Edge Coloring in W-Streams

Vikrant Ashvinkumar, Justin Kim, Michael Wang

December 14, 2021

Abstract

Edge coloring is a well-known problem that asks for an assignment of colors to the edges of a graph such that no two edges incident on the same vertex share a color. The goal in coloring the edges of a graph is to minimize the total number of colors used. In this paper, we attempt to improve on the number of colors used for edge coloring in the graph streaming model, and explore future improvements in lower and upper bounds for this problem.

1 Introduction

The edge coloring problem takes an input of a graph $G = (V, E)$ and returns a color for every edge such that every edge that meets at a vertex has a different color. The goal of this problem is to minimize the total number of colors used to color the graph.

In the classical model, Vizing's theorem [5] states that $\Delta + 1$ colors is sufficient to color any graph, where Δ is equal to the maximum degree in the graph, and it is easy to find this coloring [1]. However, in the graph semi-streaming model, the best known algorithm can only obtain a $O(\Delta^2)$ coloring or $O(\Delta^2/s)$ in $O(ns)$ space [2]. The added difficulty in the streaming model occurs because most algorithms rely on access to the neighbors of every vertex, and that idea approach does not work when edges can appear in the stream in any adversarial order. In this paper, we will explore possible improvements for edge coloring in the semi-streaming model.

1.1 The Model

The algorithm must output a color for every edge, so the output will not be able to fit in $\tilde{O}(n)$ space. For this reason, the standard streaming model is modified to also include a write-only output stream, and this is referred to as the "W-Streaming" model [3]. This restriction makes edge coloring difficult because we must color nearly the entire graph before reaching the end of the stream. For these last edges, it must color them without remembering which colors it chose for the other edges in the graph.

We will assume that the edges can arrive in any adversarial order, and we want to create an algorithm that achieves a good coloring with high probability.

2 Background

Behnezhad et al. [2] described a framework that, when given an algorithm that colors a bipartite graph, can also color general graphs in an asymptotically equivalent number of colors. First, we will describe the algorithm that colors a bipartite graph, and then explain how it is generalized to every graph.

2.1 Coloring Bipartite Graphs

First, here is a simple algorithm that achieves a valid coloring of a bipartite graph using at most $O(\Delta^2)$ colors. Here, the edges can arrive in any adversarial order, and ordered tuples are used to represent each color.

Algorithm 1 Edge coloring for bipartite graphs

```
1: Color $G = (L, R, E)$ with vertex partitions $L$ and $R$ and edges $E$. Uses $O(\Delta^2)$ colors, given by
   ordered tuples.
2: for every vertex $v$ do
3:     $c_v \leftarrow 0$
4: end for
5: for every edge $e = (u, v)$ in the stream, with $u \in L$ and $v \in R$ do
6:     Assign $(c_u, c_v)$ to $e$
7:     $c_u, c_v \leftarrow c_u + 1, c_v + 1$
8: end for
```

Theorem 1. *Given a bipartite graph $G = (L, R, E)$, this algorithm will return a valid edge coloring using $O(\Delta^2)$ colors and $\tilde{O}(n)$ memory.*

The proof of correctness follows from the observation that for some vertex $u \in L$, for every edge (u, v) with color (c_u, c_v), the value for c_u will be different for every edge. This is because we increment c_u by 1 for every edge that includes vertex u. This fact is identical for every $v \in R$ as well, meaning that the coloring will always be valid.

The total number of colors is no more than Δ^2 because a vertex will see at most Δ edges. This means that c_u and c_v can only take on a value of at most Δ, which means that there are Δ^2 combinations of colors.

Lastly, note that the algorithm only maintains n counters, and each take up $O(\log n)$ bits of space. Therefore, this algorithm uses $O(n \log n)$ bits of space. □

Note that this algorithm is deterministic and will return a Δ^2 coloring in the worst case. For the following section, we will use randomization to apply this algorithm to general graphs. When randomization is involved, we assume that the adversary is oblivious to the randomness chosen by the algorithm, so the order of the edges must be fixed before the algorithm begins execution.

2.2 Coloring General Graphs

The edges in a graph G can be divided among $O(\log n)$ bipartite graphs as follows:

For each vertex v, generate a sequence r_v of $\log n$ random bits. This will define $\log n$ bipartite graphs $B_i = (L_i, R_i, E_i)$ induced by partitions where L_i is the set of vertices with $r_{v,i} = 0$ and R_i is the set of vertices with $r_{v,i} = 1$. Each edge $e = (u, v)$ will belong to the bipartite graph with the lowest index such that $r_{u,i} \neq r_{v,i}$.

Theorem 2. *Given any graph $G = (V, E)$, this algorithm will return a valid edge coloring using $O(\Delta^2)$ colors and $\tilde{O}(n)$ memory.*

In this algorithm, colors are represented by ordered triples, and the third number in that triple is the index of the bipartite graph that the edge belongs to. Within each bipartite graph, we color the edges using Algorithm 1, which produces a valid coloring that uses Δ_i^2 unique colors, where Δ_i

Algorithm 2 Edge coloring for general graphs

```
Color $G = (V, E)$. Uses $O(\Delta^2)$ colors, given by ordered tuples.
for every vertex $v$ do
    $r_v \leftarrow$ a string of $\log n$ random bits
    for every $i \in [\log n]$ do
        $c_{v,i} \leftarrow 0$
    end for
    $c_v \leftarrow 0$
end for
for every edge $e = (u, v)$ in the stream do
    Let $i$ be the lowest index such that $r_{u,i} \neq r_{v,i}$. Say that $r_{u,i} = 0$ and $r_{v,i} = 1$.
    if $\Delta 2^{-i} < \log n$ then
        Store edge $e$
    else
        Assign $(c_u, c_v, i)$ to $e$
        $c_u, c_v \leftarrow c_u + 1, c_v + 1$
    end if
end for
Color the stored edges using a new set of colors.
```

is the maximum degree of bipartite graph B_i. For each graph the algorithm maintains n counters, so in total there are $n \log n$ counters that fit in $\tilde{O}(n)$ space.

The graph also stores some edges, and we want to show that these edges can fit in $\tilde{O}(n)$ space. Let $d_{u,v}$ be equal to the first index for which $r_{u,i} \neq r_{v,i}$, and let X_v be the number of edges that are stored incident on v. We will store an edge (u, v) if $\Delta 2^{-d_{u,v}} < \log n$. We want to show that $\mathbb{E}[X] = O(\log n)$, and then take the union over all vertices to show $O(n \log n)$ edges are stored in total. By linearity of expectation:

$$\mathbb{E}[X_v] = \sum_{i \in [\Delta]} \Pr\left(\Delta 2^{-d_{u_i,v}} < \log n\right)$$

Note that $d_{u_i,v} > \log\left(\frac{\Delta}{\log n}\right)$ if the first $\log\left(\frac{\Delta}{\log n}\right)$ bits of r_{u_i} and r_v are equal, and this occurs with probability $\frac{\log n}{\Delta}$. Therefore, because the random bits are chosen independently, we get that

$$\mathbb{E}[X_v] = \Delta \cdot \frac{\log n}{\Delta} = \log n$$

Applying a simple Chernoff bound, we can get that $\Pr(X_v > c \cdot \mathbb{E}[X_v]) < \frac{1}{\text{poly}(n)}$, meaning that

$$\Pr\left(\bigcup_{\forall v} X_v > c \cdot \mathbb{E}[X_v]\right) < \frac{1}{\text{poly}(n)}$$

With high probability, the number of stored edges is $O(n \log n)$.

Now, all that remains is to show that this algorithm uses $O(\Delta^2)$ colors. For some vertex v, and we want to consider the upper bound for each of its counters $c_{v,i}$. Let X_i be the random variable denoting the number of neighbors of v in graph B_i. At the end of the stream, $c_{v,i} = X_i$, and by bounding X_i we can bound the number of colors used in the graph.

Observe that the probability of an edge existing on some bipartite graph i is 2^{-i}. Therefore, $\mathbb{E}[X_i] = \deg(v)2^{-i}$. By applying a Chernoff bound and taking the union over all graphs, we get that

$$\Pr\left(\bigcup_{i\in[\log n]} X_i > c \cdot \mathbb{E}[X_i]\right) < \frac{1}{\text{poly}(n)}$$

We can now take a union over vertex, and get that the maximum degree of bipartite graph B_i is $O(\Delta 2^{-i})$. If an algorithm that works on a bipartite graph uses $O(\Delta^2)$ colors, then for bipartite graph B_i it will use $O(\Delta^2 2^{-2i})$ colors. Therefore, the total number of colors used is:

$$\sum_{i\in[\log n]} O(\Delta^2 2^{-2i}) = O(\Delta^2)$$

This algorithm will return a valid coloring using $O(\Delta^2)$ colors and $\tilde{O}(n)$ memory. □

2.3 Generalization

Consider some algorithm ALG$(G = (L, R, E))$ that can produce a valid coloring of a bipartite graph with k colors using $\tilde{O}(n)$ memory, where k is some function of Δ. Here we show that it is possible to obtain an algorithm that can color any graph with $O(k)$ colors.

Algorithm 3 Algorithm that converts any bipartite graph algorithm

```
1: Color $G = (V, E)$. Uses $O(k^2)$ colors, where $k$ is how many colors ALG uses to color a bipartite
   graph in the worst case.
2: for every vertex $v$ do
3:     $r_v \leftarrow$ a string of $\log n$ random bits
4: end for
5: for every $i \in \log n$ in parallel do
6:     Initialize ALG$_i$ with an empty stream, and assign each a different color palette.
7:     for every edge $e = (u, v)$ in the stream do
8:         Compute the smallest $i$ such that $r_{u,i} \neq r_{v,i}$
9:         if $\Delta 2^{-i} > \log n$ then
10:            Store $e$ to color at the end
11:        else
12:            Stream $e$ to ALG$_i$
13:        end if
14:    end for
15: end for
16: Color the stored edges using a new set of colors.
```

Theorem 3. *If there exists an algorithm ALG that can output a valid coloring of a bipartite graph in k colors using $\tilde{O}(n)$ memory, then there exists a W-Streaming algorithm that can color any graph in $O(k)$ colors.*

Note that Algorithm 3 divides the edges into bipartite graphs exactly like Algorithm 2, which means that the concentration bounds for the maximum degree of each bipartite graph still holds. This means that with high probability, the maximum degree for ALG$_i$ will be, in expectation,

$O(2^{-i}\Delta)$. Therefore, for any linear or superlinear function k of Δ, the total number of colors used by each parallel algorithm is bounded by:

$$\sum_{i \in [\log n]} O(2^{-i}k) = O(k)$$

We can color the extra stored edges using $\Delta + 1$ colors, and assuming $s = \Omega(\Delta)$, the entire coloring uses $O(k)$ colors.

The correctness of this algorithm follow from the fact that the bipartite algorithm outputs a valid coloring. Because each algorithm uses a different palette of colors, the coloring is guaranteed to be valid.

Additionally, note that ALG uses $\tilde{O}(n)$ memory, and by running $\log n$ algorithms in parallel, we still only use $\tilde{O}(n)$ memory. □

3 Improved Upper Bounds

We first propose a simple deterministic algorithm that can color a graph in $O(\Delta^2/s)$ colors in $O(ns)$ space. By setting s to be polylog(n), we get a $O(\Delta^2/\text{polylog}(n))$ coloring in $\tilde{O}(n)$ space.

3.1 Trivial Improvement

Algorithm 4 Deterministic Graph Coloring

```
1: for every edge $e$ in the stream do
2:     Store edge $e$
3:     if $n \log n$ edges are stored then
4:         Color every edge with $\Delta + 1$ colors
5:         Erase all stored edges from memory, choose a new color palette
6:     end if
7: end for
```

Theorem 4. *Given any graph $G = (V, E)$, Algorithm 4 will return a proper edge coloring using $O(\Delta^2/\log(n))$ colors and $\tilde{O}(n)$ memory.*

A graph with maximum degree Δ has at most $n\Delta/2$ edges. Algorithm 4 thus $\Delta+1$ colors at most $\frac{n\Delta/2}{n \log n}$ chunks, each with a distinct color palette. That is to say, $(\Delta+1)(\Delta/(2\log n)) = O(\Delta^2/\log n)$ colors are used. □

Note that Algorithm 4 does not even need to know Δ, nor does it need any clever idea to overcome the lack of knowledge of Δ; for each chunk of $n \log n$ edges colored, just color it with $\Delta' + 1$ colors where Δ' is the maximum degree of the graph induced by that chunk.

3.2 An Attempt at Pushing the Trivial Improvement Further

We now present an attempt at improving upon Algorithm 4. While it comes short of any improvement, there are some ideas herein which may be worth pursuing further.

We still follow the general framework of accumulating a chunk of edges and coloring the whole chunk before moving on to accumulating the next chunk and so on and so on. What changes is that we will accumulate three different kinds of chunks, and color any chunk of edges when it gets too large before promptly clearing the colored chunk from memory. What are the kinds of chunks? They are named

1. Low-Low;
2. Low-High;
3. High-High;

where Low-Low are the edges whose endpoints have a low current degree, Low-High are the edges who have one endpoint with a low current degree and the other with a high current degree, and High-High are the edges whose endpoints have a high current degree. More formally, for some fixed time and denoting the edges in memory at that time with $LL \sqcup LH \sqcup HH$ (Low-Low, Low-High, High-High respectively), the current degree of a vertex u is the degree of u in $G(V, LL \sqcup LH \sqcup HH)$. u's current degree is said to be low when it is less than $\Delta^{2/3}$, and is said to be high when it is at least $\Delta^{2/3}$. Note that unlike Algorithm 4, foreknowledge of Δ is required for now.

When any chunk of edges is to be colored, we color it in a way that depends on its kind. Modulo these coloring subroutines, our algorithm is described by Algorithm 5. It now remains to describe **LL,LH,HH-color** referred to therein.

3.2.1 LL-coloring

$G(V, LL)$ has degree $\Delta^{2/3}$. We can thus simply do a $\Delta^{2/3} + 1$ coloring on this using a fresh color palette. As there can be no more than $\Delta/\log n$ LL chunks to color, we exhaust $O(\Delta^{5/3}/\log n)$ colors from LL-coloring.

3.2.2 LH-coloring

We do the same thing as LL-coloring, but with $\Delta+1$ colors. LH-coloring thus exhausts $O(\Delta^2/\log n)$ colors.

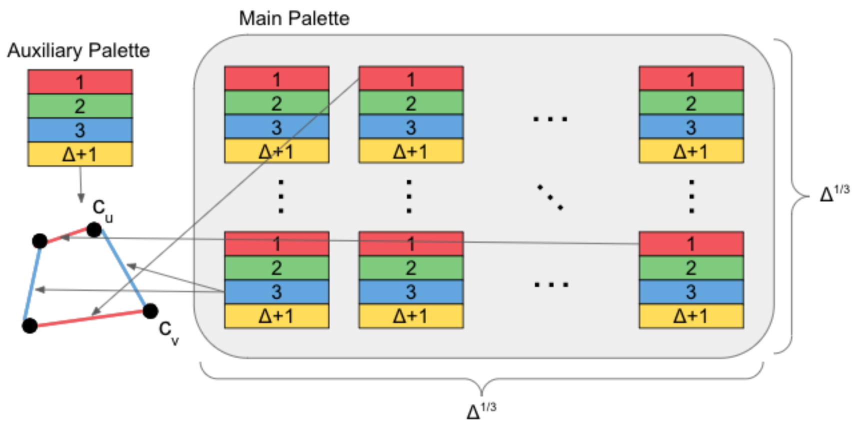
3.2.3 HH-coloring

To describe the gist of HH-coloring, let us assume the graph is bipartite, and that we also know the bipartition (e.g. vertices from $[n/2]$ are in one part and the rest are in the other). HH-coloring uses a main palette of size $O(\Delta^{5/3})$, which each HH chunk will share throughout the lifetime of the algorithm. We thus need a way to carefully choose which colors to use when coloring each chunk. To assist in this choice, there will be counters on each vertex which we increment (differently from Algorithm 1), and also an auxiliary palette. Colors from this auxiliary palette are never output. The main palette has colors of the form (i, j, k) where $i \in [\Delta+1]$ and $j, k \in [\Delta^{1/3}]$, and the auxiliary palette has colors of the form $i \in [\Delta + 1]$.

When a HH chunk of size $n \log n$ is to be colored, we use the auxiliary palette to $\Delta + 1$ color the edges in this chunk; we are not done yet as we need to convert these colors, which come from the auxiliary palette, into colors from the main palette. Let us suppose uv (where u is in the first part and v is in the second part of the bipartition) is colored i from the auxiliary palette. We will recolor uv with (i, c_u, c_v) from the main palette; which we then output. Once the whole chunk has been recolored, we will increase the counters c_u of all vertices u that have been involved in this coloring. That is to say, we will increase the counters for all non-isolated vertices in $G(V, HH)$. Note that since $d_u \geq \Delta^{2/3}$, it follows that c_u cannot increase to anything more than $\Delta^{1/3}$ and so we can always pick a color from the main palette.

Algorithm 5 LL-LH-HH Graph Coloring

```
 1: $LL \leftarrow \emptyset$
 2: $LH \leftarrow \emptyset$
 3: $HH \leftarrow \emptyset$
 4: **for** every vertex $v$ **do**
 5:     $d_v = 0$
 6: **end for**
 7: **for** every edge $uv$ in the stream **do**
 8:     $d_u, d_v \leftarrow d_u + 1, d_v + 1$
 9:     $E \leftarrow LL \sqcup LH \sqcup HH \sqcup \{uv\}$
10:     $LL \leftarrow \{u'v' \in E | d_{u'}, d_{v'} < \Delta^{2/3}\}$
11:     $LH \leftarrow \{u'v' \in E | d_{u'} \geq \Delta^{2/3}, d_{v'} < \Delta^{2/3}\}$
12:     $HH \leftarrow \{u'v' \in E | d_{u'}, d_{v'} \geq \Delta^{2/3}\}$
13:     **if** $|LL| > n \log n$ **then**
14:         **LL-color** $LL$
15:         **for** every edge $u'v' \in LL$ **do**
16:             $d_{u'}, d_{v'} \leftarrow d_{u'} - 1, d_{v'} - 1$
17:         **end for**
18:         $LL \leftarrow \emptyset$
19:     **end if**
20:     **if** $|LH| > n \log n$ **then**
21:         **LH-color** $LH$
22:         **for** every edge $u'v' \in LH$ **do**
23:             $d_{u'}, d_{v'} \leftarrow d_{u'} - 1, d_{v'} - 1$
24:         **end for**
25:         $LH \leftarrow \emptyset$
26:     **end if**
27:     **if** $|HH| > n \log n$ **then**
28:         **HH-color** $HH$
29:         **for** every edge $u'v' \in HH$ **do**
30:             $d_{u'}, d_{v'} \leftarrow d_{u'} - 1, d_{v'} - 1$
31:         **end for**
32:         $HH \leftarrow \emptyset$
33:     **end if**
34: **end for**
```



The overall approach falls short of an improvement of the $O(\Delta^2)$ color bound as we do not know yet how to efficiently LH-color. The trouble with finding out how to efficiently LH-color, however, suggests the following simple instance for which, if we know a procedure to efficiently color, may guide us towards a more general and efficient solution:

Consider $\sqrt{n}$ copies of $K_{\sqrt{n},\sqrt{n}}$ in the setting where the chunk sizes go only up to n before they're colored. The adversary then streams a maximum star from every copy of $K_{\sqrt{n},\sqrt{n}}$, then another maximum star from every copy of $K_{\sqrt{n},\sqrt{n}}$, and so on and so forth until all edges have been streamed. This instance will result in all $\Delta/2$ chunks being LH-colored.

3.3 Another Possible Approach

Moving away from the framework of coloring large chunks of edges, we seek inspiration from Algorithm 6 found in [2], which performs well in the random edge-arrival setting.

Algorithm 6 Random Edge Arrival Graph Coloring

```
1: for every vertex $v$ do
2:     $c_v = 0$
3: end for
4: for every edge $uv$ in the stream do
5:     color $uv$ with $\max(c_u, c_v)$
6:     $c_u, c_v = \max(c_u, c_v) + 1$
7: end for
```

How well does Algorithm 6 perform? With high probability, if the edges arrive in random order, it outputs an $O(\Delta)$ coloring. That this algorithm performs poorly in the adversarial edge-arrival setting can be seen from the following example: consider a P_n, the path on n vertices, where edges arrive in the order $(1,2),(2,3),\ldots,(n-1,n)$. Algorithm 6 would output $n-1$ colors when only two suffices!

This is not well explored by us, but here is how we may improve the algorithm: we can modify Algorithm 6 by adding two counters to each vertex and picking them randomly when coloring an edge. Each choice of counter-pairs uses a different palette.

Algorithm 7 immediately performs better on the bad example P_n as the longest run of $i \leftarrow 0$ or $i \leftarrow 1$ is concentrated around $\ln n$ (see [4]). Algorithm 7 thus $\tilde{O}(\Delta)$ colors the path (we are hiding this $\ln n$ multiplicative factor). Is fixing path inputs sufficient? We are not yet sure, but we are quite sure there will need to be more work done here to fully flesh this idea out.

Algorithm 7 Random Edge Arrival Graph Coloring with Multi-Counters

```
1: for every vertex $v$ and $i \in \{0,1\}$ do
2:     $c_{v,i} = 0$
3: end for
4: for every edge $uv$ in the stream do
5:     $i \leftarrow 0$ with probability $1/2$ and 1 otherwise
6:     color $uv$ with $(i, \max(c_{u,i}, c_{v,i}))$
7:     $c_{u,i}, c_{v,i} = \max(c_{u,i}, c_{v,i}) + 1$
8: end for
```

The analysis of Algorithm 6 in [2] uses the line graph $L(G)$ of the input graph G. They bound the longest monotone path in $L(G)$, which is to say path in $L(G)$ whose vertices, in the order of the path, match the relative order of edge arrivals. In their setting where edges of G arrive randomly, vertices of $L(G)$ arrive randomly. In the setting we're concerned about, on the other hand, the edges of G (thus vertices of $L(G)$) arrive in an adversarial order; however, the vertices of $L(G)$ are randomly colored one of two colors which the adversary is not aware of. The question now is, how long can monochromatic monotone paths in $L(G)$ be?

4 Lower Bounds

In the classical setting, it is well-known that finding a Δ coloring of a graph is NP-Hard. In the W-Streaming model, this is the best current lower bound, with the best upper bound being $O(\Delta^2/\log n)$. Many existing streaming lower bounds make use of reductions from communication complexity.

In these reductions, Alice and Bob have two sections of the stream, and Alice must send Bob a message that will allow Bob to output the correct answer. We can prove a lower bound for a streaming algorithm by stating that if Alice simply executes the streaming algorithm on her part of the input, then sends her current internal state to Bob, Bob should be able to obtain the correct answer by finishing the streaming algorithm on his input.

However, this reduction does not fit the W-Streaming model exactly, as Alice would be able to output part of the answer before sending her internal state to Bob. It is possible that Bob would only be required to output the solution to his part of the input (in a way that agrees with Alice's part of the solution), which is a much easier problem and therefore a much weaker lower bound.

To address this problem, we have attempted to consider a one-way communication problem between multiple parties each with small sections of the input. However, we have been unable to find any problem that could be related to edge coloring. The question of lower bounds for the W-Streaming model in general remains very open.

References

[1] E. Arjomandi. An efficient algorithm for colouring the edges of a graph with + 1 colours. *INFOR: Information Systems and Operational Research*, 20(2):82–101, 1982.

[2] S. Behnezhad, M. Derakhshan, M. Hajiaghayi, M. Knittel, and H. Saleh. Streaming and massively parallel algorithms for edge coloring. In M. A. Bender, O. Svensson, and G. Herman, editors, *editors, 27th Annual European Symposium on Algorithms, ESA 2019, September, 2019,*

Munich/Garching, Germany, volume 144 of LIPIcs, pages 15:1–15:14, pages 9–11. Schloss Dagstuhl - Leibniz-Zentrum fur Informatik, 2019.

[3] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 22–26, SODA 2006, Miami, Florida, USA, January, 2006, pages 714–723 URL, 2006.

[4] M. F. Schilling. The longest run of heads. *The College Mathematics Journal*, 21(3):196–207, 1990.

[5] V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.