Yale University
Department of Computer Science

Graph Distances in the Streaming Model: The Value of Space

Joan Feigenbaum[1]
Yale University
feigenbaum-joan@cs.yale.edu

Sampath Kannan[2]
University of Pennsylvania
kannan@cis.upenn.edu

Andrew McGregor[3]
University of Pennsylvania
andrewm@cis.upenn.edu

Siddharth Suri[4]
University of Pennsylvania
ssuri@cis.upenn.edu

Jian Zhang[5]
Yale University
zhang-jian@cs.yale.edu

YALEU/DCS/TR-1288
April, 2004

This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795.

[1]Supported in part by ONR and NSF.
[2]Supported in part by ARO grant DAAD 19-01-1-0473 and NSF grant CCR-0105337.
[3]Supported in part by NIH.
[4]Supported in part by NIH grant T32 HG000046-05.
[5]Supported by ONR and NSF.

Graph Distances in the Streaming Model: The Value of Space

Joan Feigenbaum*
Yale University
feigenbaum-joan@cs.yale.edu

Sampath Kannan†
University of Pennsylvania
kannan@cis.upenn.edu

Andrew McGregor‡
University of Pennsylvania
andrewm@cis.upenn.edu

Siddharth Suri§
University of Pennsylvania
ssuri@cis.upenn.edu

Jian Zhang¶
Yale University
zhang-jian@cs.yale.edu

Abstract

We investigate the importance of space when solving problems based on graph distance in the streaming model. In this model, the input graph is presented as a stream of edges in an arbitrary order. The main computational restriction of the model is that we have limited space; therefore, we cannot remember all the streamed data but rather are forced to make space-efficient summaries of the data as we go along. For a graph of n vertices and m edges, we show that testing many graph properties, including connectivity (*ergo* any reasonable decision problem about distances) and bipartiteness, requires $\Omega(n)$ bits of space. Given this, we then investigate how the power of the model increases as we relax our space restriction. Our main result is an efficient randomized algorithm that constructs a $(2t+1)$-spanner in one pass. With high probability, it uses $O(t\cdot n^{1+1/t}\log^2 n)$ bits of space and processes each edge in the stream in $O(t^2\cdot n^{1/t}\log n)$ time. We find approximations to diameter and girth via the constructed spanner. For $t=\Omega(\frac{\log n}{\log\log n})$, the space requirement of the algorithm is $O(n\cdot\text{polylog } n)$, and the per-edge processing time is $O(\text{polylog } n)$. We also show a corresponding lower bound of t for the approximation ratio achievable when the space restriction is $O(t\cdot n^{1+1/t}\log^2 n)$.

We then consider the scenario in which we are allowed multiple passes of the input stream. Here we investigate whether allowing these extra passes will compensate for a given space restriction. We show that finding vertices d away from a particular vertex will always take d passes, for all $d\in\{1,\ldots,t/2\}$, when the space restriction is $o(n^{1+1/t})$. For girth, we show the existence of a direct trade-off between space and number of passes in the form of a lower bound on the product of these two parameters. Finally, we conclude with two general techniques for speeding up the per-edge computation time of streaming algorithms that only increase the space by at most a log factor.

1 Introduction

In recent years, streaming has become an active area of research and an important paradigm for processing massive data sets [2, 16, 11, 17, 14, 12, 13]. Many of these papers have focused on computing statistics for a stream of data elements, *e.g.*, frequency moments [2], L^p norms [11, 17], histograms [14, 12] and wavelet decompositions [13]. More recently, the scope of streaming research has been expanded to the study of traditional graph problems [4, 10, 16]. Solving graph problems in this model has raised new challenges, because many existing approaches to the design of graph algorithms

This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795.

*Supported in part by ONR and NSF.

†Supported in part by ARO grant DAAD 19-01-1-0473 and NSF grant CCR-0105337.

‡Supported in part by NIH.

§Supported in part by NIH grant T32 HG000046-05.

¶Supported by ONR and NSF.

are rendered useless when the sequential-access and space limitations of the streaming model are enforced.

Many real world networks, *e.g.*tele-communications networks and the WWW, can be modeled as massive graphs. Dense massive graphs may also appear in applications such as structured data mining, where the relationships between the data items in the data set are represented as graphs. These massive graphs are more readily studied in a streaming, as opposed to a random-access, model.

The input to a graph algorithm in the streaming model is a stream representing the edge set of a graph. That is, for a graph $G(V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$, a *graph stream* is the sequence $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$, where $e_{i_j} \in E$ and $i_1, i_2, \ldots, i_m$ is an arbitrary permutation of $\{1, 2, \ldots, m\}$. Note that it is not required that all the edges incident on a particular vertex be grouped together in the stream.

The salient features of a streaming algorithm are: 1) sequential access to the data stream, 2) fast per-data-item processing time, 3) a small-space requirement compared to the length of the stream, and 4) access to the stream in a small number of passes. While all the existing streaming algorithms share these features, the quantities that define "small" in features (3) and (4) may vary from algorithm to algorithm. For example, most streaming algorithms use polylog n space on a stream of length n. The stream-clustering algorithm in [15], however, uses n^ϵ space. Also, the "streaming"[1] algorithm in [6] uses space $\sqrt{n}$. Most streaming algorithms access the input stream in one pass, but there are also multiple-pass algorithms [10, 20]. Thus, it is reasonable to have a broader view of the streaming model and treat the quantities in features (3) and (4) as parameters of the model. Note that it is also possible to view the quantity that defines "fast" in feature (2) as a parameter. We show that a simple technique can reduce the worst-case per-data-item complexity if the amortized complexity is small. We focus our attention on the two parameters in features (3) and (4), *i.e.*, the space $S(\cdot)$ and the number of passes $P(\cdot)$ required by the algorithm. The streaming model with these two parameters is a generalization of previous variations such the semi-streaming model [21, 10], in which the space restriction is $O(n \cdot \text{polylog } n)$, and the pass-efficient model [6]. For a better understanding of traditional graph-theory problems in the streaming model, it is necessary to investigate these problems with different values of the parameters.

1.1 Our Contributions

Our main results on one-pass streaming are upper and lower bounds for several different space restrictions on the computation of distances in undirected graphs $G = (V, E)$. For two vertices x and y in the graph G, the distance between x and y is the length of the shortest path between x and y in G. The shortest-path-distances can be approximated by graph spanners. For the graph $G = (V, E)$, a subgraph $H = (V, E')$, $E' \subseteq E$ is a *t-spanner* of G if, for any vertices $x, y \in V$, $d_G(x, y) \leq d_H(x, y) \leq t \cdot d_G(x, y)$. The number t is the *stretch factor* of the spanner. Using the spanner, we can compute approximations for girth and diameter of G.

We devise a randomized streaming algorithm that constructs a $(2t + 1)$-spanner for an unweighted undirected graph in one pass. With probability $1 - \frac{1}{n^{\Omega(1)}}$, the algorithm uses $O(t \cdot n^{1+1/t} \log^2 n)$ bits of space and processes each edge in the stream in $O(t^2 \cdot n^{1/t} \log n)$ time. Using this spanner, the all-pairs distances in the graph can be $(2t+1)$-approximated. Note that, for $t = \log n / \log\log n$, the algorithm uses $O(n \cdot \text{polylog } n)$ bits of space and processes each edge in $O(\text{polylog } n)$ time. A complementary result shows that, with $O(t \cdot n^{1+1/t} \log^2 n)$ bits of space we can not approximate the distance between u and v better than by a factor t, even if we know the vertices u and v.

In [10], a simple semi-streaming spanner construction is briefly presented. This algorithm constructs a $\log n$-spanner in one pass using $O(n \cdot \text{polylog } n)$ 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. The work of [7] studies the construction of additive spanners in the streaming model. A subgraph $H = (V, E')$ is a (ϵ, β)-*spanner* of G if, for any vertices $x, y \in V$, $d_G(x, y) \leq d_H(x, y) \leq (1 + \epsilon) \cdot d_G(x, y) + \beta$. For large distances,

[1] Instead of streaming, the authors call their algorithms "pass-efficient".

the additive spanner provides a better approximation. However, the algorithm of [7] requires multiple passes over the input stream, while our construction only needs one pass. For weighted undirected graphs, we also present a geometric grouping technique, by which our algorithm can be extended to construct $((1+\epsilon)\cdot(2t+1))$-spanners for weighted undirected graphs.

In the area of $S(n)=o(n)$, we identify a class of graph properties that we call *balanced* properties. A graph property $\mathcal{P}$ is *balanced* if, for all n sufficiently large, there exists a graph $G=(V,E)$ with $|V|=n$ and $u\in V$ such that the minimum of the size of the two sets $\{v : G=(V,E\cup\{(u,v)\})$ has $\mathcal{P}\}$ and $\{v : G=(V,E\cup\{(u,v)\})$ has $\neg\mathcal{P}\}$ is $\Omega(n)$. Using an information-theoretic approach, it can easily be shown that testing any balanced graph property requires $\Omega(n)$ bits of space. Given that many graph properties such as connectivity and bipartiteness are balanced properties, this result indicates that this area may not be very interesting for graph problems. Trivially, not being able to test for connectivity rules out testing any property based on graph distance, such as diameter or girth.

For truly massive graphs, in one pass, it may be impractical to expect that we have sufficient space to solve a graph problem. Hence, we also consider the scenario in which we are allowed multiple passes of the input stream with the idea of designing algorithms that work with tighter space constraints. We show that finding vertices d away from a particular vertex will always take d passes for all $d\in\{1,\dots t/2\}$ when the space restriction is $o(n^{1+1/t})$. Thus, the construction of a BFS tree, even for a tree with small (constant) depth, requires multiple passes over the stream. Note that the construction of BFS trees (locally or globally) has been used intensively as an important subroutine in many approximations [1, 3, 8] of the distances in graphs. Such a lower bound pushes into the domain of multiple-pass streaming any algorithm that is based on constructing BFS trees.

For girth, we show the existence of a direct tradeoff between space and passes when determining whether the girth of a graph is less than g. This takes the form of a lower bound on the product of the space S (in bits) and number of passes P:

$$SP=\Omega\left(\left(\frac{n}{g}\right)^{1+\frac{1}{g-5}}\right)$$

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

1.2 Structure of the Paper

We organize the paper in three main sections. In Section 2, we demonstrate the computational power of our model as a function of the space restriction for one-pass streaming. We first show that balanced graph properties are hard to compute with $S(n)=o(n)$. We then proceed to our main results on the estimation of distances in a graph. In Section 3, we describe the results in the multi-pass streaming domain. We show the multiple-pass lower bound for finding all the vertices distance d away from a particular vertex. We also show the tradeoff between $S(n)$ and $P(n)$ in this section for determining girth. In Section 4, we present the buffering technique that reduces the worst-case per-data-item complexity if the amortized complexity is small.

In several places, we will use communication complexity results to prove lower bounds. Specifically, we will use the results on the set-disjointness problem and the bit-vector-probing problem. We refer the reader unfamiliar with these results to [16] and [18].

2 The Importance of Space in One Pass Streaming

In this section, we investigate problems and provide results on one-pass streaming. Throughout this section, we will assume one-pass streaming when stating the results.

2.1 $S(n)=o(n)$

We start by showing that space sub-linear (in n) has no interesting computing power when it comes

[2] 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.

to solving graph problems. To do this, we identify a very general type of graph property[2] and show that testing any property of this type requires $\Omega(n)$ space.

Definition 1. *We say that 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*

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

Lemma 1. *Testing for any balanced graph property $\mathcal{P}$ requires $S(n) = \Omega(n)$.*

Proof: Let G, c, and u be as in the definition above. Consider the $(n-1)!$ streams that represent isomorphic copies of G obtained by vertex relabelings that fix the vertex u. Let $x = x_1x_2 \ldots x_{n-1}$ be an $(n-1)$-bit string where $x_i = 1$ if $v_i \in \{v : G' = (V, E \cup \{(u, v)\})$ has property $\mathcal{P}\}$ and 0 otherwise. Now over all $(n-1)!$ streams we have defined one of $\binom{n-1}{cn}$ possible binary strings x. We now stream in edge (u, v). At this point determining whether the graph $G' = (V, E \cup \{(u, v)\})$ has $\mathcal{P}$ or $\neg\mathcal{P}$ is equivalent to determining whether the bit in x corresponding to v is 1 or 0. This requires $\log \binom{n-1}{cn} = \Omega(n)$ bits of information about the string x. □

Note that many interesting properties are balanced: connectivity, bipartiteness, the existence of a vertex of a certain degree, *etc.*

2.2 $S(n, m) = \Omega(n \cdot \textbf{polylog } n)$

For this space restriction, we explore the power of the model in regard to graph distances. We present upper and lower bounds for approximating these distances.

To begin with, we provide a streaming algorithm that constructs a $(2t+1)$-spanner for an unweighted undirected graph in one pass. We then extend this algorithm to construct $((1+\epsilon)\cdot(2t+1))$-spanners for weighted undirected graphs, using a geometric grouping technique.

First, we give a high level view of the algorithm. Intuitively a dense graph will contain a small number (compared to n) of small-diameter subgraphs that are densely connected to each other. If we remove these subgraphs, 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. Thus, we can group the vertices in these subgraphs into subsets of small diameter, and each subset can be treated as a super-node. 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. Previous constructions [3, 8, 5] of such clusters all employ an approach similar to BFS, *i.e.*, the clusters are constructed layer by layer. In the streaming model, this would necessitate multiple passes over the input stream. We devise a randomized labeling scheme where the set of vertices with the same label form a cluster. By bypassing the BFS, this randomized construction enables our algorithm to run in one pass through the stream.

We now describe the labeling scheme in detail. 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 the labels $1, 2, \ldots, n$. We denote by L^0 this set of labels and call them the *level* 0 labels. Independently, and with probability $\frac{1}{n^{1/t}}$, 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 $\lfloor\frac{t}{2}\rfloor$ labels $L^{\lfloor\frac{t}{2}\rfloor}$ 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 $1, 2, \ldots, \lfloor\frac{t}{2}\rfloor$, *i.e.*, $L = \cup_0^t L^i$. 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 just $\frac{1}{2\cdot O(1)}$.

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 $i = 1, 2, \ldots, \lfloor \frac{t}{2} \rfloor$. 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 $L(v) = \{l^0, l^{k_1}, l^{k_2}, \ldots l^{k_j}\}$, $0 < k_1 < k_2 < \ldots < k_j < t$ be the collection of labels that has been assigned to the vertex v. 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'. Our algorithm will also store some edge that connects $C(l)$ and $C(l')$ where l and l' are both level $\lfloor \frac{t}{2} \rfloor$ labels. Let H be this set of edges. In addition, for each vertex v, our algorithm stores a set of edges that we denote by $M(v)$. 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 in Figure 1, and we proceed with its analysis.

Lemma 2. *At the end of the stream, for all $v \in V$, $|M(v)| \leq O(t \cdot n^{1/t} \log n)$ with probability $1 - \frac{1}{n^{\Omega(1)}}$.*

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 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, *i.e.*, $\mathbb{P}(|B| = k) \leq (1 - \frac{1}{n^{1/t}})^k$. Thus, with probability $1 - \frac{1}{n^{\Omega(1)}}$, $|M^{(i)}(v)| \leq O(n^{1/t} \log n)$. Because i can take at most $O(t)$ values, $|M(v)| = \sum_i |M^{(i)}(v)|$ is at most $O(t \cdot n^{1/t} \log n)$ with probability $1 - \frac{1}{n^{\Omega(1)}}$. □

Lemma 3. *The algorithm stores at most $O(t \cdot n^{1+1/t} \log n)$ edges, with probability $1 - \frac{1}{n^{\Omega(1)}}$.*

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 $i = 1, 2, \ldots, \lfloor \frac{t}{2} \rfloor$, a vertex is labeled with at most one label in L^i. Hence, $\cup_{l \in L^i} C(l) \subseteq V$. Thus, the overall number of edges in the BFS trees is at most $O(t \cdot n)$.

By Lemma 2, with high probability, $|M(v)| \leq O(t \cdot n^{1/t} \log n)$. By the union bound, with high probability[3] $\sum_{v \in V} |M(v)| \leq O(t \cdot n^{1+1/t} \log n)$. □

Theorem 1. *For an unweighted undirected graph of n vertices, presented as a stream of edges, and a positive number t, there is a randomized streaming algorithm that constructs a $(2t+1)$-spanner of the graph in one pass. With probability $1 - \frac{1}{n^{\Omega(1)}}$, the algorithm uses $O(t \cdot n^{1+1/t} \log^2 n)$ bits of space and processes each edge in the stream in $O(t^2 \cdot n^{1/t} \log n)$ 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). Note that 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

[3] The constants hidden in the asymptotic notation for the probability before and after applying the union bound are different.

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.

1. Generate the set L of labels. $\forall\, v_i \in V$, label vertex v_i with label i, and set $M(v) \leftarrow \phi$, $H \leftarrow \phi$.
2. Upon seeing an edge (u, v) in the stream, if $L(v) \cap L(u) = \emptyset$, consider the following cases:
   (a) If $Height(v) = Height(u) = \lfloor \frac{t}{2} \rfloor$, 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 lost of generality*, $\lfloor \frac{t}{2} \rfloor \geq Height(u) \geq Height(v)$. Consider the collection of labels $L_v(u) = \{l^{k_1}, l^{k_2}, \ldots, l^{Height(u)}\} \subseteq L(u)$, where $k_1 \geq Height(v)$ and $k_1 < k_2 < \ldots < Height(u)$. 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 $M(v)$, *i.e.*, set $M(v) \leftarrow M(v) \cup \{(u, v)\}$.
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

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 3, with high probability, the algorithm stores $O(t \cdot n^{1+1/t} \log n)$ edges and requires $O(t \cdot n^{1+1/t} \log^2 n)$ 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 $O(t^2 \cdot n^{1/t} \log n)$ time. □

Once the spanner is built, all-pairs-shortest distances of the graph can be computed from the spanner. The computation does not need to access the input stream and thus can be viewed as post-processing. Although Algorithm 1 constructs spanners for an unweighted undirected graph, we can use it, along with a geometric grouping technique to construct spanners for weighted undirected graphs.

Consider a weighted undirected graph $G = (V, E, \omega)$. Let ω_{min} be the minimum weight and let ω_{max} be the maximum weight. We divide the range $[\omega_{min}, \omega_{max}]$ into $\log_{(1+\epsilon)} \frac{\omega_{max}}{\omega_{min}}$ intervals of the form $[(1+\epsilon)^i \omega_{min}, (1+\epsilon)^{i+1}\omega_{min})$. Any weight in interval $[(1+\epsilon)^i \omega_{min}, (1+\epsilon)^{i+1}\omega_{min})$ we round down to $(1+\epsilon)^i\omega_{min}$. Let $G^i = (V, E^i)$ be the induced graph of G where E^i is the set of edges in E whose weight, after rounding, is $(1+\epsilon)^i$. For each G^i we run a copy of Algorithm 1 in parallel, and the union of the spanners constructed for G^i forms a spanner for G. Note that this can be done without prior knowledge of ω_{min} and ω_{max}.

Theorem 2. *For a weighted undirected graph of n vertices, whose maximum edge weight is ω_{max}, and whose minimum edge weight is ω_{min}, there is a randomized streaming algorithm that constructs a $((1+\epsilon)\cdot(2t+1))$-spanner of the graph in one pass. With probability $1 - \frac{1}{n^{\Omega(1)}}$, the algorithm uses $O(\log_{1+\epsilon} \frac{\omega_{max}}{\omega_{min}} \cdot t \cdot n^{1+1/t} \log^2 n)$ bits of space and processes each edge in the stream in $O(t^2 \log_{1+\epsilon} \frac{\omega_{max}}{\omega_{min}} \cdot n^{1/t} \log n)$ time.*

In the case where $t = \frac{\log n}{\log \log n}$, Algorithm 1 computes a $(2\frac{\log n}{\log\log n} + 1)$-spanner in one pass. It uses $O(n \log^4 n)$ bits of space and processes each edge in $\log^4 n$ time.

The spanner gives a $2t+1$ approximation to the distance between and two nodes and therefore, in particular, gives a $2t+1$ approximation of the diameter of the graph. It also indirectly $\frac{2t+2}{3}$ approximates the girth of the graph since, if the constructed spanner is a strict subgraph of G we know the girth of G must have been between 3 and $2t+2$.

2.2.1 A Lower Bound for One Pass Distance and Diameter Estimation

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.

A general technique to prove such results is showing the existence of *zombie* edges, edges whose presence or absence in the input graph can not be decided because of the memory limitation. The existence of zombie edges usually follows from an application of the pigeon hole principle: that for any algorithm, there exists two graphs G and G' that map to the same memory configuration and therefore there exists a zombie edge $e \in E(G) \setminus E(G')$. By making a zombie edge important in some way to the answer to the problem, we can prove lower bounds for the problem.

For distance estimation, we consider an edge $e = (u, v)$ important if it is k-*critical*, *i.e.* if the shortest path from u to v in $G \setminus (u, v)$ is of length $\geq k$. In Lemma 4, we show the existence of graphs with many k-critical edges. Furthermore, we show these graphs may have relatively small diameter. By arguing about the existence of zombie k-critical edges, we prove in Theorem 3, lower bounds for the approximation ratio possible for both short and long distances.

Lemma 4. *For $\epsilon > 0$ and sufficiently large n, there exists a graph $G = (V, E)$ with $|V| = n$, $|E| = (n^{1+\gamma})/4$ and unit edge weights such that the majority of edges are $\frac{1-\epsilon}{\gamma}$-critical and the majority of the subgraphs in the set*

$$\{G' : G' \text{ formed from } G \text{ by deleting at most half the } k\text{-critical edges}\}$$

have diameter less or equal to $\frac{1-\epsilon}{\gamma} + 1$.

We denote by $\mathcal{F}(G)$ the family of subgraphs in Lemma 4. The existence of such a family of graphs can be demonstrated by proving that almost all graphs in $\mathcal{G}_{n,n^{\gamma-1}}$ have the desired properties. We omit the details here but refer the reader to a similar proof in [10] for a slightly less general result. The following theorem generalizes and tightens a result in [10]. (The result of [10] only applies to the semi-streaming space restriction.)

Theorem 3. *In one pass and $S(n) = o(n^{1+\gamma})$, it is impossible to approximate the distance between two nodes $s, t \in V$ better than by a $\frac{1-\epsilon}{\gamma}$ ratio. Furthermore, this lower bound holds even if we are promised that $d_G(s, t)$ is equal to the diameter of G.*

Proof: Let $k = \frac{1-\epsilon}{\gamma}$, $D = k + 1$ and r, some arbitrarily large constant. Consider a graph G on n vertices with the properties in Lemma 4. Each graph in $\mathcal{F}(G)$ has $> n^{1+\gamma}/8$ edges and hence $|\mathcal{F}(G)| \geq 2^{n^{1+\gamma}/8-1} = 2^{\Omega(S(n))}$. Therefore, for any algorithm, there exists a zombie edge which is k-critical. If after streaming in $G_1 \in \mathcal{F}(G)$ and identifying zombie edge (t_1, s_1) we add edges (s, t_1) and (s_1, t) of zero weight, we can at best claim $d_G(s,t) \leq k$ when in fact $d_G(s,t)$ could be 1. This gives us the first result.

To prove a lower bound for approximating the diameter, consider a graph G on $\eta = \frac{n-2D}{rD}$ with the properties in Lemma 4. Our instance consists of rD disjoint graphs $G_1, \ldots G_{rD}$ from $\mathcal{F}(G)$. For any algorithm, there exists a zombie edge (t_i, s_i) which is k-critical in each G_i. Finally we stream in edges (s_i, t_{i+1}) for $i = 2, \ldots rD-1$ of zero weight and two disjoint length D paths, one with end points s and t_1 and the other with end points s_D and t. Because of these two paths, the diameter is realized by the shortest path between s and t. Our construction gives a graph that has diameter $2D + rD$ while the algorithm can not guarantee that the diameter is less than $2D + rDk$. Hence the best approximation ratio is $> \frac{2D+rDk}{2D+rD} \to k$ for increasingly large r. $\square$

3 Trade-offs between Passes and Storage Space

3.1 Distances

In this subsection we look at the hardness of finding all the nodes at a specified distance from a particular node. This has obvious implications for the construction of BFS-trees. Let $\Gamma_i(s)$ be all the nodes up to distance i away from s. When $i = 1$ we omit the subscript. Let $S_i(s) = \Gamma_i(s) \setminus \Gamma_{i-1}(s)$.

Consider the following construction of a family of random graphs, $\mathcal{H}_{n,\epsilon,d}$, on $n = d\eta + 1$ vertices. Let $\epsilon < \frac{1}{2d}$. Partition the vertices into $d+1$ layers, $A_0 = \{s\}, A_1, \ldots A_d$ where $|A_i| = \eta$. The graph has edges only between successive layers: s has one randomly chosen neighbour in A_1 and for $1 \leq i < d$, each node in A_i has exactly η^ϵ neighbours in A_{i+1} chosen uniformly at random. Let B_i be the subset of A_i that is exactly distance i away from s. The following lemma about the properties of this random construction follows from straightforward applications of the Chernoff bound.

Lemma 5. *For $\epsilon \leq \frac{1}{2d}$, with probability $p > 1 - \frac{1}{\Omega(n)}$, the induced subgraph of G on nodes $A_i \cup A_{i+1}$ has the following properties,*

1. *For a randomly chosen subset B_i of A_i of size $\eta^{\epsilon(i-1)}$, $|\Gamma(B_i)| = \eta^\epsilon |B_i|$.*
2. *For all $1 \leq i \leq d-1$, every subset S of size $\eta/4$ of A_i has $\Gamma(S) = A_{i+1}$.*

Lemma 6. *Let d satisfy $\left(\frac{n}{d}\right)^{1+\frac{1}{2d}} = \Omega(S(n))$ and let $\epsilon = 1/(2d)$. Partitioning A_d into B_d and $A_d \setminus B_d$ in a graph $G \in \mathcal{H}_{n,\epsilon,d}$, even when the layer number of each node is given, requires d passes in the streaming model with space restriction $S(n)$.*

Proof: Consider a randomly chosen instance constructed as above that has the properties of Lemma 5. Let E_i be the set of edges between A_{i-1} and A_i. Observe that, for $i > 1$, $|E_i| = \eta^{1+\epsilon}$. We stream the edges in the following order $E_d, E_{d-1}, \ldots E_1$, and we allow computation with unlimited time per input edge. Now, $v \in B_i$ iff there exists $u \in B_{i-1}$ such that $(u,v) \in E_i$. We call u a *witness* for v. To certify that $v \in A_i \setminus B_i$ we need to know that v has no witnesses, ie. $\Gamma(v) \cap B_{i-1} = \emptyset$. The lemma follows from the following claim.

Claim: For $1 \leq i \leq d$, classifying more than half of the elements of A_i (as elements of B_i or of $A_i \setminus B_i$) requires at least $i-1$ passes plus the part $(E_d, E_{d-1}, \ldots E_i)$ of the ith pass.

Proof of Claim. The proof is by induction on i. Clearly the claim is true for $i = 1$. Assume it is true for $j - 1$.

Consider the first $j-1$ passes of the stream. In each pass, when seeing edges from E_j there are still at least $\eta/2$ potential witnesses for vertices in B_j by the induction hypothesis. Hence, when seeing E_j we are unable to conclude any of the nodes in A_j are not in B_j by property 2 of G.

Consider the $j-1$th pass of the stream when we are now seeing edges from E_{j-1}. At this stage we may successfully partition A_{j-1}. An algorithm may have remembered some information about E_j, but we argue, because of the memory restriction, that no algorithm could have remembered a sufficient amount to use our new found partition of A_{j-1} to deduce more than half of the elements of $A_j \setminus B_j$.

Denote by T the time when we stop seeing edges from E_j in the $j-1$th pass. Let $UP_{(i,T)}$ be the vertices in A_i that are unpartitioned at time T – these are all potential witnesses for vertices in A_{i+1} at this time. (NB. when we say a vertex in A_i is unpartitioned we mean that the current memory state is insufficient to deduce whether it is in B_i or $A_i \setminus B_i$.)

Let $S_1 = UP_{(j-1,T)} \setminus \Gamma(B_j)$ and $S_2 = UP_{(j,T)}$. Let x_i' be the characteristic vector of the subset of A_j to which $v_i \in A_{j-1}$ is connected. Consider only v_i which are in S_1 and let $x_i = x_i'|_{S_2}$, the substring of x_i' on support set S_2. This gives rise to an $|S_2| \times |S_1|$ matrix such that each column, x_i, has η^ϵ 1's (since every node in A_{j-1} has n^ϵ neighbors in A_j and if $v \in A_{j-1}$ has a neighbor in $A_j \setminus S_2$ then $v \in \Gamma(B_j)$ as $A_j \setminus S_2 \subset B_j$). We call this matrix M.

Consider the situation that, while seeing E_{j-1} we fully partition A_{j-1} and learn a new vertex v_l in B_{j-1}. Note there must be at least one such vertex since we know $|B_{j-1}| = \eta^{\epsilon(j-2)}$ and if, before seeing E_{j-1}, we had already found $\eta^{\epsilon(j-2)}$ nodes in B_{j-1} we would have concluded that nodes not in this set must be in $A_{j-1} \setminus B_{j-1}$ and we would have totally partitioned A_{j-1}, which is in contradiction to the induction hypothesis. Now, v_l could have been any vertex in S_1. To deduce that more than a quarter of the vertices in S_2 are in $A_j \setminus B_j$ requires the information that some $|S_2|/4$ indices in the column x_l of M correspond to 0 entries. Since any algorithm could not have known v_l in advance, to do this for all possible v_l requires

$$\log p \binom{|S_2|}{\eta^\epsilon}^{|S_1|} - \log \binom{|S_2|/4}{\eta^\epsilon}^{|S_1|}$$

bits of information about E_j. By the induction hypothesis, this is $\Omega\left(\left(\frac{n}{d}\right)^{1+\frac{1}{2d}}\right)$. Thus, any algorithm that uses $o\left(\left(\frac{n}{d}\right)^{1+\frac{1}{2d}}\right)$ storage, cannot store enough information about E_j, and therefore it cannot guarantee, for any v_l, that it learns that some $|S_2|/4$ indices in the column x_l of M correspond to 0 entries. That is, any algorithm cannot guarantee that it partitions more than $|S_2|/4$ vertices in A_j into $A_j \setminus B_j$. Therefore the total number of vertices partitioned in A_j is

$$< |B_j| + |S_2|/4 \leq \eta/4 + \eta/4 = \eta/2$$

since at time $T < |B_j|$ vertices were partitioned. $\square$

Our main result in this subsection then follows directly from the above lemma.

Corollary 1 (Hardness of Computing $S_d(s)$). *For all $d \in \{1, \ldots t/2\}$, computing $S_d(s)$ takes d passes when the space restriction is $o(n^{1+1/t})$.*

In regards to the semi-streaming space restriction of [10], we can not find $S_d(s)$ for $d < \frac{\log n}{(\log \log n)^2}$ in less than d passes. Conversely, note that with d passes we can easily find $S_d(s)$ even in the semi-streaming model - in the ith pass we can construct $S_i(s)$ from $S_{i-1}(s)$.

3.2 Girth

In this section we prove a lower bound on the product $S(n) \cdot P(n)$ necessary to check whether a graph has girth $\leq g$. Our main tool comes from communication complexity. See [18] for a proof and further details.

Lemma 7 (Communication Complexity of Disjointness[18]). *Alice and Bob each have n bit vectors* $\mathbf{x}$ *and* $\mathbf{y}$ *and Bob wishes to determine whether they are* disjoint, *i.e. $\nexists$ index i such that $x_i = y_i = 1$. The minimum number of bits of communication sent between Alice and Bob that is sufficient for this computation, the* communication complexity, *is $\Omega(n)$.*

We also shall make use of the following result from [19],

Lemma 8. *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 on $q \leq n \leq 2q^{k-t+1}$ nodes, with number of edges $\Omega(n^{1+1/(k-t+2)})$ and with girth $\geq k+5$.*

Theorem 4 (Hardness of Finding the girth). *Consider the graph property: the girth of G is less than g. Then, to check for this property, we require*

$$P(n)S(n) = \Omega\left(\left(\frac{n}{g}\right)^{1+\frac{1}{g-5}}\right)$$

Proof: Let $G = (L \cup R, E)$ be a q regular bipartite graph with $|L| = |R| = n/g$ and $q = \Omega(n^{1/(g-5)})$. Let D_i be the set of neighbours of the ith node in L. We consider a family of graphs $\mathcal{G}$ indexed by $(S_1, \ldots S_{n/g}, T_1, \ldots T_{n/g})$ where T_i, S_i are arbitrary subsets of D_i. Each graph consists of g, size n/g sets of nodes $V_1, \ldots V_g$. There is a perfect matching between V_i and V_{i+1} for $i = 1, \ldots \lceil g/2 \rceil, \lceil g/2 \rceil + 2, \ldots g$. Let $V_{\lceil g/2 \rceil + 1} = \{u_1, \ldots u_{n/g}\}$, $V_{\lceil g/2 \rceil} = \{v_1, \ldots v_{n/g}\}$

and $V_1 = \{w_1, \ldots w_{n/g}\}$. Then we connect v_i to $\{u_j : j \in S_i\}$ and $\{w_j : j \in T_j\}$. The girth of G is g if

$$\sum_{i \in [n/g]} |T_i \cap S_i| \geq 1$$

and at least $g+1$ otherwise.

Consider the nq/g bit string, A, induced by concatenating the characteristic vectors of the S_i. Likewise B for the T_i. Then the girth of the graph G is g iff A and B are not disjoint. The bound follows from Lemma 7. □

4 Towards Fast per-data item Processing

As discussed in the introduction, in addition to using only a small number of passes and small space, we are interested in the speed at which our algorithm can process the streamed data items. In this section we explore two related methods for improving the per-edge computation time of a streaming algorithm. As a consequence, we will show how some results from [9] 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 5. *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 $\mathcal{P}$ at any point. *Partially dynamic algorithms*, on the other hand, are those that only allow edge insertions and querying. In [9] 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 2. *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 $G \cup H$.

In order to achieve their speedup, the authors of [9], 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, $S(n) = O(n \cdot \text{polylog } n)$. The following theorem states that any algorithm that could be sped up via the three major techniques described in [9], yields a one-pass, $O(n \cdot \text{polylog } n)$ space, streaming algorithm with the improved running time per input edge.

Theorem 6. *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: Denoted by $e_1, e_2, \ldots, e_m$ the edges in the stream. 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 $G_{(j+1)n}$. Because C_{jn} is sparse, $|T| = (c+1)n$. Thus, computing $C_{(j+1)n}$ takes time $f(n, O(n))$. By theorem 5 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 $k = \lfloor m/n \rfloor$, the final sparse certificate will be $C_k \cup \{e_{k+1}, e_{k+2}, \ldots, e_m\}$. $\square$

We note that for $f(n,m)$ which is linear or sublinear in m, a better speed up may be achieved by buffering more than n edges, a possibility when we have more space. In [9] the authors provide many algorithms for computing various graph properties which they speed up using sparsification. Applying theorem 6 to these algorithms yields the list of streaming algorithms outlined in the following table.

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

For $l \geq 2$, the l-vertex and l-edge connectivity problems have either not been explicitly considering in the streaming model, or have algorithms with significantly slower time per edge (see [10]).

References

[1] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani, *Fast estimation of diameter and shortest paths(without matrix multiplication)*, SIAM Journal on Computing **28** (1999), no. 4, 1167–1181.

[2] Noga Alon, Yossi Matias, and Mario Szegedy, *The space complexity of approximating the frequency moments*, Journal of Computer and System Sciences **58** (1999), no. 1, 137–147.

[3] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, *Near-linear time construction of sparse neighborhood covers*, SIAM Journal on Computing **28** (1998), no. 1, 263–277.

[4] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, *Reductions in streaming algorithms, with an application to counting triangles in graphs*, Proc. 13th ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 623–632.

[5] S. Baswana and S. Sen, *A simple linear time algorithm for computing a (2k-1)-spanner of o(n1+1/k) size in weighted graphs*, Proc. 30th International Colloq. on Automata, Languages and Computing, 2003, pp. 284–296.

[6] P. Drineas and R. Kannan, *Pass efficient algorithms for approximating large matrices*, Proc. 14th ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 223–232.

[7] M. Elkin and J. Zhang, *Efficient algorithms for constructing $(1+\epsilon,\beta)$-spanners in the distributed and streaming models*, Yale University Technical Report, YALEU/DCS/TR-1275 (2004).

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

[9] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig, *Sparsification – a technique for speeding up dynamic graph algorithms*, J. ACM **44** (1997), no. 5, 669–696.

[10] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, *On graph problems in a semi-streaming model*, To appear in the 31st International Colloquium on Automata, Languages and Programming, 2004.

[11] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, *An approximate L^1 difference algorithm for massive data streams*, SIAM Journal on Computing **32** (2002), no. 1, 131–151.

[12] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss, *Fast, small-space algorithms for approximate histogram maintenance.*, Proc. 34th ACM Symposium on Theory of Computing, 2002, pp. 389–398.

[13] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss, *Surfing wavelets on streams: One-pass summaries for approximate aggregate queries*, Proc. 27th International Conference on Very Large Data Bases, 2001, pp. 79–88.

[14] Sudipto Guha, Nick Koudas, and Kyuseok Shim, *Data-streams and histograms*, Proc. 33th ACM Symposium on Theory of Computing, 2001, pp. 471–475.

[15] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan, *Clustering data streams*, Proc. 41th IEEE Symposium on Foundations of Computer Science, 2000, pp. 359–366.

[16] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on data streams*, Technical Report 1998-001, DEC Systems Research Center (1998).

[17] P. Indyk, *Stable distributions, pseudorandom generators, embeddings and data stream computation*, Proc. 41th IEEE Symposium on Foundations of Computer Science, 2000, pp. 189–197.

[18] E. Kushilevitz and N. Nisan, *Communication complexity*, Combridge University Press, 1997.

[19] F. Lazebnik, V.A. Ustimenko, and A.J. Woldar, *A new series of dense graphs of high girth*, Bulletin of the AMS **32** (1995), no. 1, 73–79.

[20] J.I. Munro and M.S. Paterson, *Selection and sorting with limited storage*, Theoretical Computer Science **12** (1980), 315–323.

[21] S. Muthukrishnan, *Data streams: Algorithms and applications*, (2003), Available at http://athos.rutgers.edu/~muthu/stream-1-1.ps.