Online Graph Problems

Shahin Kamali

Lecture 23 - Nov. 28th, 2017
University of Manitoba
Review & Plan
Today’s plan

- Online Edge-coloring
- Online bipartite matching (marriage problem)
- Assignment 3 & logistics
Online Edge Coloring in Graphs
Graph Edge Coloring

Problem Definition

- In edge coloring, the goal is to color edges of a graph with minimum number of colors
  - No two adjacent edges (edges sharing an endpoint) should have the same color
- In the offline setting, the problem is NP-hard!
- For a graph of degree $\Delta$, at least $\Delta$ and at most $\Delta + 1$ colors are required (Vizing theorem)
  - This implies that $\text{cost}(\text{Opt}) \approx \Delta$
In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).

For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.
Problem Definition

In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).

For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.

- cost of $\text{OPT}$ is 3
In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).

For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.

Cost of $\text{OPT}$ is 3.
Problem Definition

In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).

For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.

Cost of \( \text{OPT} \) is 3
Graph Edge Coloring

Problem Definition

In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).

For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.

cost of $OPT$ is 3
Problem Definition

- In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).
- For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.
- Cost of $\text{OPT}$ is 3.
Graph Edge Coloring

Problem Definition

- In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).
- For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.
- cost of $\text{OPT}$ is 3
In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).

For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.

cost of $\text{OPT}$ is 3
In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).

For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.

cost of $\text{OPT}$ is 3
In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).

For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.

cost of $\text{OPT}$ is 3
Graph Edge Coloring

Problem Definition

- In the online setting, edges arrive one by one, and an algorithm should take an irrevocable decision on coloring the edges without any knowledge about future edges (or how graph looks).
- For example, Greedy family of algorithms maintain a set of colors and use them, if possible, before requesting a new coloring.
- cost of $\text{OPT}$ is 3
- Cost of Greedy is 4, which is not optimal
Graph Edge Coloring

Greedy algorithm

**Theorem**

*Greedy has a competitive ratio of at most 2.*

- For any graph of degree $\Delta$, cost of $OPT$ is at least $\Delta$.
- Cost of greedy is at most $2\Delta - 1$. 
No deterministic online algorithm can have a competitive ratio better than 2.

- Adversary forms a graph of degree $\Delta$ which can be colored using $\Delta$ colors
  - In doing so, any online algorithm needs to use at least $2\Delta$ colors
Graph Edge Coloring

Lower Bound

- The input starts by sending a star of degree $\Delta - 1$
  - Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.

There are at most $K = (\frac{2\Delta}{\Delta - 1})$ ways to color a star using $2\Delta$ colors.

If we have $K + 1$ stars, at least two of them have the same coloring (pigeonhole principle).

If we have $2K + 1$ stars, at least three of them have the same coloring.

If we have $(\Delta + 1)K + 1$ stars, at least $\Delta$ of them will have the same coloring.
The input starts by sending \textbf{star} of degree $\Delta - 1$

- Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.

- There are at most $K = \binom{2\Delta}{\Delta - 1}$ ways to color a star using $2\Delta$ colors.
Graph Edge Coloring

Lower Bound

- The input starts by sending a star of degree $\Delta - 1$.
  - Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.
- There are at most $K = \binom{2\Delta}{\Delta-1}$ ways to color a star using $2\Delta$ colors.
The input starts by sending **star** of degree $\Delta - 1$

- Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.

- There are at most $K = \binom{2\Delta}{\Delta - 1}$ ways to color a star using $2\Delta$ colors.
Lower Bound

- The input starts by sending a star of degree $\Delta - 1$
  - Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.

- There are at most $K = \binom{2\Delta}{\Delta - 1}$ ways to color a star using $2\Delta$ colors.
The input starts by sending star of degree $\Delta - 1$

- Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.

- There are at most $K = \binom{2\Delta}{\Delta - 1}$ ways to color a star using $2\Delta$ colors.
Graph Edge Coloring

Lower Bound

- The input starts by sending *star* of degree $\Delta - 1$
  - Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.
- There are at most $K = \binom{2\Delta}{\Delta - 1}$ ways to color a star using $2\Delta$ colors.
Graph Edge Coloring

**Lower Bound**

- The input starts by sending **star** of degree $\Delta - 1$
  - Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.
- There are at most $K = \binom{2\Delta}{\Delta - 1}$ ways to color a star using $2\Delta$ colors.
The input starts by sending a star of degree $\Delta - 1$

- Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.
- There are at most $K = \binom{2\Delta}{\Delta-1}$ ways to color a star using $2\Delta$ colors.
The input starts by sending a star of degree $\Delta - 1$

Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.

There are at most $K = \binom{2\Delta}{\Delta - 1}$ ways to color a star using $2\Delta$ colors.
Graph Edge Coloring

Lower Bound

- The input starts by sending star of degree $\Delta - 1$
  - Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.
- There are at most $K = \binom{2\Delta}{\Delta-1}$ ways to color a star using $2\Delta$ colors.
The input starts by sending star of degree $\Delta - 1$

Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.

There are at most $K = \left( \begin{array}{c} 2\Delta \\ \Delta - 1 \end{array} \right)$ ways to color a star using $2\Delta$ colors.

If we have $K + 1$ stars, at least two of them have the same coloring (pigeonhole principle).
The input starts by sending **star** of degree $\Delta - 1$

- Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.

There are at most $K = \binom{2\Delta}{\Delta-1}$ ways to color a star using $2\Delta$ colors.

- If we have $K + 1$ stars, at least two of them have the same coloring (pigeonhole principle).
- If we have $2K + 1$ stars, at least three of them have the same coloring.
The input starts by sending star of degree $\Delta - 1$

- Recall that a star of degree $d$ is a tree formed by a center vertex connected to $d$ leaves.

- There are at most $K = \binom{2\Delta}{\Delta-1}$ ways to color a star using $2\Delta$ colors.
  - If we have $K + 1$ stars, at least two of them have the same coloring (pigeonhole principle).
  - If we have $2K + 1$ stars, at least three of them have the same coloring.
  - If we have $(\Delta + 1)K + 1$ stars, at least $\Delta$ of them will have the same coloring.
Graph Edge Coloring

**Lower Bound**

- After sending \( \Delta + 1 \) edges, at least \( \Delta \) edges have the same coloring.
- Adversary reveals edges forming another star, of degree \( \Delta \), connected to centers of these stars.
  - Any of these new edges requires a color other than the \( \Delta - 1 \) colors in the old star.
  - In total \( (\Delta - 1) + \Delta = 2\Delta - 1 \) colors are used by the algorithm.
Graph Edge Coloring

Lower Bound

- After sending \( \Delta + 1 \)\( K \) + 1 stars, at least \( \Delta \) stars have the same coloring.
- Adversary reveals edges forming another star, of degree \( \Delta \), connected to centers of these stars.
  - Any of these new edges requires a color other than the \( \Delta - 1 \) colors in the old star.
  - In total \( (\Delta - 1) + \Delta = 2\Delta - 1 \) colors are used by the algorithm.
After sending $\Delta + 1)K + 1$ stars, at least $\Delta$ stars have the same coloring.

Adversary reveals edges forming another star, of degree $\Delta$, connected to centers of these stars.

- Any of these new edges requires a color other than the $\Delta - 1$ colors in the old star.
- In total $(\Delta - 1) + \Delta = 2\Delta - 1$ colors are used by the algorithm.
Graph Edge Coloring

Lower Bound

- After sending $\Delta + 1)K + 1$ stars, at least $\Delta$ stars have the same coloring.

- Adversary reveals edges forming another star, of degree $\Delta$, connected to centers of these stars.
  - Any of these new edges requires a color other than the $\Delta - 1$ colors in the old star.
  - In total $(\Delta - 1) + \Delta = 2\Delta - 1$ colors are used by the algorithm.
Lower Bound

- After sending \( \Delta + 1 \) \( K + 1 \) stars, at least \( \Delta \) stars have the same coloring.
- Adversary reveals edges forming another star, of degree \( \Delta \), connected to centers of these stars.
  - Any of these new edges requires a color other than the \( \Delta - 1 \) colors in the old star.
  - In total, \( (\Delta - 1) + \Delta = 2\Delta - 1 \) colors are used by the algorithm.
OPT colors edges adjacent to the new center (at the bottom) using $\Delta$ colors.
OPT colors edges adjacent to the new center (at the bottom) using $\Delta$ colors.

Other edges form stars connected to only of the colored edges; each star can be colored using the remaining $\Delta - 1$ colors.

OPT uses $\Delta$ colors.
OPT colors edges adjacent to the new center (at the bottom) using $\Delta$ colors.

Other edges form stars connected to only of the colored edges; each star can be colored using the remaining $\Delta - 1$ colors.

OPT uses $\Delta$ colors.
Graph Edge Coloring

Lower Bound Summary

For any given online algorithm, adversary created a graph $G$ so that:

- $G$ has degree $\Delta \rightarrow \text{OPT}$ uses $\Delta$ colors.
- the online algorithm uses $2\Delta - 1$ colors

Competitive ratio of the algorithm is at least $\frac{2\Delta - 1}{\Delta} \approx 2$
Graph Edge Coloring

Lower Bound

**Theorem**

*No deterministic online algorithm can have a competitive ratio better than 2.*

- This implies that greedy algorithms are the best deterministic algorithm
Online Bipartite Matching
(Marriage problem)
Given a bipartite graph, the goal is to create a matching (creating pair of non-adjacent edges) with maximum size.
Online Matching

Online Bipartite Matching

- Given a bipartite graph, the goal is to create a matching (creating pair of non-adjacent edges) with maximum size.
- In the online setting, vertices in one side of the graph are given, and vertices in the other side arrive one by one.
  - Upon arrival of vertex \( x \), all edges connecting \( x \) to its neighbors on the left are revealed.
  - An online algorithm should match \( x \) with another vertex, if possible, without any information about future vertices.
- **Greedy algorithm**: match with any vertex on the right if possible!
Online Bipartite Matching

- Given a bipartite graph, the goal is to create a matching (creating pair of non-adjacent edges) with maximum size.
- In the online setting, vertices in one side of the graph are given, and vertices in the other side arrive one by one:
  - Upon arrival of vertex $x$, all edges connecting $x$ to its neighbors on the left are revealed.
  - An online algorithm should match $x$ with another vertex, if possible, without any information about future vertices.
- **Greedy algorithm**: match with any vertex on the right if possible!
Online Matching

Online Bipartite Matching

- Given a bipartite graph, the goal is to create a matching (creating pair of non-adjacent edges) with maximum size.
- In the online setting, vertices in one side of the graph are given, and vertices in the other side arrive one by one.
  - Upon arrival of vertex \( x \), all edges connecting \( x \) to its neighbors on the left are revealed.
  - An online algorithm should match \( x \) with another vertex, if possible, without any information about future vertices.
- **Greedy algorithm**: match with any vertex on the right if possible!
Online Matching

Online Bipartite Matching

- Given a bipartite graph, the goal is to create a matching (creating pair of non-adjacent edges) with maximum size.
- In the online setting, vertices in one side of the graph are given, and vertices in the other side arrive one by one.
  - Upon arrival of vertex $x$, all edges connecting $x$ to its neighbors on the left are revealed.
  - An online algorithm should match $x$ with another vertex, if possible, without any information about future vertices.
- **Greedy algorithm**: match with any vertex on the right if possible!
Online Bipartite Matching

- Given a bipartite graph, the goal is to create a matching (creating pair of non-adjacent edges) with maximum size.
- In the online setting, vertices in one side of the graph are given, and vertices in the other side arrive one by one.
  - Upon arrival of vertex $x$, all edges connecting $x$ to its neighbors on the left are revealed.
  - An online algorithm should match $x$ with another vertex, if possible, without any information about future vertices.
- **Greedy algorithm**: match with any vertex on the right if possible!
Online Matching

Online Bipartite Matching

- Given a bipartite graph, the goal is to create a matching (creating pair of non-adjacent edges) with maximum size.
- In the online setting, vertices in one side of the graph are given, and vertices in the other side arrive one by one.
  - Upon arrival of vertex $x$, all edges connecting $x$ to its neighbors on the left are revealed.
  - An online algorithm should match $x$ with another vertex, if possible, without any information about future vertices.
- **Greedy algorithm:** match with any vertex on the right if possible!
Online Bipartite Matching

- Given a bipartite graph, the goal is to create a matching (creating pair of non-adjacent edges) with maximum size.

- In the online setting, vertices in one side of the graph are given, and vertices in the other side arrive one by one:
  - Upon arrival of vertex $x$, all edges connecting $x$ to its neighbors on the left are revealed.
  - An online algorithm should match $x$ with another vertex, if possible, without any information about future vertices.

- **Greedy algorithm**: match with any vertex on the right if possible!
In this example, \( \text{OPT} \) has a benefit of 4 (a perfect matching) while greedy has a benefit of 3!

Matching is a maximization problem: we would like to maximize the benefit instead of minimizing the cost.

Competitive ratio is often defined as the maximum value of

\[
\frac{\text{Benefit}(\text{OPT})}{\text{benefit}(\text{Alg})}
\]

Greedy algorithm always creates a maximal matching

- For any ‘mistake’ match, it blocks two possible matches
- The benefit of greedy is no less than twice that of \( \text{OPT} \)

**Theorem**

*Greedy has a competitive ratio of at most 2 (details in the next class).*