Simple Streaming Algorithms for Edge Coloring

Mohammad Ansari
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Mohammad Saneian
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Hamid Zarrabi-Zadeh
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Abstract

We revisit the classical edge coloring problem for general graphs in the streaming model. In this model, the input graph is presented as a stream of edges, and the algorithm must report colors assigned to the edges in a streaming fashion, using a memory of size $O(n \operatorname{polylog} n)$ on graphs of up to $O(n^2)$ edges. In ESA 2019 and SOSA 2021, two elegant randomized algorithms were presented for this problem in the adversarial edge arrival model, where the latest one colors any input graph using $O(\Delta^2/s)$ colors with high probability in $\tilde{O}(ns)$ space. In this short note, we propose two extremely simple streaming algorithms that achieve the same color and space bounds deterministically. Besides being surprisingly simple, our algorithms do not use randomness at all, and are very simple to analyze.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Graph coloring

Keywords and phrases Edge coloring, streaming model, adversarial order

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.8

1 Introduction

An edge coloring of a graph G is an assignment of colors to the edges such that no two adjacent edges receive the same color. The chromatic index of a graph G is the smallest number of colors required to properly color edges of G. Although determining the chromatic index is NP-hard [10], a classic theorem by Vizing states that any graph can be colored with at most $\Delta + 1$ colors, where Δ is the maximum degree of the graph. On the other hand, Δ colors are always required to properly color edges of a graph.

In this short note, we revisit the edge coloring problem in the streaming model, where edges of the input graph are presented to the algorithm one at a time in an adversarially chosen order. In this model, the memory available to the algorithm is less than the input size, and hence, we cannot store all the edges of the graph. We consider one-pass algorithms in which the amount of available space is $O(n \operatorname{polylog} n)$, which is also known as the semi-streaming model [9]. Moreover, since output in the edge coloring problem is as large as the input, we cannot wait to report the output after the whole stream is processed. Instead, we need to report colors of the edges in a streaming fashion, a typical approach usually referred to as W-streaming in the literature [8].

The edge coloring problem has been studied in various models of computation. In the offline model, there are polynomial-time algorithms that compute $(\Delta + 1)$-edge colorings for general graphs [1, 11]. In the online model, Bar-Noy et al. [3] were the first to show that no algorithm can do better than the greedy algorithm, which uses at most $2\Delta - 1$ colors, no matter if the input graph is revealed edge by edge or vertex by vertex. However, their lower bound only holds when $\Delta = O(\log n)$. Therefore, research has been shifted towards online edge coloring of higher-degree graphs, i.e., when $\Delta = \omega(\log n)$. In particular, Cohen et al. [7] achieved the first positive result by giving an algorithm that uses $(1 + o(1))\Delta$ colors for

Table 1 Summary of streaming algorithms for edge coloring in the adversarial edge arrival model. Here, $s = o(\Delta)$ is a space parameter.

Algorithm	Type	# Colors	Space	Ref
Behnezhad et al.	randomized	$O(\Delta^2)$ (w.h.p.)	$\tilde{O}(n)$	[4]
Charikar and Liu	randomized	$(1+o(1))\Delta^2/s$ (w.h.p.)	$\tilde{O}(ns)$	[6]
This work	deterministic	$(1+o(1))\Delta^2/s$	$O(ns)$	[here]

bipartite graphs in the adversarial vertex arrival setting. In the same setting, Saberi and Wajc [12] have recently presented a $(1.9+o(1))\Delta$ algorithm for general graphs. In the random-order edge arrival, where edges of the graph are presented in a uniformly random order, the best current algorithm uses $(1+o(1))\Delta$ colors due to Bhattacharia et al. [5], improving upon the previous 1.26Δ algorithm of Bahmani et al. [2].

In the streaming model, existing results in the online model cannot be typically applied due to memory restriction. Behnezhad et al. [4] were the first to provide $O(n \operatorname{polylog} n)$-space streaming algorithms for the edge coloring problem. They proposed an algorithm reporting a $2e\Delta$-coloring in the random edge arrival case, and another $O(\Delta^2)$ algorithm, when edges arrive in an arbitrary (adversarial) order. Charikar and Liu [6] improved these results by presenting an algorithm that uses $(1+o(1))\Delta$ colors on random streams, and another algorithm in the adversarial edge arrival case that uses $O(\Delta^2/s)$ colors with high probability in $\tilde{O}(ns)$ space.

In this work, we focus on the adversarial edge arrival model, where edges of the input graph are revealed in an arbitrary order. In this setting, the algorithm of Behnezhad et al. [4] randomly decomposes the input graph into $O(\log n)$ bipartite subgraphs, and uses a distinct palette of colors for each subgraph. In particular, for each vertex v and each bipartite subgraph i, a counter $C_{v,i}$ is stored to denote the degree of v in the subgraph i. For each incoming edge (u,v) belonging to subgraph i, the color $(C_{u,i}, C_{v,i}, i)$ is assigned to that edge, and both $C_{u,i}$ and $C_{v,i}$ are incremented by one. Behnezhad et al. showed that with high probability, the number of colors used this way is $O(\Delta^2)$. Charikar and Liu [6] used a similar technique, but modified the way each edge is randomly assigned to bipartite subgraphs. They showed that their new algorithm uses $O(\Delta^2/s)$ colors with high probability, when available space is $\tilde{O}(ns)$.

In this short note, we propose two simple deterministic algorithms for the edge coloring problem in the adversarial arrival streams. Besides being extremely simple, our algorithms achieve the current state-of-the-art bounds on the space and number of colors, without using randomization. More precisely, our algorithms use $(1+o(1))\Delta^2/s$ colors in the worst case using $O(ns)$ space, where $s = o(\Delta)$[1]. To our knowledge, these are the first deterministic algorithms for the problem in the adversarial edge arrival streams. A summary of the results for edge coloring in the adversarial edge arrival model is presented in Table 1.

2 A Simple Greedy Algorithm

Our first algorithm uses a simple greedy approach to color edges of the input graph in an online manner. The algorithm simply colors each edge arrived with the first available color, and continues coloring until the memory is full. At that point, the algorithm deletes the most

[1] Note that when $s = \Omega(\Delta)$, an $O(ns)$ space is enough to store the entire graph.

frequently used color and discards all edges which use that color to free up some memory. The pseudocode of the algorithm is presented in Algorithm 1. Throughout the algorithm, we say that a color c is *available* for coloring an edge e, if no other edge incident to either endpoint of e has already received color c. Note that during the course of the algorithm, a deleted color will never be used again.

Algorithm 1 A Simple Greedy Algorithm.

```
start with a color palette of size 1
for each edge e in the stream do
    if no color is available in the palette to color e then
        add a new color to the palette
    end
    color e with an available color in the palette
    if the memory capacity is hit then
        delete the most frequently used color c from the palette
        remove all edges colored with c from memory
    end
end
```

▶ **Theorem 1.** *Algorithm 1 colors any input graph with* $(1+o(1))\Delta^2/s$ *colors in* $O(ns)$ *space.*

Proof. It is easy to see that the algorithm reports a proper coloring. Consider two edges e_1 and e_2 incident to a vertex v, where e_1 arrives earlier than e_2 in the stream. Upon arrival of e_2, if e_1 is still in memory, then its color will not be considered for coloring e_2 by the algorithm. On the other hand, if e_1 is not in memory, then its color has been permanently deleted from the palette, and hence will not be considered for coloring e_2 again.

To bound the number of colors used by the algorithm, first note that at any point of time during the execution of the algorithm, the color palette has size at most $2\Delta - 1$. This is because for any edge $e = (u, v)$ in the stream, at most $2(\Delta - 1) = 2\Delta - 2$ distinct colors are used by the edges incident to either u or v, as both endpoints have degree at most Δ. As such, a color palette of size $2\Delta - 1$ has at least one color available for coloring e, and hence, no new color is added to such palette by the algorithm. Therefore, for a memory of size ns, whenever the memory capacity is hit, the most popular color deleted by the algorithm has frequency at least $ns/2\Delta$. Thus, the total number of colors deleted during the course of the algorithm is at most $|E|/(ns/2\Delta)$, which is upper bounded by Δ^2/s, since $|E| \leq n\Delta/2$. Therefore, the total number of colors used by the algorithm is at most $(2\Delta - 1) + \Delta^2/s$ which is $(1 + o(1))\Delta^2/s$, for $s = o(\Delta)$. ◀

3 A Simple Buffering Algorithm

Algorithm 1 colors each arrived edge instantly in an online manner. In the streaming model, however, we can use our limited space to "buffer" a chunk of input stream before processing it and producing output. Using this buffering technique, we can achieve an even simpler algorithm. The idea behind our second algorithm is to simply buffer every chunk of $O(ns)$ edges and color it using a distinct palette of $\Delta + 1$ colors. We can use any offline $\Delta + 1$ coloring algorithm to color each chunk of the input. The pseudocode of our algorithm is presented in Algorithm 2.

▶ **Theorem 2.** *Algorithm 2 colors any input graph with* $(1+o(1))\Delta^2/s$ *colors in* $O(ns)$ *space.*

Algorithm 2 A Simple Buffering Algorithm.

partition the stream into chunks of size ns
for *each chunk of edges* **do**
  color the chunk using a distinct palette of $\Delta + 1$ colors
end

Proof. As a new palette is used for coloring each chunk, it is clear that the algorithm reports a proper edge coloring. To analyze the number of colors, we observe that the number of chunks is $|E|/ns$. Therefore, the total number of colors used is

$$\frac{|E|}{ns}(\Delta+1) \leq \frac{n\Delta}{2ns}(\Delta+1) = \left(\frac{1}{2} + o(1)\right)\Delta^2/s,$$

where the above inequality holds because $|E| \leq n\Delta/2$. ◀

▶ Remark. Algorithm 2 can be easily modified to work online by just replacing the offline $\Delta+1$ coloring algorithm with an online $2\Delta-1$ greedy one. The resulting algorithm uses $(1+o(1))\Delta^2/s$ colors in the worst case.

References

1 Eshrat Arjomandi. An efficient algorithm for colouring the edges of a graph with $\Delta+1$ colours. *Information Systems and Operational Research*, 20(2):82–101, 1982.
2 Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. Online graph edge-coloring in the random-order arrival model. *Theory of Computing*, 8(1):567–595, 2012.
3 Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters*, 44(5):251–253, 1992.
4 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and massively parallel algorithms for edge coloring. In *Proceedings of the 27th European Symposium on Algorithms*, volume 144 of *LIPIcs*, pages 15:1–15:14, 2019.
5 Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms*, pages 2830–2842, 2021.
6 Moses Charikar and Paul Liu. Improved algorithms for edge colouring in the W-streaming model. In *Proceedings of the 4th SIAM Symposium on Simplicity in Algorithms*, pages 181–183, 2021.
7 Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science*, pages 1–25, 2019.
8 Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 714–723, 2006.
9 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming*, volume 3142 of *LNCS*, pages 531–543, 2004.
10 Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.
11 Jayadev Misra and David Gries. A constructive proof of Vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992.
12 Amin Saberi and David Wajc. The greedy algorithm is not optimal for on-line edge coloring. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming*, volume 198 of *LIPIcs*, pages 109:1–109:18, 2021.