Improved Algorithms for Edge Colouring in the $W$-Streaming Model

Paul Liu

Moses Charikar
Edge-colouring
Edge - Colouring

- An assignment of colours to the edges of a graph
Edge - Colouring

- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour
Edge - Colouring

- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour
- Goal: Minimize \# distinct colours used
Edge-Colouring

- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour
- Goal: Minimize number of distinct colours used
Edge - Colouring

- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour
- Goal: Minimize # distinct colours used
Edge - Colouring

- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour
- Goal: Minimize number of distinct colours used

- \( \Delta \) always possible!
  (Vizing) \( (\Delta = \max \text{ deg.}) \)

- \( \Delta \) required; NP-Complete to determine \( \Delta \) vs. \( \Delta + 1 \)
Edge-colouring in $\mathbb{W}$-Streaming
Edge-colouring in $W$-Streaming

- Up to $O(n^2)$ edges streamed in $\#\text{vertices} = n$
- $O(n \log n)$ memory for alg.
  (can assume graph is dense)
Edge-colouring in $\mathbb{W}$-Streaming

- Up to $O(n^2)$ edges streamed in $\# \text{vertices} = n$
- $\Theta(n \log n)$ memory (can assume graph is dense)
- A colour for every edge must be output by the time the stream completes
Edge-colouring in $W$-Streaming

- Up to $O(n^2)$ edges streamed in
- $O(n \cdot \text{polylog } n)$ memory
- A colour for every edge must be output by the time the stream completes
Edge-colouring in W-Streaming

- Up to $O(n^2)$ edges streamed in
- $O(n \log n)$ memory
- A colour for every edge must be output by the time the stream completes
- Memory allows for limited "buffering" of output
Edge-colouring in $W$-Streaming

- Up to $O(n^2)$ edges streamed in
- $O(n \text{ polylog } n)$ memory
- A colour for every edge must be output by the time the stream completes
- Memory allows for limited "buffering" of output
Related Work: **Online Model**

- Must output a colour for each edge *immediately*

- Allowed poly(n) memory (usually $O(E)$)
Related Work: Online Model

- Must output a colour for each edge immediately

- Allowed poly(n) memory (usually $O(E)$)

- Two regimes: vertex arrival (all edges adj. to a vertex arrive at once) edge-arrival (edges arrive one at a time)

our regime
Related Work: Online Model

- When $\Delta = \omega(\log n)$, $(1 + o(1))\Delta$ is possible:
  - Adversarial vertex arrivals [Cohen, Peng, Wajc '19]
  - Random edge arrivals [Bhattacharya, Grandoni, Wajc '20]  
  
  [SODA 21]
Related Work: Online Model

- When $\Delta = \omega(\log n)$, $(1 + o(1))\Delta$ is possible:
  - Adversarial vertex arrivals [Cohen, Peng, Wajc '19]
  - Random edge arrivals [Bhattacharya, Grandoni, Wajc '20]
  - SODA 21

- All of these algs use too much memory for $W$-streaming
Progress in N-streaming
<table>
<thead>
<tr>
<th>Progress in W-streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge ( v ) Arrival</td>
</tr>
<tr>
<td>Random Arrival</td>
</tr>
<tr>
<td>[2e(1+o(1))\Delta \text{ colours}]</td>
</tr>
</tbody>
</table>

[Behnezhad et al. '11]
## Progress in $W$-streaming

<table>
<thead>
<tr>
<th>Edge Arrival</th>
<th>Adversarial Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Random</strong></td>
<td>- $O(D^2)$ colouring</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Behnezhad et al. '19</th>
<th>- $2e(1+o(1))\Delta$ colours</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our Results</strong></td>
<td>- $(1+o(1))\Delta$ colours</td>
</tr>
<tr>
<td>(uploaded talk! + animated slides!)</td>
<td></td>
</tr>
</tbody>
</table>

- *(all other algos use $\tilde{\Omega}(n)$ space)*

- *(Today’s talk)*
A simple algorithm for adversarial Arrivals
A simple algorithm for adversarial Arrivals

- For each node $u$, generate $s := 36 \log n$ random bits
- Initialize empty graphs $B_1, B_2, \ldots, B_s$
A simple algorithm for adversarial arrivals

- For each node $u$, generate $s := 36 \log n$ random bits
- Initialize empty graphs $B_1, B_2, \ldots, B_s$
- For each edge $(u,v)$, pick a random bit where the bit strings of $u$ & $v$ differ. Let $i$ be the index of this bit. Add $(u,v)$ to $B_i$. 
A simple algorithm for adversarial Arrivals

- For each node $u$, generate $s := 36 \log n$ random bits
- Initialize empty graphs $B_1, B_2, \ldots, B_s$
- For each edge $(u,v)$, pick a random bit where the bit strings of $u$ & $v$ differ. Let $i$ be the index of this bit. Add $(u,v)$ to $B_i$. 

\[ \begin{array}{c}
 v \\
 \longrightarrow \\
 u \quad \text{(random bit)} \\
 \end{array} \]
\[ B_1 \quad B_2 \quad \ldots \quad B_s \]
A simple algorithm for adversarial arrivals

- Each of the Bi's are bipartite
  (left nodes: $i^{th}$ bit 0, right nodes: $i^{th}$ bit 1)
A simple algorithm for adversarial arrivals

- Each of the $B_i$'s are bipartite
  (left nodes: $i^{th}$ bit 0, right nodes: $i^{th}$ bit 1)

- Any index equally likely to be chosen, so $\text{max deg } B_i \approx \frac{\Delta}{5}$
  (concentration achieved when $\Delta \approx \omega(\log n)$)
A simple algorithm for adversarial arrivals

- Each of the $B_i$'s are bipartite
  (left nodes: $i^{th}$ bit $0$, right nodes: $i^{th}$ bit $1$)

- Any index equally likely to be chosen, so $\max \deg B_i \leq \frac{\Delta}{5}$
  (Concentration achieved when $\Delta \approx \omega(\log n)$)

- Choice of $36 \log n$ ensures at least one mismatched bit
Colouring a bipartite graph
Colouring a bipartite graph

- For each node \( u \in B \), initialize a counter \( C_u = 0 \).
Colouring a bipartite graph

- For each node $u \in B$, initialize a counter $C_u = 0$.
- When an edge $(u, v)$ is streamed in, output colour $(u, v)$ and increment $C_u$ if $C_v$ by 1.
**Colouring a bipartite graph**

- For each node $u \in B$, initialize a counter $C_u = 0$.
- When an edge $(u, v)$ is streamed in, output colour $(u, v)$ and increment $C_u$ and $C_v$ by 1.
Colouring a bipartite graph

- For each node \( u \in B \), initialize a counter \( C_u = 0 \).
- When an edge \( (u, v) \) is streamed in, output colour \( \langle u, v \rangle \) and increment \( C_u \) \& \( C_v \) by 1.
Colouring a bipartite graph

- For each node \( u \in B \), initialize a counter \( C_u = 0 \).
- When an edge \((u, v)\) is streamed in, output colour \((u, v)\) and increment \( C_u \) if \( C_v \) by 1.
Colouring a bipartite graph

- For each node \( v \in B \), initialize a counter \( C_v = 0 \).

- When an edge \((u, v)\) is streamed in, output colour \((u, v)\) and increment \( C_u \) and \( C_v \) by 1.
Colouring a bipartite graph

- For each node \( u \in B \), initialize a counter \( \ell_n = 0 \).
- When an edge \((u, v)\) is streamed in, output colour \((\ell_n, \ell_m)\) and increment \( \ell_u \) and \( \ell_v \) by 1.
Colouring a bipartite graph

- For each node $u \in B$, initialize a counter $C_u = 0$.
- When an edge $(u, v)$ is streamed in, output colour $(u, v)$ and increment $C_u$ and $C_v$ by 1.
Colouring a bipartite graph

- Worst case: each counter goes up to \( \max \deg B_i \) \( (\geq 0.5) \)

\[ \Rightarrow (u, v) \text{ has } \left(\frac{4}{5}\right)^2 \text{ possibilities}. \]

- Achievable:

Works for any \( p, q \leq \max \deg B_i \) \( (\geq 0.5) \).
Colouring a bipartite graph

- Worst case: each vertex goes up to \( \max \text{deg } B_i \geq 0.1 \frac{D}{5} \)
  \[ (u, v) \text{ has } \left( \frac{D}{5} \right)^2 \text{ possibilities} \]

- Achievable:

\[ p \quad n \quad q \quad v \]

Works for any \( p, q \leq \max \text{deg } B_i \geq 0.1 \frac{D}{5} \).
Wrapping Up

Total # colours: \( S \cdot \left( \left( \frac{1 + o(1)}{5} \right) \frac{\Delta}{5} \right)^2 \)

\begin{align*}
\# \text{ of } B_i & \quad \text{max deg per } B_i \\
= \frac{\Delta^2}{5} \left( 1 + o(1) \right)
\end{align*}
Wrapping up

Total # of colours

$\frac{1}{5} \left( \frac{1}{2} \right) \Delta^2 (1 + \alpha \alpha')$

max deg per B$
\rightarrow$

copies of star-shaped graphs

Worst case can be achieved by streaming polyas
Open problem: Is there an $O(A)$ colouring algorithm for adversarial orders in $W$-streaming?
Open problem: Is there a $O(d)$ colouring algorithm for adversarial orders in $W$-streaming?