Yale University
Department of Computer Science

On Graph Problems in a Semi-Streaming Model

Joan Feigenbaum [1]
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 [2]
Yale University
zhang-jian@cs.yale.edu

YALEU/DCS/TR-1262
December, 2003

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 by ONR and NSF.

On Graph Problems in a Semi-Streaming Model

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 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 is presented as a stream of edges (in adversarial order), and the storage space of an algorithm is bounded by $O(|V|\text{polylog}(|V|))$.[1] 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 approximation algorithms for the unweighted and weighted matching problems, along with a further algorithm improvement for the bipartite case. We also exhibit $\log n$ semi-streaming approximations to the girth and the problem of computing the distance between specified vertices in a weighted graph.

1 Introduction

Streaming [11] 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 [7, 2, 6, 12, 10, 9]. Yet, the problems considered fall into a small number of categories, such as computing statistics, norms, and histograms. Very few graph problems [3] have been considered in the streaming model.

Massive graphs arise naturally in several real world scenarios. For example, in a *call graph*, nodes correspond to telephone numbers and edges to calls between numbers that call each other during some time interval. The *web graph* is a graph in which the nodes are web pages, and the edges are links between pages. These graphs are all massive in nature. The streaming model is necessary for the study of the efficient processing of such graphs. Furthermore, there are situations in which the graph is revealed in a streaming fashion, such as a web crawler exploring the web graph.

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 by ONR and NSF.

[1] V is the vertex set of a graph.

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 access as a spectrum. At one end of the spectrum, we have dynamic algorithms [5] that may use memory enough for the whole input. At the other end, we have streaming algorithms that use only (poly)log 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 [14] that the middle ground, where the algorithms can use $O(|V|polylog(|V|))$ bits of space, is an interesting area that has been left largely unexplored. Our semi-streaming model covers this middle ground; the algorithms are allowed to use enough space for vertices but not enough for the edges. We consider classical graph-theory problems for dense graphs in this model. In [1], the authors introduced the semi-external model of studying graphs, *i.e.*, one in which the vertex set can be stored in memory, but the edge set cannot. However, these authors addressed the problems in an external memory model where random access to the edges, although expensive, is allowed. This is a major difference between their model and ours.

Besides taking a middle position in the memory-size spectrum, the semi-streaming model allows multiple passes over the input stream. In certain applications, one pass over the stream is all that is required, but there are other situations in which streaming is considered to be an efficient data-processing paradigm, instead of a choice made when data storage is impossible. For example, 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] have considered the multiple-pass model. Full understanding of multiple-pass computation requires further research.

We consider a set of 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 results cover the following categories of problems:

1. **Matchings**: For unweighted graphs, we can find a 1/2-approximation in 1 pass, and, in the weighted case, we can find a 1/6-approximation in 1 pass. For an unweighted bipartite graph, we can find a $(2/3 - \epsilon)$-approximation in $O(\frac{\log 1/\epsilon}{\epsilon})$ passes, where the constant is small.
2. **Distances, diameter, and girth**: We show that the shortest-path distance between any pair of graph vertices can be $(\log n)$-approximated using a semi-streaming construction of $(\log n)$-spanners for the graph that use only one pass. This leads to a $(\log n)$-approximation of the graph diameter and girth.

The rest of the paper is organized as follows. In section 2, we formally define the semi-streaming model. In section 3, we give an algorithm for finding a bipartition in a graph. We then go on to describe our constant-factor streaming approximation algorithms for finding matchings in the unweighted and weighted cases, as well as the unweighted bipartite case. Next, in section 4, we first show that computing the exact distance between two points in an unweighted graph is impossible in the streaming model. Furthermore, upper-bounding the diameter by a constant factor is also impossible in this model. Then, we give semi-streaming $(\log n)$-approximation algorithms for computing the diameter and girth of a graph. At the end of this section, we outline a few problems that have quite simple algorithms in the semi-streaming model. In section 5, we discuss some open problems.

2 Preliminaries

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

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

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 the stream of adjacency-matrix or adjacency-list models, the edges incident to the same vertex would be grouped together. This property is not always present in massive-graph data sets. For example, in a *call graph*, the edges could appear in any order. Because it does not assume any edge order, our definition is general enough to accommodate this type of massive graph.

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. *An* ***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* $O(n \cdot polylog(n))$ *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 the two vertex sets, the neighborhood of x and the neighborhood of y, have a nonempty intersection. Because set disjointness has linear-space streaming complexity [13], the length-2 path problem is impossible in the (poly)log-space streaming model.

Compared to other models with the sequential-input-access constraint, a semi-streaming algorithm would be more space efficient than a dynamic algorithm. At the same time, it would have more computational power than a (poly)log-space streaming algorithm. Better understanding of the semi-streaming model would help to give a full-range view of sequential (one-way) input-access models with various memory-size constraints.

3 Graph Matching

3.1 Unweighted Bipartite Matching

In this section, we develop an algorithm to find a matching that has approximately the maximum cardinality. Our final algorithm will only work for bipartite graphs, and, as a preliminary step, we are required to find a bipartition.

Algorithm 1 (BIPARTITION). *Initially, the vertices have labels* $\{1, 2, \dots n\}$ *and all have sign '+'. On seeing edge* (u, v) *with labels* a *and* b*,*

1. *if $a = b$ and they have the same signs then output 'NOT BIPARTITE.'*

2. *if $a = b$ and they have different signs do nothing.*

3. *if $a \neq b$ we relabel all nodes with label $\max(a, b)$ by $\min(a, b)$. If sign(a) = sign(b) then flip the signs of all the relabeled vertices.*

If, at the end of the stream, we have not outputted 'NOT BIPARTITE', then the graph is BIPARTITE and

$$L = \{ \text{ nodes with sign } +\}$$

$$R = \{ \text{ nodes with sign } -\}$$

forms a bipartition.

The correctness of the algorithm is easily seen, because 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. Note that the overall running time of this algorithm can be improved if, in step 3, instead of relabeling the set of nodes with the larger label, we relabel the set of nodes with the less common label. However, since we are interested in the worst case time, $T(n, m)$, to process each edge, such a trick doesn't buy us anything in our model.

We are now ready to start actually finding matchings in our graph G. Our first algorithm actually finds matchings in arbitrary graphs, not only bipartite graphs. For a graph G and a matching M, we will call a vertex *free* if it doesn't appear as the end point of any edge in M.

Algorithm 2 (1-Pass Greedy Matching). *We take 1 pass through the stream and maintain a matching M. When we see an edge $e = (u, v)$, we add it to M iff both u and v are free vertices.*

Theorem 1. *The above greedy matching algorithm, in one pass and with $O(n \log n)$ space, finds a matching M such that*

$$|M| \geq |\text{OPT}|/2$$

where OPT is a maximum matching.

Proof: The matching found is clearly a maximal matching and a maximal matching is a 2-approx to the maximum matching. □

From this point on, our edges are ordered pairs, the first entry is a node in partition L, and the second entry is a node in partition R. We know the partition from having run BIPARTITION. Note that BIPARTITION can be run in the same pass as the 1-Pass Greedy Matching.

Lemma 1. *A length-3 augmenting path for an edge $e = (u, v) \in M$ is a quadruple (w_l, u, v, w_r) such that there exist edges $(u, w_l), (w_r, v) \in E$, 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 length-3 augmenting paths is a set of length-3 augmenting paths whose wing-tips are disjoint. Let X be a maximum-sized set of simultaneously length-3 augmenting paths for M. Then*

$$|M|(1 + \alpha) \geq 2/3$$

where $|X| = \alpha|M|$.

Proof: Consider a maximum matching $\textsc{Opt}$ and the symmetric difference $\textsc{Opt} \bigtriangledown M$. In each connected component, there is at most 1 more edge from $\textsc{Opt}$ than there is from M. Note that no connected component consists of only a single edge that came from $\textsc{Opt}$, because M is maximal. The number of components with one edge from M and two edges from $\textsc{Opt}$ is at most $|X|$. In all other components, the ratio of edges from M to edges from $\textsc{Opt}$ is at least $2:3$. The result follows. $\square$

Algorithm 3 (A non-streaming greedy algorithm for finding length-3 augmenting paths). *We start with the graph G and a maximal matching M. Consider the edges in M in an arbitrary order. For each $e \in M$, we test whether there exists a length-3 augmenting path for e. If so, we remove the 4 nodes in the augmenting path from G. If not, we only remove the end points of e from the graph.*

Theorem 2. *Algorithm 3 will find a set of simultaneously length-3 augmentable paths of size at least 1/3 of the maximal such set.*

Proof: Let $\textsc{Opt}$ be some maximal set of simultaneously length-3 augmentable paths. Note that each path that we find destroys at most 3 paths that $\textsc{Opt}$ 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. By the same argument, if we find k augmenting paths by any means (such as finding a maximal set of left wings first and completing a maximal subset of these with right wings) we destroy at most $3k$ augmenting paths of $\textsc{Opt}$. $\square$

We first try to adapt this non-streaming algorithm for the streaming context. We do this by performing the algorithm in a number of phases, in each of which we endeavor to find a number of augmenting paths for matched edges or identify matched edges that can't be augmented. Each phase will take 3 passes.

Algorithm 4 (Find augmenting paths). *We have a graph $G = (L \cup R, E)$ and a maximal matching M.*

1. *In one pass, find a maximal set of left wings. If the number of left wings found falls below δM, terminate.*
2. *In a second pass, of the matched edges with left wings, find a maximal set of right wings. [Make a note in memory of the length-3 augmenting paths.]*
3. *Remove the endpoints of every matched edge that got a left wing. Furthermore, remove the wing tips of every matched edge that got* both *wings. Lastly, remove the endpoints of all matched edges that are no longer 3 augmentable. (This requires a third pass.)*
4. *Repeat.*

Theorem 3. *Algorithm 4 will find length-3 augmenting paths for*

$$\frac{\alpha M - 2\delta M}{3}$$

of the matched edges in $3/\delta$ passes.

Proof: The number of phases is at most $1/\delta$, because each phase rules out at least δM matched edges, and there are only M to start with. When we terminate, we have found fewer than δM left wings. Hence there were fewer than $2\delta M$ left wings that could have been found. Consequently, there were fewer than $2\delta M$ length-3 augmenting paths in the remaining graph.

Thus we have destroyed $\alpha M - 2\delta M$ augmenting paths of OPT. By the previous theorem this is only possible if we have already selected at least $\frac{\alpha M - 2\delta M}{3}$ augmenting paths.

□

We want to find a $(2/3 - \epsilon)$-approximation for matching. Therefore, we should only do something when $\alpha > \frac{3\epsilon}{2-3\epsilon}$; otherwise, we already have a

$$\frac{2}{3}\frac{1}{1+\alpha} \geq \frac{2}{3} - \epsilon$$

approximation. Let $\delta = \frac{\epsilon}{2-3\epsilon}$. Each time we run the above algorithm, we get

$$\frac{\alpha M - 2\delta M}{3} \geq \frac{\alpha M}{9}$$

augmentations. This is 1/9 of the possible total.

Algorithm 5 (An approximation algorithm for bipartite matching). *We start by using the 1-pass greedy algorithm to find a maximal matching M and simultaneously calculate the bipartition. We then do the following steps k times:*

1. *Run the algorithm 4 with $\delta = \frac{\epsilon}{2-3\epsilon}$ to find a 1/9 of the maximum possible number of length-3 augmenting paths.*

2. *For each $e = (u, v) \in M$ for which an augmenting path (w_l, u, v, w_r) is found, remove (u, v) from M and add (u, w_l) and (w_r, v), thereby increasing the size of the maximal matching.*

Theorem 4. *Algorithm 5 finds a $2/3 - 1/6(8/9)^k$ approximation.*

Proof: After i repetitions, let s_i be the approximation ratio. Before we start, $s_0 = 1/2$, because the original maximal matching is a 1/2 approximation. Augmenting the maximum number of matched edges gives a 2/3-approximation i.e.

$$s_i + \alpha s_i \geq 2/3$$

Hence in iteration $i + 1$, we have an

$$s_{i+1} = s_i + \alpha s_i/9 \geq 8/9 s_i + 2/27$$

approximation. Solving this recurrence gives the theorem. □

Theorem 5. *We can find a $(2/3 - \epsilon)$-approximation with*

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

passes.

Proof: To get a $(2/3 - \epsilon)$-approximation, we need $k = \left\lceil \frac{\log 6\epsilon}{log 8/9} \right\rceil$. Each iteration requires $\frac{6-9\epsilon}{\epsilon}$ passes from theorem 3 with the appropriate value of δ. □

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 [16] 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 [16].

We propose a new algorithm that uses only one pass yet still manages to find a matching $1/6$ the optimal size.

Algorithm 6 (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 $C = \{e' | e' \in M \text{ and } e' \text{ and } e \text{ share an end point}\}$.*

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

Proof: For any set of edges S, let $c(S) = \sum_{e \in S} w_e$. 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 $c(S)$.

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

$$\begin{aligned} C_0 &= \{e\} \\ C_1 &= \text{ the edges murdered by e} \\ C_i &= \cup_{e' \in C_{i-1}} \text{ the edges murdered by } e' \end{aligned}$$

Claim 1: $c(T(e)) \leq c(e)$

Proof of claim: For each murdering edge e, $c(e)$ is at least twice the cost of murdered edges, and an edge has at most one murderer. Hence, for all i,

$$c(C_i) \geq 2c(C_{i+1})$$

therefore

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

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 else 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 $c(o)$ to e. If two edges e_1 and e_2 are accountable for o, then we charge $\frac{c(o)c(e_1)}{c(e_1)+c(e_2)}$ to e_1 and $\frac{c(o)c(e_2)}{c(e_1)+c(e_2)}$ to e_2. In either case, the amount charged by o to any edge e is at most $2c(e)$.

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 $2c(e)$ because $c(e') \geq c(e)$. 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 $c(\textsc{Opt})$ to the survivors and their trails of the dead, and hence

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

By Claim 1,

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

and the lemma 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 2. *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 (t, r), 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 7. *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 $S(n)$. Take a directed graph $G = (\{v_1, \ldots v_n\}, E)$ with two specific nodes $s = v_1$ and $t = v_n$ without loss of generality. 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' iff 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

$$\begin{aligned} E' &= \{(v_{il}, v_{ir}) : i \in [n]\} \\ &\cup \{(v_{ir}, v_{jl}) : (v_i, v_j) \in E\} \\ &\cup \{(v_s, v_1)\} \cup \{(v_t, v_n)\} \end{aligned}$$

Let the existing matching be

$$M = \{(v_{il}, v_{ir}) : i \in [n]\}$$

There is an augmenting path in G' iff 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 between vertices, diameter, and girth on graph streams. We briefly mention several other graph-stream problems at the end of this section.

Many existing shortest-path algorithms construct the path one edge at a time. Borrowing a notion from parallel computing, one may say that such an approach is sequential in nature. With limited storage space and a small number of passes, this type of algorithm could not be implemented in a semi-streaming model. Indeed, we show that, in the semi-streaming model, computing exact shortest-path distances, or even certain approximations to them, is impossible in one pass over the graph stream. On the other hand, the semi-streaming model does enable us to store a sparse subgraph of the input graph. Thus we seek to construct a sparse graph in the semi-streaming model that approximately preserves distances in the original graph.

We first show that, even in the semi-streaming model, computing exact shortest-path distances, or a certain approximation, is not possible in one pass.

Theorem 8. *Any one-pass streaming algorithm that computes a BFS tree on an input graph stream must use $\Omega(m)$ bits of storage.*

Proof: First, we define a family of graphs F in which there is a source s and two sets of n vertices each, denoted L and R. By our definition, any graph $G \in F$ will have the following structure: a bipartite graph H between L and R containing m edges, together with a path starting at s and visiting the vertices of L in some order, using n edges. Fix a value of $m \leq n^2/2$. As before there are an exponential number of possible bipartite graphs H.

Now imagine a stream of edges in which the bipartite graph appears first, followed by the edges in the path. Consider the state retained when the algorithm has seen the bipartite graph but not the path. We claim that, for any two distinct bipartite graphs H and H', this state must be different. Suppose not. Because H and H' are distinct, there exists a vertex $v \in R$ such that the neighborhoods of v in H and H' are different. Without loss of generality, suppose that u is a vertex in L that is adjacent to v in H but not in H'.

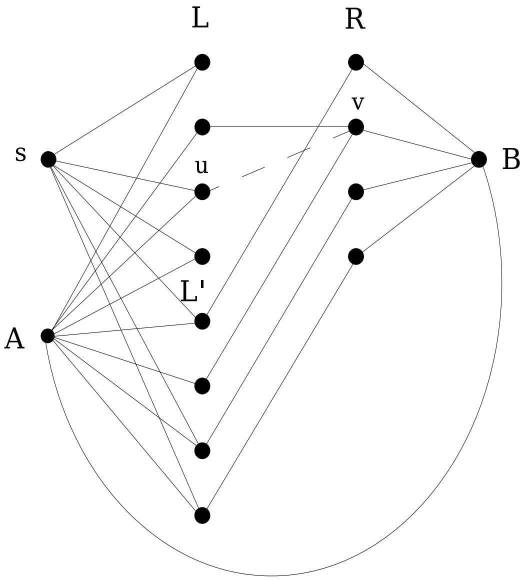

Figure 1: A picture of G and G'. Note, (u, v) is an edge in G and not in G'. For clarity, we left out a dense bipartite graph between L and L'.

After the bipartite graph H has been input, consider following up with the path that goes from s to u and then to the other vertices of L. Then in H, v must be at level 2 whereas in H', v is at a level greater than 2. However, if we kept the same state and saw the same path we would be forced to output the same tree in both cases which would be a mistake. Thus, the number of bits in the state must be at least log of the number of bipartite graphs, which is $\Omega(m)$. □

Theorem 9. *Let d be the diameter of a graph G. Any one-pass algorithm that outputs $\tilde{d}$, such that $d \leq \tilde{d} < 4/3d$, must use $\Omega(m)$ bits.*

Proof: Let F be a family of graphs with the following structure. Each graph $G \in F$ consists of a bipartite graph between two sets of vertices L and R such that $|L| = |R| = n$. In addition G contains another set of n vertices L'. The ith vertex of L' is adjacent just to the ith vertex of R. Finally, there are three special vertices labeled s, A, and B. A is connected to all vertices in $L \cup L'$, B is connected to all the vertices in R, and A is connected to B. We will specify the neighbors of s later. Fix a value for $m \leq n^2/2$.

Now imagine a stream of edges in which all edges incident on s appear after the rest of the edges in G. Consider the state retained when the algorithm has seen all of G except the edges incident on s. We claim that, for any two distinct graphs $G = (V, E)$ and $G' = (V, E')$ chosen from F, this state must be different. Suppose it is not. Since G and G' are distinct, there exists $v \in R$ such that the neighborhoods of v in G and G' are different. Without loss of generality, suppose that u is a vertex in L that is adjacent to v in G but not in G'.

Now we specify the neighborhood of s. Let $\Gamma_{G'}(v) = \{x \in L \cup L' \mid (x, v) \notin E'\}$. Consider finishing up the input with all edges of the form (s, w) where $w \in \Gamma_{G'}(v) \setminus \{B\}$, (see figure 1).

Now consider the diameter of the two graphs. We claim that the diameter of G' is at least 4. Consider performing a BFS from s. The vertex s itself will be at level 0. $L \cup L' \setminus \Gamma_{G'}(v)$ will be at level 1. A and $R \setminus \{v\}$ will be at level 2. $\Gamma_{G'}(v) \cup \{B\}$ will be at level 3. Finally, v itself will be at level 4.

Now we will show that the diameter of G is at most 3. If $x, y \in L \cup L'$, there is path of the form $x \rightsquigarrow A \rightsquigarrow y$. Similarly, if $x, y \in R$, there is a path of the form $x \rightsquigarrow B \rightsquigarrow y$. Now, if $x \in L \cup L'$ and $y \in R$, there is a path of the form $x \rightsquigarrow A \rightsquigarrow B \rightsquigarrow y$.

All that is left to compute are paths that are incident on s in G. One can get from s to any element in $L \cup L' \setminus \Gamma_{G'}(v)$ in one step. One can get from s to any element in $\Gamma_{G'}(v)$ by $s \rightsquigarrow L \cup L' \rightsquigarrow A \rightsquigarrow \Gamma_{G'}(v)$. Also, one can reach A and B by following $s \rightsquigarrow L \cup L' \rightsquigarrow A \rightsquigarrow B$. One can get from s to $R \setminus \{v\}$ by $s \rightsquigarrow L' \rightsquigarrow R \setminus \{v\}$. One can get from s to v by $s \rightsquigarrow u \rightsquigarrow v$. All of these paths require 3 hops or less. Thus the diameter of G is at most 3.

So let A be an algorithm that produces an upper bound on the diameter which is within a factor of $4/3 - \epsilon$ of the true diameter, for some $\epsilon > 0$. If A kept the same state for G and G', it would output the same diameter for both graphs thus violating either the property that it always produces an upper bound or the approximation factor. $\square$

We next show that the shortest-path distances can be $\log n$-approximated in one pass in the semi-streaming model. The approximation uses graph spanners. A graph spanner is a subgraph that preserves the approximate distances between all pairs of vertices. 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. The size of a spanner is the number of edges in the subgraph.

Lemma 3. *There is a semi-streaming graph algorithm that constructs a $\log n$-spanner of size $O(n)$ in one pass over the graph stream.*

Proof: The spanner is constructed by adding the spanner edges one by one. Each time we see an edge, we store it with the current subgraph. We then check whether this causes a cycle of length $\log n + 1$ or less in the subgraph. If so, we remove the heaviest edge from this cycle. After seeing the whole graph stream, the algorithm outputs the stored edges as a $\log n$-spanner.

The stored edges form a $\log n$-spanner, because, for each removed edge, there is a path connecting the two end vertices of the edge. The length of the path is at most $\log n$ times the length of the edge. The size of the spanner is $O(n)$ because an extremal graph-theory result [4] states that a graph whose girth is larger than k can only have $\lceil n^{1+2/(k-1)} \rceil$ edges. $\square$

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.

Theorem 10. *There is a semi-streaming graph algorithm that $\log n$-approximates the distance between any pair of vertices in a graph. The algorithm uses $O(n \log n)$ bits of space and makes one pass over the graph stream. The diameter of the graph can also be $\log n$-approximated.*

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 $\log n$-spanner thus provides a $\log n$-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 using a simple adaption of an existing on-line algorithm [15] that maintains a minimum spanning forest for the graph seen so far. This algorithm adds each edge it sees to the forest but, if a cycle is created, removes the heaviest edge in the cycle.

Planarity test is a 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. However, only a graph having at most $3n-6$ edges would require a nontrivial planarity test. Many of the existing testing algorithms can be applied in the semi-streaming model, with storage sufficient for the $3n-6$ edges.

Articulation points, bridges, biconnected, and 2-edge-connected components can be found with a second pass after the computation of a spanning forest in one pass. Let F be the spanning forest. During the second pass, for each tree T in F and for each $v \in T$, we maintain a Union-Find structure for the components of T created by the removal of v. We do not explicitly enumerate the vertices in each component since this will result in each vertex being listed n times leading to a space requirement of $\Omega(n^2)$. Instead, each component is represented by the edge incident on v leading to that component. Let (x, y) be the edge being processed. Let $x = a_0, a_1, \ldots, a_k = y$ be the path from u to v in T. For each $a_i, 0 < i < k$ we union the components represented by a_{i-1} and a_{i+1}.

If the neighborhood of some vertex v is still partitioned into more than one component at the end, v is an articulation point and the graph can be appropriately decomposed at v. Performing such decompositions at every vertex will produce the biconnected components of G. Similar techniques can be used to detect bridges and 2-edge connected components.

5 Conclusion

The semi-streaming model is an interesting one for graph problems. On the one hand, it is computationally more powerful than the (poly)log-space streaming algorithms. On the other hand, it preserves space-efficiency to a certain extent.

We considered a set of graph problems in the semi-streaming model. We showed that, although exact answers to most of these problems would still be impossible, certain approximations are possible. However, more research is needed for a complete understanding of the model. We propose the following open problems:

1. The efficiency of an algorithm in our 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 in the semi-streaming model has 4 parameters. It would be interesting to develop a better understanding for the tradeoffs among these parameters. Other more specific goals include the following:

2. For unweighted bipartite matching, find a 2/3-approximation that runs in constant number of passes.

3. Close the gap between the upper and lower bounds on approximation ratios for calculating the diameter of a graph. We suspect that it is the lower bound that is looser.

References

[1] J. Abello, A.L. Buchsbaum, J.R. Westbrook. A Functional Approach to External Graph Algorithms. *Algorithmica*, 32(3): 437–458, 2002.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, Feb. 1999.

[3] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632, 2002.

[4] B. Bollobás. *Extremal Graph Theory.* Academic Press, New York, 1978.

[5] D. Eppstein, Z. Galil, and G.F. Italiano. Dynamic graph algorithms. *CRC Handbook of Algorithms and Theory of Computation*, Chapter 8, 1999.

[6] 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.

[7] P. Flajolet and G.N. Martin. Probabilistic counting. In *IEEE Symposium on Foundation of Computer Science*, pages 76–82, 1983.

[8] Alan M. Frieze and Ravi Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.

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

[10] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *ACM Symposium on Theory of Computing*, pages 471–475, 2001.

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

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

[13] B. Kalyanasundaram and G. Schnitger. The Probabilistic Communication Complexity of Set Intersection. *SIAM Journal on Discrete Math.*, 5:545–557, 1990.

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

[15] R.E. Tarjan. Data Structures and Network Algorithms. *SIAM*, Philadelphia, 1983.

[16] Ryuhei Uehara and Zhi-Zhong Chen. Parallel approximation algorithms for maximum weighted matching in general graphs. *Information Processing Letters*, 76(1–2):13–17, 2000.