University of Glasgow | School of Computing Science

Counting Monochromatic Components in Adversarial Graph Burning

Lewis Dyer (2299195)

November 17, 2021

ABSTRACT

We introduce adversarial graph burning, an extension of the graph burning process to contagion with competition. We develop multiple results on the number of monochromatic components on graph colourings resulting from adversarial graph burning.

1. INTRODUCTION

This paper outlines the standard template for an MSci submission. In earlier years, MSci students at the School of Computing Science[1], University of Glasgow, were expected to produce a full-length dissertation. Now, the requirement is for MSci students to write a paper of up to 14 pages in length, using the supplied `mpaper` LaTeXstyle file.

The precise structure of an MSci paper is not mandated, but it should probably cover in detail the following aspects of the project.

1. General description of the problem, motivation, relevance
2. Background information, possibly including a literature survey
3. Description of approach taken to solve the problem, including high-level design and lower-level implementation details as appropriate
4. Evaluation, qualitative or quantitative as appropriate
5. Conclusion, including scope for future work

2. BACKGROUND

This LaTeXtemplate is based on the ACM `sig-alternate` class. The layout is two-column text. Generally figures and tables only extend to one column width, e.g. Table 1, but it is possible to make them stretch over both columns using the `figure*` and `table*` environments. For an example, see Figure 1.

3. THE WIZWOZ SYSTEM

Again, Simon Peyton Jones has a lovely description of how to write a paper on his website[2]. Personally, I put URLs in footnotes and *bona fide* references in the bibliography. For instance, Turing [2] and Knuth [1] would not be out of place in list of references. How many references? Hard to say. Five is not enough, 50 is pushing it.

[1]`http://www.dcs.gla.ac.uk`
[2]`http://research.microsoft.com/en-us/um/people/simonpj/papers/giving-a-talk/giving-a-talk.htm`

Operating System	*Version*	*Verdict*
Ubuntu	12.04	Everyone's favourite Linux, unless you grew up with RedHat
Slackware	xxx	Pseudo-hacker's Linux, how often do you recompile your kernel?
Mac OS	10.7	For people with more money than sense

Table 1: Single column table of figures

4. ADVERSARIAL GRAPH BURNING

4.1 Defining Adversarial Graph Burning

First, we formalise the process of Adversarial Graph Burning on a graph G. Throughout, we presume that G is a finite, simple, undirected graph.

Definition 4.1. *Adversarial Graph Burning (or AGB for short) is a discrete-time graph process for two players. Each vertex is assigned one of 4 colours:*

1. White *vertices have not been burned by either player yet.*
2. Red *vertices have been burned by player 1.*
3. Blue *vertices have been burned by player 2.*
4. Green *vertices have been burned by* both *players.*

At time $t = 0$, all vertices are initially white. Each round consists of two main steps. First, both players simultaneously choose a white vertex, burning the vertex into that player's respective colour. Secondly, all non-white vertices spread their colour onto adjacent white vertices, according to the following rules:

- *If a white vertex is burned by multiple vertices, all with the same colour, the white vertex is also burned with the same colour.*

Figure 1: An example figure stretching over two columns

- *If a white vertex is burned by both red and blue vertices, the white vertex will become green.*
- *If a white vertex is burned by either red or blue vertices (but not both), and also by green vertices, the white vertex will become red or blue, respectively.*

In particular, these rules ensure that this process is symmetrical, so the choice of player is unimportant.

Our main interest in this problem will be to count the maximum number of monochromatic components that can appear in any colouring of G resulting from an instance of the AGB process. Our definition of "monochromatic components" differs slightly from most other uses of the term, so we define it separately here to account for these differences, and use slightly different terminology to emphasise these differences:

Definition 4.2. *Given a graph G, with an assignment of colours resulting from the AGB process, a* red cluster *is a connected component in the subgraph induced by all red and green vertices in G.*

We also introduce the restriction that, in the AGB process, both players may only choose to burn the same vertex if there are no other remaining white vertices to burn.

We note that only red clusters are defined here - blue clusters could of course be defined similarly, though since the AGB process is symmetrical we need only define red clusters.

In addition, the latter restriction on AGB is relatively minor, and mainly aims to remove some degenerate cases when counting red clusters, without needing to explicitly define exceptions for later results.

4.2 Burning Sequences

We now introduce a compact notation for describing a particular instance of adversarial graph burning.

Definition 4.3. *Given a graph G, a* burning sequence *of length n, $B \in (V(G) \times V(G))^n$, is a sequence of tuples, $(r_1, b_1), (r_2, b_2), \ldots, (r_n, b_n)$, such that player 1 burns vertex r_t and player 2 burns vertex b_t at time t.*

This sequence describes possible choices for the first n rounds of the AGB process, but some further care is required to ensure this sequence describes valid choices for this process:

Definition 4.4. *Given a burning sequence B, B is said to be* valid *if:*

1. *For every vertex that appears in the i^{th} term of B, say v_i, for every vertex in the j^{th} term of B such that $j < i$, say v_j the distance between v_i and v_j is at most $(i - j)$.*
2. *After the burning sequence described by B is completed on G, every vertex is G is non-white.*

From this definition, an immediate corollary is that the original graph burning problem is a subset of adversarial graph burning:

Corollary 4.1. *Given a valid burning sequence for the original graph burning problem, say $(v_1, v_2, \ldots, v_n)$, the burning sequence $(v_1, v_1), (v_2, v_2), \ldots, (v_n, v_n)$ describes a valid burning sequence in adversarial graph burning.*

4.3 Bounding the length of burning sequences

A useful question in the adversarial graph burning problem is to consider the maximum number of rounds required for this process to terminate, providing an upper bound for the number of valid burning sequences for a graph G. The following result gives a useful bound for connected graphs:

Theorem 4.1. *For a connected graph G with n vertices, the maximum number of rounds required for the AGB process to terminate is $\lceil \frac{n}{3} \rceil$.*

Proof. Given a graph G, consider U, the subgraph induced by all white vertices in G. We aim to show that, if U contains at least 3 vertices, each round of AGB removes at least 3 vertices from U.

If U contains at least 3 vertices, then we choose 2 vertices to initially burn, say v_1 and v_2, and we need to show that at least one other distinct vertex is adjacent to v_1 or v_2.

Considering the possible degrees of v_1 and v_2 in U, there are two potentially problematic cases, while the other cases are straightforward.

If v_1 and v_2 are both adjacent and have degree 1, then no other vertices are adjacent to them. However, since U has at least 3 vertices, some other vertex v_3 must be in U, but not connected to v_1 or v_2. However, since G is connected, v_3 must be adjacent to some non-white vertex in G, so v_3 must also be removed this round. A similar argument holds when v_1 and v_2 are both degree 0.

Hence, while at least 3 vertices remain unburned, at least 3 vertices are burned each round, and if there are ever less than 3 vertices remaining they will all be burned in 1 round, so the maximum number of rounds to burn all vertices is $\lceil \frac{n}{3} \rceil$.

□

5. COUNTING MONOCHROMATIC COMPONENTS

5.1 Monochromatic components on paths

Our first key observation when counting red clusters is that at most one cluster can be added per round. This is because red vertices can be added in two different ways. If red vertices are added via spreading from adjacent vertices, no new clusters are created in this way (though existing clusters may be merged together, *reducing* the number of red clusters on the graph). So red clusters can only be created by player 1 choosing vertices to burn, but only one vertex can be burned by player 1 each turn. Coupled with Theorem 4.1, this gives a useful bound for the number of clusters on connected graphs:

COROLLARY 5.1. *Let G be a connected graph on n vertices. Then the maximum number of red clusters on G is $\lceil \frac{n}{3} \rceil$, and this maximum is attainable when G is a path on n vertices.*

PROOF. Firstly, since each round can introduce at most 1 red cluster, and since the maximum number of rounds on G is $\lceil \frac{n}{3} \rceil$ by Theorem 4.1, the maximum number of red clusters on G is $\lceil \frac{n}{3} \rceil$.

Now let G be the path on n vertices. Listing the vertices of G in order from $v_1, v_2, \dots, v_n$, the burning sequence $(v_1, v_2), (v_4, v_5), \dots, (v_{n-2}, v_{n-1})$ contains $\lceil \frac{n}{3} \rceil$ red clusters.

□

5.2 Caterpillar graphs

We now consider a natural extension of paths, and provide a method for counting clusters on these graphs.

DEFINITION 5.1. *A* caterpillar graph *is a graph obtained by taking a path, and adding any number of vertices of degree 1 (known as leaves), that are adjacent to a vertex on the original path.*

In order to count clusters on caterpillar graphs, it will be useful to introduce more compact notation for describing caterpillar graphs:

DEFINITION 5.2. *Given a caterpillar graph built on a path with n vertices, its* caterpillar string *is a sequence $C \in \mathbb{N}^n$ of the form $(c_1, \dots, c_n)$, where c_i denotes the number of leaves adjacent to the i^{th} path vertex. For instance, the caterpillar string $(1, 0, 2, 3)$ represents the caterpillar graph shown in Figure [INSERT A PICTURE HERE].*

Clearly, there are a countably infinite number of caterpillar strings of length n. However, for the purposes of counting clusters, the following lemma shows that we can reduce the set of caterpillar strings to a finite subset of strings of a given form:

LEMMA 5.1. *Given a caterpillar string C, the maximum number of red clusters in C is equal to the maximum number of red clusters in the caterpillar string given by*

$$C'_i = \begin{cases} 0 & C_i = 0 \\ 1 & C_i \geq 1 \end{cases}$$

Moreover, given such a caterpillar string C', if the first or last element in C' is 1, this can be removed and two zeros can be appended to the start or end of the sequence respectively, giving an equivalent caterpillar string, say C''. After performing these two operations, we say that C'' is a reduced caterpillar string.

PROOF. For the first reduction, suppose a given path vertex v_i has at least 2 adjacent leaves. Pick two of these leaves without loss of generality, calling them l_1 and l_2, and consider the subgraph induced by v_i, l_1 and l_2.

Suppose that this subgraph contains 2 red clusters. Then both leaves must be coloured red. However, this is only possible when v_i is also red. Since both leaf vertices are not adjacent, 2 rounds are required to colour them both red, and hence the first one coloured red will spread to v_i, colouring it red. And if v_i was already coloured blue, it would spread its colour to at least one of l_1 and l_2, so by contradiction this subgraph can contain at most 1 red cluster. And since the leaves were chosen without any loss of generality, either no leaves are red, all leaves are red along with their common path vertex, or exactly one leaf is red, so all but one leaf can be removed without changing the overall number of clusters.

For the second reduction, if one of the endpoints has a leaf, the leaf can be treated as a path vertex rather than a leaf vertex, removing a leaf from the existing endpoint and becoming the new endpoint instead. □

Therefore, the set of reduced caterpillar strings of length n is given by the set of bitstrings of length n beginning and ending with a 0, hence there are 2^{n-2} reduced caterpillar strings of length n. In practice this may be reduced even further, since reversing the reduced caterpillar string gives another string that may be different, but which always represents an isomorphic caterpillar graph.

5.3 Monochromatic components on caterpillar graphs

As discussed in Lemma 5.1, we can transform any caterpillar graph into a reduced caterpillar graph while keeping the same number of maximum red clusters on the graph. We now present the following algorithm to count the maximum number of red clusters on any reduced caterpillar graph:

Given a reduced caterpillar string, partition this string into pieces of length 3, leaving any remainder characters in the last piece which may be shorter than 3 characters, and treat each of these pieces as a (not necessarily reduced) caterpillar string.

For each of these pieces, we can easily compute the maximum number of red clusters which originate in this piece, given some starting and ending constraints. These constraints are comprised of 4 distinct cases:

- In case RW, when colouring of this piece begins, the first path vertex is white, but the previous adjacent path vertex, which if it exists is not part of this piece, is coloured red.
- Case BW is equivalent to case RW, except the previous path vertex is blue instead of red. This is also the case applied for the very first piece, in order to ensure that colouring the first path vertex red correctly counts the start of a new red cluster.
- Case R is where the first path vertex is already coloured red, since the colouring of the previous piece is not fully contained within that piece.
- Case B is analogous to case R.

These cases are defined for the start of a piece, but the ending constraints are analogous, and crucially correspond to the same starting constraint of the next piece.

Since these pieces are small, each of these computations is small and relatively simple to perform. For each piece, this then produces a weighted directed bipartite graph G_P, with vertex set $\{RW_{start}, BW_{start}, R_{start}, B_{start}, RW_{end}, BW_{end}, R_{end}, B_{end}\}$, such that an edge exists from X_{start} to Y_{end} if there exists a colouring of piece P with starting constraint X and ending constraint P, with weight equal to the number of clusters originating in P.

Once all of these bipartite graphs are computed, the ending vertices of each piece can be merged with the starting vertices of the next piece, along with adding in a common source and sink vertex, removing every case bar case BW from the first piece, and finally removing all unreachable vertices, generating a directed acyclic multipartite graph G. Then, the maximum number of red clusters on the reduced caterpillar graph is equal to the length of the longest path in G.

Acknowledgments. Firstly, I would like to thank my supervisors - Jessica Enright, William Pettersson and John Sylvester - for their continual guidance and support throughout the project. I would also like to thank my parents, for always caring for me, and supporting my goals when I thought they could never be attained. And, last but not least, I would like to thank my girlfriend Jodie, for her never-ending love and kindness.

6. REFERENCES

[1] D. E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms.* Addison Wesley, 1st edition, 1968.

[2] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society Series 2*, 42:230–265, 1937.