Burning Graphs

Xianghe Xu, Xinrui Chen, Nancy Jia, Haozhen Zheng [*]

Abstract

Graph Burning is a way to simulate the spread of information in a network. Our goal was to find an upper bound on the number of rounds it takes to burn a graph in relation to the number of its vertices, and in particular, attempt to prove the **Burning Number Conjecture**.

We explored properties that a minimal counterexample to the conjecture would have to have. We also created the concept of (k, ℓ)-burnability, which is a relaxation of regular graph burning that can be computationally easier if $k < \ell$. We created an algorithm to check if a graph is (k, ℓ)-burnable, which runs quickly on small graphs, and our hope is to use this algorithm to find either a counterexample to the burning number conjecture or to build intuition on which kinds of graphs are hard to burn.

1 Introduction

Given a finite, simple, undirected graph G, the burning of G is a discrete-time process; at each round, vertices of G may either be **burned** or **unburned**.

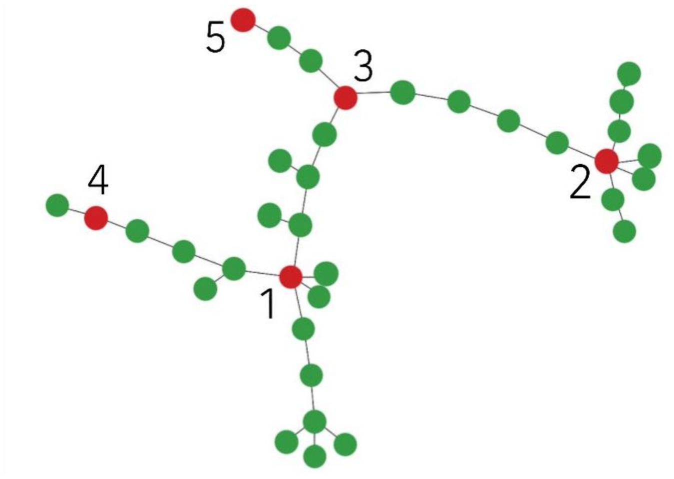
When burning a graph, we always begin with a graph G with all vertices being unburned. In the first round, we can pick any vertices as our first source, and set it on fire. Now, this vertex is burned. In the next round, fire first spread from the burned vertices, to its unburned neighbors, if it has any. In this case, the only vertex that is burned is the first source we picked in the first round. Then we are allowed to pick another vertex in the graph as our second source. Thus by the end of the second round, both the first source, the neighbors of the first source, and the second source are all burned. In later rounds, the process repeats, the fire first spreads from the vertices that are burned to all their unburned neighbors, then we proceed on picking another vertex as a new source. Once a vertex is burned, it cannot return to the unburned state. The fire will spreading from the sources to their neighbors, then to the neighbors of their neighbors.The process eventually stops when all the vertices in G are burned.

The **burning sequence** is the sources chosen, in order. The **burning number** of a graph G, $b(G)$, is the length of the shortest possible burning sequence of G, or the minimal number of rounds that is needed for G to be

[*]We like to thank Sean English, Mina Nahvi, Jingwei Xu and the IGL Program

burned entirely. The smaller the burning number of a graph, the faster a piece of information can spread in this represented network.

Figure 1: Below is a tree that can be burned in 5 rounds. The labeled red vertices are the sources that were picked, while the labels represent which round they were chosen.



Although by definition, the sources picked at each round can be any of the vertices of G, however, when we are striving for efficiency, there are certain rules that we would wish to follow. Every time we pick a source, it is the best if the vertex is unburned. This makes sense since picking a source is also an act of burning, and we do not wish to waste the opportunity of burning more unburned vertices. The only exception to this rule occurs if all of the vertices are burned after the spread of the fire, at this point, it does not matter which vertex we pick as our new source, since the graph is already fully burned. Moreover, it is intuitive to pick the vertices that have a high degree as our sources, however, this will depend on the nature of the specific graph that we are working with, and there can be exceptions to counter this intuition. [1]

As a measurement of the speed of the spread of information, there has been a strong interest in finding the upper bound for the burning number. First proposed in [2]:

Conjecture 1.1 (The Burning Number Conjecture)**.** *For a connected graph G of order n,*

$$b(G) \leq \lceil \sqrt{n} \rceil.$$

This conjecture is yet to be proved, however, it is known to be true for certain

[1]We will discuss the details of source picking for efficient graph burning later in this paper.

classes of graphs. [2] The current best bound for general graphs is presented in [3], where:

Theorem 1.2 (Bonato, Kamali, 2021)**.** *If G is a connected graph of order n, then*

$$b(G) \leq \left\lceil \frac{\sqrt{12n+64}+8}{3} \right\rceil = \left\lceil \sqrt{\frac{4n}{3}} \right\rceil + O(1)$$

In this paper, we aim for finding a minimal counter example for Burning Number Conjecture. We will first look into the Tree Reduction Theorem, which give us a tighter scope of what type of graph we should work with. We will also discussion certain lemmas which builds up on the Tree Reduction Theorem and help us to gain a better understanding of what characteristics a minimal counterexample should have for our situation. We will explain the idea of the k, ℓ burnability, and how it can be turned into a helpful algorithm and help us when studying graph burning. We will present the algorithm in Pseudo-code and talk about what we have done to optimize its run time while keeping an accurate functionality, as well as in what way we can utilize it when further studying graph burning.

2 Bound on Minimal Counterexample

We will dive into the close examination of the properties of a probable minimal counter example for the Burning Number Conjecture. Notice that we can limit our scope into trees:

Theorem 2.1 (Tree Reduction Theorem, [2])**.** *For a connected graph G we have that*

$$b(G) = min\{b(T) : T \textit{ is a spanning tree of } G\}$$

The **Tree Reduction Theorem** implies that if the burning conjecture is false, it is false for a tree, so we mainly focus on trees. In this project, we approach this problem by considering a tree that is a minimal counterexample, meaning it has the fewest number of vertices among all counterexamples. We came up with some bounds regarding the number of leaves and the length of the pendant path in a tree.

Since trees have unique properties that follows a certain pattern, we can develop some bounds on such properties. Let's first look at the upper bound of the leaves of a probable minimal counterexample.

Lemma 2.2. *If the tree T is a minimal counterexample, we can assume T has at most $2\sqrt{n}-1$ leaves.*

Proof. We prove Lemma 2.2 by contradiction. Let T be the tree with $n = b^2$ [3]number of the vertices and have more than $2\sqrt{n}$ leaves. Assume that T^* is T

[2]Note that paths realize the upper bound in the Burning Number Conjecture. That is, for any path P, $b(P_n) = \lceil\sqrt{n}\rceil$ for all n.

[3]We will use $\sqrt{n}$ many times in the proof, for convenience, we will substitute in b.

without its leaves. Thus once T^* is fully burnt, it only takes one more round for the $2\sqrt{n}$ leaves to be on fire, or, for T to be fully burnt. Thus we know that:

$$(b(T) \leq b(T^*) + 1$$

$$\textit{or}$$

$$b(T^*) \geq b(T) - 1$$

we also know that

$$\begin{aligned} n^* \leq n - 2\sqrt{n} &\leq n - 2\sqrt{n} + 1 \\ &= b^2 - 2b + 1 \\ &= (b-1)^2. \end{aligned}$$

Assume T is a minimal counterexample. Then we know that $b(T^*) \leq \lceil \sqrt{n^*} \rceil$ is true. Then

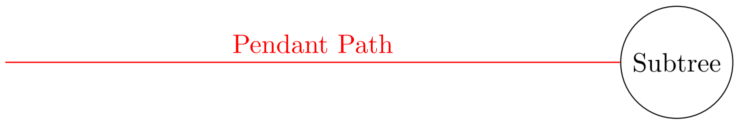
$$\begin{aligned} b(T) &\leq b(T^*) + 1 \\ &\leq \lceil \sqrt{n^*} \rceil + 1 \\ &\leq \lceil \sqrt{(b-1)^2} \rceil + 1 \\ &\leq b - 1 + 1 \\ &\leq b \\ &\leq \sqrt{n}. \end{aligned}$$

This contradicts with our definition for minimal counterexamples, thus, tree with more than $2\sqrt{n}$ leaves can not be a minimal counterexample. □

We can also look into certain unique structures that can be in a tree, and the limitations of a probable minimal counterexample regarding this structure.

Lemma 2.3. *If the tree T is a minimal counterexample, we can assume T has no pendant paths of length longer than $2\sqrt{n} - 1$.*

note: A pendant path of T is a path whose removal won't disconnected the tree. See below:



Proof. We prove Lemma 2.3 by contradiction. Take tree T that is a minimal counterexample with n vertices and with a pendant path p. Assume p has less than $\lceil 2\sqrt{n} \rceil - 1$ vertices. Take subtree T^* that is the part of T without p. Since T is a minimal counter example, T^* should comply with the conjecture, and

$b(T^*) \leq \lceil \sqrt{n^*} \rceil$, where n^* is the number of vertices in T^*. We know that for subtree T^*:

$$\begin{aligned} n^* &< n - (2\sqrt{n} - 1) \\ &< n - 2\sqrt{n} + 1 \\ &< (\sqrt{n} - 1)^2 \\ \sqrt{n^*} &< \sqrt{n} - 1 \\ \lceil \sqrt{n^*} \rceil &< \lceil \sqrt{n} \rceil - 1. \end{aligned}$$

Since T is a minimal counter example, then we know that $b(T) \geq \lceil \sqrt{n} \rceil$. From what we know about T^*:

$$\begin{aligned} b(T) &\geq \lceil \sqrt{n} \rceil \\ &\geq \lceil \sqrt{n^*} \rceil + 1 \\ &\geq b(T^*) + 1 \end{aligned}$$

Notice that, we can burn the path p of length $2k-1$ by a source in k rounds so that if the path has length shorter than $2k-1$, we can burn the path in at most 1 more round. Thus,

$$b(T^*) \leq \lceil \sqrt{n^*} \rceil \leq b(T) - 1$$

contradicted.

□

With these limitations in mind when we are trying to find a minimal counter example, we wonder if there are more characteristics that we can come up with. We came up with an coding algorithm that can simulate the burning process, and in the later section, we will explore how it optimizes the burning progress, and help us gain a better grasp on the characteristics of a potential minimal counter example.

3 (k, ℓ) - burnability

As we know, graph burning problem is NP-complete if some restrictions are given according to [1]. The burning conjecture $b(G) \leq \sqrt{n}$ means that there exists a sequence of vertices of length $\sqrt{n}$ that burns the whole graph. Its negation is that for all sequences of vertices of length $\sqrt{n}$, they cannot burn the graph. Thus, to decide this naive run-time of checking if $b(G) \leq \sqrt{n}$, we want to find the number of sequences and the time each sequence takes. The number of vertices we can choose as our first source is n; the number of vertices we can choose as our second source is $n-1$; since the length is $\sqrt{n}$, the number of sequences is

$$\leq \underbrace{n \cdot n-1 \cdot n-2 \cdot ......}_{\sqrt{n}} = \prod_{i=1}^{\sqrt{n}} (n - i + 1) \leq n^{\sqrt{n}}.$$

To check if every vertex get burned, we can check the distance from the vertex to all the sources, which should have the run-time 1. Because each sequence has n vertices to check and there are $\sqrt{n}$ sources for each vertex to check the distance from, the time each sequence of vertices takes should be $n \cdot \sqrt{n}$. So, the naive run-time for checking if $b(G) \leq \sqrt{n}$ is

$$O(n^{\sqrt{n}} \cdot n \cdot \sqrt{n}) = O(n^{\sqrt{n}+1+1/2}).$$

So, we want to consider a computationally easier problem.

Given a tree T, if it can be burned after we pick k sources and let the graph burn for ℓ rounds, then T is (k, ℓ) - burnable. A graph G is (k, ℓ) - burnable if there exists a sequence of vertices $(v_1, v_2, \ldots, v_k)$ such that if v_i burns all vertices at distance $\ell - i$, then graph G is burned. The first source v_1 will burn for $\ell - 1$ rounds; the second source v_2 will burn for $\ell - 2$ rounds; the ith source v_i will burn for $\ell - i$ rounds, where $1 \leq i \leq k$. This is equivalent to the usual burning with the sequence $(v_1, v_2, \ldots, v_k)$, and then letting the fire propagate for $\ell - k$ times. In this (k, ℓ)-burning process, we assume that $\ell \geq k$. We make this assumption because each time we choose a source, the graph will burn for one more round, then after we are done placing the sources, the fire can still spread $\ell - k$ rounds.

There are k sources that need to be chosen. For each source, we have n choices, hence the run-time is $O(n^k)$. Since every source will burn for at most ℓ rounds, the total run-time for the algorithm is $O(\ell \cdot n^k)$. This means that it mainly depends on k. Since the naive run-time for checking burning conjecture is $O(n^{\sqrt{n}+1+1/2})$, which has the leading coefficient $\sqrt{n}$, we can make the coefficient term $k \ll \sqrt{n}$. So, instead of directly asking if a graph is $(\sqrt{n}, \sqrt{n})$-burnable (which corresponds to the burning number conjecture), we came up with the concept of (k, ℓ)-burning, which allows us to reduce time complexity. By choosing smaller k (polynomial coefficient) and larger ℓ (constant coefficient), when $k \ll \ell$, we can determine if a graph is (k, ℓ)-burnable in less time.

4 k, ℓ - burnable algorithm

In order to test whether graphs are k, ℓ burnable, we implemented a series of algorithms. The pseudo code are attached in the appendix. This section will give a brief introduction about the design idea of our algorithms, the challenge we met and the experiment and result.

Design idea

The algorithm will take a networkx-graph G, k, and ℓ as input and its output are a boolean value that tells us whether this graph is k, ℓ burnable. Generally, we reproduce the process of burning the graph. That is, for each round, we let all burned vertices to burn (by Algorithm 2 Update Burned Nodes) and pick one source to burn at the end of each round. We set the condition k, ℓ, which is explained in the previous section. That means while we let the graph burn for ℓ rounds, we only pick sources in the first k rounds. When $k = \ell = \sqrt{n}$, it is the original burning process. To make our algorithm run better, instead of choosing any unchosen vertex as source in every round, we pick ith source by excluding the vertices which have been burned until ith round and whose distance to the leaves is bigger than $\ell - i$ $(i = 1, 2, \ldots, k)$. As a result, if the graph is k, ℓ burnable, it will suddenly return, and if it is not, it will walk through all possibilities and return a burning sequence which can burn most vertices among all possible burning sequence.

Challenge

While it may seem easy to implement the algorithm, the real situation involves dynamic structure and many edge cases.

Implement Dynamic structure

By the design idea, different source picked at previous round will lead to different sets of available sources at the current round. Therefore, we implemented a dynamic *depth-first-search* on the collection of all possible "reasonable" burning sequences. We will record available sources at each round by a list of sublists R, vertices distance by a list of sublists L, and new burned vertices at each round by sublist burnedrecord$[i]$. With these information, we can *move down* and generate $R[i+1]$ by applying the action sequence "pick the first node in $R[i]$ as source and record it to our array source, then remove the first node from $R[i]$ (for $i = 1, \ldots, k$)". Our algorithm can *move up* to $R[i-1]$ by removing all the nodes burned at ith round and removing the $(i-1)$st source , then go back to $(i-1)$st level and generate the new $R[i]$.

Edge cases

Besides the dynamic structure, there are also many edge cases in real situations: easy-burn graph, burning too fast, and generating an empty list $R[i]$. We take different strategies to deal with these problems.
For easy-burn graphs, we make judgement after getting the List L. There are two cases that needs further consideration. The first case is, the first sublist of L has only one node. That means we can burn the whole graph by picking only one source and let it burn ℓ rounds. The second case is, the first sublist of L is empty. That means we can burn the whole graph by picking the source(s) from the first non-empty sub-list. Since the input of algorithm is a tree, when there is empty layer above, the lower layer below the empty layer can have at most 2 vertices. It is determined by our "update L" algorithm: each layer of L should contains the vertices that can build a tree. If the layer that below the empty

layer has more than two vertices, we can always remove two vertices and move the rest vertices up to the higher layer.
For graphs that may generate an empty list $R[i]$, we will walk through $L[j]$ to produce $R[i]$ for $j = i+1, i+2, ..., \ell$. If $R[i] = L[i]/B[i]$ is empty, it means that the graph has burned all the vertices at level i. Hence, we can move forward to lower level until all the nodes have been burned. This adjustment can make sure we will not waste the opportunity to burn one more vertex at this round.
For graphs burn too fast, we set check points at each round to save time when burn the graph before picking the kth source or burning the ℓth round.

Modification to accelerate the burning process

To make the algorithm burn faster when the graph is k, ℓ burnable, we add priority weight when picking up the source: the source with higher degree will be chosen earlier.

Experiment and Result

We use the dataset from this website[3], which contains 303 graphs, and run with our algorithm.
When we set $k = \ell = \sqrt{n}$, all the connected graphs in this dataset is k, ℓ burnable. However, to extract useful information, we try to constrain our condition and find graphs hard to be burned. Most graphs are still burnable under this condition, but a few cases failed.

5 Future Thoughts

There are still many open questions about graph burning. While all the graphs in the dataset are burnable, we cannot prove the conjecture just by that. Due to the limitation of the size of the dataset, it is still hard to analyze the underlying properties of these hard-to-burn graphs. To solve this problem, we can find a larger dataset such that we can extract information or apply a random graph generator to test which graphs are burnable.

A Algorithms

Algorithm 1 Get L

```
Input: $G, N, \ell$
Output: $L$, canburned
$L \leftarrow empty\ list$
$i \leftarrow 0$
while $i < \ell - 1$ do
    $G \leftarrow delete\ leaves$
    $L[i] \leftarrow all\ nodes\ of\ G$
end while
$ReverseL$

$canburned \leftarrow False$
if $len(L[0]) = 0\ or\ len(L[0]) = 1$ then
    $canburned \leftarrow True$
end if
return $L, canburned$
```

Algorithm 2 Update Burned Nodes

```
Input: $G$, burned
Output: burned
burned $\leftarrow$ all neighbours of nodes in burned and all nodes in burned
return burned
```

Algorithm 3 Update R

```
  **Input:** $k$, $L$, $G$, $i$, burned, source, burnedrecord, $R$
  **Output:** burned, source, burnedrecord, $R$
if len($R[i]$) = 0 then
    Delete $R[i]$
    Delete burnedrecord[$i$]
    remove $i-1th$ source from burnedrecord
    burned ← all nodes in burnedrecord
    return burned, source, burnedrecord, $R$
end if

pick source
update burned and burnedrecord
$R[i+1] \leftarrow L[i]/B[i]$
num ← $i+1$
while $R[i+1]$ is empty and nums $< \ell - 1$ do
    $R[i+1] \leftarrow L[num]/B[i]$
    num ← num + 1
end while

if $i < k-2$ then
    call Update $R$
end if
```

Algorithm 4 Isklburnable

```
  Input: $G, k, \ell$
  Output: source, burned, isburnable
N ← number of nodes of $G$
burned ← empty list
source ← empty list
$R$ ← empty list
number ← 0
bestsequence ← empty list
if $k = 1$ then
    update source, burned
    if $N = 1$ then
        return source, burned, True
    end if
    return source, burned, False
end if
$L$, tellburnfromL ← GetL
if tellburnfromL = True then
    update source, burned
    return source, burned, True
end if
burned source, $R$ ← UpdateR        ▷ generate $R[i]$ and burn $i$ rounds for $i = 0, 1, \ldots, k-2$
while $len(R) \neq 0$ do      ▷ not all possible matches of sources are touched on
    while $len(R[k-1]) \neq 0$ do   ▷ not all possible last source are touched on
        update source, burned, $R$
        if len(burned) = $N$ then                   ▷ burned the whole graph
            return source, burned, True
        end if
        update number, bestsequence
    end while
    call UpdateR        ▷ return to higher level to pick other possible source
    if len(burned) = $N$ then                       ▷ burned the whole graph
        return source, burned, True
    end if
end while
return source, bestsequence, False     ▷ fail to burn the whole graph, return best way
```

References

[1] A. Bonato, J. Janssen, and E. Roshanbin. Burning a graph is hard, 2015.

[2] A. Bonato, J. Janssen, and E. Roshanbin. How to burn a graph. *Internet Math.*, 12(1-2):85–100, 2016.

[3] A. Bonato and S. Kamali. An improved bound on the burning number of graphs, 2021.