On Graph Problems in a Semi-streaming Model

Arne Kemper

Abstract Massive graphs arise naturally in a lot of applications, especially in communication networks like the internet. The size of these graphs makes it very hard or even impossible to store set of edges in the main memory. Thus, random access to the edges can't be realized, which makes most offline algorithms unusable. This essay investigates efficient algorithms that read the edges only in a fixed sequential order. Since even basic graph problems often need at least linear space in the number of vertices to be solved, the storage space bounds are relaxed compared to the classic streaming model, such that the bound is $O(n \cdot \text{polylog } n)$. The essay describes algorithms for approximations of the unweighted and weighted matching problem and gives a $o(\log^{1 - \epsilon} n)$ lower bound for approximations of the diameter. Finally, some results for further graph problems are discussed.

1 Introduction

Streaming[2, 4, 5] is a well-studied method for handling massive data sets. There are two interpretations for how the stream is actually created in the application. The first one usually assumes a client-server model, in which data item are send one by one from the client to the server, where they are processed. The other one is used for local computations of massive instances of a problem, where it is very expensive or even impossible to realize random access to the items, usually because the data is much larger than the local memory. Here the access problems are reduced by allowing only sequential access in a fixed order to the items. This practice make streaming interesting even for local computations. Obviously, streaming algorithms work in both interpretations, but the latter one allows us a new approach to solve problems on data sets that are too big for normal computations. One set of problems that’s worth to explore in this model is graphs problems. Massive graphs are common for real world applications, for example the web graph, where vertices are web pages and edges are links or the call graph, that shows phone calls between parties. Since standard graph algorithms assume random access to the edge set these algorithms are unusable for this massive graphs since the constant reloading of data into the memory would be very expensive and time consuming.
Still, an application like a web crawler that collects the edges for the web graph, would still generate edge one by one and could be used in client-server model. But since these algorithms are mostly used in the context where the edge set stored locally but too big for the memory, we can actually do multiple passes of the data without too much problems.

A big problem for solving graph problems in the streaming model is the rather low storage space bound. A lot of basic graph problems are not solvable in sublinear space, thus such a bound is not feasible. Therefore this essay discusses basic graph problems in the semi-streaming model. This model allows storage space of $O(n \cdot \text{polylog } n)$ instead of just $O(\text{polylog } n)$ and explicitly allows multiple passes over the input, which is sometimes considered in the normal streaming model, but still rather uncommon.

I will present a $(2/3 - \epsilon)$ approximation semi-streaming algorithm for the bipartite matching problem in unweighted graph in $O\left(\frac{\log \frac{1}{\epsilon}}{\epsilon}\right)$ passes, a one-pass algorithm for a $1/6$-approximation for the weighted graph matching problem. At last I will present a $\log n / \log \log n$-approximation for the diameter and a $\Omega(\log^{1-\epsilon} n)$ lower bound for the approximation of the diameter and other distance problems.

2 Preliminaries

A graph $G$ is denoted by $G = (V, E)$, where $V$ is the set of vertices and $E$ the set of edges. Note that the number of vertices is $n$, unless stated otherwise, therefore all bounds are dependent of this number. The number of edges is denoted with $m$.

**Definition 1.** A graph stream is a sequence of edges $e_{i1}, e_{i2}, \ldots, e_{im} \in E$. $i_1, \ldots, i_m$ is an arbitrary permutation, so the order of the edges is arbitrary but fixed during the execution of an algorithm.

An algorithm can access these edges one by one and only in this order. This means that if the algorithm wants to access an edge again after discarding it, it has to go through the whole set of edges again. Note that since the order of the edges isn’t random this definition also works for streams that stream the edges in order of the adjacency matrix or list, where edges adjacent to a vertex are grouped. But since this isn’t guaranteed the algorithms have to work for any order of edges.

The efficiency of these algorithms is measured in the number of bits of storage space it needs, the time it need per edge and the number of passes it needs over the input.

**Definition 2.** A semi-streaming graph algorithm is an algorithm that computes over a graph stream. It accesses the input $P(n, m)$ times via one-way passes and $T(n, m)$ time per edge. The algorithm uses $S(n, m)$ bits of space, where $S(n, m) \in O(n \cdot \text{polylog } n)$.

To show that the higher space limitation is actually necessary, consider the following Lemma:

**Lemma 1.** Every algorithm that decides if there is a directed path from $s \in V$ to $t \in V$ in a Graph $G = (V, E)$ requires $\Omega(m)$ bits of space.

*Proof.** Consider a family $\mathcal{F}$ of graphs $G = (L \cup R \cup \{s, t\}, E)$, where all edge are directed from $L$ to $R$ or of the form $(s, l)$ or $(r, t)$ with $l \in L$ and $r \in R$. Let $|L| = |R| = n$ and $|E| \leq n^2/2$. Now consider a stream that gives all edges between $L$ and $R$, then one edge of the form $(s, l)$ and at last an edge of the form $(r, t)$. Before the last two edges are streamed in, the algorithm can’t decide whether an edge it has seen before is part of a path between $s$ and $t$, therefore it needs a different memory configuration for every possible combination of edges, since the existence of any edge could mean a different answer to whether there is a path. Thus the algorithm needs at least $\Omega(\log_2 |\mathcal{F}|) \in \Omega(m)$.  


Since connectivity of two vertices appears as a subproblem in many algorithms, for example the diameter or shortest paths, linear space is needed for graph problems and the classic streaming model is not applicable.

3 Graph Matching

Definition 3. Given a graph $G = (V, E)$, a matching is a set $M \subseteq E$ such that no two edges in $M$ have common endpoint. If the size of the set can not be improved by just adding edges it's called maximal matching, if it is the biggest of all possible matchings, it's called a maximum matching.

This section will show that (approximations of) graph matching are indeed possible in the semi-streaming model. First the unweighted case will be considered, the weighted case will be discussed afterwards.

3.1 Unweighted Bipartite Matching

The first step to test for a bipartite matching is testing if the graph is bipartite. This can be done in one pass.

Algorithm 1 IsBipartite

Maintain a disjoint set data structure for the connected components found so far and associate a sign with every vertex, such that no two connected vertices have the same sign.

If a new edge connects two vertices with the same sign, try fixing it by flipping the signs of one of the connected components.

If this can't fix the signs the graph is not bipartite.

The algorithm test if an edge connects two vertices with the same sign. If it doesn't, this edge does not contradict an bipartition on the graph seen so far. The information that the connected vertices are in the same connected component is stored. Since the only way to change a sign is to flip the signs of the whole connected components the condition is met over the whole algorithm, if it is ensured for every edge. So if the algorithm doesn't find an edge that connects two vertices with the same sign, that can’t be fixed, the graph is bipartite, where the two sets are the vertices with a common sign, respectively. The disjoint data structure with union by rank and path compression can be changed, such that it maintains the signs and still uses amortized only constant time.

Similarly, a $1/2$-matching for any unweighted matching can be found in one pass. This does not need a bipartition and can therefore be done in the same pass as finding a bipartition.

Given a matching $M$, a vertex is called free if it isn't an endpoint of any edge in $M$.

Algorithm 2 GreedyMatching

Require: A graph stream of a graph $G$.

Ensure: A maximal matching in $G$.

1: $M \leftarrow \emptyset$
2: for edge $e$ that is streamed in do
3: if both ends of $e$ are free w.r.t $M$ then
4: $M \leftarrow M \cup \{e\}$
5: end if
6: end for
Since every edge is considered, this matching is obviously maximal. For the approximation factor, consider the general case:

**Lemma 2.** Every maximal matching $M$ is a $1/2$-approximation of the maximum matching $OPT$.

**Proof.** Every edge in $M$ has endpoints in common with at most two edges in $OPT$. If any edge in $M$ would not have at least one endpoint in common with an edge in $OPT$, one could add this edge to $OPT$, which would contradict the optimality. So every edge in $M$ shares at least one endpoint with an edge in $OPT$. Additionally, if any edge in $OPT$ would not have a common endpoint with an edge in $M$, this edge could be added to $M$, so $M$ would not be maximal. Thus, $|M| \geq \frac{1}{2} |OPT|$. \[\square\]

Algorithm 2 finds a $1/2$-approximation in one pass in any graph. For better approximations in bipartite graphs we need to improve this matching. Since the all considered graphs in this chapter from now on are bipartite, they are treated as if they are directed and all edges go from $L$ to $R$. Because edges are unweighted, we could improve the matching by finding two edges which end in both endpoints of an edge in the matching and their other respective endpoint is free. If we remove the edge from the matching and add the two found one instead the set is still a matching, but it’s size is improved. These three edges describe an augmenting path.

**Definition 4.** For a matching $M$ for a bipartite graph $G = (L \cup R, E)$ a (length-3) augmenting path is a tuple $(w_l, u, v, w_r)$ if $(u,v) \in M, (u,w_l), (w_r,l) \in E$ and $w_l, w_r$ are free. $w_l$ and $w_r$ are the left and right wing tip, respectively, $(u,w_l)$ is the left wing and $(w_r,l)$ the right wing.

A set of length-3 augmenting path where the paths are pairwise disjoint is called simultaneously augmentable.

This definition can be easily generalized to longer paths (see figure 1), but this algorithm uses only paths of length 3.

![Fig. 1 Two augmenting paths (of length 3 and 5) in $G$ with a matching $M$. Green vertices are free w.r.t. $M$, red edges $\in M$, blues ones $\in E \setminus M$.](image)

Note that applying a set of simultaneously augmentable length-3 augmenting paths still results in a matching since the added edges must be edge disjoint.

Now, an algorithm that finds such a set can be defined, which will be used in the main algorithm.
Algorithm 3 FindAugmentingPaths

Require: A graph stream of a bipartite graph $G$, a matching $M$ and a parameter $\delta$.
Ensure: A set of simultaneous augmentable augmenting paths.
1: In one pass, find a maximal set of disjoint left wings $L$
2: if $L \leq \delta M$ then return the set found
3: end if
4: In another pass, for every edge that has a left wing, find a maximal set of disjoint right wings (they form a set of simultaneously augmentable augmenting paths)
5: In another pass, find a set of vertices, that:
   • are endpoints of an edge $e \in M$ with a left wing
   • are the wingtips of edges with both wings
   • are endpoints of a matched edge that can’t be augmented anymore
6: Repeat

Note that this algorithm does not change the matching, but only finds augmenting paths. The change of the matching happens in the main algorithm:

Algorithm 4 UnweightedBipartiteMatching

Require: A graph stream of a bipartite graph $G$ and a parameter $\epsilon$.
Ensure: A $2/3 - \epsilon$ approximation of a maximum matching.
1: In one pass, find a maximal matching $M$ (by algorithm 2) and the bipartition (by algorithm 1)
2: for $k = 1, 2, \ldots, \left\lfloor \log_6 \frac{M}{\epsilon} \right\rfloor$ do
3: Run algorithm 3 with $G, M$ and $\delta = \frac{\epsilon}{2^k}$
4: for each augmenting path $(w_L, u, v, w_R)$ found in the last step do
5: remove $(u, v)$ from $M$ and add $(u, w_L)$ and $(w_R, v)$
6: end for
7: end for

The efficiency of this algorithm hinges on the quality of the set of augmenting paths found by algorithm 3. To evaluate this, a relationship with a maximum set has to be established.

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

Proof. Consider an augmenting path from $S$. Each of its wing tips can block at most one path in $X$, since no vertex can appear twice. Additionally there might be a different augmenting path for the matched edge in $X$, so for every path in $S$, there are at most 3 paths in $X$. Thus, $|S| \geq \frac{1}{3}|X|$. $\square$

The algorithm 3 does not produce a maximal matching, since it terminates if the number of left wing tips is less than the $\delta M$ threshold. So the next question would be how many paths (compared to a maximum set $X$) the algorithm does find.

Lemma 4. Algorithm 3 finds at least $\frac{|X| - 2\delta M}{3}$ simultaneously augmentable length-3 augmenting paths in $3/\delta$ passes, where $X$ is a maximum set of simultaneously augmentable length-3 augmenting paths.

Proof. Since every repetition of the steps in the algorithm takes passes, the number of passes is dependent on how often more than $\delta M$ many left wings can be found. Since all further repetitions at least ignore those edges in $M$ that have a left wing, at least $\delta M$ edges are removed in every round. Thus, after at most $(3/\delta)$ passes the algorithm terminates.
Let $L(M)$ be the set of the endpoints of edges in $M$ that are in $L$ and $V_{L(M)}$ be the set of possible left wing tips for edges in $M$. Since the left wings form a maximal matching between the part of $L(M)$
and \( V_k(M) \) that is not ignored and in the last repetition there are at most \( \delta M \) left wings are found, by Lemma 2, the maximum matching between these sets has size at most \( 2\delta |M| \). Furthermore, there obviously can’t be more augmenting paths than left wings. This means that all other augmenting paths are in the part of the graph that is ignored in the last repetition, so at least \(|X| - 2\delta |M|\) paths. The algorithm finds a maximal set of augmenting paths in this part of the graph. Thus, according to Lemma 3 it finds at least \( \frac{|X| - 2\delta |M|}{4} \) paths. \( \square \)

This makes no statement about how big the set \(|X|\) is compared to the matching \( M \) and a maximum matching. A relationship between them has to be established to find out what the size of the matching is at the end of the algorithm.

**Lemma 5.** Let \( X \) be a maximum set of simultaneously augmentable length-3 augmenting paths for a maximum matching \( M \) and \( OPT \) be an maximum matching. Then \(|M| + |X| \geq 2/3|OPT|\).

**Proof.** Consider the connected components of the symmetric difference \( M \triangle OPT \). Note that the edges are undirected. None of the connected components can only consist of a single edge of \( OPT \), since then both endpoints of this edge would be free with regard to \( M \), which contradicts the maximality of \( M \). Every connected component has at most one more edge from \( OPT \) than from \( M \), because no two edges from the same set can be connected. So the only connected components that do not have a ratio between \( M \) and \( OPT \) of at least \( \frac{2}{3} \) are those with one edge from \( M \) and two from \( OPT \). But these are at most \(|X|\) since they are augmenting paths. So every edge from \( M \) is either part of \( OPT \) and therefore not in \( M \triangle OPT \), part of a connected component where the ratio is \( 2/3 \) or part of one of the at most \(|X|\) components where the ratio is \( 1/2 \). Thus, \(|M| + |X| \geq 2/3|OPT|\). \( \square \)

Now, putting all of this together gives the desired result.

**Theorem 1.** For any \( 0 < \epsilon < 1/3 \) and a bipartite graph, Algorithm 4 finds a \( 2/3 - \epsilon \) approximation of a maximum matching in \( \mathcal{O}(\frac{2\epsilon}{1-\epsilon}) \) passes. The algorithm takes amortized constant time per edge in the first pass and constant time per edge in every other pass. The algorithm needs \( \mathcal{O}(n \cdot \log n) \) bits of storage space.

**Proof.** The constant processing time can be achieved by keeping the state of a vertex regarding whether is part of the matching or ignored for the time being in memory. The needed storage is then defined by this state \( (\mathcal{O}(n)) \), the bipartition \( (\mathcal{O}(n)) \) and several matchings, which are sets of at most \( n/2 \) edges, where a single edge takes \( \mathcal{O}(\log n) \) space. So the total storage space is \( \mathcal{O}(n \cdot \log n) \) bits.

The correctness of the algorithm is yet to be shown. Let \( OPT \) be the size of a maximum matching in \( G \). Consider the \( i \)th repetition of the loop in Algorithm 4. Let \( M_i \) be the matching found in this repetition, \( X_i \) a maximum sized set of simultaneously augmentable length-3 augmenting paths for \( M_i \), \( \alpha_i = \frac{|X_i|}{|M_i|} \) and \( s_i = \frac{|M_i|}{OPT} \).

Note that if \( \alpha_i \leq \frac{3\epsilon}{2-3\epsilon} \), by Lemma 5,

\[
|M_i|(1 + \alpha_i) \geq \frac{2}{3} OPT \Leftrightarrow |M_i| \geq \frac{2}{3} \cdot \frac{1}{1 + \alpha_i} OPT \Leftrightarrow |M_i| \geq (\frac{2}{3} - \epsilon)OPT.
\]

Thus, \( M_i \) is a \( \frac{2}{3} - \epsilon \) approximation. Therefore we assume \( \alpha_i \geq \frac{3\epsilon}{2-3\epsilon} \) holds for all \( i \). Remember that \( \delta = \frac{2}{2-3\epsilon} \), therefore \( \delta \leq \frac{3\epsilon}{2} \) for all \( \alpha_i \). Using this and Lemma 4 gives
On Graph Problems in a Semi-streaming Model

\[
\frac{|X_i| - 2\delta|M_i|}{3} = \alpha_i|M_i| - 2\delta|M_i| \\
\geq \alpha_i|M_i| - 2(\alpha_i/3)|M_i| \\
= \frac{1}{3} \cdot \alpha_i|M_i| \\
= \frac{\alpha_i|M_i|}{3} \\
\Leftrightarrow \frac{\alpha_i - 2\delta}{3} \geq \frac{\alpha_i}{9}.
\]

(1)

By Lemma 5

\[
|M_i| + |X_i| \geq \frac{2}{3} \cdot \text{OPT} \\
\Leftrightarrow |M_i| + \alpha_i|M_i| \geq \frac{2}{3} \cdot \text{OPT} \\
\Leftrightarrow \frac{|M_i| + \alpha_i|M_i|}{\text{OPT}} \geq \frac{2}{3} \\
\Leftrightarrow s_i + \alpha_is_i \geq \frac{2}{3} \\
\Leftrightarrow \alpha_is_i \geq \frac{2}{3} - s_i.
\]

(2)

Furthermore, with Lemma 4 we get

\[
|M_{i+1}| = |M_i| + \frac{\alpha_i|M_i| - 2\delta|M_i|}{3} \\
= |M_i| \cdot (1 + \frac{\alpha_i - 2\delta}{3}) \\
\overset{(1)}{\geq} |M_i| \cdot (1 + \frac{\alpha_i}{9}) \\
\Leftrightarrow \frac{|M_{i+1}|}{\text{OPT}} \geq \frac{|M_i| \cdot (1 + \frac{\alpha_i}{9})}{\text{OPT}} \\
\Leftrightarrow s_{i+1} \geq s_i \cdot (1 + \frac{\alpha_i}{9}) = s_i + \frac{s_i\alpha_i}{9}
\]

(3)

Putting this together we can get a closed recurrence:

\[
s_{i+1} \geq s_i + \frac{s_i\alpha_i}{9} \\
\overset{(2)}{\Leftrightarrow} s_{i+1} \geq s_i + \frac{2/3 - s_i}{9} \\
\Leftrightarrow s_{i+1} \geq s_i - \frac{s_i}{9} + \frac{2}{27} \\
\Leftrightarrow s_{i+1} \geq \frac{8}{9} s_i + \frac{2}{27}
\]

(4)

Because the algorithm finds a maximal matching in the first pass and Lemma 2 \(s_0 \geq 1/2\). Using this, a solution for (4) can be found:

\[
s_i \geq \frac{2}{3} - \frac{1}{6} \left(\frac{8}{9}\right)^i
\]

The algorithm does \(k = \left\lceil \frac{\log 6\epsilon}{\log 8/9} \right\rceil\) repetitions, so
\[ \frac{|M_k|}{OPT} = s_k \geq \frac{2}{3} - \frac{1}{6}\left(\frac{8}{9}\right)^k \geq \frac{2}{3} - \frac{1}{6}\left(\frac{8}{9}\right)^{\log_{6/9} \frac{\log 6}{\log 8/9}} = \frac{2}{3} - \frac{1}{6} \cdot 6\epsilon = \frac{2}{3} - \epsilon. \]

Conclusively, the number of passes is

\[ 1 + k \cdot \frac{3}{\delta} = 1 + k \cdot \frac{6 - 9\epsilon}{\epsilon} = 1 + \left\lceil \frac{\log 6\epsilon}{\log 8/9} \right\rceil \cdot \frac{6 - 9\epsilon}{\epsilon} \in O\left(\frac{\log 1/\epsilon}{\epsilon}\right) \]

\[ \square \]

### 3.2 Weighted Matching

In the unweighted case all edges have the same value in a matching, therefore only the number of edges influences the quality of a matching. Contrary to this, in the weighted case every edge \( e \) has a weight \( w(e) \) and the quality of a matching \( M \) is \( \sum_{e \in M} w(e) \). There are many offline algorithms to solve this problem and at least one can be adapted for the streaming model.

First the edges are partitioned into \( \left\lceil \log_{1+\epsilon/3} (\frac{3}{\epsilon} + 1) n \right\rceil \) groups and the algorithm finds maximal matchings in all those groups, starting with the group with highest weight. This leads to a \( \frac{1}{2+\epsilon} \)-approximation in \( O\left(\log_{1+\epsilon/3} n\right) \) passes and \( O(n \log n) \) storage. Details can be found in [6].

Since the number of passes isn’t constant in \( n \) anymore it is interesting if there are approximations with less passes. The following algorithm achieves a \( \frac{1}{6} \)-approximation in only 1 pass.

For a matching \( M \), let \( w_{\text{adj}}(e) \) be the sum of the weights of the (at most 2) edges in \( M \) that are adjacent to \( e \).

\[ w(e) \]

**Algorithm 5** weightedMatching

**Require:** A graph stream of a weighted graph \( G \).

**Ensure:** A \( \frac{1}{6} \)-approximation of a maximum matching.

1: \( M \leftarrow \emptyset \)  \hspace{1cm} \triangleright \text{maintain a matching at all time}
2: for every streamed edge \( e \) do
3: if \( w(e) > 2w_{\text{adj}}(e) \) then
4: remove the adjacent edges from \( M \) and add \( e \)
5: end if
6: end for

Note that removing an edge is final and can’t be reversed, even if both endpoints would be free with regard to the final matching. So an edge has to not only account for the edges it directly removes, but all edges that were removed by them and so on. Thus \( w(e) \) must be at least twice \( w_{\text{adj}}(e) \).

**Theorem 2.** In one pass and \( O(n \log n) \) storage, the algorithm constructs a \( \frac{1}{6} \)-approximation of a maximum matching.

**Proof.** For any set of edges \( A \), let \( w(A) = \sum_{e \in A} w(e) \).

An edge is called born, if it is ever part of \( M \), killed if it was born, but afterwards removed from \( M \). The removed edge is murdered by the new one. If an edge is born, but never killed, it’s called a survivor. Let \( S \) be the set of survivors. Since all edges in \( M \) have to be born, but never be killed \( S = M \), and therefore the quality of this matching is \( w(S) \).

Let the trail of dead of an edge \( e \) be \( T(e) = \bigcup_{i \in \mathbb{N}} C_i \), with \( C_i = \bigcup_{e' \in C_{i-1}} \{\text{the edges murdered by } e'\} \) and \( C_0 = \{e\} \). So \( C_1 \) is the set of edges murdered by \( e \), \( C_2 \) the of edges murdered by these edges and so on.
Claim 1: \( w(T(e)) \leq w(e) \)

**Proof of claim:** Every edge has at most one murderer and by the algorithm \( w(e) > 2w_{ad}(e) \), so an edge has at least twice the weight of the edges it murdered, so \( 2w(C_{i+1}) \leq w(C_i) \).

\[
2w(T(e)) = \sum_{i \geq 1} 2w(C_i) \leq \sum_{i \leq 0} w(C_i) = w(C_0) + \sum_{i \geq 1} w(C_i) = w(e) + w(T(e)),
\]

which implies the claim.

Now consider a maximum solution \( OPT = \{o_1, o_2, ...\} \). The weight of these edges is distributed to the survivors of the algorithm.

First, and edge \( e \) is accountable to \( o \in OPT \), if \( e = o \) or \( o \) was never born because of \( e \). In the second case two edges might be accountable to \( o \). If only one edge is accountable to \( o \), its whole weight \( w(o) \) is charged to \( e \), otherwise let \( e_1, e_2 \) be the two edges accountable to \( o \). In this case \( e_1 \) is charged with \( \frac{w(o)w(e_1)}{w(e_1) + w(e_2)} \) and \( e_2 \) with \( \frac{w(o)w(e_2)}{w(e_1) + w(e_2)} \). Since in this case

\[
w(o) < 2(w(e_1) + w(e_2)) \iff \frac{w(o)}{2} < w(e_1) + w(e_2),
\]

it holds that

\[
\frac{w(o)w(e_1)}{w(e_1) + w(e_2)} > \frac{w(o)w(e_1)}{\frac{w(o)}{2}} = 2 \cdot \frac{w(o)w(e_1)}{w(o)} = 2w(e_1).
\]

\( e_1 \) is charged at most twice its own weight. \( e_2 \) is analogous. For the case that only one edge is accountable \( e \) is also charged at most twice it’s weight since either \( o = e \) or we have \( w(o) < 2w(e) \) and \( o \) was never born because of only one edge. So any edge is charged at most twice it’s own weight by a single charge.

Note that an edge has an endpoint in common with every edge it murders and with every edge it gets charged by. So an edge can be charged by two edges from \( OPT \), one for every endpoint. Because the calculations are easier if all edges in a trail of dead are charged by only one edge in \( OPT \), the charges are redistributed as follows. For distinct \( u_1, u_2, u_3 \) and a \( v \in V \); if \( e' = (u_1, v) \) gets charged by \( (u_2, v) \) and afterwards murdered by \( e = (u_3, v) \) the charge is transferred from \( e' \) to \( e \). Edges are still charged at most twice their own weight since \( w(e) \geq w(e') \).

Although the edges in trail of death are only charged once, survivors still can be charged twice. Thus, for the set of survivors \( S \),

\[
w(OPT) \leq \sum_{e \in S} 2w(T(e)) + 2w(e) = \sum_{e \in S} 2w(T(e)) + 4w(e) = \sum_{e \in S} 2w(T(e)) + 4w(e) = 6w(S)
\]

Therefore the set of survivors and such the matching found by the algorithm is a \( \frac{1}{6} \)-approximation.

\( \square \)

Lately, a \((1+\epsilon)\) approximation in \( O((\frac{1}{\epsilon})^5) \) passes was introduced for the bipartite, unweighted graphs [3] and an algorithm for the weighted case that achieves in one pass an approximation ratio of 5.58 [7] (compared to the 6, that is presented here).

### 3.3 Lower Bounds

According to Lemma 1, deciding whether there is a directed path between two fixed vertices \( s, t \) requires \( O(m) \) bits of storage. So if "s-t-connectivity" \( \leq P " \text{find augmenting path}" \), it can be inferred that the matching algorithm is impossible in the classic streaming model, since finding an augmenting path would also take \( \Omega(m) \) bits of storage. Note that an augmenting path does not have fixed length, otherwise the definition is similar to that of length-3 augmenting paths (cf. Figure 1).
Theorem 3. **"k-t-connectivity"** \(\leq_p "\text{find augmenting path}"\)

**Proof.** Without loss of generality, let \(G = (\{v_1 = s, v_2, ..., v_{n-1}, v_n = t\}, E)\) be a directed graph. Construct the undirected graph \(G' = (V', E')\) with \(V' = \{v_i | v_i \in V\} \cup \{v_{ir} | v_i \in V\} \cup \{v_s, v_t\}\) and \(E' = \{(v_{il}, v_{ir}) | v_i \in V\} \cup \{(v_{ir}, v_{jl}) | (v_i, v_j) \in E\} \cup \{(v_s, v_{il}), (v_t, v_{nr})\}.\) The initial matching is \(M = \{(v_{il}, v_{ir}) | v_i \in V\}.\)

Since all edges \((v_{il}, v_{ir})\) are part of the matching and every other edge except \((v_s, v_{il}), (v_t, v_{nr})\) is of the form \((v_{ir}, v_{jl})\), the only way to find an augmenting path, i.e. a path that contains more edges from \(E \setminus M\) than \(M\), is for \((v_s, v_{il})\) and \((v_t, v_{nr})\) to be part of this path. Due to the construction of \(G'\), \(v_{il}\) and \(v_{nr}\), are connected exactly if the is a path from \(s\) to \(t\) in \(G\).

Thus, there is an augmenting path in \(G'\) exactly if there is a path from \(s\) to \(t\) in \(G\). \(\square\)

4 Distances and further problems

This section gives an overview for results regarding distances like shortest paths and the diameter in the semi-streaming model. Notably, a lower bound for approximations in one pass will be presented.

**Definition 5.** An edge \((u, v)\) of a graph is called \(k\)-critical if the shortest path from \(u\) to \(v\) in \((V, E \setminus \{(u, v)\})\) has length \(\geq k\).

The next Lemma will show the existence of a graph with useful properties. This will be done by using the probabilistic method, i.e. by showing that a random graph has these properties with positive probability and therefore one such graph must exist.

**Lemma 6.** For \(0 < \epsilon < 1\) and sufficiently large \(n\) there exists a Graph \(G = (V, E)\) with \(|V| = n, |E| = 2^{\log_2 - \epsilon} n/4\) such that more than half of the edges are \(k\)-critical with \(k = \frac{\log_2 - \epsilon}{2} n\) and more than half subgraphs of the set

\[\{G' \subseteq G | G' is formed by deleting a subset of the k-critical edges\}\]

have diameter less than or equal to \(8k = 4\log_2 - \epsilon\) \(n\).

**Proof.** Consider a random graph \(G \in \mathcal{G}_{n, p}\), i.e. a random graph with \(n\) vertices, where the probability that any fixed edge exists is \(p\), so the appearance of any edge is independent of every other edge. Let \(p = 2^\log_2 - \epsilon / n\). It is intuitively clear and can be shown by using the Chernoff bound, that, with high probability, the number of edges in \(G\) is at least \(2^\log_2 - \epsilon n/4\), which is 1/2 of the expectancy, since there are \(n^2/2\) many possible edges.

**Claim 1:** With high probability, the majority of edges in \(G\) is \(k\)-critical.

**Proof of Claim 1:** By using the Chernoff bound again, for any vertex \(v\), \(\Pr[d(v) \geq 2 \cdot 2^\log_2 - \epsilon n] \leq (e/4)^2^\log_2 - \epsilon\), where \(d(v)\) is the degree of \(v\). Using the union bound, the probability that any vertex has a degree that large is at most \(\sum_{v \in V} \Pr[d(v) \geq 2 \cdot 2^\log_2 - \epsilon n] \leq n \cdot (e/4)^2^\log_2 - \epsilon\).

For sufficiently large \(n\) it holds that \(2^\log_2 - \epsilon n \geq \log_2 \cdot 2n \geq 1\) and not that \((e/4) < 1\). Thus

\[n \cdot (e/4)^2^\log_2 - \epsilon n \leq n \cdot (e/4)^{(\log_2 - \epsilon) n} = \frac{1}{n^{\log_2 - \epsilon}}.\]

Therefore, the probability that no vertex has degree larger than \(2 \cdot 2^\log_2 - \epsilon n\) is at least \(1 - \frac{1}{n^{\log_2 - \epsilon}}\) and it is assumed that this is always the case. This implies that number of vertices that have distance \(i\) from any vertex \(v\) is at most \(2^\log_2 - \epsilon n^i\).

Consider any edge \((u, v)\) in \(G = (V, E)\). Let \(I_i(v)\) be the set of vertices with distance at most \(i\) from \(v\) in \((V, E \setminus \{(u, v)\})\), so
\[
|\Gamma_k(v)| \leq \sum_{0 \leq i \leq k} (2 \cdot 2^\log^* n)^i \\
= 2 \cdot (2^\log^* n)^{k+1} = 2 \cdot (2^\log^* n)^{\log_{1/2} \epsilon n + 1} \\
= 2 \cdot (2^\log^* n)^{\log_{1/2} \epsilon n + \log^* n} \\
= 2 \cdot (2^{\log^* n} + \log^* n) \\
= 2 \log^* n + \log^* n + 1.
\]

This is smaller than \(2^{2\log n/3}\), because

\[
2^{\log^* n + \log^* n + 1} \\
\leq 2^{2\log n/3} \\
\iff \frac{\log n}{2} + \log^* n + 1 \\
\leq \frac{2}{3} \log n \\
\iff \log^* n + 1 \\
\leq \frac{1}{3} \log n
\]

holds for sufficiently large \(n\). In a random graph, the vertex \(u\) is selected uniformly at random with probability \(p\) from \(V \setminus \{v\}\), so the probability that \(u\) is in \(\Gamma_k(v)\) is \(\frac{2^{2(\log n/3)}}{n}\). Since \((u, v)\) is \(k\)-critical exactly if \(u\) is not in \(\Gamma_k(v)\), the probability for it being \(k\)-critical is

\[
1 - \frac{2^{2(\log n/3)}}{n} = 1 - \frac{1}{n^{1/3}}.
\]

Using the Chernoff bound again shows that, with high probability, the majority of edges in \(G\) is \(k\)-critical. □

**Claim 2:** The diameter of a random graph \(G \in \mathcal{G}_{n,p/2}\) is, with high probability, less than \(D = 4 \log^{1-\epsilon} n\)

**Proof of Claim 2:** Consider any node \(v \in V\). Let \(S_i = \Gamma_i(v) \setminus \Gamma_{i-1}(v)\). For those \(i\) with \(\Gamma_i(v) < n/2\), the Chernoff bound gives that, with high probability, \(|S_{i+1}| > |S_i| \cdot 2^{\log^* n}/4\). Now consider the first \(t\), such that \(\Gamma_t(v) \geq n/2\). Since \(|\Gamma_1(v)| = \sum_{i=1}^{t} |S_i|\) and the above bound, \(|S_i| > n/4\) and with high probability \(|\Gamma_{t+1}| = n\). Solving this for \(t\) gives

\[
t + 1 \leq \frac{\log n}{\log^* n - 2} < 2 \log^{1-\epsilon} n.
\]

This is the maximum distance from \(v\) to every other node, so the diameter must be smaller than \(4 \log^{1-\epsilon} n\). □

Using these two claims, the actual lemma can be proven. Let \(G \in \mathcal{G}_{n,p}\). Picking a random subgraph from \(G\) is the same as picking a Graph from \(\mathcal{G}_{n,p}/2\). Note that any edge could be removed from \(G\), not only those that are \(k\)-critical. Consider the events

- \(A = \{ \ G = (V, E) \in \mathcal{G}_{n,p} \ \text{majority of edges is} \ k\text{-critical} \}\)
- \(B = \{ \ G = (V, E) \in \mathcal{G}_{n,p}/2 \ \text{diameter of} \ G \text{is less than} \ D \}\)
- \(B_H = \{ \text{for subgraphs} \ H' \text{of} \ H \text{the diameter of} \ H' \text{is less than} \ d \}\)
The former argumentation shows that $\Pr[B]$ is high and that $\Pr[A]$ is very low. Thus, the probability of $A$ and $B$ happening at the same time is

$$\Pr[A \cap B] \geq \Pr[B] - \Pr[A] > 1/2.$$  

Also,

$$\Pr[A \cap B] = \sum_{G} I[A] \Pr[G] \Pr[B_G].$$

With a simple averaging argument, it follows that there is a $G \in A$ with $\Pr[B_G] > 1/2$. This encompasses all subgraphs of $G$, not only those, where only $k$-critical are deleted. But since adding edges can not increase the diameter of graph, re-adding the non-$k$-critical edges does not change this. So the majority of the subgraphs of $G$ formed by deleting $k$-critical edges has diameter $< D$.

The following theorem can be proven by using this graph $G$ and his subgraphs.

**Theorem 4.** For $0 < \epsilon < 1$, 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 with the properties of Lemma 6 and $\eta = \frac{n-2D}{D} = \frac{n}{D} - 2$ vertices. Let $\mathcal{F}(G)$ be the set of subgraphs of $G$ with diameter less than $D$. Note that

$$|\mathcal{F}(G)| \geq 2^{2^{\log^{1-\epsilon} n/8}} = 2^{\omega(n \log n)}.$$  

So there are more graphs than possible memory configurations in the semi-streaming model, so, for any given algorithm, there are two graphs $G', G'' \in \mathcal{F}(G)$ that are indistinguishable by that algorithm. So when these graphs are streamed to this algorithm the exists an edge in those graphs which existence is undetermined at the end of the algorithm. Consider $D$ of those graphs $G_1, ..., G_D$, and let $e_i = (u_{i1}, u_{i2})$ be an edge whose existence is undetermined in a stream of $G_i$.

The stream that is actually streamed to the algorithm consists of first the graphs $G_1, ..., G_D$, then edges of the form $(u_{ir}, u_{i(i+1)r})$ for $i = 1, 2, ..., D - 1$ and finally two new paths of length $D$ each, one with endpoint $s$ and $u_{i1}$ and the other with endpoints $t$ and $u_{Dr}$. See figure 2 for the final graph. Since no path in the subgraphs $G_1, ..., G_D$ has length $D$ or longer, the diameter of the graph is the length of the path between $s$ and $t$. This path has length $4D - 1$, where $2D$ comes from the two added paths, plus $D$ from the undetermined edges and $D - 1$ from the edges connecting the subgraphs. Since all the undetermined edges might have been $k'$-critical, with $k' = \frac{\log^{1-\epsilon} n}{2}$, the minimum diameter the algorithm can ensure is $3D - 1 + k'D$. This is a $\Omega(k')$-approximation. If $k' \in \Theta(k)$, the Theorem follows. Observe that

$$k' = \frac{\log^{1-\epsilon} n}{2} = \frac{\log^{1-\epsilon} n - 2}{2} = \frac{\log^{1-\epsilon} n - 2}{4} = \Theta(\log^{1-\epsilon} n)$$

and

$$\log^{1-\epsilon} \frac{n}{4} = \log^{1-\epsilon} n - \log^{1-\epsilon} 4 \log^{1-\epsilon} n = \Theta(\log^{1-\epsilon} n) = \Theta(k),$$

thus the theorem follows. □
Additionally, shortest paths can be approximated in one pass by building a spanner graph, especially a \( \log n / \log \log n \)-spanner can be constructed for unweighted graphs by an algorithm similar to the one described in [1]. The algorithm constructs this spanner \( S \) by adding the streamed edges to \( S \), unless this would create a circle of length \( \log n / \log \log n \). The weighted case is more complicated, since the edges would have to be sorted according to their weight, which is at least very difficult in the streaming model.

Instead, there is an algorithm that groups the edges into sets of edges with similar weight and uses the unweighted algorithm on these sets. If \( w_{\text{max}} \) is the maximum weight and \( w_{\text{min}} \) the minimum one, the range \([w_{\text{min}}, w_{\text{max}}]\) is divided into intervals \([(1+\epsilon)^i w_{\text{min}}, (1+\epsilon)^{i+1} w_{\text{min}}]\) and all edges with weight in this range are treated as if they had weight \((1+\epsilon)^i w_{\text{min}}\) and the unweighted algorithm is used on these. This leads to \( \log (1+\epsilon) \frac{w_{\text{max}}}{w_{\text{min}}} \) many spanners on the subset of edges. Note that if this is more than \( \text{polylog} \ n \), the storage space needed is too big and the algorithm can’t be used. This is independent of \( \epsilon \), so some graphs can’t be handled in the semi-streaming model. The union of the single spanners gives a \((1 - \epsilon) \log n\)-spanner. The algorithm does not need prior knowledge of the bounds of the edge weights and can operate on the maximum and minimum value yet. To summarize:

**Theorem 5.** For \( \epsilon > 0 \), and a weighted undirected graph on \( n \) vertices with maximum edge weight \( w_{\text{max}} \) and minimum \( w_{\text{min}} \), where \( \log (1+\epsilon) \frac{w_{\text{max}}}{w_{\text{min}}} = \text{polylog} \ n \), there is a semi-streaming algorithm that calculates a \((1 + \epsilon) \log n\)-spanner of the graph in one pass. It needs \( O(\log (1+\epsilon) \frac{w_{\text{max}}}{w_{\text{min}}} \cdot n \log n) \) bits of space and at most \( O(\log (1+\epsilon) \frac{w_{\text{max}}}{w_{\text{min}}} \cdot n) \) time per edge.

This spanner can be used to approximate shortest paths and the diameter of the graph. Also it can be used to find an approximation of the girth, which is the shortest circle in the graph. If the girth is larger than \( k \), it can be determined exactly in a \( k \)-spanner, this spanner gives a \( \log n / \log \log n \)-approximation.

Some other graph problems, that are solvable in the semi-streaming model are the construction of a minimum spanning tree and testing for planarity. Since the number of edges in a spanning tree and in a planar graph are linear in the number of nodes, there are offline algorithms, that can be adapted without going over the storage bound. Note that at least the planarity test is impossible in the classic streaming model.

Lastly, the next algorithm finds articulation points (i.e. a vertex, such that the induced graph wouldn’t be connected). It uses a disjoint set data structure for every vertex \( v \) to keep all neighbors of \( v \), that are in the same connected component of the graph without \( v \), in the same set.
Algorithm 6 findArticulationPoints

Require: A graph stream of a unweighted, connected graph \( G \).
Ensure: The set of articulation points in \( G \).

1. \( T = (V, \emptyset) \)
2. for each \( v \in V \) do
3. \( \text{SF.makeSet}(v) \)
4. end for
5. for each streamed edge \((u, v)\) do
6. if \( \text{SF.findSet}(u) = \text{SF.findSet}(v) \) then
7. find the path \( u = a_0, a_1, ..., a_k = v \) in \( T \)
8. for each \( a_i, 1 < i < k - 1 \) do
9. \( a_i.union(a_{i-1}, a_{i+1}) \)
10. end for
11. else
12. \( \text{SF.union}(u, v) \)
13. \( T = T \cup \{(u, v)\} \)
14. \( u.makeSet(v) \)
15. \( v.makeSet(u) \)
16. end if
17. end for
18. for each \( v \in V \) do
19. if the neighbor of \( v \) w.r.t. \( T \) are in different sets then
20. output \( v \) as an articulation point
21. end if
22. end for

If \( v \) is an articulation point then there are two neighbors that have no other connection. Therefore their set in the data structure would never be unioned and the algorithm outputs \( v \).

5 Conclusion

The semi-streaming model allows a lot of new possibilities while still keeping the core idea of the streaming model. The multiple passes might not be suitable for a client-server model, but the pagewise reading of data is often still more efficient than random access. Interesting further research would be the tradeoff between the number of passes, the storage space, the per-edge time and the approximation factor.

References