On Graph Problems in a Semi-Streaming Model*

Joan Feigenbaum **[1], Sampath Kannan[2] ***, Andrew McGregor[2] †, Siddharth Suri[2] ‡, and Jian Zhang §[1]

[1] Yale University, New Haven, CT 06520, USA
{feigenbaum-joan, zhang-jian}@cs.yale.edu
[2] University of Pennsylvania, Philadelphia, PA 19104, USA
{kannan, andrewm, ssuri}@cis.upenn.edu

Abstract. We formalize a potentially rich new streaming model, the *semi-streaming model*, that we believe is necessary for the fruitful study of efficient algorithms for solving problems on massive graphs whose edge sets cannot be stored in memory. In this model, the input graph, $G = (V, E)$, is presented as a stream of edges (in adversarial order), and the storage space of an algorithm is bounded by $O(n \cdot \text{polylog } n)$, where $n = |V|$. We are particularly interested in algorithms that use only one pass over the input, but, for problems where this is provably insufficient, we also look at algorithms using constant or, in some cases, logarithmically many passes. In the course of this general study, we give semi-streaming constant approximation algorithms for the unweighted and weighted matching problems, along with a further algorithm improvement for the bipartite case. We also exhibit $\log n / \log \log n$ semi-streaming approximations to the diameter and the problem of computing the distance between specified vertices in a weighted graph. These are complemented by $\Omega(\log^{(1-\epsilon)} n)$ lower bounds.

1 Introduction

Streaming [14, 3, 10] is an important model for computation on massive data sets. Recently, there has been a large body of work on designing algorithms in this model [11, 3, 10, 15, 13, 12]. Yet, the problems considered fall into a small number of categories, such as computing statistics, norms, and histograms. Very few graph problems [5] have been considered in the streaming model.

The difficulty of graph problems in the streaming model arises from the memory limitation of the model combined with input-access constraints. We can view the amount of memory used by algorithms with sequential (one-way) input

* 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.

access as a spectrum. At one end of the spectrum, we have dynamic algorithms [9] that may use memory enough for the whole input. At the other end, we have streaming algorithms that use only polylog space. At one extreme, there is a lot of work on dynamic graph problems; on the other, general graph problems are considered hard in the (poly)log-space streaming model. Recently, it has been suggested by Muthukrishnan [19] that the middle ground, where the algorithms can use $O(n \cdot \text{polylog } n)$ bits of space is an interesting and open area. This is the area that we explore.

Besides taking a middle position in the memory-size spectrum, the semi-streaming model allows multiple passes over the input stream. In certain applications with massive data sets, a small number of sequential passes over the data would be much more efficient than many random accesses to the data. Only a few works [8, 4] have considered the multiple-pass model and a lot remains to be done.

Massive graphs arise naturally in many real world scenarios. Two examples are the *call graph*, where nodes correspond to telephone numbers and edges to calls between numbers that call each other during some time interval, and the *web graph*, where nodes are web pages, and the edges are links between pages. The streaming model is necessary for the study of the efficient processing of such massive graphs. In [1], the authors introduce the semi-external model for computations on massive graphs, *i.e.*, one in which the vertex set can be stored in memory, but the edge set cannot. However, this work addresses the problems in an external memory model in which random access to the edges, although expensive, is allowed. This is a major difference between their model and ours. Indeed, the authors of [18] 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. Furthermore, there are situations in which the graph is revealed in a streaming fashion, such as a web crawler exploring the web graph.

We consider a set of classical graph problems in this semi-streaming model. We show that, although the computing power of this model is still limited, there are semi-streaming algorithms for a variety of graph problems. Our main result is a semi-streaming algorithm that computes a $(2/3-\epsilon)$-approximation in $O(\frac{\log 1/\epsilon}{\epsilon})$ passes for unweighted bipartite graph matching. We also provide a one-pass semi-streaming algorithm for 1/6-approximating the maximum weighted graph matching. We also provide $\log n/\log\log n$ approximations for diameter and shortest paths in weighted graphs which we complement with $\Omega(\log^{(1-\epsilon)} n)$ lower bounds for these problems in unweighted graphs.

2 Preliminaries

Unless stated otherwise, we denote by $G(V, E)$ a graph G with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$. Note that n is the number of vertices and m the number of edges.

Definition 1. *A **graph stream** is a sequence of edges $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$, where $e_{i_j} \in E$ and $i_1, i_2, \ldots, i_m$ is an arbitrary permutation of $[m] = \{1, 2, \ldots, m\}$.*

While an algorithm goes through the stream, the graph is revealed one edge at a time. This definition generalizes the streams of graphs in which the adjacency matrix or the adjacency list is presented as a stream. In a stream in the adjacency-matrix or adjacency-list models, the edges incident to each vertex are grouped together. We need the more general model to account for graphs such as call graphs where the edges might generated in any order.

The efficiency of a graph algorithm in the semi-streaming model is measured by the space it uses, the time it requires to process each edge, and the number of passes it makes over the graph stream.

Definition 2. *A **semi-streaming graph algorithm** computes over a graph stream using $S(n, m)$ bits of space. The algorithm may access the input stream in a sequential order(one-way) for $P(n, m)$ passes and use $T(n, m)$ time to process each edge. It is required that $S(n, m)$ be $O(n \cdot \text{polylog}\ (n))$ bits.*

To see the limitation of the (poly)log-space streaming model for graph problems, consider the following simple problem. Given a graph, determining whether there is a length-2 path between two vertices, x and y, is equivalent to deciding whether two vertex sets, the neighborhood of x and the neighborhood of y, have a nonempty intersection. Because set disjointness has linear-space communication complexity [17], the length-2 path problem is impossible in the (poly)log-space streaming model. See [4] for a more comprehensive treatment of finding common neighborhoods in the streaming model.

3 Graph Matching

3.1 Unweighted Bipartite Matching

In this subsection, we present an algorithm for approximating unweighted bipartite matching. First, a bipartition can be found by the following algorithm. Note that the labeling of the vertices keeps track of the connected components in the graph seen so far, and the signs keep a partition for each connected component.

Algorithm 1 (Bipartition). *As edges stream in, we use a disjoint set data structure to maintain the connected components of the graph so far. We associate a sign with each vertex such that no edge joins two vertices of the same sign. If this condition ever fails and cannot be corrected by flipping the sign of a vertex and the vertices in its connected component, then we output that the graph is non-bipartite.*

The disjoint set data structure with union by rank and path compression can be augmented to maintain the signs without increasing the amortized time, $\alpha(m, n)$ needed per edge.

Given a matching M, we call a vertex *free* if it doesn't appear as the end point of any edge in M. It is easy to see that a maximal matching (thus a 1/2-approximation) for a graph can be constructed by a semi-streaming algorithm in one pass over the graph stream: when going through the stream, the algorithm adds an edge to the current matching M if both ends of the edge are free *w.r.t.* M. (This constructs a maximal matching for an arbitrary graph, not just for bipartite graphs.)

Consider a matching M for a bipartite graph $G = (L \cup R, E)$. A length-3 augmenting path for an edge $e = (u, v) \in M$, $u \in L$ and $v \in R$, is a quadruple (w_l, u, v, w_r) such that $(u, w_l), (w_r, v) \in E$, and w_l and w_r are free vertices. We call w_l and w_r the *wing-tips* of the augmenting path, (u, w_l) the *left wing* and (w_r, v) the *right wing*. A set of *simultaneously augmentable length-3 augmenting paths* is a set of length-3 augmenting paths that are vertex disjoint.

We now provide an algorithm that will be used as a subroutine in our main unweighted bipartite matching algorithm. Given a bipartite graph and a matching of the graph, this algorithm finds a set of simultaneously augmentable length-3 augmenting paths.

Algorithm 2 (Find Augmenting Paths). *The input to the algorithm is a graph $G = (L \cup R, E)$, a matching M for G and a parameter $0 < \delta < 1$.*

1. *In one pass, find a maximal set of disjoint left wings. If the number of left wings found is $\leq \delta M$, terminate.*
2. *In a second pass, for the edges in M with left wings, find a maximal set of disjoint right wings.*
3. *In a third pass we identify the set of vertices that*
   (a) *Are endpoints of a matched edge that got a left wing.*
   (b) *Are the wing tips of a matched edge that got* both *wings.*
   (c) *Are endpoints of a matched edge that is no longer 3 augmentable.*

   We remember these vertices and in subsequent passes, we ignore any edge incident on one of these vertices.
4. *Repeat.*

Our main unweighted bipartite matching algorithm increases the size of a matching by repeatedly finding a set of simultaneously augmentable length-3 augmenting paths and augmenting the matching using these paths.

Algorithm 3 (Unweighted Bipartite Matching). *The input to the algorithm is a bipartite graph $G = (L \cup R, E)$ and a parameter $0 < \epsilon < 1/3$.*

1. *In one pass, find a maximal matching M and the bipartition of G.*
2. *For $k = 1, 2, \ldots, \lceil \frac{\log 6\epsilon}{log 8/9} \rceil$ Do:*
   (a) *Run the algorithm 2 with G, M and $\delta = \frac{\epsilon}{2-3\epsilon}$.*
   (b) *For each $e = (u, v) \in M$ for which an augmenting path (w_l, u, v, w_r) is found by algorithm 2, remove (u, v) from M and add (u, w_l) and (w_r, v) to M.*

We now establish a relationship between the size of a maximal set of simultaneously augmentable length-3 augmenting paths and the size of a maximum such set.

Lemma 1. *The size of a maximal set of simultaneously augmentable length-3 augmenting paths is at least 1/3 of the size of a maximum set of simultaneously augmentable length-3 augmenting paths.*

Proof. Let AP_{max} be some maximal set of simultaneously length-3 augmentable paths. Note that each path that we find destroys at most 3 paths that AP_{max} might have used, one involving each of the wing tips that we use and a third path involving the matched edge we used. Thus we have a 1/3-approximation.

Lemma 2. *Let X be a maximum-sized set of simultaneously augmentable length-3 augmenting paths for a maximal matching M. Let $\alpha = \frac{|X|}{|M|}$ and* Opt *a maximum matching. $|M|(1+\alpha) \geq 2/3\ |\text{Opt}|$.*

Proof. See for example, [16], page 156.

Lemma 3. *Algorithm 2 finds $\frac{\alpha|M|-2\delta|M|}{3}$ simultaneously augmentable length-3 augmenting paths in $3/\delta$ passes.*

Proof. Let $L(M)$ be the set of the end vertices of the edges in M that are in L and $V_L(M) = \{v \in R \mid v \text{ is free } w.r.t.\ M \text{ and } \exists u \in L(M)\ s.t.\ (u,v) \in E\}$. We call one repetition of step 1–4 in the algorithm a phase. The number of phases is at most $1/\delta$ because at least $\delta|M|$ edges in M are removed at each phase.

When the algorithm terminates, the number of left wings found is at most $\delta|M|$. Note that the set of left wings found form a maximal matching between the remaining vertices in $L(M)$ and $V_L(M)$. Hence there are fewer than $2\delta|M|$ disjoint left wings that could have been found at this phase. Consequently, there are fewer than $2\delta|M|$ simultaneously augmentable length-3 augmenting paths in the remaining graph, which we denote G'.

Let $G'' = G \setminus G'$. Note that a maximum set of simultaneously augmentable length-3 augmenting paths in G'' would have a size at least $\alpha|M| - 2\delta|M|$. Also note that the set of length-3 augmenting paths found by the algorithm form a maximal set *w.r.t.* G''. By Lemma 1, the size of such a set is at least $\frac{\alpha|M|-2\delta|M|}{3}$.

Theorem 1. *For any $0 < \epsilon < 1/3$ and a bipartite graph, algorithm 3 finds a $2/3-\epsilon$ approximation of maximum matching in $O\left(\frac{\log 1/\epsilon}{\epsilon}\right)$ passes. The algorithm processes each edge in $O(1)$ time in each pass except the first pass, in which the bipartition is found. The amortized per-edge processing time is $\alpha(m,n)$ for finding the bipartition. The storage space required by the algorithm is $O(n \log n)$.*

Proof. It is easy to see the bounds for the per-edge processing time and storage space. We now show the correctness of the algorithm. Let Opt be the size of the maximum matching. At the ith phase, let M_i be the matching found by the algorithm and X_i a maximum-sized set of simultaneously augmentable length-3 augmenting paths for the matching M_i. Let $\alpha_i = \frac{|X_i|}{|M_i|}$ and $s_i = \frac{|M_i|}{\text{Opt}}$.

Note that we only need to consider the case where $\alpha_i > \frac{3\epsilon}{2-3\epsilon}$. Otherwise, by Lemma 2, the matching M_i is already a $\frac{2}{3}\frac{1}{1+\alpha_i} \geq \frac{2}{3} - \epsilon$ approximation. Assuming $\alpha_i > \frac{3\epsilon}{2-3\epsilon}$ for all stage i, let $\delta = \frac{\epsilon}{2-3\epsilon}$. Then $\delta \leq \frac{\alpha_i}{3}$ for all α_i. By Lemma 3, the number of simultaneously augmentable length-3 augmenting paths found by Algorithm 2 is then $\frac{\alpha_i|M_i|-2\delta|M_i|}{3} \geq \frac{\alpha_i|M_i|}{9}$.

Because M_0 is a maximal matching, $s_0 \geq 1/2$. At any stage, by Lemma 2, $|M_i| + \alpha_i|M_i| \geq 2/3 \cdot \textsc{Opt}$. This gives:

$$s_i + \alpha_i s_i \geq 2/3 \tag{1}$$

By Lemma 3, $|M_{i+1}| = |M_i| \cdot (1 + \frac{\alpha_i - 2\delta}{3}) \geq |M_i| \cdot (1 + \alpha_i/9)$. This gives:

$$s_{i+1} \geq s_i + \alpha_i s_i/9 \tag{2}$$

Putting together inequalities 1 and 2, we have: $s_{i+1} \geq 8/9 \cdot s_i + 2/27$. Solving this recurrence gives $s_i \geq 2/3 - 1/6(8/9)^i$. Note that, the algorithm runs in $k = \left\lceil \frac{\log 6\epsilon}{\log 8/9} \right\rceil$ stages. Thus $\frac{|M_k|}{\text{OPT}} = s_k \geq 2/3 - \epsilon$.

At each stage of algorithm 3, we run algorithm 2 as a subroutine. There are $k = \left\lceil \frac{\log 6\epsilon}{\log 8/9} \right\rceil$ stages and each stage requires $\frac{6-9\epsilon}{\epsilon}$ passes. The total number of passes is then:

$$\left\lceil \frac{\log 6\epsilon}{\log 8/9} \right\rceil \frac{6 - 9\epsilon}{\epsilon} = O\left(\frac{\log 1/\epsilon}{\epsilon}\right)$$

3.2 Weighted Matching

In the weighted matching problem, every edge e has a weight $w(e)$. We seek the matching M for which $\sum_{e \in M} w(e)$ is maximized. This is also a well studied problem when we do not restrict ourselves to the streaming model.

At least one existing algorithm [21] can easily be adapted to work in our model. For any $\epsilon > 0$, the streaming version finds a weighted matching that is at least $1/(2+\epsilon)$ the optimal size using $O(\log_{1+\epsilon/3} n)$ passes and $O(n \log n)$ storage. The algorithm works by geometrically grouping the weights into $\lceil \log_{1+\epsilon/3}(\lceil 3/\epsilon + 1.5 \rceil) n \rceil$ groups and then, for each group, starting at those with the largest weights, finding maximal matchings. Each maximal matching can be found with one pass. Further details and the proof of correctness can be found in [21].

We propose a new algorithm that uses only one pass yet still manages to find a matching which is at least $\frac{1}{6}$ of the optimal size.

Algorithm 4 (Weighted Matching). *We maintain a matching M at all times. When we see a new edge e, we compare $w(e)$ with $w(C)$, the sum of the weights of the edges of $C = \{e' | e' \in M$ and e' and e share an end point$\}$.*

- *If $w(e) > 2w(C)$, we update $M \leftarrow M \cup \{e\} \setminus C$.*
- *If $w(e) \leq 2w(C)$, we ignore e and wait for the next edge.*

Theorem 2. *In 1 pass and $O(n \log n)$ storage,we can construct a weighted matching that is at least $\frac{1}{6}$ the of the optimal size.*

Proof. For any set of edges S, let $w(S) = \sum_{e \in S} w(e)$. We say that an edge is *born* if it is ever part of M. We say that an edge is *killed* if it was born but subsequently removed from M by a newer heavier edge. This new edge *murdered* the killed edge. We say an edge is a *survivor* if it is born and never killed. Let the set of survivors be S. The weight of the matching we find is therefore $w(S)$.

For each survivor e, let the *Trail of the Dead* leading to this edge be $T(e) = C_1 \cup C_2 \cup \ldots$ where $C_0 = \{e\}$, $C_1 = \{\text{the edges murdered by } e\}$, and $C_i = \cup_{e' \in C_{i-1}} \{\text{the edges murdered by } e'\}$

Claim: $w(T(e)) \leq w(e)$

Proof of claim: For each murdering edge e, $w(e)$ is at least twice the cost of murdered edges, and an edge has at most one murderer. Hence, for all i, $w(C_i) \geq 2w(C_{i+1})$ therefore

$$2w(T(e)) = \sum_{i \geq 1} 2w(C_i) \leq \sum_{i \geq 0} w(C_i) = w(T(e)) + w(e)$$

The claim follows.

Now consider the optimal solution that includes edges $\textsc{Opt} = \{o_1, o_2, \ldots\}$. We are going to charge the costs of edges in $\textsc{Opt}$ to the survivors and their trails of the dead, $\cup_{e \in S} T(e) \cup \{e\}$. We hold an edge e in this set *accountable to* $o \in \textsc{Opt}$ if either $e = o$ or if o wasn't born because e was in M when o arrived. Note that, in the second case, it is possible for two edges to be accountable to o. If only one edge is accountable for o then we charge $w(o)$ to e. If two edges e_1 and e_2 are accountable for o, then we charge $\frac{w(o)w(e_1)}{w(e_1)+w(e_2)}$ to e_1 and $\frac{w(o)w(e_2)}{w(e_1)+w(e_2)}$ to e_2. In either case, the amount charged by o to any edge e is at most $2w(e)$.

We now redistribute these charges as follows: (for distinct u_1, u_2, u_3) if $e = (u_1, v)$ gets charged by $o = (u_2, v)$, and e subsequently gets killed by $e' = (u_3, v)$, we transfer the charge from e to e'. Note that we maintain the property that the amount charged by o to any edge e is at most $2w(e)$ because $w(e') \geq w(e)$. What this redistribution of charges achieves is that now every edge in a trail of the dead is only charged by one edge in $\textsc{Opt}$. Survivors can, however, be charged by two edges in $\textsc{Opt}$. We charge $w(\textsc{Opt})$ to the survivors and their trails of the dead, and hence

$$w(\textsc{Opt}) \leq \sum_{e \in S} 2w(T(e)) + 4w(e)$$

By Claim 1,

$$\sum_{e \in S} 2w(T(e)) + 4w(e) \leq 6w(S)$$

and the theorem follows.

3.3 Lower Bounds

In contrast to the above results, it is worth noting that even to check whether an existing matching is maximum in 1 pass requires $\Omega(m)$ space. First, we prove that testing s, t connectivity in a directed graph requires $\Omega(m)$ space.

Lemma 4. *Testing for $s-t$ connectivity in a directed graph $G = (V, E)$ requires $\Omega(m)$ bits of space, where $|E| = m$.*

Proof. Consider the family $\mathcal{F}$ of graphs $G = (L \cup R \cup \{s, t\}, E)$, where the induced graph on $L \cup R$ is an arbitrary bipartite graph with $|L| = |R| = n$ and $m \leq n^2/2$ edges and all the edges are directed from L to R. Say the stream gives all the edges between L and R first, then one edge of the form (s, l) and then (r, t), where $l \in L$ and $r \in R$. At the point at which all the edges from L to R have appeared any correct algorithm must have a different memory configuration for each graph in $\mathcal{F}$ since there are continuations that will result in different answers for any two graphs in $\mathcal{F}$. Thus the number of bits of space required is $\Omega(log_2|\mathcal{F}|$ which is easily seen to be $\Omega(m)$.

Theorem 3. *Consider a bipartite graph $G = (L \cup R, E)$. Testing whether there exists an augmenting path from $s \in R$ to $t \in L$ requires $\Omega(m)$ bits of storage.*

Proof. Let the storage required be $S(n)$. Let $G = (\{v_1, \ldots, v_n\}, E)$ be a directed graph and assume, without loss of generality, that $s = v_1$ and $t = v_n$. We will use the "test for an augmenting path" algorithm to answer the s, t directed-connectivity problem. We construct an undirected bipartite graph G' with nodes v_s, v_t such that there exists an augmenting path from v_s to v_t in G' if and only if there exists a directed path from s to t in G.

For each node v_i in G, create two nodes v_{il} and v_{ir} in G'. In addition, add nodes v_s and v_t to G'. Let the edges of G' be $E' = \{(v_{il}, v_{ir}) : i \in [n]\} \cup \{(v_{ir}, v_{jl}) : (v_i, v_j) \in E\} \cup \{(v_s, v_{1l})\} \cup \{(v_t, v_{nr})\}$. Let the existing matching be $M = \{(v_{il}, v_{ir}) : i \in [n]\}$. There is an augmenting path in G' if and only if there is a path from s to t in G.

4 Distances, Girth and Other Problems

In this section, we consider the problems of computing shortest-path distances, diameter, and girth on graph streams. We briefly mention several other graph-stream problems at the end of this section.

First we show that, in the semi-streaming model, computing exact shortest-path distances, even certain approximation, is not possible in one pass. In a graph G, we say an edge (u, v) is *k-critical* if the shortest path from u to v in $G \setminus (u, v)$ has length $\geq k$.

Lemma 5. *For $1 > \epsilon > 0$ and sufficiently large n there exists a graph $G = (V, E)$ with $|V| = n$, $|E| = 2^{\log^{\epsilon} n} n/4$ such that the majority of edges are $\frac{\log^{1-\epsilon} n}{2}$-critical and the majority of the subgraphs in the set*

$$\{G' : G' \text{ formed from } G \text{ by deleting a subset of the } \tfrac{\log^{1-\epsilon} n}{2}\text{-critical edges}\}$$

have diameter less or equal to $4 \log^{1-\epsilon} n$.

Proof (Sketch). We show existence by considering a random graph $G \in \mathcal{G}_{n,p}$ where n is very large and $p = 2^{\log^\epsilon n}/n$. Clearly the number of edges in G is at least $(2^{\log^\epsilon n} n)/4$ with high probability.

Claim 1: w.h.p. the majority of edges are k-critical where $k = \frac{\log^{1-\epsilon} n}{2}$.

Proof of Claim 1: By the Chernoff bound, with high probability, no vertex has degree greater than $2 \cdot 2^{\log^\epsilon n}$. We henceforth assume this to be the case. Consider an edge (u, v). Let $\Gamma_i(v)$ be the vertices up to a distance i from v in $G \setminus (u, v)$ and then

$$|\Gamma_k(v)| \leq \sum_{0 \leq i \leq k} (2 \cdot 2^{\log^\epsilon n})^i \leq (2 \cdot 2^{\log^\epsilon n})^{k+1} \leq 2^{2(\log n)/3}$$

Hence the probability that (u, v) was k-critical is $1 - 2^{2(\log n)/3 - \log n} \to 1$. Thus, by the Chernoff bound, the majority of edges are k-critical with high probability.

Claim 2: Consider $G \in \mathcal{G}_{n,p/2}$ w.h.p. the diameter of G is $< D = 4 \log^{1-\epsilon} n$.

Proof of Claim 2: Pick a node $v \in G$. Let $S_i = \Gamma_i(v) \setminus \Gamma_{i-1}(v)$. By the Chernoff bound, with high probability, for all i such that $|\Gamma_i(v)| < n/2$ we have $|S_{i+1}| > |S_i| 2^{\log^\epsilon n}/4$. Consider the first value of i for which $|\Gamma_i| \geq n/2$. By the above bound $S_i \geq n/4$ and with high probability $|\Gamma_{i+1}| = n$. Hence the distance between v and every other vertex is $\leq \frac{\log n}{\log^\epsilon n - 2} < 2 \log^{1-\epsilon} n$ and so the diameter is $< 4 \log^{1-\epsilon} n$.

Let $G \in \mathcal{G}_{n,p}$ and G' be a random subgraph of G. Picking a random subgraph of a graph picked from $\mathcal{G}_{n,p}$ is the same as picking a graph from $\mathcal{G}_{n,p/2}$. Consider the events

$$A = \{G \in \mathcal{G}_{n,p} : \text{majority of } E(G) \text{ are } k\text{-critical}\}$$
$$B = \{G \in \mathcal{G}_{n,p/2} : \text{diameter of } G \text{ is } < D\}$$
$$\text{for each graph } H, \quad B_H = \{\text{subgraphs } H' \text{ of } H : \text{diameter of } H' \text{ is } < D\}$$

From claim 2 we know $\mathbb{P}(B)$ is large and from claim 1 we know $\mathbb{P}(\neg A)$ is small. Thus $\mathbb{P}(A \cap B) \geq \mathbb{P}(B) - \mathbb{P}(\neg A) > 1/2$. $\mathbb{P}(A \cap B) = \sum_G I[A] \mathbb{P}(G) \mathbb{P}(B_G)$ and so there exists a graph $G \in A$ such that $\mathbb{P}(B_G) \geq 1/2$, i.e. the majority of subgraphs of G have diameter $< D$. Now, undeleting edges that weren't k-critical can only decrease the diameter and hence there exists a least one graph G such that majority of $\{G' : G' \text{ formed from } G \text{ by deleting a subset of } k\text{-critical edges}\}$ have diameter $< D$.

Theorem 4. *For $1 > \epsilon > 0$, it is impossible in one pass to approximate the diameter of an unweighted graph within a factor of $o(\log^{1-\epsilon} n)$ in the semi-streaming model.*

Proof. Let $k = \frac{\log^{1-\epsilon} n}{2}$ and $D = 8k$. Consider a graph G on $\eta = \frac{n-2D}{D}$ vertices with the properties in Lemma 5. Let the subgraphs of this G with diameter $< D$ be $\mathcal{F}(G)$. Observe that

$$|\mathcal{F}(G)| \geq 2^{2^{\log^\epsilon n} n/8} = 2^{\omega(n \cdot \text{polylog } n)}$$

and hence, for any algorithm there exists two graphs $G', G'' \in \mathcal{F}(G)$ that are indistinguishable from information stored by the algorithm. Consequently when we stream in one of G', G'' there exists an edge $e \in E(G') \backslash E(G'')$ whose existence in the streamed graph is undetermined. We stream in D graphs $G_1, \ldots, G_D$ from $\mathcal{F}(G)$ such that there exists an edge (t_i, s_i) in each G_i whose existence is undetermined. Finally we stream in edges (t_i, s_{i+1}) for $i = 2, \ldots D-1$ 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 $4D-1$ while the algorithm can not guarantee that the diameter is less than $3D-1+Dk$. Hence the best approximation ratio is $\Omega(k)$.

We next show that the shortest-path distances can be approximated in one pass in the semi-streaming model. The approximation uses *graph spanners*. A subgraph $G'(V, E_s)$ is a t-spanner of graph $G(V, E)$ if, between any pair of vertices, the distance in G' is at most t times the distance in G.

For an unweighted graph, a $\log n / \log\log n$-spanner S can be constructed in one pass in the semi-streaming model using a simple algorithm similar to the one in [2]. Because a graph whose girth is larger than k can only have $\lceil n^{1+2/(k-1)} \rceil$ edges [6], the algorithm constructs S by adding the edges in the stream to S, if such an edge does not cause a cycle of length less than $\log n / \log\log n$ in the spanner S constructed so far. For a weighted graph, however, the construction in [2] requires to sort the edges according to their weights, which is difficult in the semi-streaming model. So, instead of sorting, we use a geometric grouping technique to extend the spanner construction for unweighted graphs to a construction for weighted graphs. This technique is similar to the one used in [7]. Let ω_{min} be the minimum weight and let ω_{max} be the maximum weight. We divide the range $[\omega_{min}, \omega_{max}]$ into intervals of the form $[(1+\epsilon)^i \omega_{min}, (1+\epsilon)^{i+1} \omega_{min})$ and round all the weights in the interval $[(1+\epsilon)^i \omega_{min}, (1+\epsilon)^{i+1} \omega_{min})$ down to $(1+\epsilon)^i \omega_{min}$. For each induced graph $G^i = (V, E^i)$, where E^i is the set of edges in E whose weight is in the interval $[(1+\epsilon)^i \omega_{min}, (1+\epsilon)^{i+1} \omega_{min})$, a spanner can be constructed in parallel using the above construction for unweighted graphs. The union of the spanners for all the G^i, $i \in \{0, 1, \ldots, \log_{(1+\epsilon)} \frac{\omega_{max}}{\omega_{min}} - 1\}$, forms a spanner for the graph G. Note that this can be done without prior knowledge of ω_{min} and ω_{max}.

Theorem 5. *For $\epsilon > 0$, and a weighted undirected graph on n vertices, whose maximum edge weight, ω_{max}, and minimum edge weight, ω_{min}, satisfy $\log \frac{\omega_{max}}{\omega_{min}} =$ polylog n, there is a semi-streaming algorithm that constructs a $(1+\epsilon)\log n$-spanner of the graph in one pass. The algorithm uses $O(\log_{1+\epsilon} \frac{\omega_{max}}{\omega_{min}} \cdot n \log n)$ bits of space and the worst case processing time for each edge is $O(\log_{1+\epsilon} \frac{\omega_{max}}{\omega_{min}} \cdot n)$.*

Once we have the spanner, the distance between any pair of vertices can be approximated by computing their distance in the spanner. The diameter of the graph can be approximated by the spanner diameter too. Note that, if the girth of an unweighted graph is larger than k, it can be determined exactly in

a k-spanner of the graph. The construction of the $\log n/\log\log n$-spanner thus provides a $\log n/\log\log n$-approximation for the girth.

We end this section by briefly mentioning some graph problems that are simple in the semi-streaming model but may be impossible in a (poly)log-space streaming setting.

A minimum spanning tree can be constructed in one pass and $O(\log n)$ time per edge using a simple adaptation of an existing on-line algorithm [20]. Planarity testing is impossible in the (poly)log-space streaming model, because deciding the existence of a K_5 minor of a graph would require $O(n)$ bits of space. Because a planar graph would have at most $3n-6$ edges, using $O(n)$ storage, many existing algorithms can be adapted to the semi-streaming model.

The following is an algorithm for finding articulation points in the semi-streaming model. It uses one disjoint set data structure, SF, for keeping track of the connected components of the spanning forest, T. It also uses one disjoint set data structure per vertex v, in order to store v's neighbors.

Algorithm 5 (Articulation Points).

$T = (V, \emptyset)$
For each $v \in V$*: SF.makeset(v)*
For each input edge (u, v)*:*
  if SF.find-set(u) = SF.find-set(v) then:
    find the path, $u = a_0, a_1, \ldots, a_k = v$ *from* u *to* v *in* T
    For each $a_i, 0 < i < k$, a_i*.union(*a_{i-1}, a_{i+1}*).*
  else:
    SF.union(u,v)
    $T = T \cup \{(u, v)\}$
    u.makeset(v)
    v.makeset(u)
For each $v \in V$*:*
  if the neighbors of v *w.r.t.* T *lie in at least two different sets*
    then output v *as an articulation point.*

If u is an articulation point there exists two neighbors v and w of u in T such that any path from v to w passes through u. In this case, in the disjoint set structure for u the components containing v and w will never be unioned.

5 Conclusion

We considered a set of classical graph problems in the semi-streaming model. We showed that although exact answers to most of these problems are still impossible, certain approximations are possible. More research is needed for a complete understanding of the model. Particularly, the efficiency of an algorithm in the semi-streaming model is measured by $S(m, n)$, $P(m, n)$ and $T(m, n)$ as in definition 2. Together with the approximation factor, an algorithm thus has 4 parameters. It would be interesting to develop a better understanding of the tradeoffs among these parameters.

References

1. J. Abello, A.L. Buchsbaum, J.R. Westbrook. A Functional Approach to External Graph Algorithms. *Algorithmica*, 32(3): 437–458, 2002.
2. I. Althöfer, G. Das, D. Dobkin, and D. Joseph. Generating sparse spanners for weighted graphs. In *Proc. 2nd Scandinavian Workshop on Algorithm Theory (SWAT'90)*, LNCS 447, 26–37, 1990.
3. 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, Feb. 1999.
4. A.L. Buchsbaum, R. Giancarlo and J.R. Westbrook. On finding common neighborhoods in massive graphs. *Theoretical Computer Science*, 299 (1-3):707–718, 2003.
5. Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.
6. B. Bollobás. *Extremal Graph Theory.* Academic Press, New York, 1978.
7. E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. on Computing*, 28:210-236, 1998.
8. P. Drineas and R. Kannan. Pass Efficient Algorithm for approximating large matrices. In *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 223-232, 2003.
9. D. Eppstein, Z. Galil, and G.F. Italiano. Dynamic graph algorithms. *CRC Handbook of Algorithms and Theory of Computation*, Chapter 8, 1999.
10. 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.
11. P. Flajolet and G.N. Martin. Probabilistic counting. In *Proc. 24th IEEE Symposium on Foundation of Computer Science*, pages 76–82, 1983.
12. A.C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proc. 34th ACM Symposium on Theory of Computing*, pages 389–398, 2002.
13. S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proc. 33th ACM Symposium on Theory of Computing*, pages 471–475, 2001.
14. M. Rauch Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Report 1998-001, DEC Systems Research Center*, 1998.
15. P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. 41th IEEE Symposium on Foundations of Computer Science*, pages 189–197, 2000.
16. M. Karpinski and W. Rytter *Fast Parallel Algorithms for Graph Matching Problems, Oxford Lecture Series in Math. and its Appl.* Oxford University Press, 1998.
17. B. Kalyanasundaram and G. Schnitger. The Probabilistic Communication Complexity of Set Intersection. *SIAM Journal on Discrete Math.*, 5:545–557, 1990.
18. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Extracting large-scale knowledge bases from the web. In *Proceedings of the 25th VLDB Conference*, pages 639–650, 1999.
19. S. Muthukrishnan. Data streams: Algorithms and applications. 2003. *Available at* "http://athos.rutgers.edu/∼muthu/stream-1-1.ps"
20. R. Tarjan. Data Structures and Network Algorithms. *SIAM*, Philadelphia, 1983.
21. R. Uehara and Z. Chen. Parallel approximation algorithms for maximum weighted matching in general graphs. *Information Processing Letters*, 76(1–2):13–17, 2000.