first search tree, we must solve the l-round, d-valued pointer chasing problem and the lower bound above applies. This will lead to the following theorem.

Theorem 8 (BFS Lower Bound)**.** *Let γ be a constant in the range $(0,1)$. For $l \in \{1,\ldots,1/\gamma\}$, any algorithm that computes the first l layers of a BFS from a prescribed node with probability at least $2/3$ requires either $(l-1)/2$ passes or $\Omega(n^{1+\gamma})$ space.*

Formal Argument: We now present the above argument formally. In the following definition, for a function $f:[m]\to[m]$ and set $A\subset[m]$ we denote,

$$f(A) := \{j : f(i) = j \text{ for some } i \in A\} \ .$$

Definition 9 (d-valued Pointer Chasing)**.** *Let F_d be the set of all d-valued functions from $[m]$ to $[m]$. Define $g_{d,k} : F_d \times F_d \to \mathcal{P}([m])$ by $g_{d,0}(f_A, f_B) = 1$ and,*

$$g_{d,i}(f_A,f_B) = \begin{cases} f_A(g_{d,i-1}(f_A,f_B)) & \text{if } i \text{ odd} \\ f_B(g_{d,i-1}(f_A,f_B)) & \text{if } i \text{ even} \end{cases} ,$$

where $\mathcal{P}([m])$ denotes the power-set of $[m]$. Note that $g_{d,k}$ is a set of size at most d^k. Let $\bar{g}_{d,k} : F_d \times F_d \to \mathcal{P}([m])^k$ be the function,

$$\bar{g}_{d,k}(f_A,f_B) = \langle g_{d,1}(f_A,f_B),\ldots,g_{d,k}(f_A,f_B)\rangle \ .$$

Let $g_{1,k}^t : F_1^t \times F_1^t \to \mathcal{P}([m])^t$ be the t-fold direct sum of $g_{1,k}$, i.e.

$$g_{1,k}^t(\langle f_A^1,\ldots,f_A^t\rangle,\langle f_B^1,\ldots,f_B^t\rangle) = \langle g_{1,k}(f_A^1,f_B^1),\ldots,g_{1,k}(f_A^t,f_B^t)\rangle \ .$$

Let Alice have function f_A and Bob have function f_B. Let $R_\delta^r(g_{d,k})$ be the r-round randomized communication complexity of $g_{d,k}$ where Bob speaks first, i.e. the number of bits sent in the worst case (over all inputs and random coin tosses) by the best r-round protocol Π in which, with probability at least $1-\delta$, both Alice and Bob learn $g_{d,k}$.

Theorem 10 (Nisan and Wigderson [36])**.** $R^k_{\mu_1,1/3}(g_{1,k}) = \Omega(m/k^2 - k\log m)$

The following *direct-sum* theorem for $g_{1,k}$ is proved by [28] using the interesting notion of the information complexity of a function f.

Theorem 11 (Jain, Radhakrishnan, and Sen [28])**.** $R^k_{1/4}(g^t_{1,k}) = \Omega(tmk^{-3} - tk\log m - 2t)$.

Theorem 12. $R^{k-1}_{1/8}(\bar{g}_{d,k}) = \Omega(dm/k^3 - dk\log m - 2d - 12d^k \lg m - km)$.

Proof. The proof will be using a reduction from $g^d_{1,k}$. Let $(\langle f_A^1,\ldots,f_A^d\rangle,\langle f_B^1,\ldots,f_B^d\rangle)$ be an instance of $g^d_{1,k}$ Define f_A^* and f_B^* by,

$$f_A^*(j) := \{f_A^i(j) : i \in [d]\} \text{ and } f_B^*(j) := \{f_B^i(j) : i \in [d]\} \ .$$

Assume there exists a $(k-1)$-round protocol Π for $\bar{g}_{d,k}$ that fails with probability at most $1/4$ and communicates $o(dm/k^3 - dk\log m - 2d - 12d^k \lg m - km)$ bits in the worst case. We will show how to transform Π into a protocol Π' for $g^d_{1,k}$ that fails with probability at most $1/4$ and communicates $o(dm/k^3 - dk\log m - 2d)$ bits in the worst case. This will be a contradiction by Theorem 11 and hence there was no such protocol for $\bar{g}_{d,k}$.

If Π is successful then the player who sends the last message, m_{k-1}, of Π knows

$$\bar{g}_{d,k}(f_A^*, f_B^*) = \langle g_{d,1}(f_A^*, f_B^*), \ldots, g_{d,k}(f_A^*, f_B^*)\rangle \ .$$

Assume this message is sent my Alice. In Π' we append m_{k-1} with $\bar{g}_{d,k}$ and the following set of triples,

$$\{\langle i, j, f_A^i(j)\rangle : i \in [d], j \in \bigcup_{r\in\{0\}\cup[k-1]:\text{even}} g_{d,r}(f_A^*, f_B^*)\} \ .$$

Sending $\bar{g}_{d,k}(f_A^*, f_B^*)$ adds at most an extra km bits of communication. Sending the triples adds at most an extra $6d^k \lg m$ bits of communication since $|g_{d,r}(f_A^*, f_B^*)| \leq d^r$. The final message of Π' is the set of triples,

$$\{\langle i, j, f_B^i(j)\rangle : i \in [d], j \in \bigcup_{r\in[k-1]:\text{odd}} g_{d,r}(f_A^*, f_B^*)\} \ .$$

Again, sending the triples adds at most an extra $6d^k \lg m$ bits of communication. Hence Π' communicates $o(dm/k^3 - dk\log m - 2d)$ bits in the worst case.

□

Proof of Theorem 8. We do a reduction from d-valued pointer chasing. Let $m = n/(l+1)$ and let $d = m^\gamma$. Then, since l is constant by Thm 12, $R^{l-1}_{1/8}(\bar{g}_{d,l}) = \Omega(n^{1+\gamma})$.

Consider an instance (f_A, f_B) of $\bar{g}_{d,l}$. The graph described by the stream is on the following set of $n = (l+1)m$ nodes,

$$V = \bigcup_{1\leq i\leq l+1} \{v_1^i, \ldots, v_m^i\} \ .$$

For $i \in [l]$ we define a set of edges $E(i)$ between $\{v_1^i, \ldots, v_m^i\}$ and $\{v_1^{i+1}, \ldots, v_m^{i+1}\}$ in the following way:

$$E(i) = \begin{cases} \{(v_j^i, v_k^{i+1}) : k \in f_A(j)\} & \text{if } i \text{ is odd} \\ \{(v_j^i, v_k^{i+1}) : k \in f_B(j)\} & \text{if } i \text{ is even} \end{cases} \ .$$

Suppose there exists an algorithm $\mathcal{A}$ that computes the first l layers of the breadth first search tree from v_1^1 in p passes using memory M. Let L_r be set of nodes that are exactly distance r from v_1^1. Note that for all $r \in [l]$,

$$g_{d,r} = L_r \cap \{v_1^{r+1}, \ldots, v_m^{r+1}\} \ .$$

Hence by simulating $\mathcal{A}$ on a stream starting with $\bigcup_{i\in[l]:\text{even}} E(i)$ and concluding with $\bigcup_{i\in[l]:\text{odd}} E(i)$ in the natural way we deduce there exists an $2p$ round communication protocol for $g_{d,l}$ that uses only $2pM$ communication. Hence either $2p > l-1$ or $M = \Omega(n^{1+\gamma})$. □

5 Lower-Bounds

In this section we present lower-bounds on the space required to estimate graph distances, test whether a graph is connected, and compute the girth of a graph. Our lower bounds are reductions from problems in communication complexity. In Set-Disjointness, Alice has $x \in \mathbb{F}_2^n$ and Bob has $y \in \mathbb{F}_2^n$ where $\|x\|_1 = \|y\|_1 = \lceil n/4 \rceil$. If Bob is to compute

$$\textsc{Set-Disjointness}(x, y) = \begin{cases} 1 & \text{if } x.y = 0 \\ 0 & \text{if } x.y \geq 1 \end{cases}$$

with probability at least 3/4, then it is known that $\Omega(n)$ bits must be communicated between Alice and Bob [30, 37]. In Index, Alice has $x \in \mathbb{F}_2^n$ and Bob has $j \in [n]$. If Bob is to compute $\textsc{Index}(x, j) = x_j$ with probability at least 3/4 after a single message from Alice, then it is known that this message must contain $\Omega(n)$ bits (e.g. [32]). To relate our graph stream problems to these communication problems we use reductions based upon results from random graph theory and extremal combinatorics.

5.1 Connectivity and Balanced Properties

Our first result shows that a large class of problems, including connectivity, can not be solved by a single pass streaming algorithm in small space. Specifically, we identify a general type of graph property[1] and show that testing any such graph property requires $\Omega(n)$ space.

Definition 13 (Balanced Properties)**.** *We say a graph property $\mathcal{P}$ is* balanced *if there exists a constant $c > 0$ such that for all sufficiently large n, there exists a graph $G = (V, E)$ with $|V| = n$ and $u \in V$ such that:*

$$\min\{|\{v : (V, E \cup \{(u, v)\}) \text{ has } \mathcal{P}\}|, |\{v : (V, E \cup \{(u, v)\}) \text{ has } \neg\mathcal{P}\}| \} \geq cn \ .$$

Note that many interesting properties are balanced including connectivity, bipartiteness, and whether there exists a vertex of a certain degree.

Theorem 14. *Testing for any balanced graph property $\mathcal{P}$ with probability 2/3 requires $\Omega(n)$ space.*

Proof. Let c be a constant, $G = (V, E)$ be a graph on n vertices and $u \in V$ be a vertex satisfying the relevant conditions.

The proof is by a reduction to the communication complexity of Index. Let $(x, j) \in \mathbb{F}_2^{cn} \times [cn]$ be an instance of Index. Let $G(x)$ be a relabeling of the vertices of G such that $u = v_n$ and for $i \in [cn]$, $(V, E \cup \{(v_n, v_i)\})$ has $\mathcal{P}$ iff $x_i = 1$. Such a relabeling is possible because $\mathcal{P}$ does not depend on the labeling of the vertices. Let $e(j) = (v_j, v_n)$. Hence the graph determined by the edges of $G(x)$ and $e(j)$ has $\mathcal{P}$ iff $x_j = 1$. Therefore, any single pass algorithm for testing $\mathcal{P}$ using M bits of work space gives rise to a one-message protocol for solving Index. Therefore $M = \Omega(cn)$. □

For some balanced graph properties the above theorem can be generalized. For example it is possible to show that any p-pass algorithm that determines if a graph is connected requires $\Omega(np^{-1})$ bits of space [15].

[1] A graph property is simply a boolean function whose variables are the elements of the adjacency matrix of the graph but whose value is independent of the labeling of the nodes of the graph.

5.2 Graph-Distances and Graph-Diameter

If we are solely trying to estimate the distance between two nodes u and v, it may appear that constructing a graph spanner that gives no special attention to u and v, but rather approximates *all* distances, is an unnecessarily crude approach. In this section, however, we demonstrate that the spanner construction approach yields an approximation at most a factor 2 away from optimal.

In this section we show a lower-bound on the amount of space required to approximate the length of the shortest path between two nodes. Our bound also applies to estimating the diameter of the graph. Integral to our proof is the notion of an edge being *k-critical*.

Definition 15. *In a graph $G = (V, E)$, an edge $e = (u, v) \in E$ is* k-critical *if $d_{G\setminus(u,v)}(u, v) \geq k$.*

In Lemma 16 we show the existence of a graph G with a large subset of edges E' such that each edge in E' is k-critical but the removal of all edges in E' still leaves a graph with relatively small diameter. The proof is by a probabilistic argument.

Lemma 16. *For any $\gamma > 0$, $k = \lceil 1/\gamma - \epsilon \rceil$ (for some arbitrarily small constant $\epsilon > 0$) and sufficiently large n, there exists a set E of edges partitioned into two disjoint sets E_1 and E_2 on a set of n nodes V such that,*

1. *$|E_2| = n^{1+\gamma}/64$.*
2. *Every edge in E_2 is k-critical for $G = (V, E)$.*
3. *$\mathrm{Diam}(G_1) \leq 2/\gamma$ where $G_1 = (V, E_1)$.*

Proof. Consider choosing a random graph $G' = (V, E')$ on n nodes where each edge is present with probability $p = 1/(2n^{1-\gamma})$. This is commonly denoted as $G' \sim \mathcal{G}_{n,p}$. We will then construct $G_1 = (V, E_1)$ by deleting each edge in G' with probability $1/2$. We will show that with non-zero probability the sets E_1 and $E_2 = \{e \in E' \setminus E_1 : e \text{ is } k\text{-critical for } G'\}$ satisfy the three required properties. Hence there exist sets with the required properties.

The second property is satisfied by construction. It follows from the fact that if an edge is k-critical in a graph G, then it is also k-critical in any subgraph of G. We now argue that the third property is satisfied with probability at least $9/10$. First note that the process that generates G_1 is identical to picking $G_1 \sim \mathcal{G}_{n,p/2}$. It can be shown that with high probability, the diameter of such a graph is less than $2/\gamma$ for sufficiently large n [8, Corollary 10.12].

We now show that the first property is satisfied with probability at least $9/10$. Applying the Chernoff bound and the union bound proves that with probability at least $99/100$, the degree of every vertex in G' is between $n^\gamma/4$ and n^γ.

Now consider choosing a random graph and a random edge in that graph simultaneously, i.e., $G' = (V, E') \sim \mathcal{G}_{n,p}$ and an edge $(u, v) \in_R E'$. We now try to lower-bound the probability that(u, v) is k-critical in G'. Let $\Gamma_i(v) = \{w \in V : d_{G'\setminus(u,v)}(v, w) \leq i\}$. For sufficiently large n,

$$|\Gamma_k(v)| \leq \sum_{0 \leq i \leq k} n^{i\gamma} \leq 2n^{k\gamma} \leq (n-1)/100 \ .$$

As G' varies over all possible graphs, by symmetry, each vertex is equally likely to be in $\Gamma_k(v)$. Thus the probability that u is not in this set is at least $99/100$. By Markov's inequality,

$$\Pr\left(|\{(u, v) \in E' : d_{G'\setminus(u,v)}(u, v) \geq k\}| \geq |E'|/2\right) \geq 98/100 \ .$$

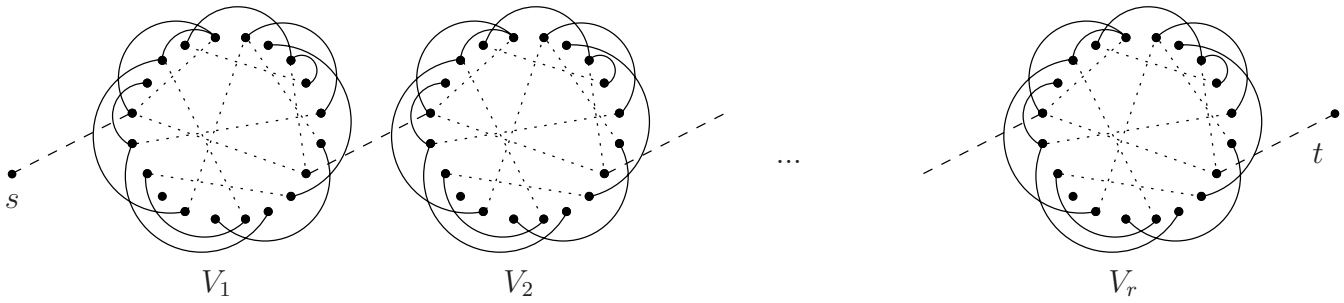

Figure 2: Diameter Lower-Bound Construction. Edges E_x are dotted, E_j are dashed, and E_m are solid.

Note that if the degree of every vertex in G' is at least $n^\gamma/4$ then $|E'| \geq n^{1+\gamma}/8$. Hence,

$$\Pr\left(|\{(u,v) \in E' : d_{G' \setminus (u,v)}(u,v) \geq k\}| \geq n^{1+\gamma}/16\right) \geq 97/100 \ .$$

Given that each edge in E' is independently deleted with probability $1/2$ to form E_1, by a further application of the Chernoff bound we deduce that,

$$\Pr\left(|\{(u,v) \in E' \setminus E_1 : d_{G' \setminus (u,v)}(u,v) \geq k\}| \geq n^{1+\gamma}/64\right) \geq 96/100 \ .$$

From this set of k-critical edges we can choose a subset whose size is exactly $n^{1+\gamma}/64$ as required by statement 1. Therefore, all three properties hold with probability at least $1 - 2/10 = 4/5$. □

Theorem 17. *For any constant $\gamma > 0$, any single pass algorithm that, with probability at least $3/4$, returns $\tilde{D}$ such that for an arbitrarily small $\epsilon > 0$,*

$$\mathrm{Diam}(G) \leq \tilde{D} \leq (\lceil 1/\gamma - \epsilon \rceil - 1)\,\mathrm{Diam}(G)$$

where G is a weighted graph on n nodes, requires $\Omega(n^{1+\gamma})$ space.

Proof. Let $(x, j) \in \mathbb{F}_2^t \times [t]$ by an instance of INDEX. We will show how to transform an algorithm $\mathcal{A}$ for approximating the diameter of a graph into a protocol for INDEX.

Let $G = (V, E = E_1 \cup E_2)$ be a graph on $n' = (64t)^{1/(1+\gamma)}$ nodes with the properties listed in Lemma 16. G is hardwired into the protocol. An enumeration of the edges in E_2, $e_1, \ldots, e_t$ is also hardwired into the protocol.

Alice forms the graph $G_x = (V, E_m \cup E_x)$ where, $E_x = \{e_i \in E_2 : x_i = 1\}$ and $E_m = E_1$. She then creates the prefix of a stream by taking r (to be determined later) copies of G_x, i.e., a graph on $n'r$ vertices $\{v_1^1, \ldots, v_{n'}^1, v_1^2, \ldots, v_{n'}^2, v_1^3, \ldots, v_{n'}^r\}$ and with edge set, $\{(v_j^i, v_k^i) : i \in [r], (v_j, v_k) \in E_x\}$. All these edges have unit weight.

Let j be the index in the instance of INDEXand let $e_j = (a, b)$. Bob determines the remaining edges E_j as follows: $r - 1$ edges of zero weight, $\{(v_b^i, v_a^{i+1}) : i \in [r-1]\}$, and two edges of weight $2k + 2$, (s, v_a^1) and (v_b^r, t). See Fig. 2 for a diagram of the construction.

Note that regardless of the values of x and j, the diameter of the graph described by the stream equals $d_G(s, t)$. Note that $x_j = 1$ implies that $d_G(s, t) = r + 4k + 3$. However, if $x_j = 0$ then

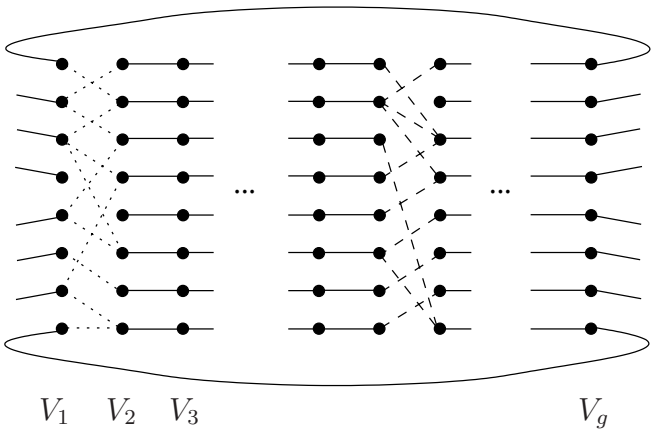

Figure 3: Girth Lower-Bound Construction. Edges E_x are dotted, E_y are dashed, and E_m are solid.

$d_G(s,t) \geq k(r-1)+4k+4$. Hence for $r = 1+(4k+4)(k-2)$, the ratio between $k(r-1)+4k+4$ and $r+4k+3$ is at least $k-1$. Hence any single-pass algorithm that approximates the diameter to within a factor $k-1$ gives rise to a one-way protocol for solving INDEX. Therefore any such algorithm requires $\Omega(n^{1+\gamma})$ bits of space since the total number of nodes in the construction is $n = O((64t)^{1/(1+\gamma)}k^2)$. □

5.3 Girth Estimation

In this section we prove a lower bound on the space required by a (multi-pass) algorithm that tests whether a graph has girth at most g. We shall make use of the following result from [33],

Lemma 18 (Lazebnik, Ustimenko, and Woldar [33])**.** *Let $k \geq 1$ be an odd integer, $t = \lceil \frac{k+2}{4} \rceil$ and q be a prime power. There exists a bipartite, q-regular graph with at most $2q^{k-t+1}$ nodes and girth at least $k+5$.*

The following lower-bound is established with a construction based on Lemma 18 that yields a reduction from SET-DISJOINTNESS to girth estimation.

Theorem 19. *For $g \geq 5$, any p-pass algorithm that tests if the girth of an unweighted graph is at most g, requires $\Omega\left(p^{-1}(n/g)^{1+4/(3g-4)}\right)$ space. If g is odd this can be strengthened to $\Omega\left(p^{-1}(n/g)^{1+4/(3g-7)}\right)$ space.*

Proof. Let q be a prime power and let $k = g-4$ if g is odd and $k = g-3$ if g is even. Let $t = \lceil \frac{k+2}{4} \rceil$. Therefore,

$$k - t + 1 \leq k - \frac{k+2}{4} + 3/4 + 1 \leq \begin{cases} (3g-7)/4 & \text{if } g \text{ is odd} \\ (3g-4)/4 & \text{if } g \text{ is even} \end{cases}.$$

Then Lemma 18 implies that there exists a q-regular graph $G' = (L \cup R, E')$ with at most $2n' \leq 2q^{k-t+1}$ nodes and girth at least $g+1$. We denote $L = \{l_1, \ldots, l_{n'}\}$ and $R = \{r_1, \ldots, r_{n'}\}$ and, for each $i \in [n']$, let $D_i = \Gamma(l_i)$.

Let $(x,y) \in \mathbb{F}_2^r \times \mathbb{F}_2^r$ by an instance of SET-DISJOINTNESS where $r = n'q$. It will be convenient to write $x = x^1 \ldots x^{n'}$ and $y = y^1 \ldots y^{n'}$ where $x^i, y^j \in \mathbb{F}_2^q$. We will show how to transform a p-pass

algorithm $\mathcal{A}$ for testing if the girth of a graph is at most g into a protocol for Set-Disjointness. If $\mathcal{A}$ uses M bits of working memory then the protocol will use use $\Omega(pM)$. Hence $M = \Omega(p^{-1}n'q)$.

Alice and Bob construct a graph G based upon G', x, and y as follows. For $i \in [g]$, let $V_i = \{v_1^i, \ldots, v_{n'}^i\}$. For each $i \in [n']$, let $D_i(x) \subset D_i$ be the subset of D_i whose characteristic vector is x^i. $D_i(y)$ is defined similarly. There are three sets of edges on these nodes;

$$\begin{aligned} E_m &= \bigcup_{j \in [g] \setminus \{1, \lceil g/2 \rceil\}} \{(v_i^j, v_i^{j+1}) : i \in [n']\}, \\ E_x &= \{(v_i^1, v_j^2) : j \in D_i(x), i \in [n']\}, \text{ and} \\ E_y &= \{(v_j^{\lceil g/2 \rceil}, v_i^{\lceil g/2 \rceil + 1}) : j \in D_i(y), i \in [n']\} \ . \end{aligned}$$

See Fig. 3 for a diagram of the construction.

Note that the $\text{Girth}(G) = g$ if there exists i such that $D_i(x) \cap D_i(y) \neq \emptyset$, i.e., x and y are not disjoint. However if x and y are disjoint then the shortest cycle is at least length $4 + 2\lceil \frac{g-2}{2} \rceil \geq g+1$. Hence determining if the girth is at most g determines if x and y are disjoint. □

6 Towards Fast per-data item Processing

In Section 3, we showed a spanner construction that processes each edge much faster than previous spanner construction algorithms. In this section we explore two general methods for decreasing the per-edge computation time of a streaming algorithm. As a consequence, we will show how some results from [18] give rise to efficient graph streaming algorithms.

Our first observation is that we can locally amortize per-edge processing by using some of our storage space as a *buffer* for incoming edges. While the algorithm processes a time intensive edge, subsequent edges can be buffered subject to the availability of space. This potentially yields a decrease in the minimal allowable time between the arrival of incoming edges.

Theorem 20. *Consider a streaming algorithm that runs in space $S(n)$ and uses computation time $\tau(m, n)$ to process the entire stream. This streaming algorithm can be simulated by a 1 pass streaming algorithm that uses $O(S(n) \log n)$ storage space, and has worst case time per-edge $\tau(m, n)/S(n)$.*

Next we turn to capitalizing on work done to speed up *dynamic graph algorithms.* Dynamic graph algorithms allow edges to be inserted and deleted in any order, and the current graph to be queried for a property $\mathcal{P}$ at any point. *Partially dynamic algorithms*, on the other hand, are those that only allow edge insertions and querying. In [18] the authors describe a technique called sparsification, which they use to speed up existing dynamic graph algorithms that decide if a graph has property $\mathcal{P}$ or not. Sparsification is based on maintaining strong certificates throughout the updates to the graph.

Definition 21. *For any graph property $\mathcal{P}$, and graph G, a strong certificate for G is a graph G' on the same vertex set such that, for any H, $G \cup H$ has the property $\mathcal{P}$ if and only if $G' \cup H$ has the property.*

It is easy to see that strong certificates obey a transitivity property. If G' is a strong certificate of property $\mathcal{P}$ for graph G, and if G'' is a strong certificate for G', then G'' is a strong certificate for G. Strong certificates also obey a compositional property. If G' and H' are strong certificates of $\mathcal{P}$ for G and H, then $G' \cup H'$ is a strong certificate for $G \cup H$.

Problem	Time/Edge
Bipartiteness	$\alpha(n)$
Connected comps.	$\alpha(n)$
2-vertex connected comps.	$\alpha(n)$
3-vertex connected comps.	$\alpha(n)$
4-vertex connected comps.	$\log n$
MST	$\log n$
2-edge connected comps.	$\alpha(n)$
3-edge connected comps.	$\alpha(n)$
4-edge connected comps.	$n\alpha(n)$
Constant edge connected comps.	$n \log n$

Table 1: One-pass, $O(n \operatorname{polylog} n)$ space streaming algorithms given by Theorem 22.

In order to achieve their speedup, the authors of [18], ensure that the certificates they maintain are not only strong but also sparse. A property is said to have *sparse certificates* if there is some constant c such that for every graph G on an n-vertex set, we can find a strong certificate for G with at most cn edges. Maintaining $\mathcal{P}$ over a sparse certificate allows an algorithm to run over a dense graph, while only using the computational time of a sparse graph.

Now, in the streaming model, we are only concerned with edges being inserted and querying the property at the end of the stream. Moreover, observe that a sparse certificate fits in space $O(n \operatorname{polylog} n)$. The following theorem states that any algorithm that could be sped up via the three major techniques described in [18], yields a one-pass, $O(n \operatorname{polylog} n)$ space, streaming algorithm with the improved running time per input edge.

Theorem 22. *Let $\mathcal{P}$ be a property for which we can find a sparse certificate in time $f(n, m)$. Then there exists a one-pass, semi-streaming algorithm that maintains a sparse certificate for $\mathcal{P}$ using $f(n, O(n))/n$ time per edge.*

Proof. Let the edges in the stream be denoted $e_1, e_2, \ldots, e_m$. Let G_i denote the subgraph given by $e_1, e_2, \ldots, e_i$. Inductively assume we have a sparse certificate C_{jn} for G_{jn}, where $1 \leq j \leq \lfloor m/n \rfloor$, constructed in time $f(n, O(n))/n$ per edge. Also, inductively assume that we have buffered the next n edges, $e_{jn+1}, e_{jn+2}, \ldots, e_{(j+1)n}$. Let $T = C_{jn} \cup \{e_{jn+1}, e_{jn+2}, \ldots, e_{(j+1)n}\}$. By the composition of strong certificates, T is a strong certificate for $G_{(j+1)n}$. Let $C_{(j+1)n}$ be the sparse certificate of T. By the transitivity of strong certificates, $C_{(j+1)n}$ is a sparse certificate of $G_{(j+1)n}$. Since C_{jn} is sparse, $|T| = (c+1)n$. Thus, computing $C_{(j+1)n}$ takes time $f(n, O(n))$. By Theorem 20 this results in $f(n, O(n))/n$ time per edge, charged over $e_{jn+1}, e_{jn+2}, \ldots, e_{(j+1)n}$. This computation can be done while the next n edges are being buffered. Let $k = \lfloor m/n \rfloor$, then the final sparse certificate will be $C_k \cup \{e_{k+1}, e_{k+2}, \ldots, e_m\}$. □

We note that for $f(n, m)$ which is linear or sub-linear in m, a better speed up may be achieved by buffering more than n edges, a possibility when we have more space. In [18] the authors provide many algorithms for computing various graph properties which they speed up using sparsification. Applying Theorem 22 to these algorithms yields the list of streaming algorithms outlined in Table 1. For $l \geq 2$, the l-vertex and l-edge connectivity problems have either not been explicitly considered in the streaming model, or have algorithms with significantly slower time per edge [20].

References

[1] J. Abello, A. L. Buchsbaum, and J. Westbrook. A functional approach to external graph algorithms. *Algorithmica*, 32(3):437–458, 2002.

[2] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. *IEEE Symposium on Foundations of Computer Science*, pages 540–549, 2004.

[3] R. Albert, H. Jeong, and A.-L. Barabasi. The diameter of the world wide web. *Nature*, 401:130, 1999.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[5] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM J. Comput.*, 28(1):263–277, 1998.

[6] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.

[7] S. Baswana and S. Sen. A simple linear time algorithm for computing a $(2k-1)-$spanner of $O(n^{1+1/k})$ size in weighted graphs. In *International Colloquium on Automata, Languages and Programming*, pages 384–296, 2003.

[8] B. Bollobás. *Random Graphs*. Academic Press, London, 1985.

[9] A. L. Buchsbaum, R. Giancarlo, and J. Westbrook. On finding common neighborhoods in massive graphs. *Theor. Comput. Sci.*, 1-3(299):707–718, 2003.

[10] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 253–262, 2006.

[11] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for computing the entropy of a stream. In *ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[12] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28(1):210–236, 1998.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, New York, NY, USA, 2001.

[14] G. Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 271–282, 2005.

[15] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 714–723, 2006.

[16] M. Elkin and J. Zhang. Efficient algorithms for constructing $(1+\epsilon, \beta)$-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006.

[17] M. L. Elkin. Computing almost shortest paths. In *Proc. 20th ACM Symposium on Principles of Distributed Computing*, pages 53–62, 2001.

[18] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification – a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

[19] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the streaming model: the value of space. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 745–754, 2005.

[20] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[21] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1 difference algorithm for massive data streams. *SIAM Journal on Computing*, 32(1):131–151, 2002.

[22] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *ACM Symposium on Theory of Computing*, pages 389–398, 2002.

[23] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD International Conference on Management of Data*, pages 58–66, 2001.

[24] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. Database Syst.*, 31(1):396–438, 2006.

[25] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *External memory algorithms*, pages 107–118, 1999.

[26] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. *IEEE Symposium on Foundations of Computer Science*, pages 189–197, 2000.

[27] P. Indyk and D. P. Woodruff. Optimal approximations of the frequency moments of data streams. In *ACM Symposium on Theory of Computing*, pages 202–208, 2005.

[28] R. Jain, J. Radhakrishnan, and P. Sen. A direct sum theorem in communication complexity via message compression. In *International Colloquium on Automata, Languages and Programming*, pages 300–315, 2003.

[29] H. Jowhari and M. Ghodsi. New streaming algorithms for counting triangles in graphs. In *COCOON*, pages 710–716, 2005.

[30] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992.

[31] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Extracting large-scale knowledge bases from the web. In *VLDB*, pages 639–650, 1999.

[32] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[33] F. Lazebnik, V. Ustimenko, and A. Woldar. A new series of dense graphs of high girth. *Bulletin of the AMS*, 32(1):73–79, 1995.

[34] A. McGregor. Finding graph matchings in data streams. In *APPROX-RANDOM*, pages 170–181, 2005.

[35] S. Muthukrishnan. Data streams: Algorithms and applications. *Now Publishers*, 2006.

[36] N. Nisan and A. Wigderson. Rounds in communication complexity revisited. *SIAM J. Comput.*, 22(1):211–219, 1993.

[37] A. A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992.

[38] M. Thorup and U. Zwick. Approximate distance oracles. In *ACM Symposium on Theory of Computing*, pages 183–192, 2001.

[39] M. Zelke. k-connectivity in the semi-streaming model. *CoRR*, cs/0608066, 2006.