Hypergraph Burning

by

© **Caleb Jones**

A thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Master of Science.

Department of Mathematics and Statistics

Memorial University

November 2023

St. John's, Newfoundland and Labrador, Canada

Abstract

Graph burning is a combinatorial game or process that models the spread of influence throughout a network. We introduce a generalization of graph burning which applies to hypergraphs. One of our key results is that arbitrary hypergraphs do not satisfy a bound analogous to the one in the Burning Number Conjecture. We also introduce a variant called "lazy" hypergraph burning, along with a new parameter, the lazy burning number. Interestingly, lazily burning a graph is trivial, while lazily burning a hypergraph can be quite complicated. Moreover, the lazy burning model is a useful tool for analyzing the round-based model. We obtain bounds on the burning number and lazy burning number of a hypergraph in terms of its parameters, as well as stronger bounds that apply to Steiner triple systems. We also discuss the complexity of both the round-based and lazy models. Finally, we introduce two further variants of (lazy) hypergraph burning.

Lay summary

A *graph* is a collection of points (called *vertices*) and lines (called *edges*) where each line connects two specific points. Graphs are useful for describing many different concepts. A helpful analogy is as follows: the vertices can represent people, and if two people are friends then their corresponding vertices are connected by an edge (these vertices are *adjacent*). *Graph burning* is a well-studied game or process that is played on graphs in which vertices are set on fire, and the fire spreads throughout the graph via edges. If we let the graph represent a social network as in our analogy, then the fire can represent a "social contagion" spreading throughout the network. The game takes place over a sequence of rounds, and there is a player called the *arsonist*. In each round, the arsonist manually burns a vertex while the fire spreads or *propagates* from burned vertices to adjacent unburned vertices. The *burning number* of a graph is the earliest round at which it could be completely burned, and the existing research is mostly concerned with finding this number. In this thesis, we extend the rules of graph burning so that it can be played on a *hypergraph*. A hypergraph is like a graph, except the edges can now connect any number of vertices (notice that every graph is a hypergraph). A hypergraph can also represent a social network, where edges of size two represent friendships and larger edges represent social media friend groups – collections of people who are not necessarily all acquainted, but who nonetheless spread information to one another. Since hypergraphs can model social networks in greater detail than graphs, they should be able to model the spread of a "social contagion" in more detail as well. We therefore introduce a model called *hypergraph burning*, as well as a related model with an alternate set of rules called *lazy hypergraph burning*. For each version of the game, we prove several results related to optimal play in a general hypergraph as well as some specific classes of hypergraphs. We implement code that computes the (lazy) burning number of a hypergraph, and apply it to several classes of hypergraphs. Since the main innovation in hypergraph burning is the way the fire propagates within an edge of size three or greater, we also introduce two alternative propagation rules that could be used to make hypergraph burning a more accurate model of different scenarios.

Acknowledgements

I would like to acknowledge the help of both of my supervisors, whose expertise was invaluable. They directed the research and contributed to the proofs of multiple results. They also suggested inclusions that would enhance the thesis, and provided editorial comments on the manuscript, such as how best to adhere to mathematical conventions. The idea for the proof of Theorem 3.2.7 arose from a discussion with Trent Marbach and Peter Danziger (Toronto Metropolitan University), and the remainder of Chapter 3 is inspired by this idea. This thesis was funded by an NSERC CGS M scholarship, an AARMS graduate scholarship, and baseline funding from the MUN School of Graduate Studies.

Statement of contribution

My supervisors (Dr. Andrea Burgess and Dr. David Pike) directed the research, offered expertise on which avenues would be best to explore, and conceived some key results. I (Caleb Jones) wrote the manuscript and code (found in sections A.2 and A.1), and conceived and proved several results either independently or with the help of my supervisors. The idea for Theorem 3.2.7 (as well as the algorithm and lemma used in its proof) arose from a discussion with Trent Marbach and Peter Danziger at the summer 2022 CMS conference.

Table of contents

List of tables

List of figures

Chapter 1

Introduction

A *graph* is a mathematical object which consists of a set of *vertices* and a set of *edges*, where each edge connects a pair of vertices. Graphs can be used to model anything with a network-like structure, for example, a social network. The vertices can represent people, and if two people are friends then their corresponding vertices can be connected by an edge. *Graph burning* is a combinatorial game or process that models the spread of influence throughout a network. We use the concept of a fire as a stand-in for the influence that is being spread. Throughout the game, vertices of the graph catch on fire and the fire spreads via edges. If we view the graph as a model for a social network, and the fire as a piece of misinformation, then the game can serve as a model for the spread of misinformation throughout a network of people.

A *hypergraph* is a mathematical object that is much like a graph, except edges can now connect any number of vertices, not just two. In general, hypergraphs are a better model for a network than graphs, because every graph is also a hypergraph. Indeed, a hypergraph can model a social network in greater detail, as edges of size two can represent one-on-one friendships, and larger edges can represent *social media friend groups*. In this thesis, we extend the rules of graph burning so that it can be played on a hypergraph, thus creating a better model for the spread of influence throughout a network. For example, this new game is a better model for the spread of misinformation in a social network, as it accounts for the spread of misinformation via social media friend groups. We will call this new game *hypergraph burning*.

1.1 Graph Theory

The topic of this thesis is an extension of a graph-theoretic process known as graph burning. We therefore provide a brief review of the basic definitions and concepts in graph theory. See [39] for further reference.

A *graph* G is an ordered pair (V, E) where V and E are disjoint finite sets with $V \neq \emptyset$, together with an *incidence function* that maps each element of E to an unordered pair of (not necessarily distinct) elements in V. The elements of $V = V(G)$ are called *vertices* and the elements of $E = E(G)$ are called *edges*. The two vertices associated with an edge e via the incidence function are called the *endpoints* of e. The *order* of the graph is $|V(G)|$ and the *size* is $|E(G)|$.

A *loop* in a graph is an edge whose endpoints are equal. *Multiple edges* are edges having the same pair of endpoints. The *multiplicity* of an edge e is the number of edges with the same endpoints as e, including e itself. A graph is *simple* if it has no loops or multiple edges. In a simple graph we treat each edge as an unordered pair of vertices and write $e = uv$ (or $e = vu$) for an edge e with endpoints u and v.

When u and v are the endpoints of an edge, they are *adjacent* to one another and are called *neighbours*. If vertex v is an endpoint of edge e, then v and e are *incident*. The *degree* of vertex v is the number of non-loop edges incident to v, plus twice the number of loops on v. The *open neighbourhood* of v, denoted $N(v)$, is the set of neighbours of v. The *closed neighbourhood* of v is $N(v) \cup \{v\}$, denoted $N[v]$. By convention, if we simply refer to the "neighbourhood of v," then we mean the open neighbourhood. If $S \subseteq V(G)$, then the *closed neighbourhood* of S is the set of all vertices that are either in S or adjacent to a vertex in S; it is denoted $N[S]$. The maximum degree of all vertices in G is denoted $\Delta(G)$, and the minimum degree is denoted $\delta(G)$. If every vertex in G has degree k, then G is *k-regular*.

A *clique* in a graph is a set of pairwise adjacent vertices. A *complete graph* is a simple graph whose vertices form a clique. The complete graph on n vertices is denoted K_n. An *independent set* in a graph is a set of pairwise nonadjacent vertices.

A graph G is *bipartite* if $V(G)$ can be partitioned into two *partite sets* A and B such that no two vertices in the same partite set are adjacent. Notice that A and B are both independent sets of vertices. A *complete bipartite graph* is a simple bipartite graph such that two vertices are adjacent if and only if they are in different partite sets. When the sets have sizes r and s, the complete bipartite graph is denoted $K_{r,s}$.

A *path* is a simple graph whose vertices can be listed in order and without repetition so

that two vertices are adjacent if and only if they are consecutive in the list. The path on n vertices is denoted by P_n. A (u, v)-*path* is a path whose vertices of degree 1 (its *endpoints*) are u and v. The non-endpoint vertices in a path are called *internal vertices*. A *cycle* is a graph with an equal number of vertices and edges whose vertices can be placed around a circle so that two vertices are adjacent if and only if they appear consecutively along the circle. The cycle on n vertices is denoted C_n, and is often called an n-cycle.

A *walk* is a list $v_0, e_1, v_1, \ldots, e_k, v_k$ of vertices and edges (not necessarily distinct) such that for $1 \leq i \leq k$, the edge e_i has endpoints v_{i-1} and v_i. A *trail* is a walk with no repeated edge. A (u, v)-*walk* or (u, v)-*trail* has first vertex u and last vertex v; these are its *endpoints*. The *length* of a walk, trail, path, or cycle is its number of edges. A walk or trail is *closed* if its endpoints are the same. A closed trail is called a *circuit*.

If $u, v \in V(G)$ then the *distance* between u and v, denoted $d(u, v)$, is the length of a shortest path between u and v. For a nonnegative integer k and a vertex v, the k^{th} *closed neighbourhood* of v is the set $N_k[v] = \{u \in V(G) \mid d(u, v) \leq k\}$. Of course, $N_0[v] = \{v\}$ and $N_1[v] = N[v]$.

A *subgraph* of a graph G is a graph H such that $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$, and the assignment of endpoints to edges in H is the same as in G. We then write $H \subseteq G$ and say that G *contains* H. A subgraph H of G is *induced* by a set of vertices $T \subseteq V(G)$ if H was obtained by deleting all vertices not in T from G (as well as deleting all edges with one or more endpoints not in T from G). We then write $H = G[T]$. We write $G - e$ or $G - M$ for the subgraph of G obtained by deleting an edge e or set of edges M. We write $G - v$ or $G - S$ for the subgraph obtained by deleting a vertex v or set of vertices S (along with all the edges incident to a vertex in S). A subgraph H of G is said to be a *spanning subgraph* of G if $V(G) = V(H)$, so H was obtained from G by deleting only edges.

A graph G is *connected* if there exists a (u, v)-walk for any two vertices $u, v \in V(G)$; otherwise, G is *disconnected*. The *connected components* of a graph G are its maximal connected subgraphs. An *isolated vertex* is a vertex of degree 0.

The *girth* of a graph with a cycle is the length of its shortest cycle. A graph with no cycle has infinite girth, and if it is connected it is called a *tree*. A vertex of degree 1 in a tree is called a *leaf*. A *spanning subtree* T of a graph G is a subgraph of G that is a tree and satisfies $V(T) = V(G)$. A *spider* is a tree with exactly one vertex of degree strictly greater than two. A *caterpillar* is a tree that contains a path P such that every edge in the tree is either contained in P, or incident to a vertex of P.

A *rooted tree* is a tree in which one vertex has been designated as the *root*. The *depth* of

a vertex v in a rooted tree is the distance between v and the root. The *height* of a rooted tree is the maximum depth among vertices in the tree. A *rooted tree partition* of a graph G is a collection of rooted trees which are subgraphs of G, and whose vertex sets partition $V(G)$.

An *isomorphism* from a simple graph G to a simple graph H is a bijection $f : V(G) \to V(H)$ such that $uv \in E(G)$ if and only if $f(u)f(v) \in E(H)$. We say G is *isomorphic* to H and write $G \cong H$. A graph G is *H-free* if G has no induced subgraph isomorphic to H. An *automorphism* of G is an isomorphism from G to itself. A graph G is *vertex-transitive* if for every pair $u, v \in V(G)$ there is an automorphism of G that maps u to v.

1.2 Hypergraph Theory

We give a brief overview of the foundational definitions in hypergraph theory, mostly using [4], [16], and [21] for reference.

A *hypergraph* H is an ordered pair (V, E) where V and E are disjoint finite sets with $V \neq \emptyset$, together with an *incidence function* $\varphi : E \to \mathcal{P}(V) \setminus \{\emptyset\}$. The elements of $V = V(H)$ are called *vertices* or *points*, and the elements of $E = E(H)$ are called *edges* or *hyperedges*. The *order* of H is $|V(H)|$ and the *size* of H is $|E(H)|$. A hypergraph with a single vertex is called *trivial*, and a hypergraph with no edges is called *empty*.

Informally, a hypergraph is like a graph, except edges can contain any number of vertices, not just two. In the definitions that follow, let $H = (V, E)$ be a hypergraph, and write $V(H) = \{v_1, ..., v_n\}$ and $E(H) = \{e_1, ..., e_m\}$. When referring to two arbitrary vertices in H, we will call them u and v for the sake of simplicity.

Two edges e_1 and e_2 are *parallel* if $\varphi(e_1) = \varphi(e_2)$. The number of edges parallel to some edge e, including e itself, is the *multiplicity* of e. A hypergraph is called *simple* if no edge has multiplicity greater than 1 (i.e. φ is injective), and no edge contains one or fewer vertices. An edge containing one vertex is called a *loop*. A hypergraph is *linear* if any two distinct edges intersect in at most one vertex.

Note that we mostly consider simple hypergraphs, as parallel edges and singleton/empty edges have no effect on the burning game on hypergraphs (whose rules will be defined shortly in Section 1.4). Thus, E may be viewed as a subset of $\mathcal{P}(V) \setminus \{\emptyset\}$ (containing no singletons) by identifying each edge $e \in E$ with its image under φ. Since the incidence function is rather cumbersome, even when dealing with non-simple hypergraphs we will consider each edge to be a subset of V, in which case E is a multiset that can contain singletons.

If there is some $e \in E$ such that $u, v \in e$, then u and v are *adjacent* in H and are called *neighbours*. Similarly, if $d, e \in E$ are distinct and there is some $v \in V$ such that $v \in d \cap e$, then d and e are *adjacent* in H. The *degree* of a vertex $v \in V$ is the number of edges e such that $v \in e$; it is denoted $\deg_H(v)$, or $\deg(v)$ when H is clear from context. For example, in Figure 1.1, the vertices b, c, f, and i all have degree two, and the rest have degree one. A vertex of degree zero is called *isolated*. A hypergraph is called *r-regular* if each of its vertices has degree r, and *k-uniform* if each of its edges contains exactly k vertices.

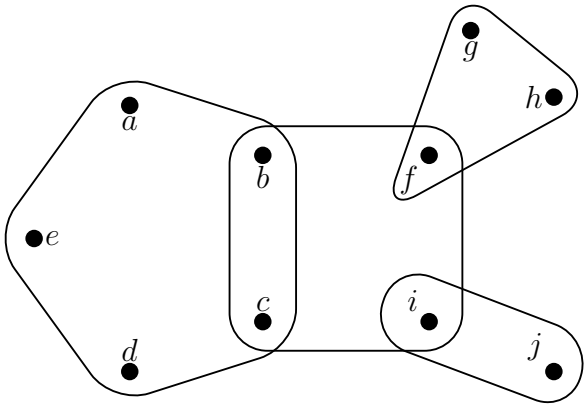

Figure 1.1: A hypergraph with edges $\{a, b, c, d, e\}$, $\{b, c, f, i\}$, $\{f, g, h\}$, and $\{i, j\}$.

The *incidence matrix* of H is the $n \times m$ matrix where the entry in row i, column j is 1 if $v_i \in e_j$, and 0 otherwise. The incidence matrix of a hypergraph is unique up to permutation of rows and columns.

A hypergraph $H' = (V', E')$ is called a *weak subhypergraph* (or just *subhypergraph*) of H if $V' \subseteq V$ and either $E' = \emptyset$, or, after a suitable permutation of its rows and columns, the incidence matrix for H' is a submatrix of the incidence matrix of H. Thus, each edge $e' \in E'$ has $e' = e \cap V'$ for some $e \in E$ (so H' may not be simple). If, in addition, $E' = \{e \cap V' \mid e \in E, e \cap V' \neq \emptyset\}$, then H' is said to be *induced* by V' (that is, all nonempty edges $e \cap V'$, singletons included, are present in E').

A hypergraph $H'' = (V'', E'')$ is called a *strong subhypergraph* (or *hypersubgraph*) of H if $V'' \subseteq V$ and $E'' \subseteq E$. If all edges induced by V'' (i.e. all $e \subseteq V''$) are present in E'' then H'' is said to be *induced* by V'', and we write $H'' = H[V'']$. If $V'' = \cup_{e \in E''} e$ then H'' is said to be *induced* by E'', and we write $H'' = H[E'']$. Note that every strong subhypergraph of H is also a weak subhypergraph of H, but the converse is not necessarily true.

A *(u, v)-walk of length k* in H is an alternating sequence $v_1 e_1 v_2 e_2 \ldots v_{k-1} e_{k-1} v_k$ of vertices and edges such that $v_i \in V(H)$ and $e_i \in E(H)$ for each i, $v_1 = u$, $v_k = v$, and for all $i = 1, 2, \ldots, k-1$, the vertices v_i and v_{i+1} are adjacent in H via the edge e_i. The *end*

vertices of W are $u = v_1$ and $v = v_k$, and the *internal vertices* of W are $v_2, \ldots, v_{k-1}$. If $v_1, \ldots, v_k$ are pairwise distinct and $e_1, e_2, \ldots, e_{k-1}$ are pairwise distinct (parallel edges among the e_i are allowed) then W is called a *path* (or sometimes a *Berge path*).

If $V(H) = \{v_1, \ldots, v_n\}$ and $E(H) = \big\{\{v_1, \ldots, v_k\}, \{v_2, \ldots, v_{k+1}\}, \ldots, \{v_{n-k+1}, \ldots, v_n\}\big\}$ then H is called a k-uniform *tight path*. If $V(H) = \{v_1, \ldots, v_n\}$ and $E(H) = \big\{\{v_1, \ldots, v_k\}, \{v_k, \ldots, v_{2k-1}\}, \ldots, \{v_{n-k+1}, \ldots, v_n\}\big\}$, then H is called a k-uniform *loose path*.

Vertices $u, v \in V(H)$ are *connected* if there exists a (u, v)-walk in H; otherwise u and v are *separated* from one another. The hypergraph H is *connected* if every pair of distinct vertices are connected in H; otherwise H is *disconnected*. For example, in Figure 1.2, H is connected and H_1 is disconnected. A *connected component* of H is a maximal connected weak subhypergraph of H. Denote the number of connected components of H by $c(H)$. By the maximality condition, connected components of a hypergraph H are also strong subhypergraphs of H.

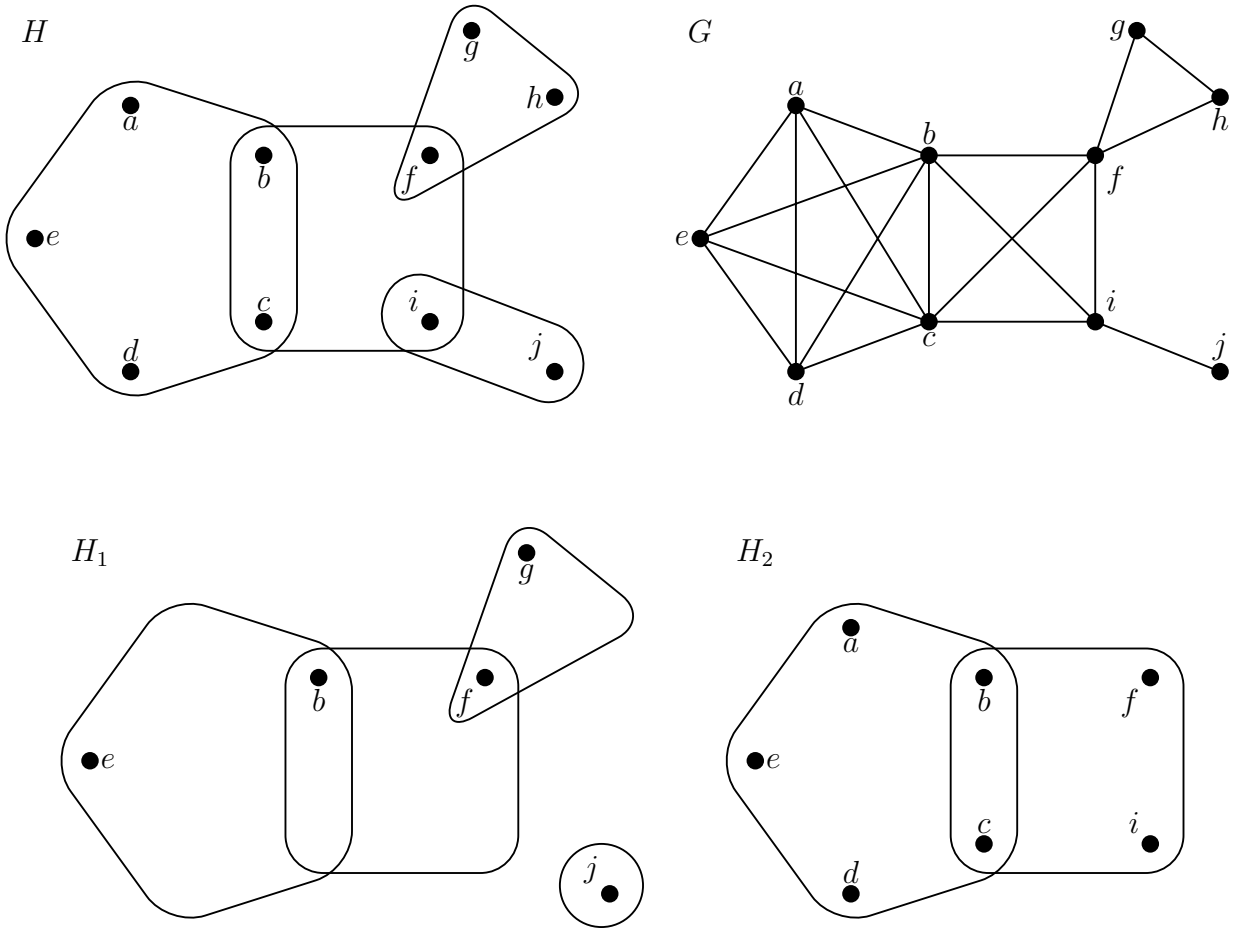

Figure 1.2: A hypergraph H, its 2-section G, the weak subhypergraph H_1 of H induced by $V_1 = \{b, e, f, g, j\}$, and the strong subhypergraph H_2 of H induced by $V_2 = \{a, b, c, d, e, f, i\}$.

The 2-*section* of a hypergraph H is the graph G on the same vertex set whose edges are precisely those of the form $\{u, v\}$ such that $\{u, v\} \subseteq e$ for some $e \in E(H)$. Thus, every edge $e = \{v_{i_1}, ..., v_{i_k}\}$ in H induces a clique among the vertices $v_{i_1}, ..., v_{i_k}$ in G. If H is connected (respectively, disconnected), then its 2-section is connected (respectively, disconnected) as a graph.

A set of vertices $\{x_1, x_2, \ldots, x_k\} \subseteq V(H)$ is called *independent* if there is no edge e in H such that $e \subseteq \{x_1, x_2, \ldots, x_k\}$. The size of a largest independent set in H is the *independence number* of H, denoted $\alpha(H)$. Note that there are multiple definitions of independence for hypergraphs, but we will use only this definition throughout.

An *isomorphism* from a simple hypergraph G to a simple hypergraph H is a bijection $f : V(G) \to V(H)$ such that $e \in E(G)$ if and only if $f(e) \in E(H)$, where $f(e) = \{f(v) \mid v \in e\}$. We say G is *isomorphic* to H and write $G \cong H$. An *automorphism* of a hypergraph H is an isomorphism from H to itself. One may view an automorphism of H as a permutation of the vertices of H. A hypergraph H is *cyclic* if there exists an automorphism of H that is a permutation consisting of a single cycle of length $|V(H)|$.

1.3 Graph Burning

Graph burning is a single-player game played on finite, simple, undirected graphs over a discrete sequence of rounds. The player, whom we call the *arsonist*, attempts to set fire to every vertex of the graph in as short a time as possible. The arsonist manually sets fire to vertices, and the fire also spreads or *propagates* from burned vertices to adjacent unburned vertices in each round. Once a vertex is set on fire, it remains on fire until the end of the game. Of course, the game ends when every vertex in the graph is on fire.

A problem equivalent to a special case of graph burning was posed in 1992 by Brandenburg and Scott as an internal problem at Intel [3]. Their model only applied to the n-cube, whose vertices were called *processors*. Instead of the arsonist, the player was called the *sender*, and the sender's goal was to transmit a message to every processor as fast as possible. In each round, the sender "sends the message" to a specified vertex, and simultaneously each processor that "knows the message" sends it to each of its neighbours. Hence, this process was equivalent to burning the n-cube.

In 2014 and independently from [3], A. Bonato, J. Janssen, and E. Roshanbin [13] introduced graph burning as a combinatorial process which could be applied to any graph. They introduced the term "graph burning" for the first time, and developed much of the theory.

We now describe the rules for graph burning in detail. Rounds are indexed by $\mathbb{N}$ starting at one. Denote the set of vertices that are on fire at the end of round r by F_r and write $f_r = |F_r|$. Thus, $F_r \subseteq F_{r+1} \subseteq V(G)$ and $f_r \leq f_{r+1} \leq |V(G)|$ for each $r \in \mathbb{N}$. By convention, write $F_0 = \emptyset$. During each round $r \geq 1$, the following two things happen simultaneously.

- For each $v \in V(G) \setminus F_{r-1}$, if there is $u \in F_{r-1}$ such that $vu \in E(G)$ then v catches fire.
- The arsonist chooses a vertex $u_r \in V(G) \setminus F_{r-1}$ and sets it on fire (u_r is called a *source*).

Thus $F_r = N[F_{r-1}] \cup \{u_r\}$. Note that the arsonist may choose as a source a vertex that also catches fire due to propagation that round, although it is never advantageous to do so. We will call such a source *redundant*. A redundant source that is burned in round r is in $V(G) \setminus F_{r-1}$, and would catch fire in round r due to propagation if it were not burned as a source. In round 1, no vertices catch fire due to propagation, and the arsonist chooses the first source, so $F_1 = \{u_1\}$ and $f_1 = 1$. If the arsonist chooses a non-redundant source in round 2 then $f_2 = |N[u_1]| + 1$. In the last round the arsonist may have no choice but to choose a redundant source. This occurs when all of the remaining unburned vertices will catch fire in the final round due to propagation; the arsonist must still choose a source, and the only options are all redundant.

Of course, the arsonist is of our own creation, and given a graph, *we* are the ones trying to find an optimal sequence of sources. Two problems can arise if we decide on the sequence of sources we want to burn ahead of time. First, it is possible that the n^{th} source in our pre-determined sequence catches fire in round $n-1$ or earlier. Thus, once we get to the n^{th} round, we cannot follow through on our pre-determined sequence of sources, because it requires us to burn as a source a vertex that has been on fire for one or more rounds (such invalid sources are not to be confused with redundant sources). The other problem that can arise is that the sequence may end up being "too long." That is, after following the sequence for n rounds, the graph is fully on fire, but the sequence continues on with an $(n+1)^{th}$ source, and possibly more. Again, this would mean our pre-determined sequence prompts us to burn as a source a vertex that has been on fire since the previous round or earlier, which is not allowed. We therefore say that a source is *legal* if it is manually burned in round r, and it is not in F_{r-1}. Otherwise, a source is *non-legal*. Note that every redundant source is also legal. If every source in a sequence is legal, we will call the sequence *valid*. Note that it is completely fine for a sequence of sources to unsuccessfully burn the graph. That is, if the graph is not completely burned after burning the final source, the sequence can still be valid, although unsuccessful.

A valid sequence of sources $(u_1, u_2, \ldots, u_k)$ that leaves the graph completely burned when the arsonist burns u_i in round i is called a *burning sequence*. When we wish to emphasize G and/or the burning sequence S we may sometimes write F_r as $F(G, S, r)$ and f_r as $f(G, S, r)$. Also define $\psi(G, r) = \min\{f(G, S, r) \mid S \text{ is a valid burning sequence in } G\}$ and $\Psi(G, r) = \max\{f(G, S, r) \mid S \text{ is a valid burning sequence in } G\}$. Write these as ψ_r and Ψ_r respectively when G and/or S are obvious. Note that none of F_r, f_r, $\psi(G, r)$, $\Psi(G, r)$, or the term *redundant* can be found elsewhere in the literature, as they were introduced for use in this thesis. Also, when referring to non-source vertices catching fire through propagation, we will often omit the words "through propagation" for the sake of brevity.

A burning sequence of minimum possible length is called *optimal*. A graph may have many different optimal burning sequences. Given a graph G, the *burning number* of G, denoted $b(G)$, is a measure of how fast the fire can possibly spread to all vertices of G.

Definition 1.3.1. *The* burning number *of G, denoted $b(G)$, is the length of an optimal burning sequence.*

Note that $b(G)$ is also equal to the earliest round at which G could become completely burned, since the number of rounds coincides with the number of sources that are chosen. See Figure 1.3 for an example of the burning game being played on a graph G. The burning sequence $S = (a, b, c)$ (where c is a redundant source) is indeed optimal, since there are no burning sequences of length two. To see this, suppose (x, y) is a burning sequence for G for some vertices $x, y \in V(G)$. At the end of the game, the vertices that are on fire are exactly those in $F_2 = N[x] \cup \{y\}$. But there are no two vertices $x, y \in V(G)$ such that $N[x] \cup \{y\} = V(G)$, so in fact (x, y) is not a burning sequence for G. Thus, $b(G) = 3$.

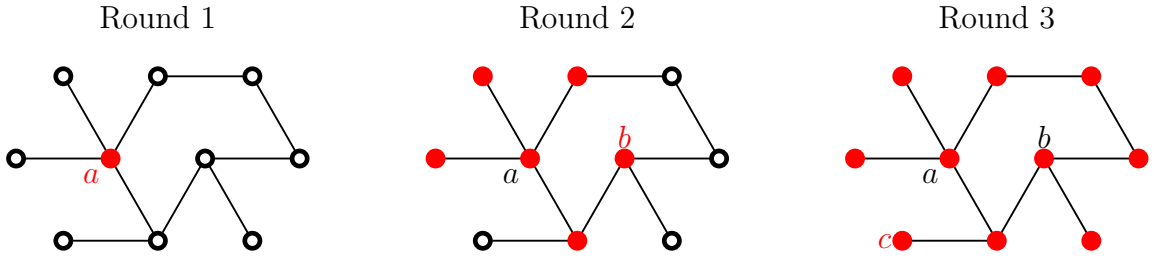

Figure 1.3: An optimal burning sequence in a graph.

As stated above, a burning sequence must satisfy two criteria: each source must be legal, and the sequence must leave the entire graph burned by the end. The following result captures these two criteria using precise mathematical language. Note that it is not presented as a lemma in [14], but it is rather a part of the discussion.

Lemma 1.3.2. ([14]) *Let G be a graph and $k \in \{1, 2, \ldots, |V(G)|\}$. Then $(x_1, x_2, \ldots, x_k)$ is a burning sequence for G if and only if both of the following hold:*

(a) For each pair $i, j \in \{1, 2, \ldots, k\}$ with $i < j$, $d(x_i, x_j) \geq j - i$.

(b) $N_{k-1}[x_1] \cup N_{k-2}[x_2] \cup \cdots \cup N_1[x_{k-1}] \cup N_0[x_k] = V(G)$.

Proof. Let $(x_1, x_2, \ldots, x_k)$ be a burning sequence for G. After a source x_i is burned in the i^{th} round, the fire will propagate $k - i$ more times since there are $k - i$ rounds remaining. Therefore, a source x_i will cause every vertex within distance $k - i$ from itself (i.e. all vertices in $N_{k-i}[x_i]$) to burn by the end of the k^{th} round. Indeed, every vertex $v \in V(G)$ is either a source in the burning sequence, or else it caught fire due to propagation from one of the sources by the end of round k. If either is true for some source x_i then $v \in N_{k-i}[x_i]$. Hence $V(G) \subseteq N_{k-1}[x_1] \cup N_{k-2}[x_2] \cup \cdots \cup N_1[x_{k-1}] \cup N_0[x_k]$, which proves (b) since the reverse inclusion is obvious. Now, suppose for some i and j with $1 \leq i < j \leq k$, that $d(x_i, x_j) = \ell < j - i$. Then, ℓ rounds after x_i is burned in round i, x_j catches fire due to propagation from x_i. But $\ell + i < j$, so x_j is on fire at the end of a strictly earlier round than round j. Hence, x_j is not a legal source, so $(x_1, x_2, \ldots, x_k)$ is not a burning sequence for G, which is a contradiction. Therefore, $d(x_i, x_j) \geq j - i$ must be true, which proves (a).

Now, for the reverse implication, assume $x_1, x_2, \ldots, x_k \in V(G)$, and that both (a) and (b) hold. Due to (a), if we burn the x_i one-by-one then we will never be prompted to burn a non-legal source. Of course, (b) implies that the sequence leaves G completely burned, so $(x_1, x_2, \ldots, x_k)$ is a burning sequence for G. □

The following theorem is presented as a corollary in [14].

Theorem 1.3.3. ([14]) *If $x_1, x_2, \ldots, x_k \in V(G)$ and $N_{k-1}[x_1] \cup N_{k-2}[x_2] \cup \cdots \cup N_1[x_{k-1}] \cup N_0[x_k] = V(G)$, then $b(G) \leq k$.*

Proof. The arsonist can burn G in no more than k rounds by burning x_i in round i if it is legal, and burning any redundant source in round i otherwise. Then x_i is on fire by the end of round i for each $i \in \{1, 2, \ldots, k\}$. Hence, $N_{k-1}[x_1] \cup N_{k-2}[x_2] \cup \cdots \cup N_1[x_{k-1}] \cup N_0[x_k] = V(G)$ will be on fire at the end of round k (or possibly earlier). Since we have constructed a burning sequence of length k (or shorter), we have $b(G) \leq k$. □

The following result is not presented as a corollary in [14], but is rather a part of the discussion.

Corollary 1.3.4. ([14]) *If H is a spanning subgraph of G then $b(G) \leq b(H)$.*

Proof. Let $(x_1, x_2, \ldots, x_k)$ be an optimal burning sequence for H. Then, by Lemma 1.3.2, $N_{k-1}[x_1] \cup N_{k-2}[x_2] \cup \cdots \cup N_1[x_{k-1}] \cup N_0[x_k] = V(H) = V(G)$. Thus, by Theorem 1.3.3, we have $b(G) \leq k = b(H)$. □

Of course, for each source x_i in a burning sequence there is a tree rooted at x_i with height $k - i$ that is a subgraph of $N_{k-i}[x_i]$. This fact motivates the following theorem.

Theorem 1.3.5. ([14]) *Burning a graph G in k steps is equivalent to finding a rooted tree partition of G into k trees $T_1, T_2, \ldots, T_k$ such that both of the following hold:*

(a) The height of tree T_i is at most $k - i$.

(b) For each pair $i, j \in \{1, 2, \ldots, k\}$, the distance between the roots of T_i and T_j is at least $|i - j|$.

Proof. For the forward implication, assume $(x_1, x_2, \ldots, x_k)$ is a burning sequence for G. After a source x_i is burned in round i, in each round $t > i$, every unburned vertex in $N_{t-i}[x_i]$ will catch fire through propagation. Also, vertex v "receives" fire from a shortest path of burned vertices between v and a source x_i (the path has length zero if $v = x_i$). Define a function $f : V(G) \to \{x_1, x_2, \ldots, x_k\}$ such that $f(v) = x_i$ if v receives fire from x_i, where i is the lowest possible index. Clearly f is surjective, since every source causes at least one vertex to receive fire (even if it is itself). Now, $\{f^{-1}(x_1), f^{-1}(x_2), \ldots, f^{-1}(x_k)\}$ forms a partition of $V(G)$, and for each i, $G[f^{-1}(x_i)]$ is a connected subgraph of G. Every vertex v in $G[f^{-1}(x_i)]$ receives fire from x_i via a shortest (x_i, v)-path, so we can delete the extraneous edges from $G[f^{-1}(x_i)]$ in order to form a tree (rooted at x_i). The height of a tree rooted at x_i is $k - i$, since the fire continues to spread from x_i for $k - i$ more rounds after the source is burned. By the same logic that was used in the proof of Lemma 1.3.2, if the distance between some x_i and x_j is strictly less than $|i-j|$, then the source of higher index is not a legal source, as it must have caught fire at a strictly earlier round than its index. Thus, the distance between the roots of T_i and T_j is at least $|i - j|$.

To prove the reverse implication, assume $T_1, T_2, \ldots, T_k$ is a rooted tree partition of G with roots $x_1, x_2, \ldots, x_k$ that satisfies both (a) and (b). We will show that $(x_1, x_2, \ldots, x_k)$ is a burning sequence for G. By (a), every vertex in G will catch fire through propagation, since the distance from x_i to any vertex in T_i is at most $k - i$, and there are $k - i$ rounds in which the fire propagates after burning x_i. By (b), every source in the sequence is "legal." If a source x_j is already on fire at the start of round j, then it received fire from some source

x_i in some round $i < \ell < j$. Then, the distance between x_i and x_j is $|i - \ell| < |i - j|$, which is a contradiction. Thus, $(x_1, x_2, \ldots, x_k)$ is a burning sequence for G. □

The following corollary is one of the most significant results in the field of graph burning, as it allows us to only consider trees when we try to prove statements about the burning number of a general graph.

Corollary 1.3.6. (Tree Reduction Theorem, [13]) *For a graph G, $b(G) = \min\{b(T) \mid T$ is a spanning subtree of $G\}$.*

Proof. Let $(x_1, x_2, \ldots, x_k)$ be an optimal burning sequence for G. By Theorem 1.3.5, this sequence induces a rooted tree partition $T_1, T_2, \ldots, T_k$ of G where each T_i is rooted at x_i. Let T be a spanning subtree of G that is created by adding edges between the T_i's in such a way that a cycle is not created. We will show that $b(G) = b(T)$. First, T can be burned in k steps via the sequence $(x_1, x_2, \ldots, x_k)$, so $b(T) \leq k = b(G)$. Also, T is a spanning subgraph of G, so $b(G) \leq b(T)$ by Corollary 1.3.4.

Thus, $b(G)$ is equal to the burning number of one of its spanning subtrees T. Indeed, by Corollary 1.3.4, none of the spanning subtrees of G can have a strictly lower burning number than $b(G)$ $(= b(T))$, so T must have the minimum burning number among all spanning subtrees of G. □

Theorem 1.3.7 states the burning number of a path, and motivates Conjecture 1.3.8.

Theorem 1.3.7. ([13]) *For a path P_n on n vertices, $b(P_n) = \lceil \sqrt{n} \rceil$.*

Proof. Let $(x_1, x_2, \ldots, x_k)$ be an optimal burning sequence for P_n. By Lemma 1.3.2 we have $|N_{k-1}[x_1] \cup N_{k-2}[x_2] \cup \cdots \cup N_1[x_{k-1}] \cup N_0[x_k]| = |V(P_n)| = n$. Now, observe that for any $v \in V(P_n)$, $N_i[v] \leq 2i + 1$. We therefore have

$$\begin{aligned} n &\leq |N_{k-1}[x_1]| + |N_{k-2}[x_2]| + \cdots + |N_1[x_{k-1}]| + |N_0[x_k]| \\ &\leq \big(2(k-1)+1\big) + \big(2(k-2)+1\big) + \cdots + \big(2(1)+1\big) + \big(2(0)+1\big) \\ &= 2\big((k-1) + (k-2) + \cdots + 1 + 0\big) + k \\ &= 2\left(\frac{(k-1)(k)}{2}\right) + k \\ &= k^2 \end{aligned}$$

Thus, $k \geq \sqrt{n}$. Since k is an integer, we conclude $b(P_n) = k \geq \lceil \sqrt{n} \rceil$.

We must now show that we can burn P_n in $\lceil \sqrt{n} \rceil$ steps. Label the vertices of P_n as $v_1, v_2, \ldots, v_n$ from left to right (with v_1 and v_n as the end vertices). First, assume that n is a square number, and write $k = \sqrt{n}$. In this case, the sequence $(x_1, x_2, \ldots, x_k)$ leaves P_n completely burned, where $x_{k-i} = v_{n-i^2-i}$ for $i = 0, 1, \ldots, k-1$. Now, assume n is not a square number, and let n^* be the smallest square number that is larger than n. Write $k = \lceil \sqrt{n} \rceil = \sqrt{n^*}$. By our previous argument, we can burn P_{n^*} in k steps by letting $x_{k-i} = v_{n^*-i^2-i}$ for $i = 0, 1, \ldots, k-1$. But P_n is strictly shorter than P_{n^*}. Hence, we can burn P_n in no more than k steps by letting $x_{k-i} = v_{n^*-i^2-i}$ for $i = 0, 1, \ldots, k-1$, unless $n^* - i^2 - i > n$, in which case we simply burn a redundant source. Thus, $b(P_n) \leq k = \lceil \sqrt{n} \rceil$. □

Intuitively, it seems like paths are the "hardest" graphs to burn. Indeed, since the fire spreads via edges, it makes sense that a graph with very few edges would be hard to burn quickly. Of course, paths (and trees in general) are the sparsest connected graphs, having exactly $|V| - 1$ edges. Vertices of high degree should make a graph easier to burn, since such a vertex can spread the fire to a large proportion of the graph. Again, this is a reason why paths are hard to burn, since they have the lowest maximum degree (two) among all trees. Therefore, [13] poses the following conjecture on $b(G)$, which is perhaps the deepest open problem in the field of graph burning.

Conjecture 1.3.8. (Burning Number Conjecture, [13]) *For a connected graph G of order n, $b(G) \leq \lceil \sqrt{n} \rceil$.*

Corollary 1.3.6 is a useful tool for making progress towards the Burning Number Conjecture – if we are trying to find an upper bound on $b(G)$ in terms of $|V(G)|$ for an arbitrary graph G, we may assume G is a tree.

The Burning Number Conjecture has been proven for several families of graphs, including spiders [15, 20], caterpillars [31], and sufficiently large graphs with minimum degree four or greater [8]. It is also known that the burning number of a graph with minimum degree three can be bounded above by $\lceil \sqrt{n} \rceil$ plus some small constant [8].

Recent progress towards the Burning Number Conjecture has yielded the following two results, whose proofs make extensive use of the Tree Reduction Theorem.

Theorem 1.3.9. ([8]) *For a connected graph G of order n, $b(G) \leq 1 + \left\lceil \sqrt{\frac{4n}{3}} \right\rceil$.*

Theorem 1.3.10 shows that the Burning Number Conjecture holds asymptotically. Its proof involves probabilistic methods, as well as the concept of *metric trees*, which are trees whose edges have been replaced with real intervals.

Theorem 1.3.10. ([35]) *For a connected graph G of order n, $b(G) \leq (1 + o(1))\sqrt{n}$.*

In Section 1.4 we introduce a model much like graph burning which applies to hypergraphs. It is unlikely that Theorems 1.3.9 and 1.3.10 could be adapted to apply to this new model, since the proof techniques for both theorems rely heavily on properties of trees in graphs. Indeed, we will see in Section 2.1 that there is no analogy to the Burning Number Conjecture in the context of hypergraphs. That is, there is no sublinear upper bound on the burning number of a hypergraph in terms of its order.

1.4 The Burning Game on Hypergraphs

The topic of this thesis is an alternate version of the burning game that is played on hypergraphs. The altered game should look much the same - each round, the arsonist burns a vertex, and the fire spreads to other "nearby" vertices based on some propagation rule. This rule should ensure that the game on hypergraphs reduces to the original game when each edge of the hypergraph contains exactly two vertices (i.e., when we play it on a graph). What is the best choice for how the fire propagates?

Consider the following potential rule: if v is on fire at the end of round r and $\{v, u_1, \ldots, u_k\}$ is an edge, then each unburned vertex in $\{u_1, \ldots, u_k\}$ catches fire in round $r+1$. This propagation rule is rather uninteresting, since burning H according to this rule is identical to playing the original burning game on the 2-section of H.

We therefore formulate the following rule for how the fire propagates. Fire spreads to a vertex v in round r if and only if there is an edge $\{v, u_1, \ldots, u_k\}$ such that each of $u_1, u_2, \ldots, u_k$ was on fire at the end of round $r-1$. Clearly this reduces to the original game when the hypergraph has only edges of size 2. Rounds are indexed by $\mathbb{N}$ starting at one, and they have the same structure as discussed in Section 1.3. That is, each round the fire spreads via the propagation rule, and simultaneously the arsonist manually burns a vertex. Of course, the definitions of a *source*, *redundant source*, *burning sequence*, and *burning number* are all analogous to those in Section 1.3, but we present them here for the sake of precision.

Let H be a hypergraph. A *source* is a vertex in $V(H)$ that is manually burned by the arsonist. If a vertex catches fire via propagation in some round i and it is also chosen as the i^{th} source, then it is called a *redundant source*. A source is called *legal* if it is the i^{th} source in a sequence, and it is not on fire at the end of round $i-1$; otherwise it is *non-legal*. Note that every redundant source is also legal. A sequence of legal sources is called *valid*. A valid sequence of sources that leaves H completely burned is called a *burning sequence*. A

burning sequence of minimum possible length is called *optimal*, and its length is the *burning number* of H, denoted $b(H)$. We denote the round-based burning game that takes place on a hypergraph H when S is chosen as the burning sequence by $BG(H, S)$.

For a hypergraph H, define $F(H, S, r)$, $f(H, S, r)$, $\psi(H, r)$, and $\Psi(H, r)$ analogously to the definitions in Section 1.3. Again, we reiterate these definitions in the context of hypergraph burning for the sake of precision.

Let H be a hypergraph, and S be a burning sequence for H. Define $F(H, S, r)$ as the set of vertices that are on fire in H at the end of round r when burning according to S, and write $f(H, S, r) = |F(H, S, r)|$. Also define $\psi(H, r) = \min\{f(G, S, r) \mid S \text{ is a valid burning sequence for } G\}$ and $\Psi(H, r) = \max\{f(G, S, r) \mid S \text{ is a valid burning sequence for } G\}$. We denote these by F_r, f_r, ψ_r, and Ψ_r respectively when H and S are obvious.

Indeed, by expanding our model to hypergraphs we need some new definitions. Define $f'(H, S, r)$ as the number of edges that are fully burned at the end of round r when burning H according to the burning sequence S, $\psi'(H, r) = \min\{f'(H, S, r) \mid S \text{ is a valid burning sequence in } H\}$, and $\Psi'(H, r) = \max\{f'(H, S, r) \mid S \text{ is a valid burning sequence in } H\}$. We denote these by f'_r, ψ'_r, and Ψ'_r respectively when H and S are obvious.

Consider the hypergraph H in Figure 1.1 on page 5. If the arsonist burns a, d, e, g, and j as the first five sources, then at the end of round five no propagation has occured. Thus, $\psi(H, 5) = 5$ and $\psi'(H, 5) = 0$. Conversely, by inspection we can see that starting with the sources g, f, i, b, and d (for example) yields the highest number of burned vertices and fully burned edges at the end of round five. In particular, we have $\Psi(H, 5) = 8$ and $\Psi'(H, 5) = 3$.

Note that it is possible to have two optimal burning sequences S_1 and S_2 in a hypergraph H such that for some round $r < b(H)$ we have $f(H, S_1, r) \neq f(H, S_2, r)$. For example, in the hypergraph H in Figures 1.4 and 1.5, $S_1 = (b, c, a, f)$ and $S_2 = (b, a, c, d)$ are both optimal burning sequences. However, $f(H, S_1, 3) = 5$ and $f(H, S_2, 3) = 4$. This is also possible in graph burning. Let G be a path on five vertices, which are labelled u_1 to u_5, with edges u_1u_2, u_2u_3, u_3u_4, and u_4u_5. Then, both (u_3, u_1, u_5) and (u_2, u_1, u_5) are optimal burning sequences. When burning according to the former sequence, four vertices are on fire at the end of round two, and when burning according to the latter, three vertices are on fire at the end of round two.

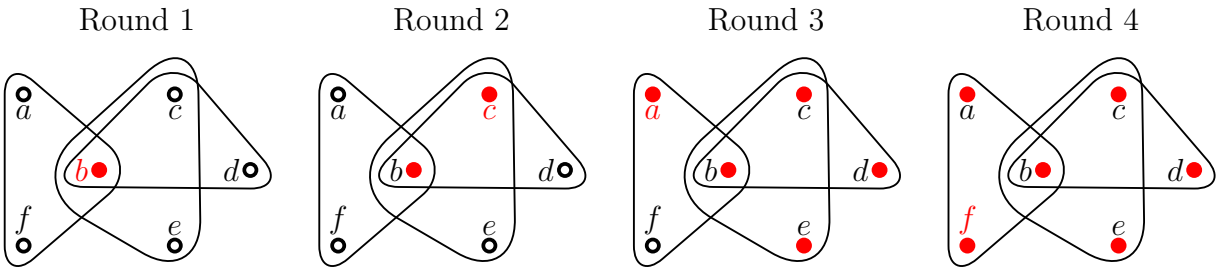

Figure 1.4: An optimal burning sequence $S_1 = (b, c, a, f)$.

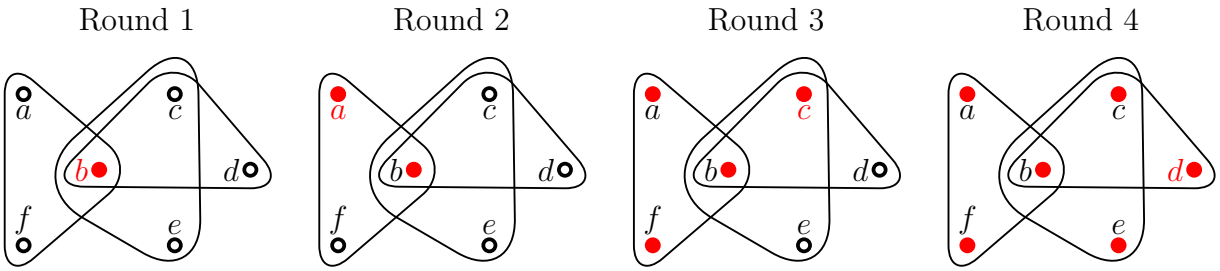

Figure 1.5: An optimal burning sequence $S_2 = (b, a, c, d)$.

1.5 The Lazy Burning Game

We also introduce an alternate set of rules for how the arsonist burns a hypergraph. Suppose the arsonist is very lazy, and does not wish to be present while the hypergraph burns. In particular, they wish to set fire to a select few vertices simultaneously in such a way that the hypergraph is eventually completely burned through subsequent propagation. Of course, the arsonist wishes to set fire to as few vertices as possible while still ensuring that the hypergraph becomes completely burned through propagation. We call the set of vertices the arsonist initially sets fire to a *lazy burning set*. The size of a smallest or *optimal* lazy burning set for a hypergraph H is called the *lazy burning number* of H, denoted $b_L(H)$. Of course, we no longer care about the number of time steps required for the hypergraph to become fully burned, just the size of the lazy burning set. The lazy burning game that takes place on a hypergraph H when $T \subseteq V(H)$ is chosen as the lazy burning set is denoted by $LBG(H, T)$.

The lazy burning game on graphs is trivial. The arsonist must simply set fire to exactly one vertex in each connected component of the graph to achieve a minimum lazy burning set. However, lazily burning a hypergraph is much more interesting. Furthermore, the lazy burning model is a useful tool for analyzing the round-based model of hypergraph burning.

Note that a minimal lazy burning set is not necessarily optimal. For example, in Figure 1.6, $\{c, e, f\}$ is a minimal lazy burning set, since any strict subset of $\{c, e, f\}$ will not successfully burn H. However, it is not an optimal lazy burning set since $b_L(H) = 2$ (for example, $\{a, b\}$ successfully burns H).

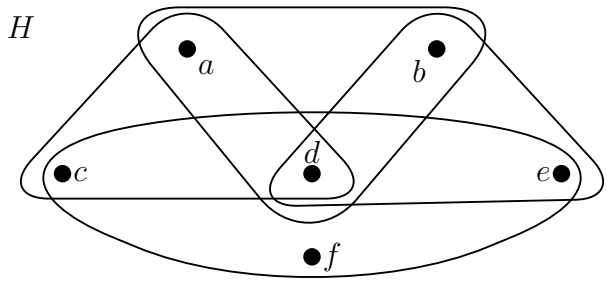

Figure 1.6: $\{c, e, f\}$ is a minimal lazy burning set that is not optimal.

input : a hypergraph $H = (V, E)$
output: a lazy burning set for H

initialize L to an empty array
repeat
  burn any unburned vertex u and add it to L
  repeat
    propagate the fire
  until *no vertex catches fire through propagation*;
until H *is fully burned*;
return L (a lazy burning set)

Algorithm 1: Naïvely construct a lazy burning set

Algorithm 1 constructs a lazy burning set for a hypergraph H that is not necessarily optimal, but at least avoids adding extraneous vertices to the lazy burning set (i.e. vertices that would catch fire via propagation if they were not burned manually). Informally, the algorithm works as follows: choose any unburned vertex and manually set it on fire, allow the fire to propagate until it stops naturally, and repeat until the hypergraph is fully burned. Then the manually burned vertices form a lazy burning set. We use this algorithm in Chapter 3 to obtain an upper bound on the lazy burning number of a Steiner triple system.

1.5.1 Connections with Bootstrap Processes

Lazy hypergraph burning already exists in the literature, and was introduced in [5] under the name $\mathcal{H}$-*bootstrap percolation* (or the $\mathcal{H}$-*bootstrap process*). While it is common in the literature to denote a hypergraph using a calligraphic uppercase letter, to remain consistent we will continue to use non-calligraphic notation. Therefore, we will henceforth refer to $\mathcal{H}$-bootstrap percolation and the $\mathcal{H}$-bootstrap process as H-bootstrap percolation and the H-bootstrap process respectively. In [5], unburned vertices are called *healthy*, burned vertices are called *infected*, and an initial set of infected vertices is said to *percolate* or H-*percolate* if it is a successful lazy burning set in a hypergraph H. Furthermore, the size of a smallest H-percolating set is denoted $m(H)$ (so $m(H)$ is equal in value to $b_L(H)$). The existing results on H-bootstrap percolation are mostly probabilistic or extremal in nature, or apply to specific families of hypergraphs such as hypercubes. Our results on lazy hypergraph burning take a different approach than those in the literature, focusing on the connection between the lazy and round-based versions of the game. They are deterministic in nature, and apply to general hypergraphs and Steiner triple systems.

The H-bootstrap process is motivated by a model called *weak saturation* or the *graph bootstrap process* introduced in [11]. Given graphs G and F, the F-*bootstrap process* on G is defined as follows. Initially, a subset of $E(G)$ is infected, and at each time step, the infection spreads to more edges of G. In particular, a healthy edge e becomes infected if there exists a copy of F in G such that e is the only healthy edge in the copy.

Before we present the results from [5], we must review some notation, and define two families of hypergraphs. If A is a set and $d \in \mathbb{N}$, recall that A^d denotes the cartesian product of d copies of A. For example, $A \times A \times A = A^3$, and elements of A^3 are ordered triples of elements in A. Also recall that if G is a graph and $S \subseteq V(G)$, then $G[S]$ denotes the subgraph of G induced by the vertices in S. That is, we keep the vertices in S and all edges with both endpoints in S, and delete the rest. Let K_n^d denote the graph with vertex set $\{1, 2, \ldots, n\}^d$ in which two vertices are adjacent if they differ in exactly one coordinate. Note that K_2^r is the r-dimensional hypercube. Define $\mathcal{K}(n, d, t, r)$ as the hypergraph with vertex set $V = \{1, 2, \ldots, n\}^d$ and edge set $E = \{S \subset V \mid K_n^d[S] \cong K_t^r\}$. Equivalently, the edges of $\mathcal{K}(n, d, t, r)$ are all sets S of the form $S = I_1 \times I_2 \times \cdots \times I_d$ where r of the sets $I_j \subseteq \{1, 2, \ldots, n\}$ have size t, and the rest are singletons. Finally, define $\mathcal{P}(n, d, t, r)$ as the hypergraph with vertex set $V = \{1, 2, \ldots, n\}^d$, and whose edges are all sets S of the form $S = I_1 \times I_2 \times \cdots \times I_d$ where r of the sets $I_j \subseteq \{1, 2, \ldots, n\}$ are intervals of size t, and the rest are singletons.

Observe that $\mathcal{P}(n,d,t,r)$ is a subhypergraph of $\mathcal{K}(n,d,t,r)$. This is because, if a set $S = I_1 \times \cdots \times I_d$ satisfies the criteria for being an edge in $\mathcal{P}(n,d,t,r)$, then it also satisfies the criteria for being an edge in $\mathcal{K}(n,d,t,r)$ (of course, every interval of size t is also a set of size t). The first two major results in [5] focus on the lazy burning number of these two classes of hypergraphs.

Theorem 1.5.1. ([5]) *For every $n \geq t \geq 2$ and $d \geq r \geq 1$,*

$$m(\mathcal{K}(n,d,t,r)) = m(\mathcal{P}(n,d,t,r)) = \sum_{s=0}^{r-1} \binom{d}{s} (t-1)^{d-s}(n+1-t)^s.$$

The following theorem presents a much simpler expression for the lazy burning number of these two classes of hypergraphs in the special case where the parameters d and r are equivalent. In this case, for an edge $S = I_1 \times \cdots \times I_d$ in $\mathcal{K}(n,d,t,d)$, *all* of the intervals I_j in S have size t. Similarly, for an edge $S = I_1 \times \cdots \times I_d$ in $\mathcal{P}(n,d,t,d)$, *all* of the intervals I_j in S are intervals of size t.

Theorem 1.5.2. ([5]) *For every $n \geq t \geq 2$ and $d \geq 1$,*

$$m(\mathcal{K}(n,d,t,d)) = m(\mathcal{P}(n,d,t,d)) = n^d - (n+1-t)^d.$$

The last major result from [5] deals with a more abstract class of hypergraphs. The vertices of such a hypergraph H index the spanning vectors of a vector space, and for each edge $e \in E(H)$, the vectors corresponding to the vertices in e are linearly independent.

Theorem 1.5.3. ([5]) *Let H be an arbitrary hypergraph, and suppose there exists a vector space $\mathcal{W}$ whose spanning vectors are indexed by the vertices of H. For each $v \in V(H)$, denote its corresponding vector by f_v. Also suppose that for every $e \in E(H)$, there exist $|e|$ non-zero integers $\{\lambda_{e,v} \mid v \in e\}$ such that $\sum_{v \in e} \lambda_{e,v} f_v = 0$. Then, $m(H)$ is bounded below by the dimension of $\mathcal{W}$.*

Several papers in the literature focus on the *critical probability* of H-bootstrap percolation; see [26, 33] for example. Again, we require some notation and definitions. Let X be a finite set, $p \in [0,1]$, and let X_p denote a random subset of X which is obtained by including each element of X with probability p independently of one another. Then, given a hypergraph H, the *critical probability* of the H-bootstrap process is $p_c(H) = \inf\{p \in (0,1) \mid \mathbb{P}(V(H)_p \text{ percolates}) \geq \frac{1}{2}\}$. One may think of it as the "infimal density" at which a random subset of $V(H)$ is likely to percolate in H. Existing results in this area of study mostly focus on

calculating the critical probability of a class of hypegraphs with "high probability" as one or more of its parameters tends to infinity.

There are also several alternate rule sets for bootstrap percolation in a hypergraph. First, consider bootstrap percolation with an *infection threshold* $r \in \mathbb{N}$; see [28,29,34] for example. In this version of the game, a vertex becomes infected when it has at least r infected neighbours. Another alternate rule set is *degree-proportional* bootstrap percolation; see [25] for example. In this version of the game, we are given a number $p \in (0,1)$, and a vertex becomes infected when at least a p-proportion of its neighbours are infected. One may consider both of these to be processes on a graph, since the given hypergraph can be replaced with its 2-section. Indeed, both of these variants are distinct processes from H-bootstrap percolation, as well as from the alternative propagation rules for (lazy) hypergraph burning which will be discussed in Chapter 4. In bootstrap percolation with an infection threshold, vertices become infected individually based on their number of infected neighbours. Conversely, the "threshold-based" propagation rule discussed in Chapter 4 focuses on the number of burned vertices in an edge, and causes entire edges to catch fire at once. In degree-proportional bootstrap percolation, vertices become infected individually based on the proportion of their neighbours that are infected. In contrast, the "proportion-based" propagation rule discussed in Chapter 4 focuses on the proportion of burned vertices in an edge, and again, causes entire edges to catch fire at once.

Before moving on, we reiterate that none of the existing results in H-bootstrap percolation are replicated in this thesis. Indeed, we have taken a new direction by focusing on finite hypergraphs in general, Steiner triple systems, and the connections with round-based hypergraph burning.

Chapter 2

General Results and Bounds

In this chapter we obtain bounds that apply to the burning number and lazy burning number of arbitrary hypergraphs. We then investigate how to express the (lazy) burning number of a disconnected hypergraph in terms of the (lazy) burning numbers of its connected components. We also consider what happens to the (lazy) burning number of a hypergraph when we take different types of subhypergraphs. We finish this chapter by discussing the complexity of the hypergraph burning and lazy hypergraph burning decision problems.

2.1 Arbitrary Hypergraphs

The burning and lazy burning numbers of a hypergraph can indeed be bounded above and below by simple graph parameters. The following results establish these bounds individually, and they are combined in Theorem 2.1.13 on page 25.

In the following theorem, note that if an edge has multiplicity greater than one, then we choose one of them to be the "original," and the rest are "duplicates." Thus, an edge with multiplicity greater than one contributes 1 to the sum $\mathcal{E}$. Of course, if H is simple then $\mathcal{E} = |E(H)|$.

Theorem 2.1.1. *Let H be a hypergraph, and let $\mathcal{E}$ be the number of edges in H that are not singleton, empty, or duplicate edges. Then $|V(H)| - \mathcal{E} \leq b_L(H)$.*

Proof. Let S be a lazy burning set for H. Denote by z_S the number of vertices that become burned throughout the game through propagation, so $z_S + |S| = |V(H)|$. Observe that each edge (that is not a singleton, empty, or duplicate edge) may "cause" fire to spread

to at most one vertex throughout the lazy burning game. Thus, $z_S \leq \mathcal{E}$. But then $|S| = |V(H)| - z_S \geq |V(H)| - \mathcal{E}$. In particular, if S is a minimum lazy burning set we get $b_L(H) = |S| \geq |V(H)| - \mathcal{E}$. □

Corollary 2.1.2. *For any simple hypergraph H, $|V(H)| - |E(H)| \leq b_L(H)$.*

Lemma 2.1.3. *$b_L(H) \leq b(H)$ for all hypergraphs H.*

Proof. Every burning sequence $(u_1, u_2, ..., u_k)$ for H is also a lazy burning set for H. This is because burning the u_i simultaneously as a lazy burning set results in the same vertices of $V(H) \setminus \{u_1, u_2, ..., u_k\}$ being burned through propagation as when we burn the u_i one-by-one. Since $(u_1, u_2, ..., u_k)$ is a burning sequence for H, all of $V(H) \setminus \{u_1, u_2, ..., u_k\}$ burns through propagation when we take $\{u_1, u_2, ..., u_k\}$ as a lazy burning set.

In particular, a shortest burning sequence S of length $b(H)$ is also a lazy burning set. But $b_L(H) \leq |X|$ for any lazy burning set X. Thus, $b_L(H) \leq |S| = b(H)$. □

Corollary 2.1.4. *Let H be a hypergraph, and let $\mathcal{E}$ be the number of edges in H that are not singleton, empty, or duplicate edges. Then $|V(H)| - \mathcal{E} \leq b(H)$.*

Corollary 2.1.5. *For any simple hypergraph H, $|V(H)| - |E(H)| \leq b(H)$.*

The bounds in Corollary 2.1.2, Lemma 2.1.3, and Corollary 2.1.5 are tight. Consider the simple hypergraph H in Figure 2.1. A minimum lazy burning set is $\{x, y, z\}$, an optimal burning sequence is (x, z, y), and $|V(H)| - |E(H)| = 3$. Hence, $|V(H)| - |E(H)| = b_L(H) = b(H)$, so the aforementioned bounds are all tight simultaneously. Of course, this example generalizes to an infinite family of hypergraphs that show tightness. One may construct such a hypergraph H with $V(H) = \{v_1, \ldots, v_n\}$ and $E(H) = \big\{\{v_1, \ldots, v_{n-1}\}\big\}$, where $n \geq 3$. Then a minimum lazy burning set is $\{v_2, \ldots, v_n\}$, an optimal burning sequence is $(v_2, \ldots, v_n)$, and $|V(H)| - |E(H)| = n - 1$.

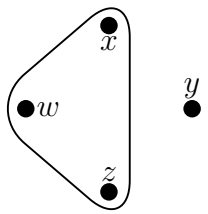

Figure 2.1: An example where all the bounds in Corollary 2.1.2, Lemma 2.1.3, and Corollary 2.1.5 are tight simultaneously.

The bound in Lemma 2.1.3 can be improved to a strict inequality if H satisfies certain conditions; see Theorem 2.1.6 and its corollaries.

Theorem 2.1.6. *If there is an optimal burning sequence $(u_1, u_2, \ldots, u_{b(H)})$ in a hypergraph H such that the last source $u_{b(H)}$ is not an isolated vertex, then $b_L(H) < b(H)$.*

Proof. From the proof of Lemma 2.1.3 we know that $\{u_1, u_2, \ldots, u_{b(H)-1}, u_{b(H)}\}$ is a lazy burning set for H. We claim that $\{u_1, u_2, \ldots, u_{b(H)-1}\}$ is also a lazy burning set for H. Since $u_{b(H)}$ is the final source, burning $u_1, u_2, \ldots, u_{b(H)-1}$ one-by-one (as in the original game) or simultaneously (as a lazy burning set) will eventually result in all of $V(H) \setminus \{u_{b(H)}\}$ being burned. So, let us burn each vertex in $\{u_1, u_2, \ldots, u_{b(H)-1}\}$ simultaneously. All we need to show is that $u_{b(H)}$ will eventually burn through propagation. If $u_{b(H)}$ is a redundant source then clearly fire will propagate to $u_{b(H)}$. Otherwise, $u_{b(H)}$ is not a redundant source. But eventually all of $V(H) \setminus \{u_{b(H)}\}$ will burn through propagation, and $u_{b(H)}$ belongs to an edge since it is not isolated. All other vertices in the edge containing $u_{b(H)}$ are on fire, and thus $u_{b(H)}$ will catch on fire.

Since $\{u_1, u_2, \ldots, u_{b(H)-1}\}$ is a lazy burning set for H we have $b_L(H) \leq b(H) - 1$, or equivalently, $b_L(H) < b(H)$. □

Corollary 2.1.7. *If H has no isolated vertices then $b_L(H) < b(H)$.*

Corollary 2.1.8. *If H is connected then $b_L(H) < b(H)$.*

A hypergraph H for which $b_L(H) = b(H) - 1$ can be seen in Figure 2.2 (on page 25), so the above bounds are tight. There is indeed an infinite family of hypergraphs that exhibit the tightness of these bounds – the family of loose paths with minimum edge size three. Informally, a *loose path* is any hypergraph that can be created from a k-uniform loose path by deleting vertices of degree one while ensuring no edge becomes a singleton. Label the edges in such a hypergraph "from left to right" as e_1 up to e_m. One may construct an optimal burning sequence in such a hypergraph (with minimum edge size three) by burning all the degree-one vertices in e_1 as sources, followed by all the degree-one vertices in e_2, and so on. Indeed, by following this process, every degree-one vertex will be a source in the burning sequence, and the last source will be redundant. Furthermore, one may construct a minimum lazy burning set by taking the vertices in an optimal burning sequence as a set and deleting any one vertex. Hence, each loose path H with minimum edge size three has $b_L(H) = b(H) - 1$.

We now consider upper bounds on $b_L(H)$ and $b(H)$ which make use of the independence number, $\alpha(H)$.

Lemma 2.1.9. *Any optimal lazy burning set in a hypergraph H is an independent set.*

Proof. Let H be an arbitrary hypergraph. Suppose S is an optimal lazy burning set in H that is not independent. Then there is some edge $e = \{x_1, x_2, \ldots, x_k\} \subseteq S$ in H. But then $S \setminus \{x_k\}$ is a smaller lazy burning set in H, which contradicts S being optimal. □

Lemma 2.1.10. *For a hypergraph H, any independent set of vertices of size $\alpha(H)$ is a lazy burning set for H.*

Proof. Let $S \subseteq V(H)$ be an independent set with $|S| = \alpha(H)$. Consider any vertex $v \in V(H) \setminus S$. Clearly $\{v\} \cup S$ is not an independent set by the maximality of S. Thus, there is an edge $\{v, x_1, x_2, \ldots, x_k\} \subseteq \{v\} \cup S$ in H. But then if we set S on fire, each of $x_1, x_2, \ldots, x_k$ will be set on fire, and thus the fire will spread to v through propagation. The vertex v was arbitrarily chosen, so each vertex in $V(H) \setminus S$ will burn through propagation. Therefore S is a lazy burning set for H. □

Note that both Lemma 2.1.9 and Lemma 2.1.10 imply Corollary 2.1.11.

Corollary 2.1.11. *$b_L(H) \leq \alpha(H)$ for all hypergraphs H.*

The bound in Corollary 2.1.11 is tight; see Figure 2.1 on page 22 for an example. The hypergraph pictured has independence number and lazy burning number three, as $\{x, y, z\}$ is both a maximum independent set and a minimum lazy burning set. Indeed, there is an infinite family of hypergraphs that exhibit the tightness of Corollary 2.1.11. One may construct such a hypergraph H with $V(H) = \{v_1, \ldots, v_n\}$ and $E(H) = \big\{\{v_1, \ldots, v_{n-1}\}\big\}$, where $n \geq 3$. Then $\{v_2, \ldots, v_n\}$ is both a maximum independent set and an minimum lazy burning set for H, so $b_L(H) = \alpha(H)$.

Lemma 2.1.12. *$b(H) \leq \alpha(H) + 1$ for all hypergraphs H.*

Proof. Let $S = \{u_1, u_2, \ldots, u_{\alpha(H)}\}$ be a maximum independent set in H. Let the arsonist burn the u_i in order as a burning sequence. If at some round the arsonist expects to burn some u_j as a source, but it is already on fire, then the arsonist skips u_j (which only shortens the burning sequence). Thus, at the end of some round $r \leq \alpha(H)$, each vertex in S is on fire (and possibly some others). Note that it is possible that H is fully burned at the end of round r. In this case, we are done, as we have constructed a burning sequence no longer than $\alpha(H) + 1$. We therefore assume that at the end of round r, H is not fully burned. We claim that in the following round $r+1$ the rest of H will burn through propagation (and the arsonist chooses a redundant source).

Consider any vertex $v \in V(H)$ that was not on fire at the end of round r. Suppose v does not catch fire in round $r + 1$. Then, at the end of round r, there was no edge e containing

v such that all of $e \setminus \{v\}$ was on fire. In particular, since S is completely burned at the end of round r, there is no edge e containing v such that $e \setminus \{v\} \subseteq S$. But then $S \cup \{v\}$ is an independent set in H that is strictly larger than S, which is a contradiction. Therefore, each vertex of H is on fire at the end of round $r+1 \leq \alpha(H)+1$. So $b(H) \leq r+1 \leq \alpha(H)+1$. □

The bound in Lemma 2.1.12 is also tight; see Figure 2.2 for an example. Again, there is an infinite family of hypergraphs that exhibit the tightness of this bound. Consider the family of hypergraphs that consist only of disjoint non-singleton edges. In such a hypergraph, one may construct an optimal burning sequence by burning the vertices of a maximum independent set in any order, with an additional redundant source in the final round. Hence, each hypergraph H in this family has $b(H) = \alpha(H) + 1$.

For a hypergraph H with no isolated vertices, if $b_L(H) = \alpha(H)$ then $b(H) = \alpha(H) + 1$ (due to Corollary 2.1.7 and Lemma 2.1.12). Is the reverse implication true? Are there other sufficient conditions for $b_L(H) = \alpha(H)$? We leave these as open questions.

Finally, by combining the bounds in Corollary 2.1.2, Corollary 2.1.7, and Lemma 2.1.12, we get the series of inequalities in the following result.

Theorem 2.1.13. *Let H be a simple hypergraph with no isolated vertices. Then*

$$|V(H)| - |E(H)| \leq b_L(H) < b(H) \leq \alpha(H) + 1.$$

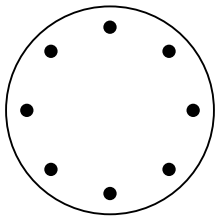

Figure 2.2: An example where all the bounds in Theorem 2.1.13 are tight simultaneously.

Each inequality in Theorem 2.1.13 is tight, and Figure 2.2 shows an example of a hypergraph H where all of them are tight simultaneously. It has $7 = |V(H)| - |E(H)| = b_L(H)$ and $8 = b(H) = \alpha(H) + 1$. Of course, this example can be expanded to an infinite family of hypergraphs that exhibit tightness for each inequality in Theorem 2.1.13 simultaneously. Simply consider the family of hypergraphs that consist of a single edge containing all of their vertices (excluding the hypergraph which is a single vertex in an edge).

We now briefly investigate k-uniform tight paths. This family of hypergraphs is used to prove Theorem 2.1.17, which states that the difference between $b_L(H)$ and $b(H)$ can be arbitrarily large.

Definition 2.1.14. *Given a k-uniform tight path H on n vertices, define a* seed *as a set of $k-1$ vertices all belonging to a common edge. Define a* burned seed *as a seed whose $k-1$ vertices were chosen as sources in the arsonist's burning sequence.*

Observe Figure 2.3. An example of a seed in G is $\{a, b\}$, and an example of a seed in H is $\{c, d, e\}$.

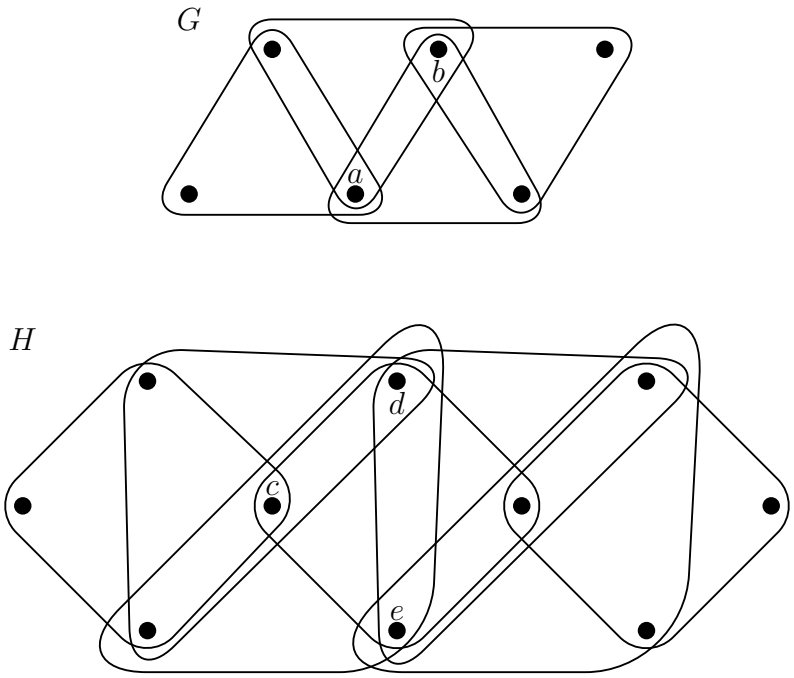

Figure 2.3: A tight 3-uniform path G and a tight 4-uniform path H.

Lemma 2.1.15. *For any tight 3-uniform path H on n vertices, $b(H) = \lceil \sqrt{2n-1} \rceil$.*

Proof. Write $V(H) = \{u_1, \ldots, u_n\}$. Clearly, no fire will propagate in H without the existence of a burned seed. Also, once a burned seed $\{u_i, u_{i+1}\}$ is created in round r, it will cause fire to spread to u_{i-1} and u_{i+2} in round $r+1$ (if these vertices are not already on fire). In round $r+2$, it will cause fire to spread to u_{i-2} and u_{i+3}, etc. The arsonist's best strategy is to create a burned seed every second round, spaced such that no vertex u_j will catch fire due to propagation from two burned seeds until possibly the final round when H is completely burned. By following this strategy, in round r, the maximum number of vertices that can catch fire through propagation is twice the number of burned seeds that existed at the end of round $r-1$. Define $n(r)$ as the number of vertices that become burned in round r when following this strategy (including sources). Clearly $n(1) = n(2) = 1$ since no propagation can occur in the first two rounds. There is *one* burned seed at the start of rounds three and

four, so $n(3) = n(4) = 2(1) + 1 = 3$. Similarly, there are *two* burned seeds at the start of rounds five and six, so $n(5) = n(6) = 2(2) + 1 = 5$. The sequence of $n(r)$-values is therefore $1, 1, 3, 3, 5, 5, 7, 7, \ldots$. Also write $S(r) = \Psi(H, r)$. Of course, $S(r) = \sum_{i=1}^{r} n(i)$, so the sequence of $S(r)$-values is $1, 2, 5, 8, 13, 18, 25, 32, 41, 50, \ldots$. The formula for this sequence is known to be $S(r) = \left\lfloor \frac{r^2+1}{2} \right\rfloor$; see sequence A000982 in [36].

It is clear that for any hypergraph G, the least $r \in \mathbb{N}$ such that $\Psi(G, r) \geq |V(G)|$ is a lower bound on $b(G)$. This is because, by the end of round $r-1$, no more than $\Psi(G, r-1) < |V(G)|$ vertices could possibly be on fire. In the case of our hypergraph H, $b(H)$ is exactly equal to the least $r \in \mathbb{N}$ such that $S(r) \geq n$. We now show that this is $r_n = \lceil \sqrt{2n-1} \, \rceil$.

If r is an integer strictly less than $\sqrt{2n-1}$, then we have

$$S(r) = \left\lfloor \frac{r^2+1}{2} \right\rfloor \leq \frac{r^2+1}{2} < \frac{(\sqrt{2n-1})^2+1}{2} = n.$$

We therefore have the restriction $r \geq \sqrt{2n-1}$, and the least integer that fulfills this restriction is $r_n = \lceil \sqrt{2n-1} \, \rceil$. Indeed, since $\lceil \sqrt{2n-1} \, \rceil \geq \sqrt{2n-1}$ and $\left\lfloor \frac{r^2+1}{2} \right\rfloor$ is monotonic increasing function, we have

$$S(r_n) = \left\lfloor \frac{r_n^2+1}{2} \right\rfloor = \left\lfloor \frac{\lceil \sqrt{2n-1} \, \rceil^2+1}{2} \right\rfloor \geq \left\lfloor \frac{(\sqrt{2n-1})^2+1}{2} \right\rfloor = \lfloor n \rfloor = n. \qquad \square$$

Figure 2.4 is a visual representation of the sequence $S(r)$ from Lemma 2.1.15. In round 6, The arsonist will burn a source in the same edge as x_5, and fire will spread to four new vertices due to the burned seeds (x_1, x_2) and (x_3, x_4). Thus, $S(6) = S(5) + 5 = 18$.

Lemma 2.1.16 generalizes Lemma 2.1.15, although only Lemma 2.1.15 is used in the proof of Theorem 2.1.17.

Lemma 2.1.16. *For H, a tight k-uniform path on n vertices, define $S_{(k)} = \sum_{i=1}^{\infty} a_i$ where $a_i = 1$ if $i \in \{1, 2, \ldots, k-1\}$, $a_i = 3$ if $i \in \{k, k+1, \ldots, 2(k-1)\}$, $a_i = 5$ if $i \in \{2k-1, 2k, \ldots, 3(k-1)\}$,* etc. *Let b_r be the r^{th} partial sum of $S_{(k)}$. Then $b(H)$ is equal to the least $r \in \mathbb{N}$ such that $b_r \geq n$.*

Proof. Write $V(H) = \{u_1, \ldots, u_n\}$. Clearly, no fire will propagate in H without the existence of a burned seed. Also, once a burned seed $\{u_i, \ldots, u_{i+k-2}\}$ is created in round r, it will cause fire to spread to u_{i-1} and u_{i+k-1} in round $r+1$ (if these vertices were not already on fire). In round $r+2$, it will cause fire to spread to u_{i-2} and u_{i+k}, etc. The arsonist's best strategy is to create a burned seed every $k-1$ rounds, spaced such that no vertex u_j will

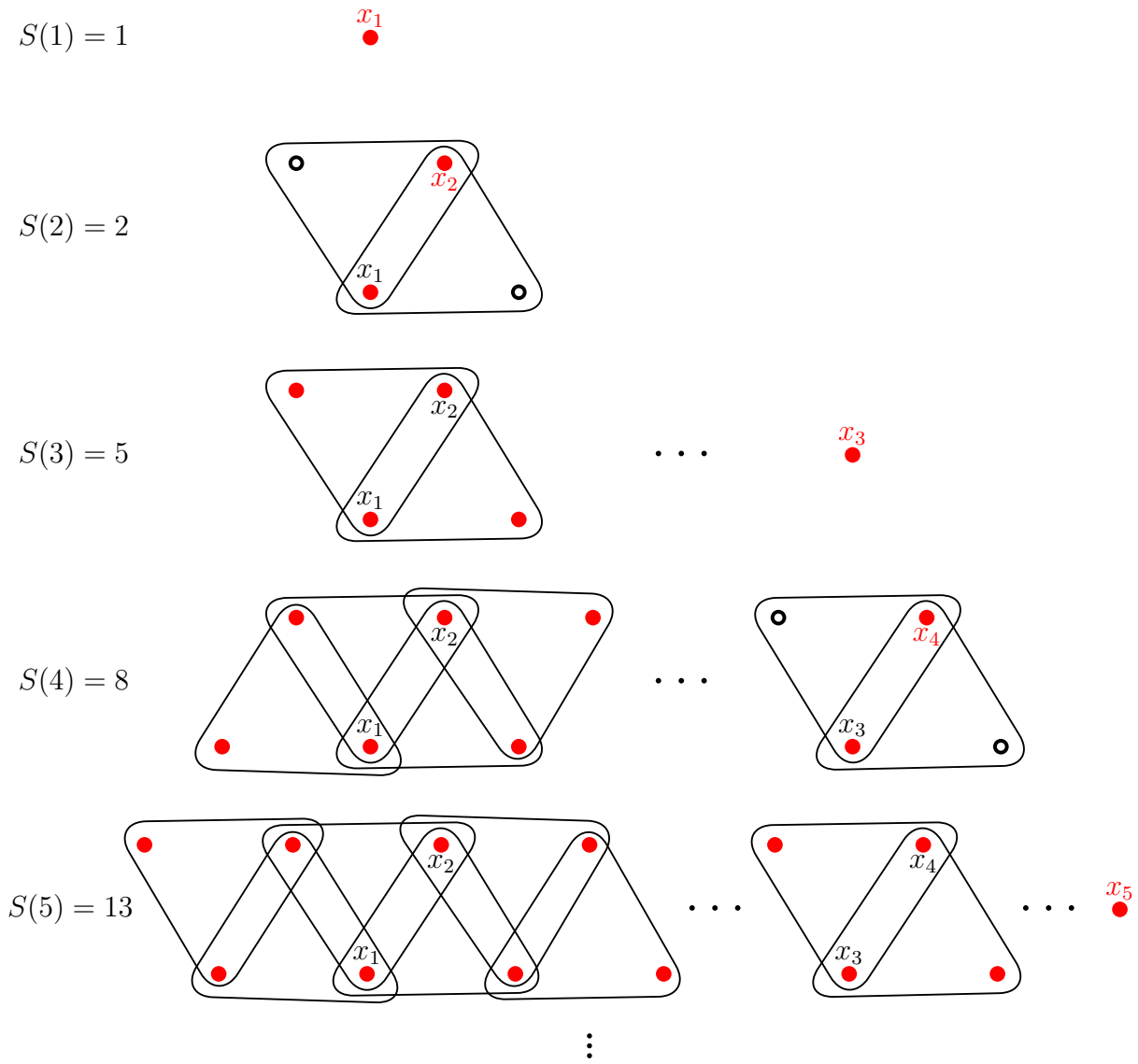

Figure 2.4: The sequence $S(r)$ from Lemma 2.1.15.

catch fire due to propagation from two burned seeds until possibly the final round when H is completely burned. By following this strategy, in round r, the maximum number of vertices that can catch fire through propagation is twice the number of burned seeds that existed at the end of round $r - 1$.

That is, in the first $k - 1$ rounds, at most one vertex catches fire each round. Then, from round k to $2k - 2$, at most three vertices catch fire each round (two from the burned seed, plus one from the source vertex). From round $2k - 1$ to round $3k - 3$, at most five vertices catch fire each round (two from each of the two burned seeds, plus one from the source vertex), etc.

Clearly, Ψ_r is the r^{th} partial sum of the series

$$S_{(k)} = 1_{(1)} + \cdots + 1_{(k-1)} + 3_{(1)} + \cdots + 3_{(k-1)} + 5_{(1)} + \cdots + 5_{(k-1)} + 7_{(1)} + \cdots .$$

Now, the least $r \in \mathbb{N}$ such that $\Psi(H, r) \geq |V(H)|$ is a lower bound on $b(H)$, call it r_0. But by following the strategy above, the arsonist can indeed successfully burn each vertex of H by the r_0^{th} round. Hence, $b(H) = r_0$, where the r_0^{th} partial sum of $S_{(k)}$ is the least partial sum that exceeds $|V(H)|$. □

When r is a multiple of $k-1$, the r^{th} partial sum of $S_{(k)}$ can be computed as follows. Write $r = \ell(k-1)$ for some $\ell \in \mathbb{N}$. Then, the r^{th} partial sum is

$$\begin{aligned}
&1_{(1)} + \cdots + 1_{(k-1)} + 3_{(1)} + \cdots + 3_{(k-1)} + \cdots + (2\ell-1)_{(1)} + \cdots + (2\ell-1)_{(k-1)} \\
&= \big(1 + 3 + \cdots + (2\ell-1)\big)(k-1) \\
&= \ell^2(k-1) \\
&= \frac{\ell^2(k-1)^2}{k-1} \\
&= \frac{r^2}{k-1} \ .
\end{aligned}$$

If instead $\ell(k-1) < r < (\ell+1)(k-1)$, then choose $t = r - \ell(k-1)$. Then, $r-t$ and $r-t+k-1$ are both multiples of $k-1$. Hence, the r^{th} partial sum of $S_{(k)}$ is bounded below by $\frac{(r-t)^2}{k-1}$ and above by $\frac{(r-t+k-1)^2}{k-1}$, which are both approximately $\frac{r^2}{k-1}$. Therefore, the the r^{th} partial sum of $S_{(k)}$ is approximately $\frac{r^2}{k-1}$ in general.

Theorem 2.1.17. *Given any $k \in \mathbb{N}$, there exist hypergraphs G and H such that $b(G) - b_L(G) > k$ and $\frac{b(H)}{b_L(H)} > k$.*

Proof. Denote by H_n the 3-regular tight path on n vertices. For all $n \geq 3$, $b_L(H_n) = 2$. But $b(H_n) = \lceil \sqrt{2n-1} \ \rceil$. Thus, $b(H_n) - b_L(H_n) = \lceil \sqrt{2n-1} \ \rceil - 2$ and $\frac{b(H_n)}{b_L(H_n)} = \frac{\lceil \sqrt{2n-1} \ \rceil}{2}$, which are both unbounded. □

We now consider an alternative method for finding lower bounds on $b_L(H)$ and $b(H)$ which involves counting degree-one vertices. For each edge $e \in E(H)$, let $n(e)$ be the number of degree-one vertices in e, and define

$$m(e) = \begin{cases} 0 & \text{if } n(e) = 0 \\ n(e) - 1 & \text{otherwise.} \end{cases}$$

Theorem 2.1.18. *Let H be a connected hypergraph. Then*

$$\sum_{e \in E(H)} m(e) \leq b_L(H).$$

Furthermore, if H contains edges that intersect, then

$$1 + \sum_{e \in E(H)} m(e) \leq b_L(H).$$

Proof. If H is an isolated vertex, then $E(H) = \emptyset$. Similarly, if H consists of a single vertex contained in an edge e, then $m(e) = 0$. In either case, $\sum_{e \in E(H)} m(e) = 0 \leq 1 = b_L(H)$.

Otherwise, if H contains no intersecting edges, then H consists of a single edge e containing $|V(H)| > 1$ vertices. In this case, $b_L(H) = |V(H)| - 1 = n(e) - 1 = m(e) = \sum_{e \in E(H)} m(e)$.

Otherwise, H contains intersecting edges. If any edge $e \in E(H)$ contains multiple degree-one vertices that are not part of the lazy burning set, none of them will ever catch fire through propagation. Thus, for each edge $e \in E(H)$, at least $m(e)$ of its degree-one vertices must be in the lazy burning set, and hence $b_L(H) \geq \sum_{e \in E(H)} m(e)$. We will show that this is in fact a strict inequality.

Suppose the lazy burning set consists only of these $\sum_{e \in E(H)} m(e)$ degree-one vertices. Recall that, since every edge intersects some other edge, every edge contains at least one vertex of degree greater than one. None of these vertices are in the lazy burning set, and if the edge in question contains degree-one vertices, one of them is not in the lazy burning set either. Thus, each edge contains at least two vertices not in the lazy burning set, so no fire will propagate. We therefore have $b_L(H) > \sum_{e \in E(H)} m(e)$, so $b_L(H) \geq 1 + \sum_{e \in E(H)} m(e)$. □

Corollary 2.1.19. *Let H be a connected hypergraph. Then*

$$\sum_{e \in E(H)} m(e) < b(H).$$

Furthermore, if H contains edges that intersect, then

$$1 + \sum_{e \in E(H)} m(e) < b(H).$$

The lower bounds in Theorem 2.1.18 and Corollary 2.1.19 are tight; see Figure 2.5. Edge e_1 contains four degree-one vertices, so $m(e_1) = 3$, edge e_2 contains three degree-one vertices, so $m(e_2) = 2$, and e_1 and e_2 intersect. Indeed, the hypergraph H pictured has $b_L(H) = 6$

and $b(H) = 7$, so the bounds in Theorem 2.1.18 and Corollary 2.1.19 are both tight. This example can be expanded to an infinite family of hypergraphs that exhibit the tightness of the bounds in Theorem 2.1.18 and Corollary 2.1.19 simultaneously. Simply consider the family of connected hypergraphs H containing two non-singleton edges e_1 and e_2 which intersect in exactly one vertex. A minimum lazy burning set for such a hypergraph consists of any $m(e_1)+1$ vertices in e_1, as well as any $m(e_2)$ degree-one vertices in e_2, so the bound in Theorem 2.1.18 is tight. One may construct an optimal burning sequence for H by burning the vertices of a minimum lazy burning set in any order, followed by a redundant source for a total of $b_L(H) + 1$ sources. Thus, the bound in Corollary 2.1.19 is tight.

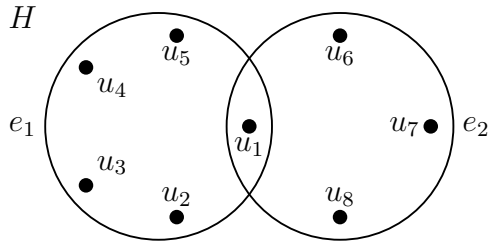

Figure 2.5: A hypergraph H for which the lower bounds in Theorem 2.1.18 and Corollary 2.1.19 are both tight.

Most often, Theorem 2.1.18 will simply tell us that $b_L(H) \geq 1 + \sum_{e \in E(H)} m(e)$, since we usually consider connected nontrivial hypergraphs with intersecting edges when playing the lazy burning game. Theorem 2.1.18 may be combined with Lemma 2.2.1 from Section 2.2 to obtain the following: $b_L(H) \geq x + y + \sum_{e \in E(H)} m(e)$, where x is the number of isolated vertices plus the number of components that are one vertex in an edge, and y is the number of connected components of H featuring edges that intersect.

We leave the following as an open question: do Theorem 2.1.18 and Corollary 2.1.19 ever provide better lower bounds than Theorem 2.1.13? Even if the answer is no, the proof at least provides the insight that for each edge e, at least $m(e)$ degree-one vertices in e must be part of the lazy burning set or chosen as sources in the burning sequence.

We finish this section by considering the following question: what might an analogous conjecture to the *Burning Number Conjecture* look like for hypergraphs? That is, for arbitrary hypergraphs H, is there a sublinear upper bound on $b(H)$ in terms of $|V(H)|$? What about $b_L(H)$? It turns out that the answer to both is *no* even when we insist H is k-uniform and linear. Consider the family of k-uniform loose paths (which are all linear; see Figure 2.6). Indeed, $b_L(H), b(H) \in \mathcal{O}(|V(H)|)$ are the best possible upper bounds for this family of hypergraphs. To see this, recall that $|V(H)| - |E(H)| \leq b_L(H) < b(H) \leq |V(H)|$, and observe that we can choose the size of the edges large enough so that $|V(H)| - |E(H)| \in \Theta(|V(H)|)$

(informally meaning $|V(H)| - |E(H)| \approx |V(H)|$).

Furthermore, for any *fixed* k, this family of hypergraphs has no sublinear upper bound on its (lazy) burning number in terms of its order. Indeed, any k-uniform loose path H has $|E(H)| = \frac{|V(H)|-1}{k-1}$. Thus, a *lower* bound on $b_L(H)$ and $b(H)$ is

$$|V(H)| - |E(H)| = |V(H)| - \left(\frac{|V(H)|-1}{k-1}\right) = \left(\frac{k-2}{k-1}\right)|V(H)| + \frac{1}{k-1}\ ,$$

which is linear in $|V(H)|$.

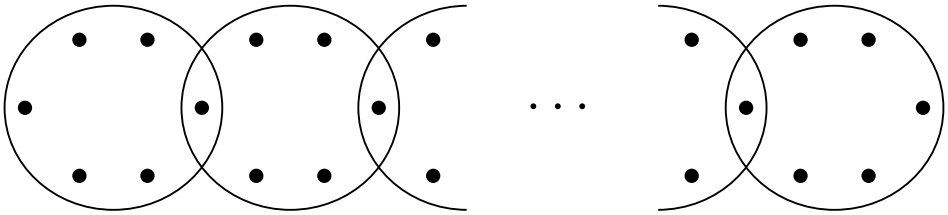

Figure 2.6: A family of uniform, linear hypergraphs H for which no upper bounds exist on $b(H)$ or $b_L(H)$ that are sublinear in terms of $|V(H)|$.

2.2 Disconnected Hypergraphs

In this section we consider how to write the (lazy) burning number of a disconnected hypergraph in terms of the (lazy) burning numbers of its connected components. The solution is trivial in the lazy case (see Lemma 2.2.1), but the round-based case is more complicated.

Lemma 2.2.1. *If H is disconnected with connected components $G_1, G_2, \ldots, G_k$, then $b_L(H) = b_L(G_1) + b_L(G_2) + \cdots + b_L(G_k)$.*

Proof. Clearly if S_i is a minimum lazy burning set for G_i for each $i \in \{1, 2, \ldots, k\}$, then a minimum lazy burning set for H is $S_1 \cup S_2 \cup \cdots \cup S_k$. Thus, $b_L(H) = |S_1 \cup S_2 \cup \cdots \cup S_k| = b_L(G_1) + b_L(G_2) + \cdots + b_L(G_k)$. □

We cannot say in general that the same equality as in Lemma 2.2.1 holds for $b(H)$. It is indeed possible that $b(H) < b(G_1) + b(G_2) + \cdots + b(G_k)$. The arsonist may "save time" by initially burning enough vertices to start propagation in one component, then moving on to subsequent components. For example, consider the disconnected hypergraph H in Figure 2.7. By Lemma 2.1.15, the "left" component of H has burning number 4, and the

"right" component of H has burning number 3. However, $(3, 4, 9, 10, 7)$ is an optimal burning sequence for H. Indeed, there are no burning sequences of length four or less. To see this, observe that any burning sequence in a 3-uniform tight path must create a burned seed with two of its sources (otherwise no propagation occurs). If the arsonist creates a burned seed in the "left" component first, then they only have two more rounds to burn the "right" component, which is impossible. A similar problem arises if the arsonist creates a burned seed in the "right" component first. Thus, H has burning number five, which is less than the sum of the burning numbers of its connected components.

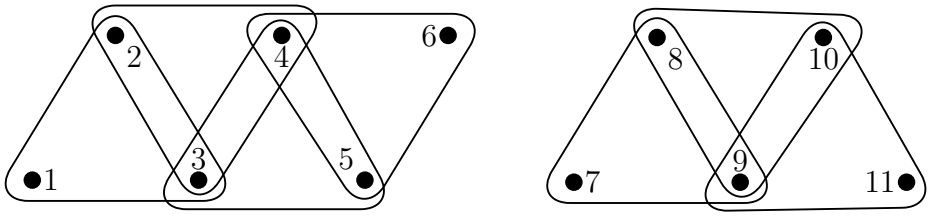

Figure 2.7: A hypergraph whose burning number is strictly less than the sum of the burning numbers of its connected components.

For the purposes of the following proof, denote the sequence created by concatenating the sequences $S_1, S_2, \ldots, S_n$ (in that order) by $S_1 S_2 \cdots S_n$. Clearly the length of $S_1 S_2 \cdots S_n$ is $|S_1| + |S_2| + \cdots + |S_n|$.

Lemma 2.2.2. *If H is disconnected with connected components $G_1, G_2, \ldots, G_k$, then* $\max\{b(G_1), b(G_2), \ldots, b(G_k)\} \leq b(H) \leq b(G_1) + b(G_2) + \cdots + b(G_k)$.

Proof. If S_i is an optimal burning sequence for each G_i, then clearly $S_1 S_2 \cdots S_k$ is a burning sequence for H. Thus $b(H) \leq |S_1| + |S_2| + \cdots + |S_k| = b(G_1) + b(G_2) + \cdots + b(G_k)$.

Now, suppose $b(H) < \max\{b(G_1), b(G_2), \ldots, b(G_k)\}$. Let G_i be a connected component of H with $b(G_i) = \max\{b(G_1), b(G_2), \ldots, b(G_k)\}$, so $b(H) < b(G_i)$. Let $S = (u_1, \ldots, u_{b(H)})$ be an optimal burning sequence for H. Clearly, $S \cap V(G_i) \neq \emptyset$, so let u_j be the lowest-indexed vertex in $S \cap V(G_i)$. We construct a burning sequence S' for G_i from S as follows. Let the first source in S' be u_j. Then, for all rounds $r > j$ in $BG(H, S)$:

- If G_i became completely burned at the end of the previous round in $BG(G_i, S')$, we may stop adding to S' (thus, S' is a burning sequence for G_i no longer than $b(H)$).
- Otherwise, if $u_r \notin V(G_i)$, choose any unburned vertex in G_i as the $(r - j + 1)^{th}$ source in S'.

- Otherwise, if $u_r \in V(G_i)$, but u_r is already on fire in $BG(G_i, S')$ at the end of round $r - j$, choose any unburned vertex in G_i as the $(r - j + 1)^{th}$ source in S'.
- Otherwise, if $u_r \in V(G_i)$ and u_r is not on fire in $BG(G_i, S')$ at the end of round $r - j$, choose u_r as the $(r - j + 1)^{th}$ source in S'.

Then, $F(H, S, r) \cap V(G_i) \subseteq F(G_i, S', r - j + 1)$ for each round $r \geq j$. Roughly, G_i burns in $BG(G_i, S')$ *at least as fast* as in $BG(H, S)$. Since G_i is completely burned in $BG(H, S)$ in at most $b(H)$ rounds, G_i is completely burned in $BG(G_i, S')$ in at most $b(H) - j + 1$ rounds.

Thus, S' is a burning sequence for G_i of length at most $b(H) - j + 1$ for some $j \in \{1, 2, \ldots, b(H)\}$. We therefore have $b(G_i) \leq |S'| \leq b(H) < b(G_i)$, which is a contradiction. So $\max\{b(G_1), b(G_2), \ldots, b(G_k)\} \leq b(H)$. □

We can improve the upper bound from Lemma 2.2.2 if we assume none of the connected components of H are isolated vertices.

Theorem 2.2.3. *If H is disconnected with connected components $G_1, G_2, \ldots, G_k$, none of which are isolated vertices, then $b(H) \leq b(G_1) + b(G_2) + \cdots + b(G_k) - k + 1$.*

Proof. For each G_i let $S_i = (u_1^{(i)}, \ldots, u_{b(G_i)}^{(i)})$ be an optimal burning sequence. We claim

$$S = (u_1^{(1)}, \ldots, u_{b(G_1)-1}^{(1)}, u_1^{(2)}, \ldots, u_{b(G_2)-1}^{(2)}, \ldots, u_1^{(k)}, \ldots, u_{b(G_k)-1}^{(k)}, u_{b(G_k)}^{(k)})$$

is a burning sequence for H.

For each $i \in \{1, 2, \ldots, k-1\}$ burning $S_i \setminus \{u_{b(G_i)}^{(i)}\}$ one-by-one and in order will leave the entirety of G_i burned (to see this, consider the two possible cases: $u_{b(G_i)}^{(i)}$ is a redundant source or a non-redundant source). We must show that for each $i < k$, $u_{b(G_i)}^{(i)}$ will catch fire through propagation before the game ends. For each $i \in \{1, 2, \ldots, k\}$, at the same time as the arsonist burns $u_1^{(i)}$ as a source, one of the following will happen: either all of G_{i-1} will catch fire through propagation (if $u_{b(G_{i-1})}^{(i-1)}$ was a redundant source), or all of $G_{i-1} \setminus \{u_{b(G_{i-1})}^{(i-1)}\}$ will catch fire through propagation (if $u_{b(G_{i-1})}^{(i-1)}$ was not a redundant source). In the latter case, $u_{b(G_{i-1})}^{(i-1)}$ will catch fire through propagation in one more round since the rest of G_{i-1} is on fire. If $i = k$ then since $|V(G_k)| \geq 2$, we have that $b(G_k) \geq 2$, so even if $u_{b(G_{k-1})}^{(k-1)}$ is a non-redundant source, it will burn through propagation before the game ends.

Since S is a valid burning sequence for H we have

$$b(H) \leq |S| = (b(G_1) - 1) + \cdots + (b(G_{k-1}) - 1) + b(G_k) = b(G_1) + \cdots + b(G_k) - k + 1. \quad \square$$

Thus, if H is disconnected with connected components $G_1, G_2, \ldots, G_k$, none of which are isolated vertices, then we have the following inequality:

$$b_L(H) = \sum_{i=1}^{k} b_L(G_i) < b(H) \leq 1 - k + \sum_{i=1}^{k} b(G_i) \tag{2.1}$$

The upper bound from Theorem 2.2.3 is tight, as illustrated by the hypergraph in Figure 2.8. The subhypergraph induced by *one* of the seven edges has burning number 3, and since $(1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20, 21)$ is an optimal burning sequence for the entire hypergraph, it has burning number $15 = (3)(7) - 7 + 1$. Indeed, this example can be expanded to an infinite family of hypergraphs which exhibit the tightness of the bound in Theorem 2.2.3. Such a hypergraph H consists of multiple disjoint non-singleton edges $e_1, \ldots, e_k$, with no other edges and no isolated vertices. For each $i \in \{1, \ldots, k\}$, let G_i be the subhypergraph of H induced by e_i. Then, one may construct an optimal burning sequence for H by burning $|e_i| - 1 = b(G_i) - 1$ sources in e_i for each $i \in \{1, \ldots, k-1\}$, followed by $|e_k| = b(G_k)$ sources in e_k (where the last source is redundant). This shows the tightness of the bound in Theorem 2.2.3, since the total number of sources is

$$\big(b(G_1) - 1\big) + \cdots + \big(b(G_{k-1}) - 1\big) + b(G_k) = b(G_1) + \cdots + b(G_k) - k + 1.$$

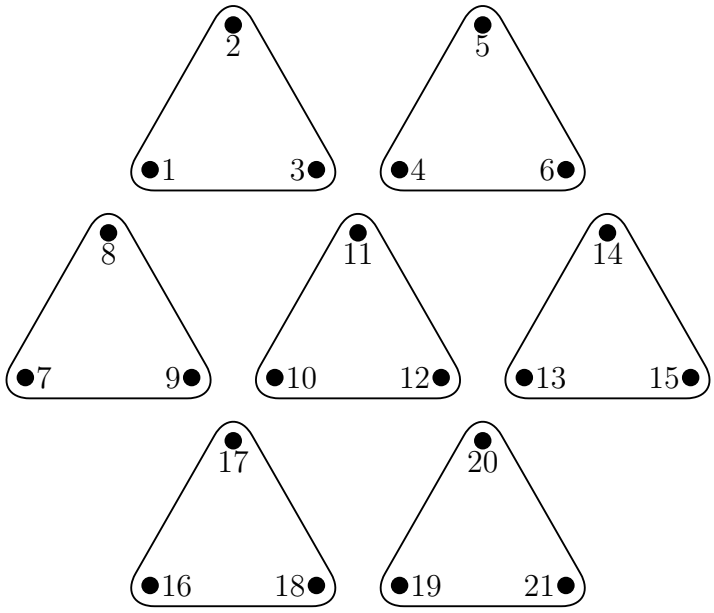

Figure 2.8: An example where the bound in Theorem 2.2.3 is tight.

There are also examples where the upper bound from Theorem 2.2.3 is a strict inequality.

Consider the hypergraph in Figure 2.9. The burning numbers of the left, middle, and right connected components are 5, 3, and 2 respectively. But $(u_1, u_2, u_3, u_4, u_5, u_6)$ is an optimal burning sequence for the entire hypergraph, so it has burning number $6 < (5+3+2)-3+1$. Is there an infinite family of hypegraphs for which the bound in Theorem 2.2.3 is strict? We leave this as an open question.

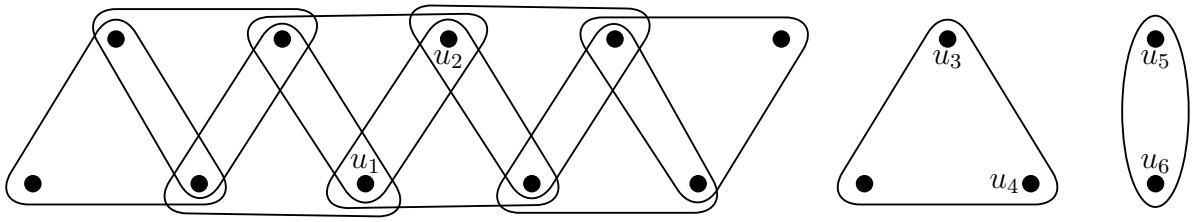

Figure 2.9: An example where the inequality in Theorem 2.2.3 is strict.

2.3 Subhypergraphs

In this section we compare $b(H)$ and $b_L(H)$ to the burning and lazy burning numbers of various types of subhypergraphs of H. We allow the existence of singleton edges, as these often arise when taking a weak induced subhypergraph.

Lemma 2.3.1. *Let G_1 and G_2 be two hypergraphs with the same vertex set such that $E(G_1) \subseteq E(G_2)$. Then $b_L(G_2) \leq b_L(G_1)$ and $b(G_2) \leq b(G_1)$.*

Proof. Let L be an optimal lazy burning set for G_1. Let us burn L as a lazy burning set in G_2. Since $E(G_1) \subseteq E(G_2)$, the same propagation that occurs in $LBG(G_1, L)$ will occur in $LBG(G_2, L)$, and it may even happen in fewer time steps due to the presence of the extra edges in G_2. Hence, L is a successful lazy burning set for G_2, so $b_L(G_2) \leq |L| = b_L(G_1)$.

Now, let S be an optimal burning sequence for G_1. Let us burn G_2 in the following way: in each round i, we burn the i^{th} source in S, unless it is already on fire, in which case we burn a redundant source that round. Again, since $E(G_1) \subseteq E(G_2)$, the same propagation that occurs in $BG(G_1, S)$ will occur in $BG(G_2, S)$, and it may even happen in fewer rounds due to the presence of the extra edges in G_2. Thus, following this strategy will leave G_2 fully burned after no more than $|S|$ rounds, so $b(G_2) \leq |S| = b(G_1)$. □

Our next result compares the (lazy) burning numbers of a weak and a strong subhypergraph induced by the same set of vertices.

Theorem 2.3.2. *Let H be a hypergraph with $V(H) = \{v_1, \ldots, v_n\}$ and let $I \subseteq \{1, \ldots, n\}$. If G_1 is a strong subhypergraph induced by $\{v_i \mid i \in I\}$ and G_2 is a weak subhypergraph induced by $\{v_i \mid i \in I\}$ then $b(G_2) \leq b(G_1)$ and $b_L(G_2) \leq b_L(G_1)$.*

Proof. Both inequalities follow from Lemma 2.3.1 and the fact that $V(G_1) = V(G_2)$ and $E(G_1) \subseteq E(G_2)$. □

The bounds in Theorem 2.3.2 are both tight. Consider the hypergraph H in Figure 2.10. The subset of vertices $\{u_2, u_3, u_4, u_5\}$ induces the strong subhypergraph G_1 and the weak subhypergraph G_2. Indeed, $\{u_2, u_3\}$ is a minimum lazy burning set in both G_1 and G_2. The subgraphs also have the same burning number, as (u_2, u_3, u_5) is an optimal burning sequence in G_1, and (u_2, u_3, u_4) is an optimal burning sequence G_2. Indeed, we may expand this to an infinite family of examples by adding any number of degree-one vertices to the edge containing u_1 in H (*without* renaming u_1 through u_5). Then, the strong and weak subhypergraphs induced by $\{u_2, u_3, u_4, u_5\}$ will be G_1 and G_2 (from Figure 2.10) respectively.

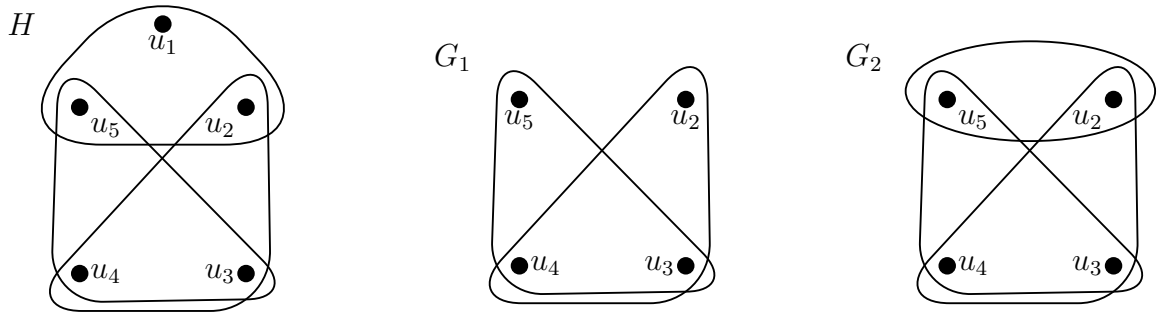

Figure 2.10: An example showing that equality can hold in Theorem 2.3.2.

The inequalities in Theorem 2.3.2 can also be strict. Consider the hypergraph H in Figure 2.11. The subset of vertices $\{u_1, u_2, u_3\}$ induces the strong subhypergraph G_1 and the weak subhypergraph G_2. Indeed, $b_L(G_2) = 1 < 2 = b_L(G_1)$, and $b(G_2) = 2 < 3 = b(G_1)$. Again, we can expand this example to an infinite family of hypergraphs H in which the inequality in Theorem 2.3.2 is strict. Construct such a hypergraph H by letting $V(H) = \{u_1, \ldots, u_{k+1}\}$ and $E(H) = \big\{\{u_1, \ldots, u_k\}, \{u_2, \ldots, u_{k+1}\}\big\}$ for some $k \geq 3$. Let G_1 and G_2 be the strong and weak subhypergraphs respectively that are induced by $\{u_1, \ldots, u_k\}$. Then, $b(G_1) = k$ and $b_L(G_1) = k - 1$ since G_1 consists only of k vertices in an edge. However, $b(G_2) = k - 1$ and $b_L(G_2) = k - 2$. To see this, observe that $E(G_2) = \big\{\{u_1, \ldots, u_k\}, \{u_2, \ldots, u_k\}\big\}$, so an optimal burning sequence in G_2 is $(u_2, u_3, \ldots, u_{k-1}, u_1)$ and a minimum lazy burning set is $\{u_2, \ldots, u_{k-1}\}$. Hence, we have an infinite family of hypergraphs for which the inequality in Theorem 2.3.2 is strict.

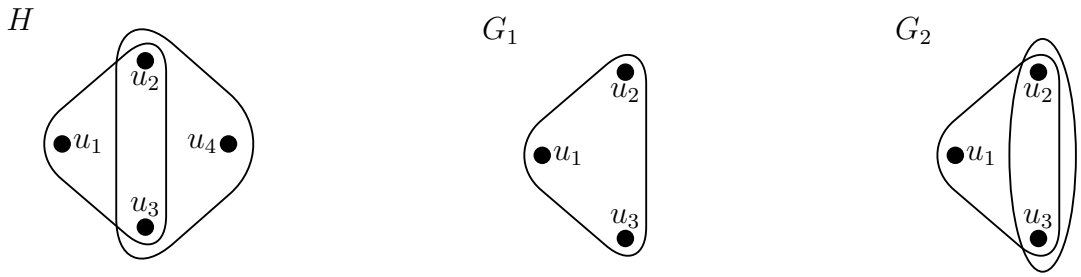

Figure 2.11: An example showing that the inequality in Theorem 2.3.2 can be strict.

We now give sufficient conditions for when a weak induced subhypergraph has (lazy) burning number no larger than that of its parent hypergraph. In fact, these conditions are the same for the lazy and round-based cases.

Theorem 2.3.3. *If G is a weak induced subhypergraph of H such that every edge in H contains a vertex of G and $E(G)$ contains no singleton edges, then $b_L(G) \leq b_L(H)$.*

Proof. Let S be a minimum lazy burning set for H. Since every edge in H contains at least two vertices that also belong to G, no fire can propagate in H unless a vertex of G is in the lazy burning set. Hence, $S \cap V(G) \neq \emptyset$. Let $S' = S \cap V(G)$. We claim S' is a lazy burning set for G.

Consider the set $V(G) \setminus S'$ of vertices in G that were not part of the lazy burning set. Label these vertices in chronological order with respect to the time step in which they catch fire in $LBG(H, S)$. That is, label $V(G) \setminus S'$ as $u_1, \ldots, u_k$ such that if $i < j$ then u_i catches fire in the same time step or in an earlier time step than u_j in $LBG(H, S)$. Note that vertices of $V(G) \setminus S'$ that catch fire in the same time step in $LBG(H, S)$ may be listed in any order relative to one another.

Suppose that S' is not a lazy burning set for G. Then, in $LBG(G, S')$, some vertex in $V(G) \setminus S'$ does not catch fire through propagation by the end of the process. Let u_q be the lowest-indexed vertex in $V(G) \setminus S'$ that never catches fire. Then, all the vertices in $V(G) \setminus S'$ that catch fire at a strictly earlier time step than u_q in $LBG(H, S)$ will catch fire in $LBG(G, S')$. Without loss of generality, these vertices are $u_1, \ldots, u_{q-1}$.

Now, consider how u_q catches fire in $LBG(H, S)$. Eventually, u_q is part of some edge e such that all other vertices in e are on fire. But e contains other vertices in G apart from u_q, all of which are on fire (they were either in the lazy burning set or caught fire strictly earlier than u_q). Hence, in the corresponding edge $e \cap V(G)$ in G, the vertices apart from u_q are either in S' or of the form u_p with $p < q$.

Therefore, in $LBG(G, S')$, eventually all vertices in $(e \cap V(G)) \setminus \{u_q\}$ will be on fire. Thus, u_q catches fire through propagation due to the edge $e \cap V(G)$. This is a contradiction, so our latest supposition must be false. That is, S' is a lazy burning set for G. Thus, $b_L(G) \leq |S'| \leq |S| = b_L(H)$. □

Theorem 2.3.4. *If G is a weak induced subhypergraph of H such that every edge in H contains a vertex of G and $E(G)$ contains no singleton edges, then $b(G) \leq b(H)$.*

Proof. Let $S = (u_1, u_2, \ldots, u_{b(H)})$ be an optimal burning sequence for H. We construct a burning sequence S' for G in which the r^{th} source is either u_r or an arbitrary vertex in $V(G)$. In particular, at each round $r \in \{1, 2, \ldots, b(H)\}$, choose the r^{th} source in S' as follows:

(1) If G became fully burned at the end of the previous round, then stop.

(2) Otherwise, if $u_r \notin V(G)$, then choose any unburned vertex in $V(G)$ as the r^{th} source in S'.

(3) Otherwise, if $u_r \in V(G)$ but u_r was on fire at the end of round $r-1$, then choose any unburned vertex in $V(G)$ as the r^{th} source in S'.

(4) Otherwise, if $u_r \in V(G)$ and u_r was not on fire at the end of round $r-1$, then choose u_r as the r^{th} source in S'.

Clearly, by the above construction, $|S'| \leq |S|$. Moreover, $|S'| < |S|$ is only true if (1) occurs, in which case S' is indeed a burning sequence for G. We therefore assume that (1) does not occur, and hence $|S'| = |S|$. We must show that S' leaves G fully burned after the final round, $b(H)$. In particular, we show $F(H, S, r) \cap V(G) \subseteq F(G, S', r)$ for each round $r \in \{1, 2, \ldots, b(H)\}$ by induction.

Base case. Consider the earliest round r_0 at which $F(H, S, r_0) \cap V(G) \neq \emptyset$, since the inclusion clearly holds for all earlier rounds. Recall that every edge in H containing a vertex of G must in fact contain two or more vertices of G. Hence, for any fire to spread to a vertex of G in $BG(H, S)$, a vertex of G must first be chosen as a source in S at a strictly earlier round. Hence, $F(H, S, r_0) \cap V(G) = \{u_{r_0}\}$ (i.e. the first vertex of G to be on fire in $BG(H, S)$ is a source). But by the above construction, in $BG(G, S')$, either u_{r_0} is chosen as the r_0^{th} source, or it was on fire in a strictly earlier round. Therefore, $F(H, S, r_0) \cap V(G) = \{u_{r_0}\} \subseteq F(G, S', r_0)$.

Inductive hypothesis. Suppose $F(H, S, r) \cap V(G) \subseteq F(G, S', r)$ for some round $r \geq r_0$.

Inductive step. By the construction of S', if $u_{r+1} \in V(G)$ then u_{r+1} is on fire at the end of round $r+1$. Consider any non-source vertex $v \in V(G)$ that catches fire through propagation

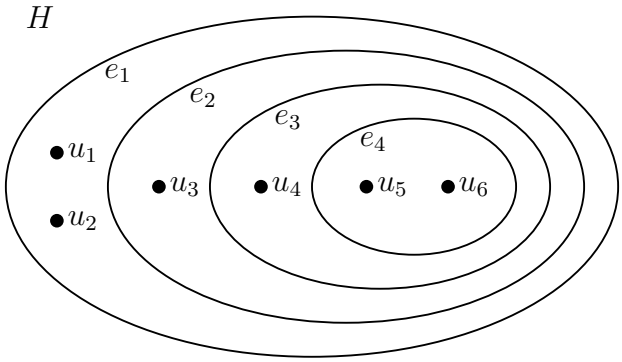

Figure 2.12: A hypergraph H that contains a weak non-induced subhypergraph with larger burning number and lazy burning number.

in $BG(H, S)$ in round $r + 1$. If v was on fire in $BG(G, S')$ at an earlier round then there is nothing to show, so assume the contrary. We must show that v catches fire through propagation in $BG(G, S')$ in round $r + 1$.

Consider an edge $e \in E(H)$ that caused fire to spread to v in $BG(H, S)$. At the end of round r, each vertex in $e \setminus \{v\}$ was on fire. Recall that $e \cap V(G)$ is a non-singleton edge in G. By the inductive hypothesis, $F(H, S, r) \cap V(G) \subseteq F(G, S', r)$, and in particular, $(e \cap V(G)) \setminus \{v\} \subseteq F(H, S, r) \cap V(G) \subseteq F(G, S', r)$. Therefore, each vertex of $(e \cap V(G)) \setminus \{v\}$ was on fire in $BG(G, S')$ at the end of round r. But then the edge $e \cap V(G)$ causes fire to spread to v in $BG(G, S')$ in round $r + 1$.

Hence, $F(H, S, r+1) \cap V(G) \subseteq F(G, S', r+1)$. We therefore have $V(G) = F(H, S, b(H)) \cap V(G) \subseteq F(G, S', b(H))$, so $F(G, S', b(H)) = V(G)$. That is, S' is a burning sequence for G, so $b(G) \leq |S'| = |S| = b(H)$. □

If we assume the weak subhypergraphs are not induced, then the analogous results to Theorems 2.3.3 and 2.3.4 do not hold in general. See Figure 2.12, and consider the non-induced weak subhypergraph G of H with $V(G) = \{u_2, u_3, u_4, u_5, u_6\}$ and $E(G) = \{e_1 \cap V(G)\} = \big\{\{u_2, u_3, u_4, u_5, u_6\}\big\}$. Note that G is not induced by its vertex set since $E(G) \neq \{e \cap V(G) \mid e \in E(H),\ e \cap V(G) \neq \emptyset\}$ (for example, e_2 is not an edge in G). But G has $b_L(G) = 4 > 2 = b_L(H)$ and $b(G) = 5 > 4 = b(H)$. To see this, first observe that $\{u_3, u_4, u_5, u_6\}$ is a minimum lazy burning set in G, whereas $\{u_1, u_6\}$ is a minimum lazy burning set in H. Also, $(u_2, u_3, u_4, u_5, u_6)$ is an optimal burning sequence in G, whereas (u_6, u_4, u_1, u_2) is an optimal burning sequence in H.

The following result shows that weak induced subhypergraphs that do not meet the other conditions in Theorems 2.3.3 and 2.3.4 may have larger (lazy) burning numbers than their

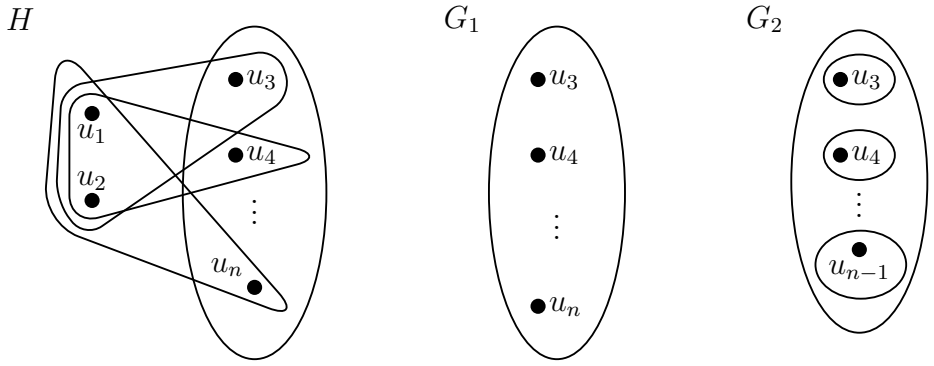

Figure 2.13: H, G_1, and G_2 from Lemma 2.3.5.

parent hypergraphs. Indeed, in Figure 2.13, the weak induced subhypergraph G_2 of H contains singleton edges, has one fewer edge than H, and has burning and lazy burning numbers strictly larger than those of H. The result also shows that strong induced subhypergraphs may have larger (lazy) burning numbers than their parent hypergraphs.

Lemma 2.3.5. *There exist hypergraphs H with strong induced subhypergraphs G_1 and weak induced subhypergraphs G_2 such that $0 < b(G_1) - b(H)$, $0 < b(G_2) - b(H)$, $0 < b_L(G_1) - b_L(H)$, and $0 < b_L(G_2) - b_L(H)$, and these differences can be arbitrarily large.*

Proof. As an example, consider $H = (V, E)$ where $V(H) = \{u_1, \ldots, u_n\}$ and $E(H) = \big\{\{u_1, u_2, u_3\}, \{u_1, u_2, u_4\}, \{u_1, u_2, u_5\}, \ldots, \{u_1, u_2, u_n\}, \{u_3, u_4, \ldots, u_n\}\big\}$. Then (u_1, u_2, u_3) is an optimal burning sequence for H, and $\{u_1, u_2\}$ is a lazy burning set for H. Hence $b(H) = 3$ and $b_L(H) = 2$.

Now, let $G_1 = (V_1, E_1)$ where $V_1 = \{u_3, u_4, \ldots, u_n\}$ and $E_1 = \big\{\{u_3, u_4, \ldots, u_n\}\big\}$, so G_1 is strongly induced by $u_3, u_4, \ldots, u_n$. Clearly, $b(G_1) = n - 2$ and $b_L(G_1) = n - 3$.

Finally, let $G_2 = (V_2, E_2)$ where $V_2 = \{u_3, u_4, \ldots, u_{n-1}\}$ and $E_2 = \big\{\{u_3\}, \{u_4\}, \ldots, \{u_{n-1}\}, \{u_3, u_4, \ldots, u_{n-1}\}\big\}$, so G_2 is weakly induced by $u_3, u_4, \ldots, u_{n-1}$. Clearly, $b(G_2) = n - 3$ and $b_L(G_2) = n - 4$. □

Notice that the weak induced subhypergraph G_2 of H in Lemma 2.3.5 need not contain fewer edges than H. That is, if instead G_2 was induced by $u_3, \ldots, u_n$, it would still have a larger (lazy) burning number than H. It does however contain singleton edges. Does this have to be the case? That is, does there exist a class of hypergraphs H with weak induced subhypergraphs G_2 such that $b(G_2) - b(H)$ and $b_L(G_2) - b_L(H)$ are both unbounded and G_2 contains no singleton edges? Of course, in light of Theorems 2.3.3 and 2.3.4, such a hypergraph G_2 would need to have fewer edges than H. We leave these as open questions.

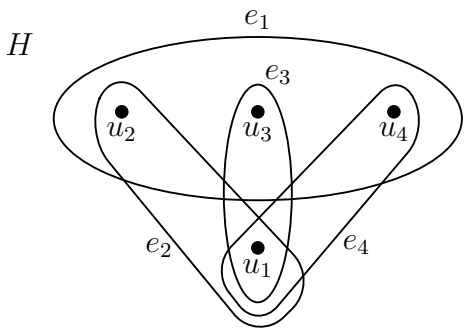

Figure 2.14: A hypergraph H that contains an induced strong subhypergraph with larger burning number and lazy burning number.

When one takes a strong subhypergraph, whether it was induced by a set of vertices or not, it is possible that the burning and lazy burning numbers both increase, and it is also possible that they both decrease. We will show this using four examples.

For an example of an *induced* strong subhypergraph G with *larger* burning number and lazy burning number than its parent hypergraph H, see Figure 2.14. Let $V(G) = \{u_2, u_3, u_4\}$ and $E(G) = \{e \in E(H) \mid e \subseteq V(G)\} = \{e_1\}$, so G is a strong subhypergraph of H that is induced by its vertex set. Then, $b_L(G) = 2 > 1 = b_L(H)$ and $b(G) = 3 > 2 = b(H)$. To see this, observe that G is simply an edge containing three vertices. But $\{u_1\}$ is a minimum lazy burning set for H and (u_1, u_2) is an optimal burning sequence for H, so $b_L(H) = 1$ and $b(H) = 2$.

Also observe that this example can be extended to provide an infinite family of hypergraphs H with the same property. One may construct H by taking $K_{1,n}$ and adding an edge e containing all n of the vertices of degree one. Then, $b_L(H) = 1$ and $b(H) = 2$. Indeed, by taking the strong subhypergraph of H induced by the vertices in e, we obtain a hypergraph that consists of n vertices in an edge, which has lazy burning number $n - 1$ and burning number n. Thus, the differences in (lazy) burning numbers can be arbitrarily large when taking an induced strong subhypergraph.

For an example of a *non*-induced strong subhypergraph G with *larger* burning number and lazy burning number than its parent hypergraph H, see Figure 2.15. Let $V(G) = \{u_1, u_2, u_3, u_4\}$ and $E(G) = \{e_3\}$, so G is a strong subhypergraph of H that is not induced by its vertex set (since $e_1, e_2 \notin E(G)$). Then $b_L(G) = 3 > 1 = b_L(H)$ and $b(G) = 4 > 3 = b(H)$. To see this, observe that G is simply an edge containing four vertices, so $b_L(G) = 3$ and $b(G) = 4$. But $\{u_1\}$ is a minimum lazy burning set for H and (u_1, u_3, u_5) is an optimal burning sequence for H, so $b_L(H) = 1$ and $b(H) = 3$.

Again, the concept illustrated in this example can be extended to an infinite class of hypergraphs. Let the vertex set of such a hypergraph H be $\{u_1, u_2, \ldots, u_n\}$, and let the edge set contain $\{u_1, u_2\}$, $\{u_1, u_2, u_3\}$, $\ldots$, and $\{u_1, \ldots, u_n\}$. Then, the hypergraph G with vertex set $\{u_1, \ldots, u_{n-1}\}$ with a single edge containing all of its vertices is a non-induced strong subhypergraph of H. Of course, $b_L(G) = n - 2$ and $b_L(H) = 1$, so the differences in lazy burning numbers can be arbitrarily large. Now, observe that $b(H) = \left\lceil \frac{n+1}{2} \right\rceil$, since an optimal burning sequence for H is $(u_1, u_3, \ldots, u_n)$ when n is odd, and $(u_1, u_3, \ldots, u_{n-1}, u_n)$ when n is even. Since $b(G) = n - 1$, the differences in the burning numbers of G and H can also be arbitrarily large.

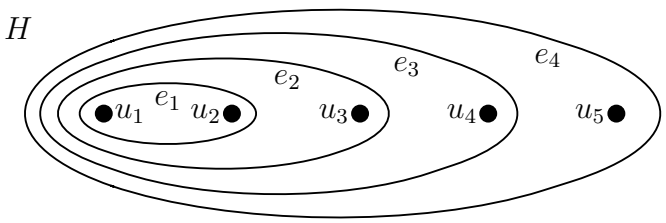

Figure 2.15: A hypergraph H that contains a non-induced strong subhypergraph with larger burning number and lazy burning number.

For an example of an *induced* strong subhypergraph G with *smaller* burning number and lazy burning number than its parent hypergraph H, see Figure 2.16. Let $V(G) = \{u_6, u_7, u_8\}$ and $E(G) = \{e \in E(H) \mid e \subseteq V(G)\} = \{e_2, e_3\}$, so G is a strong subhypergraph of H that is induced by its vertex set. Then, $b_L(G) = 1 < 5 = b_L(H)$ and $b(G) = 2 < 6 = b(H)$. To see this, observe that $\{u_6\}$ is a minimum lazy burning set in G and (u_6, u_8) is an optimal burning sequence in G, so $b_L(G) = 1$ and $b(G) = 2$. But $\{u_1, u_2, u_3, u_4, u_5\}$ is a minimum lazy burning set for H and $(u_6, u_1, u_2, u_3, u_4, u_5)$ is an optimal burning sequence for H, so $b_L(H) = 5$ and $b(H) = 6$.

Finally, for an example of a *non*-induced strong subhypergraph G with *smaller* burning number and lazy burning number than its parent hypergraph H, see Figure 2.16 again. This time, let $V(G) = \{u_6, u_7, u_8\}$ and $E(G) = \{e_2\}$, so G is a strong subhypergraph of H that is not induced by its vertex set (since $e_3 \notin E(G)$). Then, since G is simply an edge containing three vertices, and both $b_L(H)$ and $b(H)$ were found previously, it is clear that $b_L(G) = 2 < 5 = b_L(H)$ and $b(G) = 3 < 6 = b(H)$.

Both of the previous two examples (an induced/non-induced strong subhypergraph with smaller burning number) can be extended to an infinite family of hypergraphs. Indeed, one may construct H with vertex set $\{u_1, \ldots, u_n\}$, and containing edges $\{u_1, \ldots, u_\ell\}$, $\{u_\ell, u_{\ell+1}\}$,

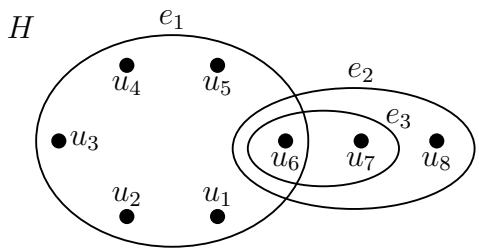

Figure 2.16: A hypergraph H that contains both an induced and a non-induced strong subhypergraph with smaller burning number and lazy burning number.

$\{u_\ell, u_{\ell+1}, u_{\ell+2}\}, \ldots, \{u_\ell, \ldots, u_n\}$ for some ℓ. Then, the induced strong subhypergraph may be obtained by simply deleting the edge $\{u_1, \ldots, u_\ell\}$, and the non-induced strong subhypergraph can be obtained by deleting all edges *but* $\{u_\ell, \ldots, u_n\}$. Of course, we also delete any isolated vertices that result from this. In both cases, we can ensure the resulting subhypergraph has smaller (lazy) burning number than H by making ℓ large enough. Informally, we can increase the (lazy) burning number of H as much as we want by making sure the edge $\{u_1, \ldots, u_\ell\}$ is very large, without affecting the (lazy) burning numbers of the two subhypergraphs in question. Of course, this means that the differences in the (lazy) burning numbers between H and the two subhypergraphs can be arbitrarily large.

2.4 Complexity

In this section we introduce two brute-force algorithms. Given a hypergraph H, the first algorithm checks to see if a given sequence of vertices in H is a valid burning sequence, and the second checks to see if a given subset of $V(H)$ is a lazy burning set. We show that both algorithms run in polynomial time with respect to the size their inputs, and hence that both round-based and lazy hypergraph burning are in **NP**. We prove that the round-based model is **NP**-complete; however, the analogous result for lazy burning is left open.

Lemma 2.4.1. *Algorithm 2 runs in time at most* $\mathcal{O}(n^3m)$ *where* $|V(H)| = n$ *and* $|E(H)| = m$.

Proof. Let F and T be initialized to arrays of length n in which v_i is in F (resp. T) if and only if the i^{th} entry in F (resp. T) is a 1. If v_i is not in F (resp. T) then the corresponding entry is 0. Hence, we may assume it takes constant time to check if a vertex v is in F or T, and similarly it takes constant time to add a vertex to F or T.

The outermost for loop (lines 2-16) executes at most ℓ times, and $\ell \leq n$. The next nested

for loop (lines 6-13) executes at most n times. The innermost for loop (lines 7-12) executes at most m times. In line 8 we must check if $e \setminus \{v\} \subseteq F$. That is, for every element of $e \setminus \{v\}$, we must check if that element is also in F. This accounts for at most $|e \setminus \{v\}| \leq n$ operations.

The operation in line 14 takes at most time n since $|T \cup \{u_i\}| \leq n$, and it occurs at the end of the outermost loop. Thus, the total run time of the algorithm is $\mathcal{O}\big(n\big((n \cdot m \cdot n) + n\big)\big) = \mathcal{O}(n^3 m)$. □

```
   input  : hypergraph H = (V, E), sequence S = (u_1, ..., u_ℓ) of vertices in H
   output: YES if S is a valid burning sequence for H, NO otherwise
 1 initialize F = ∅ and T = ∅
 2 for i = 1 ; i ≤ ℓ ; i++ do
 3 |   if u_i ∈ F then
 4 |   |   return NO
 5 |   end
 6 |   foreach v ∈ V(H) \ F do
 7 |   |   foreach e ∈ E(H) such that v ∈ e do
 8 |   |   |   if e \ {v} ⊆ F then
 9 |   |   |   |   T ← T ∪ {v}
10 |   |   |   |   Break
11 |   |   |   end
12 |   |   end
13 |   end
14 |   F ← F ∪ T ∪ {u_i}
15 |   T ← ∅
16 end
17 if F = V(H) then
18 |   return YES
19 end
20 else
21 |   return NO
22 end
```

Algorithm 2: Check potential burning sequence

Notice that Algorithm 2 takes the *incidence matrix* of the hypergraph as input. Hence, if the hypergraph has n vertices and m edges, then the size of the input is $n \cdot m$ (plus the

size of the potential burning sequence, which is negligible in comparison). In light of Lemma 2.4.1, Algorithm 2 therefore runs in polynomial time with respect to the size of its inputs.

We now formally state the decision problem for round-based hypergraph burning.

Problem: Hypergraph Burning
Instance: A Hypergraph H and a natural number k
Question: Is $b(H) \leq k$?

Theorem 2.4.2. *The hypergraph burning problem is **NP**-complete.*

Proof. Since every graph is also a hypergraph, the graph burning problem is contained within the hypergraph burning problem. Hence, the hypergraph burning problem is, in general, at least as difficult as the graph burning problem, which is **NP**-complete [9]. Since a potential solution to the hypergraph burning problem (i.e. a potential burning sequence) can be checked in polynomial time via Algorithm 2, the problem must be **NP**-complete. □

Lemma 2.4.3. *Algorithm 3 runs in time at most $\mathcal{O}(n^3m)$ where $|V(H)| = n$ and $|E(H)| = m$.*

Proof. Let F and T be initialized in the same way as Lemma 2.4.1, so look-ups and adding vertices take constant time. The outermost loop (lines 2-16) executes at most n times, since it stop once F is unchanged, and the worst case scenario is that exactly one vertex gets added to F with each iteration. Now, by the same logic as in the proof of Lemma 2.4.1, the next two nested for loops take time n and m respectively, and the operation in line 5 takes at most time n. The operation in line 11 also takes at most time n since $|T| \leq n$, and it occurs at the end of the outermost loop. We may assume the operation in line 13 takes constant time since all that is required is to check if F contains any zeros. Thus, the total run time of the algorithm is $\mathcal{O}\big(n\big((n \cdot m \cdot n) + n\big)\big) = \mathcal{O}(n^3m)$. □

input : hypergraph $H = (V, E)$, set $L \subseteq V(H)$
output: YES if L is a successful lazy burning set for H, NO otherwise

```
initialize F = L and T = ∅
repeat
    foreach v ∈ V(H) \ F do
        foreach e ∈ E(H) such that v ∈ e do
            if e \ {v} ⊆ F then
                T ← T ∪ {v}
                Break
            end
        end
    end
    F ← F ∪ T
    T ← ∅
    if F = V(H) then
        return YES
    end
until F is unchanged;
return NO
```

Algorithm 3: Check potential lazy burning set

We now formally state the decision problem for lazy hypergraph burning.

Problem: Lazy Hypergraph Burning
Instance: A Hypergraph H and a natural number k
Question: Is $b_L(H) \leq k$?

By the same logic that was used previously, Algorithm 3 runs in polynomial time with respect to the size of its inputs. Since this algorithm checks a potential solution for the lazy hypergraph burning problem, this problem is in **NP**. Intuitively, lazy burning does not seem to be a significantly easier problem than round-based burning. Hence, we pose the following conjecture (which we could not find in the literature on H-bootstrap percolation).

Conjecture 2.4.4. *The lazy hypergraph burning problem is **NP**-complete.*

To prove Conjecture 2.4.4, it would suffice to show that a solution to Lazy Hypergraph Burning yields a solution to some other decision problem which is **NP**-complete. One obvious

candidate is round-based hypergraph burning, although it is unclear how a solution to lazy burning could yield a solution to its round-based counterpart. The round-based version focuses on the number of rounds in the game, and since sources must be chosen with the goal of spreading the fire as fast as possible, it feels like there is "more strategy" involved in the selection of sources versus the selection of a lazy burning set.

Another potential candidate is a process called *zero-forcing*, which is known to be **NP**-complete [1, 17, 24, 40]. We will use the terminology of burning when discussing zero-forcing for the sake of simplicity. In zero-forcing, a subset of vertices in a graph is initially set on fire. Then, in each round, for each burned vertex with exactly one unburned neighbour, that unburned neighbour catches fire. Indeed, zero-forcing and lazy hypergraph burning are quite similar, since they both involve an initial set of burned vertices, and a propagation rule which allows for the "automated" spread of the fire. See [2, 6, 7, 27] for more on zero-forcing.

Chapter 3

Steiner Triple Systems

In this chapter we investigate the burning game on Steiner triple systems, which are a well-studied family of combinatorial designs. We obtain a lower bound on the burning number and an upper bound on the lazy burning number of an arbitrary Steiner triple system. We also use a "doubling construction" to show that there exist Steiner triple systems with arbitrary lazy burning number.

3.1 Definitions and Motivation

We briefly discuss the basic definitions from design theory, and explain how a design induces a hypergraph on which the burning game can be played. Although it is not the origin of the definitions presented in this section, we use [38] as a reference for the theory of designs.

Definition 3.1.1. ([38]) *A* design *is a pair* $(X, \mathcal{A})$ *such that* X *is a set of elements called* points *and* $\mathcal{A}$ *is a multiset consisting of nonempty subsets of* X*, called* blocks*.*

Definition 3.1.2. ([38]) *Let* $v, k, \lambda \in \mathbb{N}$ *such that* $v > k \geq 2$*. A* (v, k, λ)-balanced incomplete block design*, abbreviated* (v, k, λ)-BIBD *or* BIBD(v, k, λ)*, is a design* $(X, \mathcal{A})$ *that satisfies the following:*

- $|X| = v$*,*
- *each block contains exactly* k *points, and*
- *every pair of distinct points is contained in exactly* λ *blocks.*

It is common to use a slight abuse of notation when writing the blocks of a BIBD for the sake of readability. A block which contains the points a, b, and c is technically the set $\{a, b, c\}$, but when writing multiple blocks in a row it is easier to write such a block as simply abc.

Definition 3.1.3. ([38]) *A* Steiner triple system *of* order v*, or* STS(v)*, is a* $(v, 3, 1)$-BIBD.

An STS(v) exists if and only if $v \equiv 1, 3 \pmod 6$ [30]. Hence, if $v \equiv 1, 3 \pmod 6$ we will call v *admissible*, and otherwise we will call v *inadmissible*. For example, consider the STS(7) and STS(9) (which are unique up to isomorphism). Their block sets may be written as $\{124, 235, 346, 457, 561, 672, 713\}$ and $\{123, 456, 789, 147, 258, 369, 159, 267, 348, 168, 249, 357\}$ respectively.

A design $(X, \mathcal{A})$ induces a hypergraph H on which the burning game can be played, since we can take $V(H) = X$ and $E(H) = \mathcal{A}$. Hence, we may view an STS as a 3-uniform hypergraph in the context of burning.

Many alternative propagation rules will coincide with the original propagation rule when burning a 3-uniform hypergraph (see Chapter 4). In particular, many of them are equivalent to the following propagation rule: if two vertices are on fire in an edge, then the third vertex catches fire in the next time step. This is one reason we are interested in burning 3-uniform hypergraphs – the results that we obtain with the original propagation rule in this chapter will also apply for many variants of hypergraph burning. Of course, we devote our attention to Steiner triple systems because they have the most well-studied structure among 3-uniform hypergraphs.

3.2 General Results on Steiner Triple Systems

Denote the set of vertices of the STS(7) by $V = \{1, 2, ..., 7\}$, and the set of edges by $E = \big\{\{1, 2, 4\}, \{2, 3, 5\}, \{3, 4, 6\}, \{4, 5, 7\}, \{5, 6, 1\}, \{6, 7, 2\}, \{7, 1, 3\}\big\}$. Then, without loss of generality, the arsonist may choose vertices 1 and 2 as the first and second source respectively. In round 3, the fire spreads to 4 and the arsonist chooses as a source any vertex other than 1, 2, or 4. In round 4, the fire spreads to the remaining vertices and the arsonist must choose a redundant source. Thus, when burning the STS(7), if the arsonist plays optimally by picking no redundant sources until the final round, we have $f_1 = 1$, $f_2 = 2$, $f_3 = 4$, $f_4 = 7$, $f'_1 = f'_2 = 0$, $f'_3 = 1$, and $f'_4 = 7$. So, the STS(7) has burning number 4.

Lemma 3.2.1. *For any valid burning sequence on an STS(v) with $v > 7$, if the arsonist never chooses a redundant source in the first four rounds, we have $f_1 = 1$, $f_2 = 2$, $f_3 = 4$, and $f_4 = 8$. Also, $f'_1 = f'_2 = 0$, $f'_3 = 1$, and $4 \leq f'_4 \leq 7$.*

Proof. Observe Figure 3.1, in which the first four sources in a burning sequence for an arbitrary STS are u_1, u_2, u_3, and u_4. Clearly only the first source u_1 is burning in round 1, so $f_1 = 1$ and $f'_1 = 0$. In round 2, there is no propagation and the arsonist picks a second source u_2. Since u_1 and u_2 are the only vertices that are on fire in round 2 we have $f_2 = 2$ and $f'_2 = 0$. Since we are burning a Steiner triple system, there is a unique edge $\{u_1, u_2, a\}$. Thus, in round 3, a catches on fire and the arsonist picks a non-redundant source $u_3 \neq a$. In round 3, u_1, u_2, u_3, and a are on fire, and the edge $\{u_1, u_2, a\}$ is completely burned so $f_3 = 4$ and $f'_3 = 1$. Now, since we are burning an STS, our most recent source u_3 appears in an edge with each of u_1, u_2, and a. Let these edges be $\{u_3, u_1, b\}$, $\{u_3, u_2, c\}$, and $\{u_3, a, d\}$. Note that b, c, and d are all distinct. For example, if $b = c$ then u_3 appears with the vertex $b = c$ in more than one edge. Similarly, $\{b, c, d\} \cap \{u_1, u_2, a\} = \emptyset$. For example, if $b = u_2$ then u_3 would appear with $b = u_2$ in more than one edge. The arsonist also chooses some vertex u_4 different from u_1, u_2, u_3, a, b, c, and d as a source in this round, so $f_4 = 8$. For the lower bound on f'_4, observe that it is possible that $\{u_1, u_2, a\}$, $\{u_3, u_1, b\}$, $\{u_3, u_2, c\}$, and $\{u_3, a, d\}$ are the only fully burned edges at round 4, so $4 \leq f'_4$. However, it is also possible that each of $\{u_1, c, d\}$, $\{u_2, b, d\}$, and $\{a, b, c\}$ are edges in the STS, in which case $f'_4 = 7$, so $4 \leq f'_4 \leq 7$. □

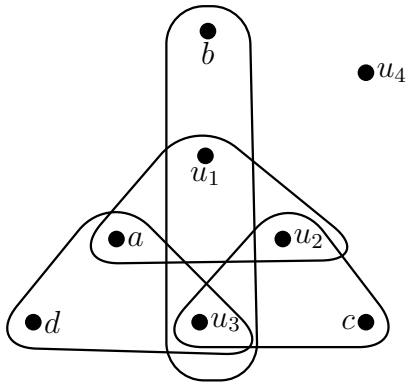

Figure 3.1: The first four edges that burn when burning an STS(v).

Our next major result is a lower bound on the burning number of an arbitrary STS. Recall that for $a, b \in \mathbb{N}$ with $a < b$, $\binom{a}{b} = 0$ by convention.

Theorem 3.2.2. *For any STS(v) and any round $r \geq 1$, $\Psi_r \leq h(r)$ where $h(r) = \binom{h(r-1)}{2} - 2h(r-1) + 3r - 2$ and $h(1) = 1$.*

Proof. Consider the maximum number of vertices that could possibly be on fire in an STS at the end of round r, and call this value $h(r)$. Thus, by definition, $\Psi_r \leq h(r)$ for all rounds $r \geq 1$ for any STS. When the fire spreads this way in an STS (i.e. so that $f_r = h(r)$), denote the number of edges that are on fire at the end of round r by $g(r)$. Note that $g(r)$ is *not* necessarily the maximum number of edges that could be completely on fire at the end of round r.

For $h(r)$ vertices to be on fire at the end of round r, two things must happen. First, $h(r-1)$ vertices must be on fire at the end of round $r-1$, and second, the fire must spread to as many distinct vertices as possible in round r. Assume $h(r-1)$ vertices and $g(r-1)$ edges are on fire at the end of round $r-1$. One of the $h(r-1)$ vertices is the source chosen by the arsonist in round $r-1$, call it z. To maximize the number of vertices that are on fire by the end of round r, assume the arsonist was able to pick a non-redundant z such that there is no edge $\{x, y, z\}$ where x and y are both on fire at the end of round $r-1$. We must now determine the maximum number of distinct vertices the fire could spread to in round r.

The best case scenario is that every unordered pair of burned vertices spreads fire to a distinct new vertex. Since $h(r-1)$ vertices are on fire at the end of round $r-1$, this would account for $\binom{h(r-1)}{2}$ newly burned vertices. However, this is an overestimate, as we know some unordered pairs of vertices do not cause any fire to spread in round r. In particular, since each unordered pair of vertices appears together in exactly one edge, a pair of vertices $\{a, b\}$ does not cause any fire to spread in round r if and only if a and b belong to an edge that was fully burned at the end of round $r-1$. The number of such edges is $g(r-1)$, and for each such edge there are $\binom{3}{2} = 3$ unordered pairs of vertices that we know do not spread any fire in round r. Of course, the arsonist picks a non-redundant source in round r as well. Thus, the maximum number of newly burned vertices in round r is $\binom{h(r-1)}{2} - 3g(r-1) + 1$. Since $h(r-1)$ vertices were already on fire in round $r-1$, no more than $h(r-1) + \binom{h(r-1)}{2} - 3g(r-1) + 1$ vertices could possibly be on fire by the end of round r, so $\Psi_r \leq h(r-1) + \binom{h(r-1)}{2} - 3g(r-1) + 1 = h(r)$.

Now, when fire spreads to a new vertex, it means that some edge of the STS has become completely burned. In the process above, we showed that at most $\binom{h(r-1)}{2} - 3g(r-1)$ (assumedly distinct) vertices could be set on fire in round r due to propagation, and thus at least $\binom{h(r-1)}{2} - 3g(r-1)$ edges become completely burned in round r when the fire spreads as fast as possible (recall that a non-redundant source chosen by the arsonist never "completes an edge"). But $g(r-1)$ edges were on fire at the end of round $r-1$ so the number of edges that are on fire at the end of round r is at least $g(r-1) + \binom{h(r-1)}{2} - 3g(r-1) = \binom{h(r-1)}{2} - 2g(r-1)$. But since we want the fire to spread as fast as possible at each round, we assume that all

newly burned vertices in round r are distinct and no set of 3 of the newly burned vertices form an edge in the STS. That is, if vertices a, b, and c became burned in round r, then $\{a, b, c\}$ is not an edge in the STS, since, if it were, then the pairs (a, b), (a, c), and (b, c) would not spread fire to any new vertices in the next round. Thus, $g(r) = \binom{h(r-1)}{2} - 2g(r-1)$.

Now, observe that $h(r) - g(r) = r$. Specifically, we have

$$\begin{aligned} h(r) - g(r) &= \left[h(r-1) + \binom{h(r-1)}{2} - 3g(r-1) + 1\right] - \left[\binom{h(r-1)}{2} - 2g(r-1)\right] \\ &= h(r-1) - g(r-1) + 1 \end{aligned} \tag{3.1}$$

But the maximum number of vertices that can be on fire at the end of round 1 is $h(1) = 1$, and the number of edges that are on fire in this case is $g(1) = 0$. Thus, $h(1) - g(1) = 1$. As an inductive hypothesis, assume $h(r) - g(r) = r$ for some $r \in \mathbb{N}$. Then, using this hypothesis and equation (3.1), we have $h(r+1) - g(r+1) = h(r) - g(r) + 1 = r + 1$. Therefore, by induction, $h(r) - g(r) = r$ for all $r \in \mathbb{N}$.

Finally, we have

$$\begin{aligned} h(r) &= h(r-1) + \binom{h(r-1)}{2} - 3g(r-1) + 1 \\ &= h(r-1) + \binom{h(r-1)}{2} - 3[h(r-1) - r + 1] + 1 \\ &= h(r-1) + \binom{h(r-1)}{2} - 3h(r-1) + 3r - 3 + 1 \\ &= \binom{h(r-1)}{2} - 2h(r-1) + 3r - 2 \end{aligned}$$

□

Corollary 3.2.3. *Let H be a Steiner triple system on v vertices and let h be the function defined in the proof of Theorem 3.2.2. If r is the smallest natural number satisfying $v \leq h(r)$, then $r \leq b(H)$.*

Proof. Recall that in any STS(v), no more than $h(r)$ vertices can possibly be on fire at the end of round r. Since $h(r-1) < v$, a burning sequence of length $r-1$ cannot possibly burn all v vertices of the STS. Thus, a sequence of length r or greater is required, so $r \leq b(H)$. □

Our next objective is to obtain an upper bound on the lazy burning number of an arbitrary STS. This bound can be seen in Theorem 3.2.7, which combines Algorithm 1, Lemma 3.2.5, and Theorem 3.2.6. Contrary to normal conventions, we allow STSs of orders 1 and 3 in

r	$h(r)$	$g(r)$
1	1	0
2	2	0
3	4	1
4	8	4
5	25	20
6	266	260
7	34732	34725
8	603069104	603069096

Table 3.1: The first few values for the functions h and g defined in the proof of Theorem 3.2.2.

the following results. The STS(1) is a pair (V, E) with $|V| = 1$ and $E = \emptyset$, and the STS(3) consists of three vertices in an edge. The STS(1) and the STS(3) are called *trivial* Steiner triple systems, and all other STSs are called *nontrivial*.

Definition 3.2.4. *Let $H = (X, \mathcal{A})$ be a BIBD(v, k, λ) and $G = (Y, \mathcal{B})$ be a BIBD(u, k, λ) with $u < v$. If $Y \subsetneqq X$ and $\mathcal{B} \subsetneqq \mathcal{A}$ then G is a* sub-design *of H and we write $G \subsetneqq H$ (a slight abuse of notation). If, in addition, both H and G are Steiner triple systems, we say G is a* sub-STS *of H.*

Lemma 3.2.5. *When applying Algorithm 1 to an STS(v), each time the inner loop finishes, the edges that are completely on fire induce a sub-STS.*

Proof. Observe that once the inner loop of Algorithm 1 has finished (within any iteration of the outer loop), there are only fully burned edges, edges containing a single burned vertex, and edges containing no burned vertices.

Denote the STS(v) by H. Denote the design induced by the fully burned edges of H after the inner loop of Algorithm 1 has finished by G. Since H is an STS, no two vertices of G appear together in more than one edge of G. We must show that every pair of vertices in G appear together in some edge of G.

Let $u, w \in V(G)$ and suppose there is no edge in $E(G)$ containing both u and w. Since H is an STS, there is some edge $B \in E(H)$ such that $u, w \in B$. Write $B = \{u, w, z\}$ for some $z \in V(H) \setminus V(G)$ ($z \notin V(G)$ since, otherwise, B would be an edge in $E(G)$). Since $z \notin V(G)$, z is not on fire. But then the inner loop of Algorithm 1 has not yet ended, since fire must propagate to z due to $B = \{u, w, z\}$. This is a contradiction, so every pair of vertices in G appears together in exactly one edge of G. Hence, G is an STS. □

We present the proof of the following well-known theorem for completeness, along with Figure 3.2 for additional clarity.

Theorem 3.2.6. *If T is an STS and A is a sub-STS of T then $|V(T)| \geq 2|V(A)| + 1$.*

Proof. Consider any $v \in V(T) \setminus V(A)$. Each $u_i \in V(A)$ must appear in an edge of T with v. Suppose the third vertex in such an edge also belongs to $V(A)$ (i.e. there is an edge $\{u_1, u_2, v\}$ with $u_1, u_2 \in V(A)$). Then $\{u_1, u_2, v\} \in E(A)$ and hence $v \in V(A)$, which is a contradiction. Therefore, for each $u_i \in V(A)$, there is an edge $\{u_i, v, z_i\} \in E(T)$ such that $z_i \in V(T) \setminus V(A)$.

Now, suppose for some distinct $u_i, u_j \in V(A)$ that $z_i = z_j$. Then v and z_i appear together in two edges of T, $\{u_i, v, z_i\}$ and $\{u_j, v, z_i\}$, which is a contradiction.

Thus, for each $u_i \in V(A)$ there is a unique vertex $z_i \in V(T) \setminus V(A)$ that is different from v. Since $u_1, \ldots, u_{|V(A)|}, v, z_1, \ldots, z_{|V(A)|}$ are $(2|V(A)| + 1)$ unique vertices in T, the result follows. □

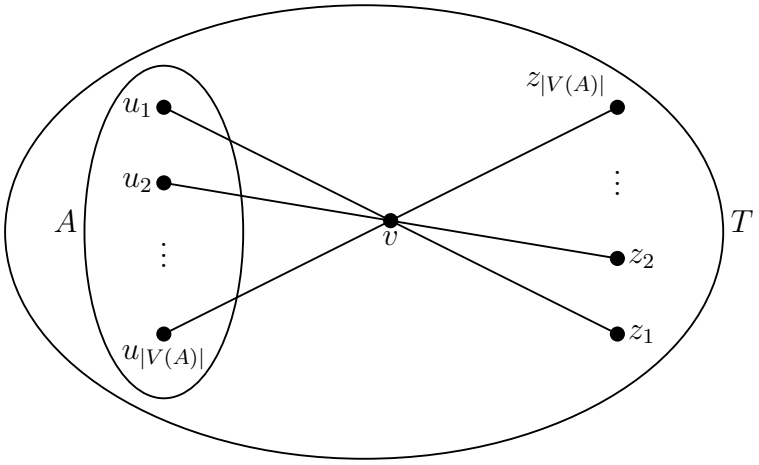

Figure 3.2: A visual proof of the bound $|V(T)| \geq 2|V(A)| + 1$.

Theorem 3.2.7. *For any Steiner triple system H on v vertices, $b_L(H) \leq \lfloor \log_2(v+1) \rfloor$.*

Proof. Consider the worst-case scenario when applying Algorithm 1 to H. By Lemma 3.2.5, each time the inner loop of the algorithm finishes, a sub-STS of H is induced by the fully burned edges. Hence, the worst-case scenario is that H contains the maximum possible number of nested sub-STSs, and the algorithm chooses vertices as "poorly" as possible such that each nested sub-STS contributes a vertex to the lazy burning set. That is, each time the inner loop finishes, the sub-STS that is induced by the burned edges is as small as possible.

The smallest two nested sub-STSs contained in any STS are the STS(1) and the STS(3), which have orders 1 and 3 respectively. Write a_r as the order of the r^{th} smallest nested sub-STS, so clearly $a_1 = 1$ and $a_2 = 3$. By Theorem 3.2.6, we know that $a_r = 2a_{r-1} + 1$. We claim that $a_r = 2^r - 1$ is the closed formula for this sequence. Indeed, $a_1 = 2^1 - 1$, and if we assume that $a_{r-1} = 2^{r-1} - 1$ for some $r > 1$ (as an inductive hypothesis), we can calculate $2^r - 1 = 2^r - 2 + 1 = 2\left(2^{r-1} - 1\right) + 1 = 2a_{r-1} + 1 = a_r$. The first few terms in this sequence are $1, 3, 7, 15, 31, \ldots$.

Now, the worst case scenario is that H has nested sub-STSs of order $a_1, a_2, a_3, \ldots, a_\ell, v$, where ℓ is the *largest* natural number such that $v \geq 2a_\ell + 1$. Thus, including H itself, H has at most $\ell + 1$ nested sub-STSs. We solve for ℓ:

$$\begin{aligned} v &\geq 2a_\ell + 1 \\ v &\geq 2(2^\ell - 1) + 1 \\ v &\geq 2^{\ell+1} - 1 \\ \log_2(v+1) &\geq \ell + 1 \\ \log_2(v+1) - 1 &\geq \ell \\ \lfloor \log_2(v+1) - 1 \rfloor &= \ell \end{aligned}$$

Therefore, the largest lazy burning set the algorithm could produce for H is of size $\ell + 1 = \lfloor \log_2(v+1) - 1 \rfloor + 1 = \lfloor \log_2(v+1) \rfloor$. Hence, $b_L(H) \leq \lfloor \log_2(v+1) \rfloor$. □

Definition 3.2.8. *Let H be an STS(v) and $H_1 \subsetneqq H_2 \subsetneqq \cdots \subsetneqq H_\ell = H$ be a chain of nested sub-STSs for H (where H_1 is an STS(1) and H_2 is an STS(3)). We call this chain a* tight sub-STS chain *of length ℓ if for every sub-STS G of H, $H_i \subsetneqq G \subsetneqq H_{i+1}$ is false for each $i \in \{1, 2, \ldots, \ell - 1\}$.*

Informally, a sub-STS chain for H is tight if there is no sub-STS G of H that is "in between" two consecutive members of the chain (with respect to inclusion). The proof of Theorem 3.2.7 implicitly uses the concept of sub-STS chains, and we can use the concept of tight sub-STS chains to obtain further results on lazily burning STSs.

Lemma 3.2.9. *Let H be a Steiner triple system. There exists a tight sub-STS chain for H.*

Proof. Let H_1 be an STS(1) and H_2 be an STS(3) with $V(H_1) \subseteq V(H_2) \subseteq V(H)$. Then $H_1 \subsetneqq H_2 \subsetneqq H$ is a chain of nested sub-STSs for H. If there is no STS G with $H_2 \subsetneqq G \subsetneqq H$ then the chain $H_1 \subsetneqq H_2 \subsetneqq H$ is tight, so assume such a G exists. Then, we can "add" G

to our chain, to obtain the chain $H_1 \subsetneqq H_2 \subsetneqq G \subsetneqq H$. If this chain is not tight, then there exists an STS J such that either $H_2 \subsetneqq J \subsetneqq G$ or $G \subsetneqq J \subsetneqq H$. In either case, we can "add" J to our chain. Clearly, this process can be repeated until no more sub-STSs can be added to the chain. The resulting chain must therefore be tight. □

Theorem 3.2.10. *Let H be an STS(v) and choose any tight sub-STS chain for H. Denote the length of this chain by ℓ. Then $b_L(H) \leq \ell$.*

Proof. Let $H_1 \subsetneqq H_2 \subsetneqq \cdots \subsetneqq H_\ell = H$ be a tight sub-STS chain of length ℓ for H. We claim that, while applying Algorithm 1 to H, we can choose the vertices to add to L in such a way that H_i is on fire after the i^{th} iteration of the inner loop for each $i \in \{1, 2, \ldots, \ell\}$. As an inductive basis, clearly we can ensure H_1 is on fire after the first iteration of the inner loop, since $V(H_1)$ is a single vertex. Now suppose for some i that H_i is on fire after the i^{th} iteration of the inner loop. We must show that we can choose the next vertex to add to L in such a way that H_{i+1} is on fire after the next iteration. In particular, we claim that we can choose any vertex $u \in V(H_{i+1}) \setminus V(H_i)$.

Denote by $H_i + u$ the sub-STS of H that is on fire after we burn u and let the fire propagate until it stops. We must show $H_i + u = H_{i+1}$.

First, clearly $H_i + u \subsetneqq H_{i+1}$ is *false*. This is because $H_i \subsetneqq H_i + u$, and we assumed there is no sub-STS G of H with $H_i \subsetneqq G \subsetneqq H_{i+1}$.

Now, suppose $V(H_i + u) \setminus V(H_{i+1}) \neq \emptyset$. Recall that $V(H_i) \cup \{u\} \subseteq V(H_{i+1})$, and let v be a vertex in $V(H_i + u) \setminus V(H_{i+1})$ that catches fire the earliest during the inner loop of Algorithm 1 (i.e. during propagation). Then, there is an edge $\{x, y, v\}$ such that x and y caused fire to spread to v during the inner loop. But $x, y \in V(H_{i+1})$ since they were on fire before v. Thus, $\{x, y, v\} \in E(H_{i+1})$, and hence $v \in V(H_{i+1})$, which is a contradiction. Therefore $V(H_i + u) \setminus V(H_{i+1}) = \emptyset$.

Since $H_i + u \subsetneqq H_{i+1}$ is *false*, $V(H_i + u) \setminus V(H_{i+1}) = \emptyset$, and $H_i \subsetneqq H_i + u \subsetneqq H$, we must have $H_i + u = H_{i+1}$. Thus, for each $i \in \{1, 2, \ldots, \ell\}$, we can ensure that H_i is on fire after the i^{th} iteration of the inner loop. This yields a lazy burning set for H of size ℓ, so $b_L(H) \leq \ell$. □

Corollary 3.2.11. *Let H be an STS(v), and let ℓ be the length of a* shortest *tight sub-STS chain for H. Then $b_L(H) \leq \ell$.*

We have been unable to prove that the length of a shortest sub-STS chain for an STS H is a *lower* bound on $b_L(H)$. The obvious direction for a proof is to "plug in" a minimum

lazy burning set to Algorithm 1, and show that the resulting sub-STS chain is tight, and of the shortest possible length. It is unclear if either of these properties are guaranteed, so we present the following conjecture.

Conjecture 3.2.12. *Let H be an STS(v), and let ℓ be the length of a* shortest *tight sub-STS chain for H. Then $b_L(H) = \ell$.*

Next, we combine the following corollary with a result from [23] to show that there are infinitely many STSs with lazy burning number three; see Corollary 3.2.15.

Corollary 3.2.13. *If H is a Steiner triple system on $v > 3$ vertices that has no nontrivial subsystems, then $b_L(H) = 3$.*

Proof. Any lazy burning set of size 2 would cause fire to spread to a third vertex and no others, so $b_L(H) \geq 3$. Also, $\mathrm{STS}(1) \subsetneqq \mathrm{STS}(3) \subsetneqq H$ is a tight sub-STS chain for H. Thus, by Thereom 3.2.10, $b_L(H) \leq 3$. □

Theorem 3.2.14. ([23]) *For every admissible $v \in \mathbb{N}$ there is an STS(v) with no nontrivial subsystems.*

Corollary 3.2.15. *For every admissible $v \in \mathbb{N}$ with $v \geq 7$ there is an STS(v) with lazy burning number equal to 3.*

Corollary 3.2.3 (on page 53) provides a lower bound on the burning number of an STS(v) that increases monotonically with v. Hence, in light of Corollary 3.2.15, the burning number and lazy burning number of an STS can differ by an arbitrarily large amount. In the following result, we also see that the lazy burning number of an STS can differ from the upper bound in Theorem 3.2.7 by an arbitrarily large amount.

Corollary 3.2.16. *For any fixed $k \in \mathbb{N}$ there are infinitely many Steiner triple systems H such that $b_L(H)$ and the upper bound in Theorem 3.2.7 differ by at least k.*

Proof. Let $H_1, H_2, H_3, \ldots$ be an infinite list of Steiner triple systems with no nontrivial subsystems such that $n_i = |V(H_i)|$ and $n_i < n_{i+1}$ for all i. Such a list must exist by Theorem 3.2.14 [23]. Note that $b_L(H_i) = 3$ for all i. Now, since $\lim_{n\to\infty} \lfloor \log_2(n+1) \rfloor = \infty$, there is i large enough such that $\lfloor \log_2(n_i+1) \rfloor - 3 > k$. Furthermore, $\lfloor \log_2(n_j+1) \rfloor - 3 > k$ for all $j > i$. Thus, $H_i, H_{i+1}, H_{i+2}, \ldots$ is an infinite list of STSs whose lazy burning numbers (all equal to 3) differ from the upper bound in Theorem 3.2.7 by at least k. □

Is it true that for any $k \in \mathbb{N} \setminus \{1, 2\}$, there are infinitely many Steiner triple systems with a shortest tight sub-STS chain of length k? If the answer is "yes," and if Conjecture 3.2.12 is true, then the following Conjecture 3.2.17 would also be true. We leave these as open questions.

Conjecture 3.2.17. *For any fixed $k \in \mathbb{N} \setminus \{1, 2\}$, there are infinitely many Steiner triple systems H with $b_L(H) = k$.*

3.3 Results on Doubling Steiner Triple Systems

Given a Steiner triple system H on v vertices, there are several constructions for an STS G on $2v + 1$ vertices which contains H as a sub-STS. We use Construction 2.15 from [18] (pp. 59) that produces such an STS, which we will refer to as the "double and add one" method. If H is the given STS, then denote the STS produced by this method by H^*. If $V(H) = \{1, 2, \ldots, v\}$ then $V(H^*) = \{1, 2, \ldots, v, 1', 2', \ldots, v', \infty\}$, and $E(H^*)$ is constructed as follows:

- $E(H) \subseteq E(H^*)$
- $\{x, \infty, x'\} \in E(H^*)$ for each $x \in V(H)$
- For each edge $\{x, y, z\}$ in $E(H)$, the edges $\{x', y', z\}$, $\{x', y, z'\}$, and $\{x, y', z'\}$ are in $E(H^*)$.

We may often wish to apply this method repeatedly, obtaining H^*, then $(H^*)^*$, and so on. We will refer to the hypergraph produced from H by repeatedly applying the double and add one method n times by H^{n*}. Note that $H \subsetneqq H^* \subsetneqq H^{2*} \subsetneqq \cdots \subsetneqq H^{(n-1)*} \subsetneqq H^{n*}$ is a chain of nested sub-STSs for H^{n*} such that, for every sub-STS G of H^{n*}, $H^{i*} \subsetneqq G \subsetneqq H^{(i+1)*}$ is false for each $i \in \{0, 1, \ldots, n\}$ (this is by Theorem 3.2.6 on page 55). Hence, $H \subsetneqq H^* \subsetneqq H^{2*} \subsetneqq \cdots \subsetneqq H^{(n-1)*} \subsetneqq H^{n*}$ is a tight sub-STS chain for H^{n*} of maximum length.

Lemma 3.3.1. *If H is an STS(v) and G is an STS($2v+1$) such that $H \subsetneqq G$ then $b_L(G) \leq b_L(H) + 1$.*

Proof. Let T be an optimal lazy burning set for H. Let $v \in V(G) \setminus V(H)$. We claim $T \cup \{v\}$ is a lazy burning set for G. Consider what happens when we apply Algorithm 1 to G using $T \cup \{v\}$. After burning the last vertex of T, the fire will propagate until $V(H)$ is on fire. We claim that after burning v, the fire will propagate until all of $V(G)$ is on fire.

Suppose that a strict subset of $V(G)$ is on fire after we burn v and allow the fire propagate (via the inner loop of Algorithm 1). Then by Lemma 3.2.5, the burned edges form an STS, call it J, with $H \subsetneqq J \subsetneqq G$. But then $|V(J)| < |V(G)| = 2|V(H)| + 1$, which contradicts Theorem 3.2.6.

Since applying Algorithm 1 to G using $T \cup \{v\}$ results in $V(G)$ being completely burned, $T \cup \{v\}$ must be a lazy burning set for G. Thus, $b_L(G) \leq |T \cup \{v\}| = b_L(H) + 1$. □

Theorem 3.3.2. *Let H be a Steiner triple system. Then $b_L(H) \leq b_L(H^*) \leq b_L(H) + 1$.*

Proof. The upper bound is by Lemma 3.3.1. Alternatively, it is easy to show that if T is an optimal lazy burning set for H, then $T \cup \{\infty\}$ is a lazy burning set for H^*.

We now prove the lower bound. Let S' be an optimal lazy burning set for H^*. Let S be the subset of $V(H)$ created from S' in the following way: if $S' = \{x_1, \ldots, x_k, y'_1, \ldots, y'_\ell, \infty\}$ or $S' = \{x_1, \ldots, x_k, y'_1, \ldots, y'_\ell\}$, then $S = \{x_1, \ldots, x_k\} \cup \{y_1, \ldots, y_\ell\}$ (i.e. we remove infinity if necessary and "un-prime" the "primed" vertices). We claim S is a lazy burning set for H.

In particular, we prove the following statement for all admissible $i \in \mathbb{N} \cup \{0\}$ by strong induction: If x or x' is on fire in $LBG(H^*, S')$ after the i^{th} propagation step, then x is on fire in $LBG(H, S)$ after the i^{th} propagation step.

First, consider the base case $i = 0$ (i.e. just after the lazy burning sets are set on fire and before any propagation). If x (or x') is on fire at this vertex in time in $LBG(H^*, S')$, then it is because $x \in S'$ (or $x' \in S'$). In either case, by the definition of S, $x \in S$, so x is on fire in $LBG(H, S)$ after $i = 0$ propagation steps.

For the inductive hypothesis, we may assume the following for all $k \in \{0, 1, \ldots, i\}$: If x or x' is on fire in $LBG(H^*, S')$ after the k^{th} propagation step, then x is on fire in $LBG(H, S)$ after the k^{th} propagation step.

Finally, for the inductive step, we must show that if x or x' is on fire in $LBG(H^*, S')$ after the $(i+1)^{th}$ propagation step then x is on fire in $LBG(H, S)$ after the $(i+1)^{th}$ propagation step.

Suppose x' is on fire in $LBG(H^*, S')$ after the $(i+1)^{th}$ propagation step. If x' was also on fire after an earlier propagation step then we simply apply the inductive hypothesis, so assume that x' catches fire in the $(i+1)^{th}$ propagation step. There are two cases to consider. First, suppose x' catches fire due to an edge of the form $\{x, \infty, x'\}$. Then x and ∞ were on fire in $LBG(H^*, S')$ after the i^{th} propagation step. By the inductive hypothesis, this means x was on fire in $LBG(H, S)$ after the i^{th} propagation step. Second, suppose x' catches fire due to an edge of the form $\{x', y', z\}$. Then y' and z were on fire in $LBG(H^*, S')$ after the

i^{th} propagation step. By the inductive hypothesis, y and z must be on fire in $LBG(H, S)$ after the i^{th} propagation step. Hence, if x was not already on fire in $LBG(H, S)$, then the edge $\{x, y, z\}$ causes fire to spread to x in the $(i+1)^{th}$ propagation step of $LBG(H, S)$.

Now, suppose x is on fire in $LBG(H^*, S')$ after the $(i+1)^{th}$ propagation step. If x was also on fire after an earlier propagation step then we simply apply the inductive hypothesis, so assume that x catches fire in the $(i+1)^{th}$ propagation step of $LBG(H^*, S')$. There are three cases to consider. First, suppose x catches fire in $LBG(H^*, S')$ due to an edge of the form $\{x, \infty, x'\}$. Then ∞ and x' were on fire in $LBG(H^*, S')$ after the i^{th} propagation step, and hence x was on fire in $LBG(H, S)$ after the i^{th} propagation step by the inductive hypothesis. Second, suppose x catches fire in $LBG(H^*, S')$ due to an edge of the form $\{x, y, z\}$. Then y and z were on fire in $LBG(H^*, S')$ after the i^{th} propagation step. By the inductive hypothesis, y and z were also on fire in $LBG(H, S)$ after the i^{th} propagation step. Hence, if x was not yet on fire in $LBG(H, S)$, the edge $\{x, y, z\}$ causes fire to spread to x in $LBG(H, S)$ in the $(i+1)^{th}$ propagation step. Third, suppose x catches fire in $LBG(H^*, S')$ due to an edge of the form $\{x, y', z'\}$. Then y' and z' were on fire in $LBG(H^*, S')$ after the i^{th} propagation step. By the inductive hypothesis, y and z were on fire in $LBG(H, S)$ after the i^{th} propagation step. Hence, if x was not yet on fire in $LBG(H, S)$, the edge $\{x, y, z\}$ causes fire to spread to x in $LBG(H, S)$ in the $(i+1)^{th}$ propagation step.

Now, since S' is a lazy burning set for H^*, and by the statement we just proved by induction, S must be a lazy burning set for H. If $\infty \in S'$ then we have $b_L(H) \leq |S| < |S'| = b_L(H^*)$, and otherwise we have $b_L(H) \leq |S| \leq |S'| = b_L(H^*)$. In either case we may conclude $b_L(H) \leq b_L(H^*)$. □

For the rest of this section we write the edge in an STS containing the vertices x, y, and z as xyz. This slight abuse of notation serves the purpose of making an edge easily distinguishable from a lazy burning set of size three.

Both the upper and lower bounds in Theorem 3.3.2 are tight. Indeed, we will show the necessary and sufficient conditions for when each bound is tight in Theorem 3.3.7 and Corollary 3.3.8 (on page 66).

First, we give an example of a triple system H such that $b_L(H) = b_L(H^*)$. Let $H = (V, E)$ be the STS(9), so $V = \{1, \ldots, 9\}$, and a unique way of writing the edges up to isomorphism is $E = \{123, 456, 789, 147, 258, 369, 159, 267, 348, 168, 249, 357\}$. Write the vertex set of H^* as $V(H^*) = \{1, \ldots, 9, 1', \ldots, 9', \infty\}$. Also write $E^\infty = \{11'\infty, 22'\infty, \ldots, 99'\infty\}$ and $E' = \{a'b'c, a'bc', ab'c' \mid abc \in E\}$. Then $E(H^*) = E \cup E^\infty \cup E'$. Clearly $b_L(H) > 2$ and $b_L(H^*) > 2$ since burning two vertices as an attempted lazy burning set would result in just one vertex

catching fire through propagation. Since $\{1, 2, 4\}$ is a lazy burning set for H, $b_L(H) = 3$. Now, it is easy to check that $\{1, 2, 4'\}$ is a lazy burning set for H^*. If we burn this set, during the first propagation the vertices 3, $7'$, and $9'$ will catch fire. During the next propagation, the vertices 8, $5'$, $6'$, and $8'$ will catch fire. Finally, during the last propagation, the vertices 4, 5, 6, 7, 9, $1'$, $2'$, $3'$, and ∞ will catch fire, leaving the entire hypergraph burned. Hence, $b_L(H^*) = 3$. We are currently unaware of any infinite families of STSs H for which this bound is tight (i.e. for which $b_L(H) = b_L(H^*)$), so we leave this as an open problem.

Now, we give an example where the upper bound in Theorem 3.3.2 is tight. Let $H = (V, E)$ be the STS(7), so $V = \{1, \ldots, 7\}$ and a unique way of writing the edges up to isomorphism is $E = \{124, 235, 346, 457, 561, 672, 713\}$. Write $V(H^*) = \{1, \ldots, 7, 1', \ldots, 7', \infty\}$, $E^\infty = \{11'\infty, 22'\infty, \ldots, 77'\infty\}$, and $E' = \{a'b'c, a'bc', ab'c' \mid abc \in E\}$. Then $E(H^*) = E \cup E^\infty \cup E'$. Clearly $b_L(H) = 3$ since there is no lazy burning set of size two and $\{1, 2, 3\}$ is a lazy burning set for H. It is easy to check that $\{1, 2, 3, 1'\}$ is a lazy burning set for H^*, so $b_L(H^*) \leq 4$. We show $b_L(H^*) = 4$ in Theorem 3.3.4, which uses Lemma 3.3.3 in its proof. Indeed, one infinite family of STSs H for which this bound is tight (i.e. for which $b_L(H^*) = b_L(H) + 1$) are the cyclic STSs; see Theorem 3.3.9 on page 66.

Lemma 3.3.3. *Suppose three vertices a, b, and c are burned in the STS(7) as a lazy burning set, where abc is not an edge. Then, during the first propagation, three new distinct vertices d, e, and f will catch fire such that def is an edge in the STS(7).*

Proof. Let $H = (V, E)$ be the STS(7) with V and E written the same as above. Clearly, each unordered pair of vertices among a, b, and c will cause fire to spread to a distinct new vertex. Without loss of generality, denote the vertex that catches fire due to a and b by d, denote the vertex that catches fire due to a and c by e, and denote the vertex that catches fire due to b and c by f. So, $abd, ace, bcf \in E$. Clearly $d, e, f \notin \{a, b, c\}$ and d, e, and f are all distinct. Now, suppose $def \notin E$. Then each unordered pair of vertices among d, e, and f will cause fire to spread to a distinct new vertex during the next propagation. Say d and e spread fire to g, d and f spread to h, and e and f spread to i. So $deg, dfh, efi \in E$. Remark that $g, h, i \notin \{a, \ldots, f\}$. For example, if $g = a$ then both abd and dea are edges in E, which is a contradiction since then a and d appear together in two different edges. We therefore have that $a, \ldots, i$ are nine distinct vertices in the STS(7), which is a contradiction. Our latest supposition must be false, so $def \in E$. □

Theorem 3.3.4. *If H is the STS(7), then $b_L(H^*) = 4$.*

Proof. Let $H = (V, E)$ be the STS(7) with V and E written the same as above. Similarly, let H^* have its vertex set and edge set written the same as above. We will show that if any three

vertices (not all together in an edge of H^*) are burned as an attempted lazy burning set in H^*, then the fire will spread to a sub-STS(7) of H^* and then stop (and hence $b_L(H^*) > 3$). Throughout the proof, if a, b, and c are arbitrary vertices in V such that $abc \notin E$, define d, e, and f as the unique vertices in V such that $abd, ace, bcf \in E$. By the same logic that was used in the proof of Lemma 3.3.3, d, e, and f are distinct, different from a, b, and c, and $def \in E$. We proceed by considering the possible cases with respect to the attempted lazy burning set.

Case 1. The attempted lazy burning set has the form $\{a, b, c\}$ such that $a, b, c \in V$ and $\{a, b, c\} \notin E$. Then clearly the fire will propagate in H^* until all the vertices of V are on fire and then stop (clearly no "primed" vertices or ∞ will catch fire).

Case 2. The attempted lazy burning set has the form $\{a', b', c'\}$ such that $abc \in E$. Then, due to the edges $a'b'c$, $a'bc'$, and $ab'c'$ in E', the vertices a, b, and c will catch fire during the first propagation. Then, due to the edges $aa'\infty$, $bb'\infty$, and $cc'\infty$ in E^∞, the vertex ∞ catches fire during the second propagation. There is no further propagation since currently abc, $a'b'c$, $a'bc'$, $ab'c'$, $aa'\infty$, $bb'\infty$, and $cc'\infty$ are on fire, which forms a sub-STS(7) on the vertices $\{a, b, c, a', b', c', \infty\}$.

Case 3. The attempted lazy burning set has the form $\{a', b', c'\}$ such that $abc \notin E$. Then by Lemma 3.3.3, $abd, ace, bcf, def \in E$ for distinct vertices $d, e, f \in V \setminus \{a, b, c\}$. So, during the first propagation, d, e, and f catch fire due to the edges $a'b'd$, $a'c'e$, and $b'c'f$ respectively. Currently the edges $a'b'd$, $a'c'e$, $b'c'f$, and def are on fire in H^*. The second propagation will be due to the pairs of vertices d and c', e and b', and f and a'. Define g as the unique vertex in V such that $cdg \in E$. First, it is easy to show that $g \notin \{a, b, c, d, e, f\}$. If $g = a$ then $cda, ace \in E$, and hence $d = e$ which contradicts Lemma 3.3.3. Similar contradictions can be reached by assuming g is equal to any of the vertices b, c, d, e, or f. Now, clearly during the second propagation, the vertices d and c' cause fire to spread to g' via the edge $c'dg' \in E'$. We claim that both of the other two the pairs (e and b', and f and a') cause fire to spread to g'. First, consider the pair e and b'. Clearly they will cause a vertex of the form x' to catch fire via the edge $eb'x' \in E'$. We will show that $x' = g'$. First, suppose $x' = a'$. Then $eb'a' \in E'$, so $eba \in E$. But $ace \in E$, so $b = c$, which is a contradiction. If $x' = c'$ then $eb'c' \in E'$, so $ebc \in E$. But $bcf \in E$ so $e = f$ which is a contradiction. Similar contradictions can be reached when we assume either $x' = d'$ or $x' = f'$, and both $x' = b'$ and $x' = e'$ are obviously false. Hence, $x' = g'$. A similar argument can be used to show that f and a' cause fire to spread to g' during the second propagation. Hence, after the second propagation the edges $a'b'd$, $a'c'e$, $b'c'f$, def, $g'dc'$, $g'eb'$, and $g'fa'$ are on fire. No more fire will spread, since this forms a sub-STS(7) on the vertices $\{a', b', c', d, e, f, g'\}$.

Case 4. The attempted lazy burning set has the form $\{a', b', c\}$. Of course, we also assume that $\{a', b', c\}$ is not an edge in H^*. If $abc \in E$ then $a'b'c \in E'$, which is a contradiction since we assume a', b', and c do not belong together in an edge of H^*. Thus, $abc \notin E$. Now, in the first propagation, d, e', and f' will catch on fire due to the edges $a'b'd$, $a'ce'$, and $b'cf'$. Then, since $def \in E$ (by Lemma 3.3.3), we know $de'f' \in E'$. From here, the same argument as in case 3 shows that the fire will propagate to a sub-STS(7) on the vertices $\{a', b', c, d, e', f', g\}$ and then stop.

Case 5. The attempted lazy burning set has the form $\{a', b, c\}$ such that $abc \in E$. In the first propagation, the vertices c', b', and a will catch fire due to the edges $a'bc'$, $a'b'c$, and abc respectively. Note that since $abc \in E$, we know $ab'c' \in E'$. From here, the same argument as in case 2 shows that the fire will propagate to a sub-STS(7) on the vertices $\{a, b, c, a', b', c', \infty\}$ and then stop.

Case 6. The attempted lazy burning set has the form $\{a', b, c\}$ such that $abc \notin E$. In the first propagation, the vertices d', e', and f will catch fire due to the edges $a'bd'$, $a'ce'$, and bcf respectively. Note that since $def \in E$ (by Lemma 3.3.3), we know $d'e'f \in E'$. From here, the same argument as in case 3 shows that the fire will propagate to a sub-STS(7) on the vertices $\{a', b, c, d', e', f, g'\}$ and then stop.

Case 7. The attempted lazy burning set has the form $\{a', b', \infty\}$. Define d as the unique vertex such that $abd \in E$. Then, a similar argument to case 2 shows that the fire will propagate to a sub-STS(7) on the vertices $\{a, b, d, a', b', d', \infty\}$ and then stop.

Case 8. The attempted lazy burning set has the form $\{a', b, \infty\}$. Define d as the unique vertex such that $abd \in E$. In the first propagation the vertices a, b', and d' will catch fire due to the edges $aa'\infty$, $bb'\infty$, and $a'bd'$ respectively. Note that since $abd \in E$, we know $ab'd' \in E'$. From here, a similar argument to case 2 shows that the fire will propagate to a sub-STS(7) on the vertices $\{a, b, d, a', b', d', \infty\}$ and then stop.

Case 9. The attempted lazy burning set has the form $\{a, b, \infty\}$. Define d as the unique vertex such that $abd \in E$. In the first propagation the vertices a', b', and d will catch fire due to the edges $aa'\infty$, $bb'\infty$, and abd respectively. Note that since $abd \in E$, we know $a'b'd \in E'$. From here, a similar argument to case 2 shows that the fire will propagate to a sub-STS(7) on the vertices $\{a, b, d, a', b', d', \infty\}$ and then stop.

We therefore have that $b_L(H^*) > 3 = b_L(H)$ where H is the STS(7). Using Theorem 3.3.2, we can conclude that $b_L(H^*) = 4$. □

A much quicker proof of the result in Theorem 3.3.4 can be found in Corollary 3.3.10 on

page 66. Next, we obtain necessary and sufficient conditions for when $b_L(H^*) = b_L(H) + 1$ for an arbitrary STS H. This result can be seen in Theorem 3.3.7.

Lemma 3.3.5. *Let H be a Steiner triple system and let $T, S \subseteq V(H^*)$. If $\infty, x, x' \in T$ for some $x \in V(H)$ then T is not an optimal lazy burning set for H^*. If $x, x', y, y' \in S$ for some $x, y \in V(H)$ then S is not an optimal lazy burning set for H^*.*

Proof. Suppose T is an optimal lazy burning set for H^* with $\infty, x, x' \in T$ for some $x \in V(H)$. Since $\{x, \infty, x'\}$ is an edge in H^*, $T \setminus \{x\}$ is a smaller lazy burning set for H^*, which is a contradiction.

Suppose S is an optimal lazy burning set for H^* with $x, x', y, y' \in S$ for some $x, y \in V(H)$. We claim $S \setminus \{y'\}$ is a lazy burning set for H^*, which would give us the desired contradiction. We only need to prove that y' catches fire through propagation in $LBG(H^*, S \setminus \{y'\})$. Consider the unique vertex $z \in V(H)$ such that $\{x, y, z\}$ is an edge in H. Clearly $z' \notin S$ since otherwise $S \setminus \{z'\}$ would be a smaller lazy burning set for H^* (the edge $\{x', y, z'\}$ would cause z' to catch fire). Hence, the edge $\{x', y, z'\}$ causes fire to spread to z' in $LBG(H^*, S \setminus \{y'\})$. Then, the edge $\{x, y', z'\}$ causes fire to spread to y' in $LBG(H^*, S \setminus \{y'\})$. □

Lemma 3.3.6. *Let H be a Steiner triple system. There is an optimal lazy burning set T for H^* with $\infty \in T$ if and only if there is an optimal lazy burning set S for H^* with $x, x' \in S$ for some $x \in V(H)$.*

Proof. Let T be an optimal lazy burning set for H^* with $\infty \in T$. Then by Lemma 3.3.5 there is no vertex $x \in V(H)$ such that $x, x' \in T$. There are three cases. First, suppose there is a vertex $y \in V(H)$ such that $y \in T$. Then $y' \notin T$ and $(T \setminus \{\infty\}) \cup \{y'\}$ is an optimal lazy burning set for H^*. Clearly this set is the same size at T, and it contains both y and y'. It is indeed a lazy burning set since the edge $\{y, \infty, y'\}$ causes ∞ to catch fire in $LBG(H^*, (T \setminus \{\infty\}) \cup \{y'\})$. Second, suppose there is a vertex $z \in V(H)$ such that $z' \in T$. Then $z \notin T$, and by a similar argument $(T \setminus \{\infty\}) \cup \{z\}$ is an optimal lazy burning set for H^*. Third, suppose neither of the previous two cases hold. Then $T = \{\infty\}$, which is a contradiction, so at least one of the previous two cases holds.

Let S be an optimal lazy burning set for H^* with $x, x' \in S$ for some $x \in V(H)$. By Lemma 3.3.5, $\infty \notin S$. Then, $(S \setminus \{x\}) \cup \{\infty\}$ is an optimal lazy burning set for H^*, since this set is the same size as S, and the edge $\{x, \infty, x'\}$ causes x to catch fire in $LBG(H^*, (S \setminus \{x\}) \cup \{\infty\})$. □

Theorem 3.3.7. *Let H be a Steiner triple system. Then, $b_L(H^*) = b_L(H) + 1$ if and only if there is an optimal lazy burning set S' for H^* such that $\infty \in S'$.*

Proof. For the reverse implication, construct S from S' in the same way as in the proof of Theorem 3.3.2. Then, S is a lazy burning set for H with $|S| < |S'|$, so $b_L(H) \leq |S| < |S'| = b_L(H^*)$. By Theorem 3.3.2, since $b_L(H) < b_L(H^*)$ we must have $b_L(H^*) = b_L(H) + 1$.

For the forward implication, assume $b_L(H^*) = b_L(H) + 1$. Let T be an optimal lazy burning set for H. We claim $T \cup \{\infty\}$ is an optimal lazy burning set for H^*. It is the correct size since $|T \cup \{\infty\}| = |T| + 1 = b_L(H) + 1 = b_L(H^*)$. It is also a lazy burning set, since each vertex $x \in V(H)$ will burn due to T in $LBG(H^*, T \cup \{\infty\})$ (recall $E(H) \subseteq E(H^*)$), and each vertex x' will burn due to the edge $\{x, \infty, x'\}$ in $LBG(H^*, T \cup \{\infty\})$. □

Corollary 3.3.8. *Let H be a Steiner triple system. Then, $b_L(H) = b_L(H^*)$ if and only if there is* no *optimal lazy burning set S' for H^* such that $\infty \in S'$.*

The conditions in Theorem 3.3.7 and Corollary 3.3.8 are not in terms of any design-theoretic properties of H or H^*. Instead, they require that we know something about the existence of an optimal lazy burning set in H^* in order to determine whether or not $b_L(H^*) = b_L(H) + 1$. We therefore pose the following open question: are there necessary and sufficient conditions for determining whether $b_L(H^*) = b_L(H) + 1$ that are only in terms of design-theoretic properties of H and/or H^*? Indeed, we have found one such sufficient condition.

Theorem 3.3.9. *If H^* is cyclic then $b_L(H^*) = b_L(H) + 1$.*

Proof. Write $|V(H^*)| = v$ and let α be an automorphism of H^* that is a permutation consisting of a single cycle of length v. Write $b_L(H^*) = b$ and let $\{u_1, u_2, \ldots, u_b\}$ be a minimum lazy burning set for H^*. Recall that each vertex of H^* belongs to one of the following sets: $\{x \mid x \in V(H)\}$, $\{x' \mid x \in V(H)\}$, or $\{\infty\}$. If one of the u_i is ∞ then we get the desired result by Theorem 3.3.7, so assume $u_i \notin \{\infty\}$ for all i. Now, since H^* is isomorphic to $\alpha^k(H^*)$ for any $k \in \mathbb{N}$, we have that $\{\alpha^k(u_1), \alpha^k(u_2), \ldots, \alpha^k(u_b)\}$ is a minimum lazy burning set in both $\alpha^k(H^*)$ and H^* for any $k \in \mathbb{N}$. Now, choose $k \in \mathbb{N}$ such that $\alpha^k(u_1) = \infty$. Then $\{\alpha^k(u_1), \alpha^k(u_2), \ldots, \alpha^k(u_b)\}$ is a minimum lazy burning set for H^* containing ∞, so the result follows by Theorem 3.3.7. □

Next, we use Theorem 3.3.9 to prove the result from Theorem 3.3.4 much more easily.

Corollary 3.3.10. *If H is the STS(7), then $b_L(H^*) = 4$.*

Proof. Observe that for each edge $\{x, y, z\}$ in an STS H, there is a sub-STS(7) on the vertices $\{x, y, z, x', y', z', \infty\}$ in H^*.

Let H be the STS(7). Then H^* is an STS(15) that has at least eight sub-STS(7)s; H itself, and one for each of the seven edges in H. As seen in [32], the only STS(15) with eight or more nontrivial subsystems is the one labelled #1, so this STS(15) must be H^*. The STS(15) #1 is cyclic [32], so by Theorem 3.3.9 we must have $b_L(H^*) = b_L(H) + 1 = 3 + 1 = 4$. □

We now turn our attention to doubling the STS(7) specifically. For the purposes of the following definition, first define the operation $\oplus$ on $\mathbb{Z}_2^n$ such that, if $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, then $x \oplus y = (x_1 + y_1, \ldots, x_n + y_n)$ where each $x_i + y_i$ is reduced modulo 2. Clearly $\oplus$ is commutative and associative.

Definition 3.3.11. ([19]) *A* projective triple system of order n *is a* 3*-uniform hypergraph* H *on* $2^n - 1$ *vertices with the following properties.*

- *Each vertex in* $V(H)$ *can be labelled with an element of* $\mathbb{Z}_2^n \setminus \{0\}$ *(i.e. an* n*-tuple of zeros and ones that is not all zeros).*
- *For all* $x, y, z \in V(H)$, xyz *is an edge in* H *if and only if* $x \oplus y \oplus z$ *is the zero vector in* $\mathbb{Z}_2^n$.

Theorem 3.3.12. ([19]) *For each* $n \in \mathbb{N} \setminus \{1, 2\}$ *there exists exactly one projective triple system of order* n*, and it is an STS*$(2^n - 1)$.

It is well-known that applying the doubling construction discussed in this section to a projective triple system of order n yields the projective triple system of order $n + 1$. We present a proof of this fact for completeness.

Lemma 3.3.13. *If* H *is the projective triple system of order* n *then* H^* *is the projective triple system of order* $n + 1$.

Proof. Write $V(H) = \{x_1, \ldots, x_{2^n-1}\} = \mathbb{Z}_2^n \setminus \{0\}$. We first need to determine how vertices of the form x_i, x_i', and ∞ are defined in H^* as elements of $\mathbb{Z}_2^{n+1} \setminus \{0\}$. For each $x_i = (x_i^{(1)}, \ldots, x_i^{(n)}) \in V(H)$, define the vertices x_i and x_i' in $V(H^*)$ as $x_i = (x_i^{(1)}, \ldots, x_i^{(n)}, 0)$ and $x_i' = (x_i^{(1)}, \ldots, x_i^{(n)}, 1)$. Also define $\infty = (0^{(1)}, \ldots, 0^{(n)}, 1)$. Then clearly $x_i, x_i', \infty \in \mathbb{Z}_2^{n+1} \setminus \{0\}$ (where this x_i is the one in H^*). Then let $V(H^*) = \{x_i, x_i' \mid x_i \in V(H)\} \cup \{\infty\}$. Notice that if $i \neq j$ then x_i, x_j, x_i', x_j', and ∞ are all distinct vertices in H^*. Hence, $|V(H^*)| = 2|V(H)| + 1 = 2(2^n - 1) + 1 = 2^{n+1} - 1 = |\mathbb{Z}_2^{n+1} \setminus \{0\}|$, so $V(H^*) = \mathbb{Z}_2^{n+1} \setminus \{0\}$ as desired. Now we must show that the rules for how the edges of H^* are constructed according to Definition 3.3.11 coincide with those from Construction 2.15 from [18].

We will show $E(H) \subseteq E(H^*)$. If $x_ix_jx_k \in E(H)$ then $x_i \oplus x_j \oplus x_k = 0$ in $\mathbb{Z}_2^n$. Since the $(n+1)^{th}$ coordinate of these vertices is zero (as elements of $V(H^*)$), we have that $x_i \oplus x_j \oplus x_k = 0$ in $\mathbb{Z}_2^{n+1}$, so $x_ix_jx_k \in E(H^*)$.

Now we show that for each edge $x_ix_jx_k \in E(H)$, the triple $x_i'x_j'x_k$ is an edge in H^* (the proofs for $x_i'x_jx_k'$ and $x_ix_j'x_k'$ are similar). Calculating in $\mathbb{Z}_2^{n+1}$, we have

$$x_i' \oplus x_j' \oplus x_k = (x_i^{(1)} + x_j^{(1)} + x_k^{(1)}, \ldots, x_i^{(n)} + x_j^{(n)} + x_k^{(n)}, 1+1+0) = (0, \ldots, 0, 0) = 0,$$

so $x_i'x_j'x_k \in E(H^*)$. Also notice that $x_i'x_j'x_k', x_i'x_jx_k \notin E(H^*)$ since

$$x_i' \oplus x_j' \oplus x_k' = (x_i^{(1)} + x_j^{(1)} + x_k^{(1)}, \ldots, x_i^{(n)} + x_j^{(n)} + x_k^{(n)}, 1+1+1) = (0, \ldots, 0, 1) \neq 0, \text{ and}$$

$$x_i' \oplus x_j \oplus x_k = (x_i^{(1)} + x_j^{(1)} + x_k^{(1)}, \ldots, x_i^{(n)} + x_j^{(n)} + x_k^{(n)}, 1+0+0) = (0, \ldots, 0, 1) \neq 0.$$

Similarly, $x_ix_j'x_k, x_ix_jx_k' \notin E(H^*)$.

For each $x_i \in V(H)$ we have $x_ix_i'\infty \in E(H^*)$. Calculating in $\mathbb{Z}_2^{n+1}$, we have

$$x_i \oplus x_i' \oplus \infty = (x_i^{(1)} + x_i^{(1)} + 0, \ldots, x_i^{(n)} + x_i^{(n)} + 0, 0+1+1) = (2x_i^{(1)}, \ldots, 2x_i^{(n)}, 2) = (0, \ldots, 0, 0).$$

Notice that ∞ does not appear in any edges of H^* except those of the form $x_ix_i'\infty$. If $i \neq j$ then $x_i \oplus x_j \neq 0$ since x_i and x_j differ in at least one of the first n coordinates (i.e. one of them has a 1 and the other has a 0 in a common coordinate). Since $\infty = (0^{(1)}, \ldots, 0^{(n)}, 1)$, there will be a 1 somewhere in the first n coordinates of $x_i \oplus x_j \oplus \infty$, so $x_ix_j\infty \notin E(H^*)$ whenever $i \neq j$. A similar argument shows that none of $x_i'x_j\infty$, $x_ix_j'\infty$, or $x_i'x_j'\infty$ are edges in H^* when $i \neq j$. □

Corollary 3.3.14. *The projective triple systems are exactly those hypergraphs of the form H^{n*}, $n \in \mathbb{N} \cup \{0\}$ where H is the STS(7).*

The following theorem is stated as a passing remark in [19] (Section 7.2.1, pp. 102-103), and can also be found in [37] (see the theorem on page 379).

Theorem 3.3.15. *All projective triple systems are cyclic.*

By combining Theorem 3.3.9 (on page 66), Corollary 3.3.14, and Theorem 3.3.15 we get the following result. It essentially says that, if we repeatedly double the STS(7), the lazy burning number increases by one each time. Hence, there exist Steiner triple systems with arbitrary lazy burning number; see Corollary 3.3.18.

Theorem 3.3.16. *Let H be the STS(7). Then $b_L(H^{n*})+1 = b_L(H^{(n+1)*})$ for all $n \in \mathbb{N} \cup \{0\}$.*

Corollary 3.3.17. *If H is the projective triple system of order n then $b_L(H) = n$.*

Corollary 3.3.18. *For all $n \in \mathbb{N} \setminus \{1, 2\}$ there exists a Steiner triple system with lazy burning number n.*

Is it true that if an STS H is cyclic then H^* is also cyclic? If yes, then this would yield *infinitely many* hypergraphs H with the same property as in Theorem 3.3.16. That is, by repeatedly doubling H, the lazy burning number increases by one each time since each double is cyclic. It would also be nice to determine if there is a hypergraph G with the "opposite" property to the one in Theorem 3.3.16. That is, does there exist a hypergraph G with $b_L(G^{n*}) = b_L(G^{(n+1)*})$ for all $n \in \mathbb{N} \cup \{0\}$? Are there in fact infinitely many such hypergraphs G? We leave these as open questions.

3.4 Computational Results

To accompany our theoretic results on Steiner triple systems, a C program was written that computes the burning and lazy burning numbers of a hypergraph; see Appendices A.1 and A.2. It uses the standard propagation rule featured throughout the vast majority of this thesis, and considers only those burning sequences in which the arsonist avoids redundant sources until possibly the final round. It was tested on several STSs of interest, and we summarize the results in this section.

First, the (lazy) burning numbers for the four smallest nontrivial STSs were found; see Table 3.2. Since these STSs are quite small, it was also possible to obtain the number of optimal burning sequences and minimum lazy burning sets. Note that optimal burning sequences that differ only in the final redundant source are counted separately. Another notable result is that there are no valid burning sequences of length strictly greater than 4 for the STS(7), and no valid burning sequences of length greater than 5 for the other three STSs.

All 80 STS(15)s were tested as well, and the results are summarized here in paragraph format for the sake of brevity. Each of them has burning number 5 and exactly 141120 optimal burning sequences. There are no valid burning sequences of length greater than 5 for any STS(15).

The STS(15) labelled number 1 in [32] (and commonly elsewhere in the literature) is the only STS(15) with lazy burning number 4, and it has 840 minimum lazy burning sets. All

H	$b(H)$	# of optimal burning sequences	$b_L(H)$	# of minimum lazy burning sets
STS(7)	4	504	3	28
STS(9)	5	864	3	72
cyclic STS(13)	5	46800	3	260
non-cyclic STS(13)	5	46800	3	260

Table 3.2: Burning numbers and lazy burning numbers for some small Steiner Triple Systems.

the other STS(15)s have lazy burning number 3, and 224, 336, 392, or 420 minimum lazy burning sets. Only number 2 has 224 minimum lazy burning sets. Numbers 3 to 7 have 336 minimum lazy burning sets. Numbers 8 to 22 and 61 have 392 minimum lazy burning sets. The rest have 420 minimum lazy burning sets.

The unique characteristics of STS(15) number 1 may provide insight into what influences the lazy burning number of a hypergraph. Most notably, STS(15) number 1 has a much higher automorphism group order (20160), and has many more subsystems (15), parallel classes (56), and resolutions (240) than the other STS(15)s [32].

The lazy burning numbers of the first few doubles of the four smallest nontrivial STSs were also found; see Table 3.3. Of course, the lazy burning number of any double of the STS(7) is known (see Theorem 3.3.16), but the first row of the table was still obtained computationally. The lazy burning number of the fourth double of the STS(7) is absent from the table since the program ran for too long. This may seem odd considering the STS(7) is the smallest nontrivial STS. However, its fourth double has lazy burning number seven (which we know from Theorem 3.3.16), and in general the program takes longer to test hypergraphs with larger lazy burning numbers.

H	$b_L(H^*)$	$b_L(H^{2*})$	$b_L(H^{3*})$	$b_L(H^{4*})$
STS(7)	4	5	6	
STS(9)	3	3	3	4
cyclic STS(13)	3	3	3	4
non-cyclic STS(13)	3	3	3	4

Table 3.3: Lazy burning numbers for the first few doubles of some small STSs.

The lazy burning numbers of the first two doubles of each STS(15) were also found computationally. STS(15) number 1 had $b_L(H^*) = 5$ and $b_L(H^{2*}) = 6$, which coincides with Theorem 3.3.16. STS(15) number 2 had $b_L(H^*) = 4$ and $b_L(H^{2*}) = 5$. The STS(15)s numbered 3 to 7 all had $b_L(H^*) = 3$ and $b_L(H^{2*}) = 4$. Finally, the STS(15)s numbered 8 to 80 all had $b_L(H^*) = b_L(H^{2*}) = 3$.

A *projective plane* of order n is a BIBD$(n^2+n+1, n+1, 1)$. Indeed, for each $n \in \mathbb{N} \setminus \{1\}$, there exists a unique projective plane of order n [38], which we will denote by $PG(2, n)$. Table 3.4 shows the (lazy) burning numbers of some projective planes of small order.

H	$b_L(H)$	$b(H)$
$PG(2,2)$	3	4
$PG(2,3)$	6	8
$PG(2,4)$	11	12
$PG(2,5)$	16	18

Table 3.4: Burning numbers and lazy burning numbers for some small projective planes.

Chapter 4

Alternative Propagation Rules

The original analogy for graph burning was the spread of a social contagion through social media [13]. Each vertex of the graph represents a person, two people who are friends are connected by an edge, and the fire represents some idea, opinion, or piece of knowledge. Hypergraph burning has the potential to be an even more comprehensive model for the spread of information, as information does not only spread between close friends. Hyperedges may represent any large group of people who may not all have direct communication with one another, but nonetheless spread information to other members of the group. In particular, hyperedges can represent friend groups on social media.

In order to make the added complication of hyperedges worthwhile, the rule for how the fire spreads within a hyperedge should represent the spread of information among members of a large group as "realistically" as possible. The original propagation rule may not always be the most realistic in this sense. Hence, we introduce two new propagation rules that are more conducive to the way information spreads within groups in the real world.

4.1 A Propagation Rule that Uses Proportions

Consider the following propagation rule: For some fixed *proportion* $p \in (0, 1)$, if $\lceil p|e| \rceil$ or more vertices are on fire in an edge e, then in the next round all unburned vertices in e catch fire. Instead of writing $b(H)$ and $b_L(H)$, when using this propagation rule we will write $b_p(H)$ and $b_{L,p}(H)$ respectively. For the entirety of Section 4.1 we will assume that we are using this propagation rule, unless it is otherwise stated. Recall that this is indeed a different process from the degree-proportional bootstrap percolation process discussed in Section 1.5.1.

Notice that under this propagation rule, if the proportion p is high enough, there may be edges in the hypergraph that cannot possibly cause fire to spread. These edges are the subject of our first few results on this propagation rule.

Remark 4.1.1. *Let $p \in (0,1)$, H be a hypergraph, and $e \in E(H)$. If $\lceil p|e| \rceil = |e|$ then e cannot cause fire to propagate among its vertices.*

Definition 4.1.2. *Let $p \in (0,1)$, H be a hypergraph, and $e \in E(H)$. If $\lceil p|e| \rceil = |e|$ then e is* non-flammable. *Otherwise, e is* flammable.

Lemma 4.1.3. *An edge e is non-flammable if and only if $|e| < \frac{1}{1-p}$.*

Proof. Observe that $\lceil p|e| \rceil = |e|$ if and only if $|e| - p|e| < 1$. Keeping in mind that $1 - p$ is positive, we can simplify to obtain $|e| < \frac{1}{1-p}$. □

Lemma 4.1.4. *Let $p \in (0,1)$, H be a hypergraph, and $e_1, e_2, \ldots, e_k$ be all of the non-flammable edges in H (for this value of p). Let H' be the hypergraph with vertex set $V(H)$ and edge set $E(H) \setminus \{e_1, e_2, \ldots, e_k\}$. Then $b_p(H) = b_p(H')$ and $b_{L,p}(H) = b_{L,p}(H')$.*

Proof. Non-flammable edges have no effect on the burning game. Thus, in the context of the burning game, H and H' behave identically. □

Remark 4.1.5. *Let $p \in (0,1)$. If every edge in H is non-flammable (for this value of p), then $b_{L,p}(H) = b_p(H) = |V(H)|$.*

Of course, in a finite hypergraph we can always choose p close enough to 1 such that every edge is non-flammable.

Lemma 4.1.6. *Let $p \in (0, \frac{1}{2}]$ and H be a hypergraph. Then the only non-flammable edges in H are singleton edges.*

Proof. Since $p \leq \frac{1}{2}$ we can calculate $\frac{1}{1-p} \leq 2$. Now, for this value of p, an edge e is non-flammable if and only if $|e| < \frac{1}{1-p} \leq 2$, that is, if $|e| < 2$. Hence, all non-flammable edges e must have $|e| = 1$. □

Lemma 4.1.7. *Let H be a connected hypergraph. Then there exists $p \in (0,1)$ such that $b_{L,p}(H) = 1$.*

Proof. Choose p small enough such that $p|e| < 1$ for all $e \in E(H)$. Then for any edge e, $\lceil p|e| \rceil = 1$, so one vertex on fire in any edge will cause the rest of the edge to catch fire. Since H is connected, a single vertex on fire in H will cause the whole hypergraph to burn through subsequent propagation. So $b_{L,p}(H) = 1$. □

Theorem 4.1.8. *Let H be a connected hypergraph and $p \in (0,1)$. Then*

$$\min\{\lceil p|e|\rceil \mid e \in E(H)\} \leq b_{L,p}(H).$$

Proof. Suppose a lazy burning set for H contains strictly less than $\min\{\lceil p|e|\rceil \mid e \in E(H)\}$ vertices. Then, for every edge e in H, strictly less than $\lceil p|e|\rceil$ vertices in e are on fire once the lazy burning set is burned. Hence, no fire will propagate, so in fact this is not a lazy burning set for H. Therefore, any lazy burning set (including a minimum lazy burning set) must contain at least $\min\{\lceil p|e|\rceil \mid e \in E(H)\}$ vertices. □

The bound in Theorem 4.1.8 is tight for all $p \in (0,1)$ – simply consider a hypergraph consisting of a single edge that contains all of its vertices. A less trivial example that exhibits tightness when $p = \frac{1}{2}$ can be seen in Figure 4.1 on page 76.

Our next major result is a tight upper bound on the lazy burning number of a hypergraph when using a proportion of the form $p = \frac{1}{n}$, see Theorem 4.1.10. The following result is used in its proof.

Lemma 4.1.9. *If a and b are real numbers, then $\lceil a\rceil + \lfloor b\rfloor \leq \lceil a+b\rceil$.*

Proof. Fix $a \in \mathbb{R}$, and first consider the case where b is an integer. Then $\lceil a\rceil + \lfloor b\rfloor = \lceil a\rceil + b = \lceil a+b\rceil$. Now, assume b is not an integer, and write $b' = \lfloor b\rfloor$. Since $b' \leq b$, we have $\lceil a+b'\rceil \leq \lceil a+b\rceil$. Now, $\lceil a\rceil + \lfloor b\rfloor = \lceil a\rceil + b' = \lceil a+b'\rceil \leq \lceil a+b\rceil$. □

Theorem 4.1.10. *Let H be a connected hypergraph and $n \in \mathbb{N}$. Then $b_{L,1/n}(H) \leq \left\lceil \frac{|V(H)|}{n} \right\rceil$.*

Proof. We use induction on $|E(H)|$, assuming singleton edges are not allowed. The bound in this theorem will still be valid for hypergraphs containing singleton edges, as they have no effect on the burning game.

Base Case. If $|E(H)| = 1$ then H consists of a single edge e containing all of $V(H)$. Thus, $b_{L,1/n}(H) = \left\lceil \frac{|e|}{n} \right\rceil = \left\lceil \frac{|V(H)|}{n} \right\rceil$.

Inductive Hypothesis. Suppose that for some $k \in \mathbb{N}$, every connected hypergraph H with $|E(H)| = k$ satisfies the bound $b_{L,1/n}(H) \leq \left\lceil \frac{|V(H)|}{n} \right\rceil$.

Inductive Step. Let G be a connected hypergraph with $k+1$ edges. Observe that, since G is connected, there must exist some edge $e \in E(G)$ such that $(V(G), E(G)\setminus\{e\})$ is connected, except for possibly some isolated vertices (degree-one vertices in e would become isolated vertices in $(V(G), E(G)\setminus\{e\})$). Choose any such $e \in E(G)$, and write $e = \{u_1, \ldots, u_x\} \cup \{v_1, \ldots, v_y\}$ where each u_i is a vertex of degree greater than one in e, and each v_j is a

degree-one vertex in e. Since G is connected, e must contain at least one vertex of degree greater than one, so $x \geq 1$. Also, clearly $y \geq 0$. Now, define H as the hypergraph with $V(H) = V(G) \setminus \{v_1, \ldots, v_y\}$ and $E(H) = E(G) \setminus \{e\}$. Then H is a connected hypergraph with k edges, so $b_{L,1/n}(H) \leq \left\lceil \frac{|V(H)|}{n} \right\rceil = \left\lceil \frac{|V(G)|-y}{n} \right\rceil$. Let S be a minimum lazy burning set in H, so $|S| \leq \left\lceil \frac{|V(G)|-y}{n} \right\rceil$.

Now, we claim $S' = S \cup \{v_1, \ldots, v_{\lfloor y/n \rfloor}\}$ is a lazy burning set for G. All of $V(G) \setminus \{v_1, \ldots, v_y\}$ will catch fire because $S \subseteq S'$. In particular, $u_1, \ldots, u_x, v_1, \ldots,$ and $v_{\lfloor y/n \rfloor}$ will all eventually be on fire either due to propagation or because they were included in S'. We now show that the rest of e will catch fire. In particular, we must show that $x + \left\lfloor \frac{y}{n} \right\rfloor \geq \left\lceil \frac{|e|}{n} \right\rceil$, since $x + \left\lfloor \frac{y}{n} \right\rfloor$ vertices are sure to be on fire in e. In our calculations, we will use the well-known identities $\left\lfloor \frac{a}{b} \right\rfloor = \left\lceil \frac{a+1}{b} \right\rceil - 1$ and $\lceil c \rceil + a = \lceil c + a \rceil$ for any $a, b \in \mathbb{N}$ and $c \in \mathbb{R}$.

First, observe $1 \leq x \implies (n-1) \leq x(n-1) \implies 0 \leq x(n-1)+1-n \implies 0 \leq \frac{(n-1)x+1-n}{n}$, which will be used in the final line of the calculations below. Now,

$$\begin{aligned}
x + \left\lfloor \frac{y}{n} \right\rfloor &= x + \left\lceil \frac{y+1}{n} \right\rceil - 1 \\
&= \left\lceil x + \frac{y+1}{n} - 1 \right\rceil \\
&= \left\lceil \frac{nx + y + 1 - n}{n} \right\rceil \\
&= \left\lceil \frac{x+y}{n} + \frac{(n-1)x + 1 - n}{n} \right\rceil \\
&= \left\lceil \frac{|e|}{n} + \frac{(n-1)x + 1 - n}{n} \right\rceil \\
&\geq \left\lceil \frac{|e|}{n} \right\rceil.
\end{aligned}$$

Therefore e will cause fire to spread to each of the vertices $v_{\lfloor y/n \rfloor + 1}, \ldots, v_y$. This will leave G completely burned, so S' is a lazy burning set for G. Now, using Lemma 4.1.9, we calculate $b_{L,1/n}(G) \leq |S'| = |S| + \left\lfloor \frac{y}{n} \right\rfloor \leq \left\lceil \frac{|V(G)|-y}{n} \right\rceil + \left\lfloor \frac{y}{n} \right\rfloor \leq \left\lceil \frac{|V(G)|}{n} \right\rceil$. □

The bounds in Theorems 4.1.8 and 4.1.10 can both be tight simultaneously when $p = \frac{1}{n}$ for any $n \in \mathbb{N}$ – consider a connected hypergraph consisting of one edge that contains all of its vertices. For additional examples with $p = \frac{1}{2}$, see Figures 4.1 and 4.2. Are there any nontrivial examples where the bounds in Theorem 4.1.8 (with $p = \frac{1}{n}$) and Theorem 4.1.10 are tight for $n > 2$? We leave this as an open question.

The hypergraph H_1 in Figure 4.1 has $b_{L,1/2}(H_1) = 3 = \min\left\{\left\lceil \frac{|e|}{2} \right\rceil \mid e \in E(H_1)\right\}$. A minimum lazy burning set is $\{x, y, z\}$. Hence, the bound in Theorem 4.1.8 is tight when $p = \frac{1}{2}$. Indeed, this example can be extended to an infinite family of hypergraphs which exhibit the tightness of this bound when $p = \frac{1}{2}$. Fix $t \in \mathbb{N}$, let e_1 contain $t+1$ vertices, and let the remaining edges $e_2, e_3, \ldots, e_m$ all contain $2t$ vertices each. Let one vertex in e_1 have degree one, and let the other t vertices also belong to e_2. Let the t vertices in $e_2 \setminus e_1$ also belong to e_3. Let the t vertices in $e_3 \setminus e_2$ also belong to e_4, and so on. Finally, $e_m \cap e_{m-1}$ will contain t vertices, and e_m will contain t vertices of degree one. In such a hypergraph, a minimum lazy burning set consists of any $\left\lceil \frac{|e_1|}{2} \right\rceil$ vertices in e_1. This shows the tightness of the bound in Theorem 4.1.8 (when $p = \frac{1}{2}$), since e_1 is the smallest edge in the hypergraph.

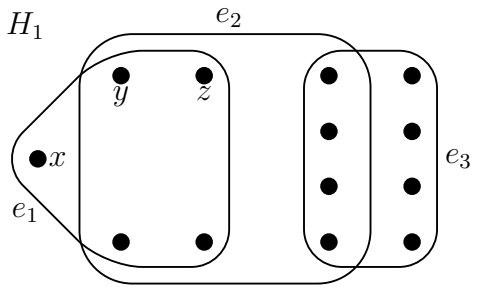

Figure 4.1: An example that shows the bound in Theorem 4.1.8 is tight when $p = \frac{1}{2}$.

The hypergraph H_2 in Figure 4.2 has $b_{L,1/2}(H_2) = 5 = \left\lceil \frac{|V(H_2)|}{2} \right\rceil$, and a minimum lazy burning set is $\{p, q, r, s, t\}$. Hence, the bound in Theorem 4.1.10 is tight. Again, this example can be extended to an infinite family of hypergraphs which exhibit tightness when $p = \frac{1}{2}$. To construct such a hypergraph H, fix an odd number $t > 1$, and let $|V(H)| = 2t$. Let $E(H)$ contain disjoint edges e_1 and e_3 each of size t, and add an edge e_2 which intersects e_1 and e_3 in exactly one vertex each. Then, a minimum lazy burning set for H consists of any $\left\lceil \frac{t}{2} \right\rceil$ vertices in e_1, and any $\left\lfloor \frac{t}{2} \right\rfloor$ degree-one vertices in e_3, for a total of $\left\lceil \frac{t}{2} \right\rceil + \left\lfloor \frac{t}{2} \right\rfloor = t = \frac{|V(H)|}{2} = \left\lceil \frac{|V(H)|}{2} \right\rceil$ vertices. This shows the tightness of the bound in Theorem 4.1.10 when $p = \frac{1}{2}$.

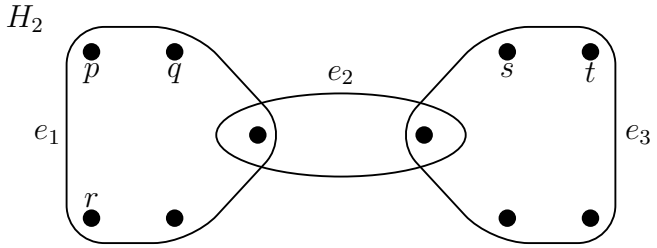

Figure 4.2: An example that shows the bound in Theorem 4.1.10 is tight when $p = \frac{1}{2}$.

Of course, it is desirable to extend the result in Theorem 4.1.10 to proportions not just of the form $\frac{1}{n}$. Hence, we have the following two conjectures.

Conjecture 4.1.11. *Let H be a connected hypergraph and $p \in (0,1) \cap \mathbb{Q}$ such that H contains no non-flammable edges. Then $b_{L,p}(H) \leq \lceil p|V(H)| \rceil$.*

Conjecture 4.1.12. *Let H be a connected hypergraph and $p \in (0,1)$ such that H contains no non-flammable edges. Then $b_{L,p}(H) \leq \lceil p|V(H)| \rceil$.*

Note that Conjecture 4.1.11 would imply Conjecture 4.1.12. If we are given $p \in (0,1) \setminus \mathbb{Q}$, then we can simply choose $p' \in (0,1) \cap \mathbb{Q}$ close enough to p such that $b_{L,p}(H) = b_{L,p'}(H)$ and $\lceil p|V(H)| \rceil = \lceil p'|V(H)| \rceil$, and then apply the bound from Conjecture 4.1.11. Clearly Conjecture 4.1.12 implies Conjecture 4.1.11, so the statements in the two conjectures are logically equivalent.

Theorem 4.1.13. *Let $p, q \in (0,1)$ and $p < q$. Then for any hypergraph H, $b_{L,p}(H) \leq b_{L,q}(H)$ and $b_p(H) \leq b_q(H)$.*

Proof. Observe that for any edge e, $\lceil p|e| \rceil \leq \lceil q|e| \rceil$. Hence, if at least $\lceil q|e| \rceil$ vertices are on fire in e, then it is also true that at least $\lceil p|e| \rceil$ vertices are on fire in e. Thus, if fire would propagate within e when using proportion q, it would also propagate within e when using proportion p. This means that any lazy burning set that is successful when using proportion q is also successful when using proportion p (if anything the propagation will be faster using proportion p). Therefore, $b_{L,p}(H) \leq b_{L,q}(H)$.

Now, let S be an optimal burning sequence for a hypergraph H when using proportion q. Suppose the arsonist burns H using proportion p by following S. Recall that, when using proportion p, the propagation may happen faster than when using proportion q. Thus, when the arsonist tries to burn H using proportion p by following S, it is possible that they will be prompted to burn a vertex that was already on fire for at least one round, which is not allowed. When this occurs, the arsonist can simply deviate from S for one round and choose an arbitrary source instead. Clearly, at the end of any round r, the set of vertices that are on fire when using proportion q and following S is a subset of the vertices that are on fire when following this strategy using proportion p. Therefore, the arsonist can burn H using proportion p in no more than $|S|$ rounds, so $b_p(H) \leq |S| = b_q(H)$. □

Since hypergraphs are discrete structures, being able to choose any real proportion $p \in (0,1)$ seems like a redundant amount of accuracy. Given a hypergraph H, there are uncountably many p that all yield the same value for $b_p(H)$, since for any $p \in (0,1)$, $b_p(H) \in \{1, 2, \ldots, |V(H)|\}$ (and similarly for $b_{L,p}(H)$). Moreover, in light of Theorem 4.1.13, there must surely be "cut-off points" for p. That is, there must be proportions $q_k \in (0,1)$

such that if $p < q_k$ then $b_p(H) < k$, and if $p > q_k$ then $b_p(H) \geq k$ (and similarly for the lazy version). We therefore introduce the following definition, which gives insight into the range of possible values for $b_p(H)$ and $b_{L,p}(H)$ given a hypergraph H (where p ranges over $(0,1)$), as well as what the aforementioned "cut-off points" are for p when burning H.

Definition 4.1.14. *Let H be a hypergraph with $|V(H)| = n$. The* burning distribution *of H is a partition $\mathcal{P}$ of $(0,1)$ into n (possibly empty) intervals $P_1, P_2, \ldots, P_n$ such that $b_p(H) = k$ if and only if $p \in P_k$. The* lazy burning distribution *is defined analogously, and denoted $\mathcal{P}_L$. As a convention, we will denote the intervals in $\mathcal{P}_L$ by $Q_1, Q_2, \ldots, Q_n$.*

As an easy first example of the (lazy) burning distribution of a family of hypergraphs; see Theorem 4.1.16, which describes the (lazy) burning distribution of an arbitrary Steiner triple system. In order to prove Theorem 4.1.16, we need the following result.

Lemma 4.1.15. *Let H be a BIBD(v,k,λ) and let $p \in (0, \frac{1}{k}]$. Then $b_{L,p}(H) = 1$ and $b_p(H) = 2$.*

Proof. First we calculate for any edge e, $\lceil p|e| \rceil = \lceil pk \rceil \leq \left\lceil \frac{1}{k} k \right\rceil = 1$. Since $p > 0$, we must therefore have $\lceil p|e| \rceil = 1$. Thus, one or more vertices on fire in an edge will cause the entire edge to catch fire. Moreover, since every pair of vertices appears together in at least one edge, if one vertex is on fire in H, then in the next time step the entire hypergraph will catch fire. Hence, any one vertex constitutes a lazy burning set, and in the round-based game only two rounds are required. □

Theorem 4.1.16. *Let H be an STS(v). The burning distribution for H contains $P_2 = \left(0, \frac{1}{3}\right]$, $P_{b(H)} = \left(\frac{1}{3}, \frac{2}{3}\right]$, and $P_v = \left(\frac{2}{3}, 1\right)$, where all other P_i are empty. The lazy burning distribution contains $Q_1 = \left(0, \frac{1}{3}\right]$, $Q_{b_L(H)} = \left(\frac{1}{3}, \frac{2}{3}\right]$, and $Q_v = \left(\frac{2}{3}, 1\right)$, where all other Q_j are empty.*

Proof. If $p \in \left(0, \frac{1}{3}\right]$ then $b_{L,p}(H) = 1$ and $b_p(H) = 2$ by Lemma 4.1.15, so $\left(0, \frac{1}{3}\right] \subseteq Q_1$ and $\left(0, \frac{1}{3}\right] \subseteq P_2$. If $p \in \left(\frac{1}{3}, \frac{2}{3}\right]$ then fire propagates within an edge in a manner identical to the original propagation rule (i.e. 2 vertices on fire in an edge cause the fire to propagate). Hence, $b_{L,p}(H) = b_L(H)$ and $b_p(H) = b(H)$, so $\left(\frac{1}{3}, \frac{2}{3}\right] \subseteq Q_{b_L(H)}$ and $\left(\frac{1}{3}, \frac{2}{3}\right] \subseteq P_{b(H)}$. Finally, if $p \in \left(\frac{2}{3}, 1\right)$ then every edge is non-flammable, so $b_{L,p}(H) = b_p(H) = v$ and therefore $\left(\frac{2}{3}, 1\right) \subseteq Q_v$ and $\left(\frac{2}{3}, 1\right) \subseteq P_v$. Since the intervals in the (lazy) burning distribution partition the interval $(0,1)$, the above inclusions are in fact equalities, so the result follows. □

Lemma 4.1.17. *For any $n \in \mathbb{N}$, there is a hypergraph H with $|V(H)| = n$ such that every interval in the lazy burning distribution is nonempty.*

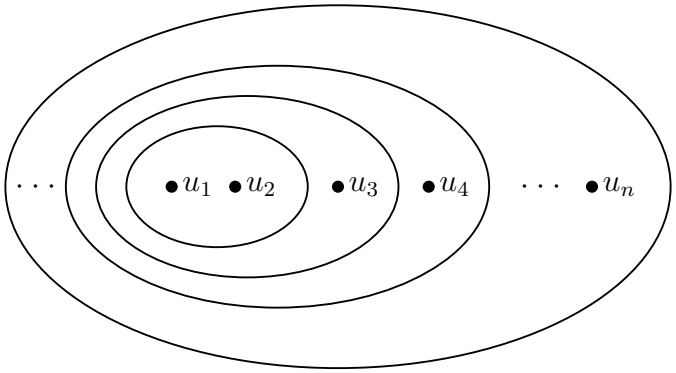

Figure 4.3: A hypergraph in which every interval in the lazy burning distribution is nonempty.

Proof. Consider the hypergraph H seen in Figure 4.3, which has $V(H) = \{u_1, \ldots, u_n\}$ and $E(H) = \big\{\{u_1, u_2\}, \{u_1, u_2, u_3\}, \{u_1, u_2, u_3, u_4\}, \ldots, \{u_1, u_2, \ldots, u_n\}\big\}$. There exists a lazy burning set of size one if and only if $p \in (0, \frac{1}{2}]$, so $Q_1 = (0, \frac{1}{2}]$ (one example of such a lazy burning set is $\{u_1\}$). There exists a minimum lazy burning set of size two if and only if $p \in (\frac{1}{2}, \frac{2}{3}]$, so $Q_2 = (\frac{1}{2}, \frac{2}{3}]$ (one example of such a lazy burning set is $\{u_1, u_2\}$). In general, for any $k < n$, $\{u_1, u_2, \ldots, u_k\}$ is a minimum lazy burning set if and only if $p \in (\frac{k-1}{k}, \frac{k}{k+1}]$, so $Q_k = (\frac{k-1}{k}, \frac{k}{k+1}]$. Finally, a minimum lazy burning set of size n exists if and only if $p > \frac{n-1}{n}$, since then every edge is non-flammable. Thus, $Q_n = (\frac{n-1}{n}, 1)$. □

4.2 A Propagation Rule that Uses Thresholds

Consider the following propagation rule: For some fixed *threshold* $t \in \mathbb{N}$, if t or more vertices are on fire in an edge e, then in the next round all unburned vertices in e catch fire. Instead of writing $b(H)$ and $b_L(H)$, when using this propagation rule we will write $b^{(t)}(H)$ and $b_L^{(t)}(H)$ respectively. For the entirety of Section 4.2 we will assume that we are using this propagation rule, unless it is otherwise stated. Recall that this is indeed a different process from bootstrap percolation with an infection threshold, which was discussed in Section 1.5.1.

Burning a hypergraph with such a propagation rule does not reduce to the original game when played on a 2-uniform hypergraph with $t > 1$. However, this propagation rule is still interesting, since a certain number of distinct vertices define an edge in many types of designs. For example, this rule with $t = 2$ is interesting when applied to a BIBD$(v, k, 1)$.

Theorem 4.2.1. *If H is a hypergraph with no isolated vertices and $t \in \mathbb{N}$ then $t \leq b_L^{(t)}(H) \leq t|E(H)|$.*

Proof. For the lower bound, clearly if fewer than t vertices are initially set on fire as part of the lazy burning set, then no fire will spread. Hence, any lazy burning set must be of size at least t, including a minimum lazy burning set, so $t \leq b_L^{(t)}(H)$.

For the upper bound, we can set t vertices on fire in each edge as our lazy burning set. Notice that because edges overlap, many edges may have more than t vertices on fire (but this is okay). Since every edge has at least t vertices on fire, it will only take one time step for H to be fully burned. Since this is an example of a lazy burning set of size $t|E(H)|$, we must have $b_L^{(t)}(H) \leq t|E(H)|$. □

Both bounds in Theorem 4.2.1 are tight — consider a connected hypergraph consisting of one edge that contains all of its vertices. Moreover, it seems as though the upper bound can be improved in the case that H is connected.

Conjecture 4.2.2. *If H is a connected hypergraph and $t \in \mathbb{N}$ then $b_L^{(t)}(H) \leq |E(H)|(t-1)+1$.*

One potential argument for a proof of Conjecture 4.2.2 is as follows. Set $t-1$ vertices on fire in each edge as part of the lazy burning set, then choose any edge e. Set an additional vertex on fire in e (so our lazy burning set contains $|E(H)|(t-1)+1$ vertices). Then all of e will catch fire in the first time step. In the second time step, if the vertices of the lazy burning set were chosen carefully, each edge adjacent to e should now contain at least t burned vertices. Hence, these edges will become completely burned in the second time step. We then consider each edge adjacent to a completely burned edge and repeat. Since H is connected, we can continue like this until H is completely burned. Can a rigorous proof of Conjecture 4.2.2 be established using this argument? We leave this as an open question.

We now turn our attention to burning BIBD$(v, k, 1)$s using the threshold $t = 2$. In particular, we find an upper bound on $b_L^{(2)}(H)$ where H is a BIBD$(v, k, 1)$ using an argument very similar to the one used to prove Theorem 3.2.7 (on page 55).

Lemma 4.2.3. *When applying Algorithm 1 to a BIBD$(v, k, 1)$, each time the inner loop finishes, the edges that are completely on fire induce a sub-BIBD$(v', k, 1)$ with $v' \leq v$ (using the propagation rule where $t = 2$ vertices on fire in an edge cause the entire edge to catch fire in the next time step).*

Proof. Observe that once the inner loop of Algorithm 1 has finished (within any iteration of the outer loop), there are only fully burned edges, edges containing a single burned vertex, and edges containing no burned vertices.

Denote the BIBD$(v, k, 1)$ by H. Denote the hypergraph induced by the fully burned edges of H after the inner loop of Algorithm 1 has finished by G. Since H is a BIBD$(v, k, 1)$, no two vertices of G appear together in more than one edge of G. We must show that every pair of vertices in G appear together in some edge of G.

Let $u, w \in V(G)$ and suppose there is no edge in $E(G)$ containing both u and w. Since H is a BIBD$(v, k, 1)$, there is some edge $B \in E(H)$ such that $u, w \in B$. Since B contains at least two burned vertices, B must in fact be completely burned (recall that every edge is completely burned, completely unburned, or contains exactly one burned vertex). Hence, $B \in E(G)$, which is a contradiction. Therefore, every pair of vertices in G appears together in some edge of G. So G is a BIBD. □

The following result is known; see [22] for example. We present its proof here for completeness (also see Figure 4.4 for a visual proof). It is a natural extension of Theorem 3.2.6.

Theorem 4.2.4. *If T is a BIBD$(v, k, 1)$ and A is a sub-BIBD$(v', k, 1)$ of T (with $v' \leq v$) then $|V(T)| \geq (k-1)|V(A)| + 1$.*

Proof. Consider any $z \in V(T) \setminus V(A)$. Each $x_i \in V(A)$ must appear in an edge of T with z. In particular, such an edge has the form $B_i = \{x_i, z, y_{i,1}, y_{i,2}, \ldots, y_{i,k-2}\}$ where each $y_{i,j}$ is a vertex in $V(T) \setminus V(A)$. This is because if some $y_{i,j}$ was a vertex in A, then x_i and $y_{i,j}$ would appear together in two different edges of T, one of them being B_i and the other being some edge in $E(A)$.

Furthermore, any two edges B_i and B_ℓ intersect only at z, since if $y_{i,p} = y_{\ell,q}$ for some $p, q \in \{1, 2, \ldots, k-2\}$, then z and $y_{i,p}$ appear together in two edges of T. Hence,

$$x_1, x_2, \ldots, x_{|V(A)|}, z, y_{1,1}, y_{1,2}, \ldots, y_{1,k-2}, y_{2,1}, y_{2,2}, \ldots, y_{2,k-2}, \ldots, y_{|V(A)|,1}, y_{|V(A)|,2}, \ldots, y_{|V(A)|,k-2}$$

are $(k-1)|V(A)| + 1$ distinct vertices in $V(T)$. □

Theorem 4.2.5. *If H is a BIBD$(v, k, 1)$ then $b_L^{(2)}(H) \leq \lfloor \log_{k-1}(v(k-2)+1) \rfloor$.*

Proof. Consider the worst-case scenario when applying Algorithm 1 to H. By Lemma 4.2.3, each time the inner loop of the algorithm finishes, a sub-BIBD of H is induced by the fully burned edges. Hence, the worst-case scenario is that H contains the maximum possible number of nested sub-BIBDs, and the algorithm chooses vertices as "poorly" as possible such that each nested sub-BIBD contributes a vertex to the lazy burning set. That is, each

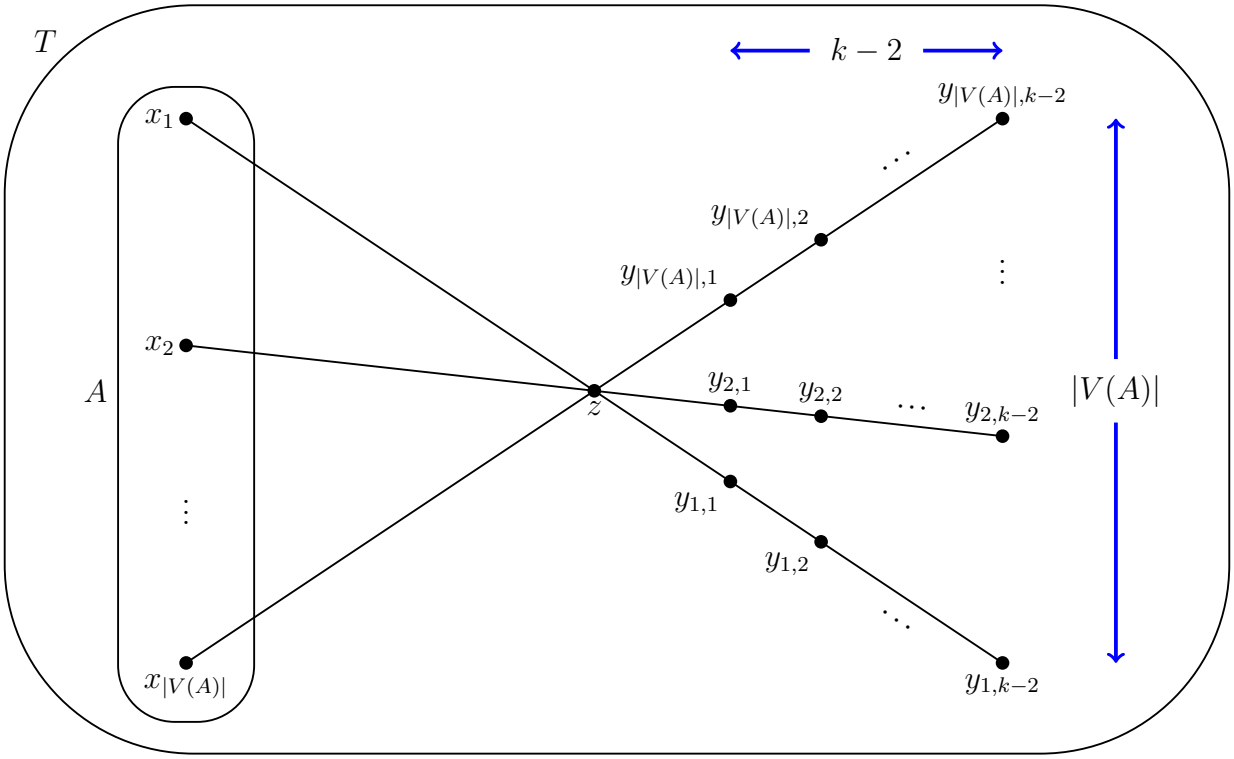

Figure 4.4: A visual proof of the bound $|V(T)| \geq (k-1)|V(A)| + 1$.

time the inner loop finishes, the sub-BIBD that is induced by the burned edges is as small as possible.

The smallest two nested sub-BIBDs contained in any BIBD$(v, k, 1)$ are the BIBD$(1, k, 1)$ (a single vertex) and the BIBD$(k, k, 1)$ (a single edge and each vertex within). Write $a_1 = 1$ and $a_2 = k$. By Theorem 4.2.4, the next smallest nested sub-BIBDs are of order $a_3, a_4, a_5, \ldots$ where $a_\ell = (k-1)a_{\ell-1} + 1$. We must solve for a_ℓ.

First, write $m = k - 1$. Then $a_\ell = ma_{\ell-1} + 1$, and it is easy to see that $a_2 = m + 1$, $a_3 = m^2 + m + 1$, and in general,

$$a_\ell = 1 + m + m^2 + \cdots + m^{\ell-1} = \frac{m^\ell - 1}{m - 1} = \frac{(k-1)^\ell - 1}{k - 2} \ .$$

Now, the worst-case scenario is that H has nested sub-BIBDs of order $1, k, a_3, a_4, \ldots, a_\ell, v$, where ℓ is the *largest* natural number such that $v \geq (k-1)a_\ell + 1$. Thus, including H itself, H has at most $\ell + 1$ nested sub-BIBDs. Observe that

$$v \geq (k-1)a_\ell + 1 = (k-1)\frac{(k-1)^\ell - 1}{k-2} + 1 = \frac{(k-1)^{\ell+1} - k + 1}{k-2} + 1.$$

Using this inequality, we solve for ℓ:

$$\begin{aligned}
& (v-1)(k-2)+k-1 \geq (k-1)^{\ell+1} \\
\implies \quad & v(k-2)-k+2+k-1 \geq (k-1)^{\ell+1} \\
\implies \quad & v(k-2)+1 \geq (k-1)^{\ell+1} \\
\implies \quad & \log_{k-1}(v(k-2)+1) \geq \ell+1 \\
\implies \quad & \left\lfloor \log_{k-1}(v(k-2)+1)-1 \right\rfloor = \ell
\end{aligned}$$

Therefore, the largest lazy burning set the algorithm could produce for H is of size $\ell+1 = \left\lfloor \log_{k-1}(v(k-2)+1) \right\rfloor$. Hence, $b_L^{(2)}(H) \leq \left\lfloor \log_{k-1}(v(k-2)+1) \right\rfloor$. □

Observe that Theorem 4.2.5 implies Theorem 3.2.7.

Chapter 5

Summary and Future Work

In Section 1.4 we introduced the round-based model for hypergraph burning and the basic propagation rule for the fire within a hyperedge. We also showed that the "obvious" propagation rule in which one vertex on fire in a hyperedge causes the whole hyperedge to catch fire is not interesting, as it is essentially a special case of graph burning. In Section 1.5 we introduced the rules for lazy hypergraph burning, and showed that it is trivial on graphs but not on hypergraphs. Lazy hypergraph burning is equivalent to a process known as H-bootstrap percolation, however our results are entirely different from those in the literature. We also introduced Algorithm 1, which naïvely constructs a lazy burning set for a hypergraph, and is therefore useful for finding upper bounds on the lazy burning number.

By combining our first few results in Section 2.1, we obtained our first major result on hypergraph burning, which can be found in Theorem 2.1.13 on page 25. This result essentially states that, if H is simple and has no isolated vertices (or if a less strict condition is met; see Theorem 2.1.6), then $|V(H)|-|E(H)| \leq b_L(H) < b(H) \leq \alpha(H)+1$. Moreover, all of these bounds are tight. Another interesting result was Theorem 2.1.17, which essentially says that $b_L(H)$ and $b(H)$ can differ by an arbitrarily large amount. We also showed that no analogous bound to the Burning Number Conjecture exists in (lazy) hypergraph burning (i.e. a universal bound on $b_L(H)$ or $b(H)$ that is sublinear in terms of $|V(H)|$), even if we only consider hypergraphs that are both uniform and linear. We therefore want to know: what restrictions must we impose on H for such a bound to exist? And what would this bound be? One possibility is that such a bound exists when we insist that $|E(H)|$ is at least $|V(H)|-1$ (which is always the case in a connected graph).

We now state several miscellaneous open questions which arose from Section 2.1.

- Given an optimal burning sequence $S = (x_1, x_2, \ldots, x_{b(H)})$, does there always exist a

subset of $\{x_1, x_2, \ldots, x_{b(H)}\}$ that is a minimum lazy burning set? This is certainly true when H is a graph.

- The *complement* of H, denoted $\overline{H}$, is the hypergraph with $V(\overline{H}) = V(H)$ such that, for each subset e of $V(H)$, e is an edge in $\overline{H}$ if and only if e is not an edge in H. Can the (lazy) burning number of $\overline{H}$ be bounded in terms of the (lazy) burning number of H? One might first investigate the case where H is k-uniform, and we only allow edges of size k in $\overline{H}$.

- Does the "density" of edges in H affect $b(H)$ or $b_L(H)$? What is the best way to define density for this purpose? (one option is to define density as $\frac{|E(H)|}{|V(H)|}$). Intuitively, it makes sense that a hypergraph with a higher density of edges would have a lower burning number and lazy burning number.

- Does there exist a hypergraph (that is not a path) such that rearranging the order of an optimal burning sequence results in a non-valid burning sequence? What are the necessary and sufficient conditions for a hypergraph to have this property?

- A hypergraph is uniquely burnable if it has a unique optimal burning sequence up to isomorphism. What are the necessary and sufficient conditions for a hypergraph to be uniquely burnable? We may also ask the analogous question for lazy burning.

- Given $n \in \mathbb{N}$, what is the set of all possible burning numbers for hypergraphs of order n? Is this set an interval? Can we at least determine this for certain families of hypergraphs?

- Is there an analogous result in hypergraph burning to the Tree Reduction Theorem from graph burning? And would such a result involve hypertrees (a family of tree-like hypergraphs)?

- Can we find stronger bounds or even exact values of $b(H)$ and $b_L(H)$ for specific classes of hypergraphs such as projective planes, Kneser hypergraphs, pairwise balanced designs, and BIBDs with block size greater than three?

We began Section 2.2 by considering a disconnected hypergraph H with connected components $G_1, \ldots, G_k$, and proved two unsurprising results. The first says that $b_L(H)$ is equal to the sum of all the $b_L(G_i)$ (Lemma 2.2.1 on page 32), and the second says that $b(H)$ is bounded below by the maximum of the $b(G_i)$, and above by the sum of the $b(G_i)$ (Lemma 2.2.2). We still do not know if either bound in Lemma 2.2.2 is tight. In Theorem 2.2.3, we improved the upper bound from Lemma 2.2.2, assuming that none of the G_i are isolated

vertices, and this bound is indeed tight. It seems like the bound from Theorem 2.2.3 also applies to disconnected hypergraphs with up to one isolated vertex, although a proof for this eludes us.

In Section 2.3 we investigated taking a strong/weak induced subhypergraph and the effect this has on the burning and lazy burning numbers. We first proved that taking either a strong or weak induced subhypergraph can potentially increase the burning and lazy burning numbers by an arbitrarily large amount (see Lemma 2.3.5 on page 41). However, in the only example we could find in which taking the weak induced subhypergraph increases the burning and lazy burning number, the resulting hypergraph contains singleton edges; see Figure 2.13. Are there examples where this does not happen? That is, can we find a hypergraph H with weak induced subhypergraph G such that G contains no singleton edges, $b(G) > b(H)$, and $b_L(G) > b_L(H)$? Can these differences be arbitrarily large? We next proved, unsurprisingly, that if G_1 is a strong subhypergraph of H, G_2 is a weak subhypergraph of H, and G_1 and G_2 are induced by the same set of vertices, then G_1 is "harder" to burn than G_2 i.e. $b_L(G_2) \leq b_L(G_1)$ and $b(G_2) \leq b(G_1)$. We also found sufficient conditions for which taking the weak induced subhypergraph will *not* increase the lazy burning number (Theorem 2.3.3) or the burning number (Theorem 2.3.4). Finally, we showed by examples that when one takes a strong subhypergraph, whether it was induced by a set of vertices or not, it is possible that the burning and lazy burning numbers both increase, and it is also possible that they both decrease (for a total of four different scenarios); see Figures 2.14, 2.15, and 2.16. We would also like to determine the necessary and sufficient conditions a hypergraph and its strong subhypergraph must satisfy in order to behave in these four ways.

In Section 2.4 we introduced Algorithms 2 and 3, which check potential burning sequences and potential lazy burning sets respectively. We showed that both algorithms run in polynomial time with respect to the size of their inputs, and hence that both the round-based and lazy hypergraph burning problems have complexity in **NP**. Since graph burning is **NP**-complete and is a sub-problem of hypergraph burning, we were able to show that the round-based model is **NP**-complete. It is still unknown whether or not lazy hypergraph burning is **NP**-complete. One approach could be to prove **NP**-completeness for a specific family of hypergraphs (although using 2-uniform hypergraphs for this purpose does not work, unlike in the round-based case). Since the restriction is a sub-problem of lazy hypergraph burning, this would prove **NP**-completeness for the general problem as well. Another approach could be to prove that a "solution" to lazy hypergraph burning somehow yields a "solution" to zero-forcing, as zero-forcing is **NP**-complete [1, 17, 24, 40] (recall that the two processes share similarities).

Our first major result on Steiner triple systems is Lemma 3.2.1 (see page 50), which essentially describes how the first four rounds of the game proceed assuming the arsonist never chooses a redundant source. Next, we defined a function h that maps the round number r to the maximum number of vertices that could possibly be on fire in an arbitrary STS at the end of round r. Corollary 3.2.3 uses this function to get a lower bound on the burning number of an STS. Observe Table 3.1, and notice that h grows extremely fast. As an example, the lemma and table combined tell us that an STS on approximately $30,000$ vertices has burning number at least seven. This either means that very large STSs can have comparatively small burning numbers, or that the bound from Corollary 3.2.3 is very weak. Our next significant result combined Algorithm 1 and the concept of nested sub-STSs to get an upper bound on the lazy burning number of an STS. Of course, it would be nice to also obtain a lower bound on the lazy burning number of an STS and an upper bound on the burning number of an STS. For the latter, one approach could be to replicate Theorem 3.2.2 with a function that maps the round number r to the *minimum* number of vertices that could possibly be on fire in an STS at the end of round r. We finished Section 3.2 by showing that for every admissible $v \in \mathbb{N}$, there is an STS(v) with lazy burning number three. In fact, any three vertices not all belonging to an edge form a lazy burning set in these STSs. Is an analogous statement true for any natural numbers apart from three?

In Section 3.3 we showed that by applying a well-known doubling construction to an STS, the lazy burning number either stays the same or increases by one; see Theorem 3.3.2 on page 60. We also determined exactly when doubling an STS will increase the lazy burning number, although the sufficient conditions in this result do not depend on the parameters of the hypergraph; see Theorem 3.3.7. Can we find sufficient conditions in terms of the parameters of the STS that yield the same conclusion as in Theorem 3.3.7? We also showed that by repeatedly doubling the STS(7), the lazy burning number increases each time, and hence there exist Steiner triple systems with arbitrary lazy burning number. Is there a Steiner triple system for which no amount of doubling will change the lazy burning number?

We now state several miscellaneous open questions which arose from Chapter 3.

- Two STSs on the same number of vertices can have different lazy burning numbers. Is the same true for the burning number? And what is the smallest order where this happens?

- Certain properties can greatly affect the lazy burning number of an STS. For example, the lazy burning numbers of all projective triple systems are known; see Corollary 3.3.17. What other properties are relevant to the lazy burning number of an STS?

Does being cyclic or resolvable matter? Do these properties also affect the burning number?

- Our computational results on STSs seem to suggest that all valid burning sequences in an STS are also optimal. Is there an STS with a valid burning sequence strictly larger than its burning number?

- The existence of sub-systems in an STS clearly has an impact on its lazy burning number. Is the same true for the burning number?

The first substantial result in Section 4.1 is a tight lower bound on $b_{L,p}(H)$ for any proportion $p \in (0,1)$; see Theorem 4.1.8 on page 74. We also found a tight upper bound on $b_{L,1/n}(H)$; see Theorem 4.1.10. The analogous result is still open for proportions *not* of the form $\frac{1}{n}$. Indeed, both the upper and lower bounds can be achieved simultaneously if we consider the hypergraph consisting of a single edge containing all of its vertices. We also found examples where only the upper/lower bound is achieved when $p = \frac{1}{2}$. We introduced the concepts of the *burning distribution* and *lazy burning distribution* of a hypergraph, which describe all the possible values for b_p and $b_{L,p}$ as p ranges over $(0,1)$. It would be interesting to investigate the burning distribution of BIBDs and other k-uniform hypergraphs. Of course, our results in this section are mostly about lazy proportion-based burning. Are there analogous results for the round-based version of the game with this propagation rule?

In Section 4.2 we considered a propagation rule that uses thresholds, and we first found tight upper and lower bounds on $b_L^{(t)}(H)$ when H has no isolated vertices; see Theorem 4.2.1 on page 79. However, the only example we could find where these bounds are tight is the hypergraph consisting of a single edge containing all of its vertices. Again, in this case both the upper and lower bounds are achieved simultaneously. Are there other examples where equality in these bounds is achieved, and not at the same time? We also conjecture that the upper bound can be improved in the case that H is connected; see Conjecture 4.2.2. Using similar techniques to those used in Chapter 3, we found an upper bound on the lazy burning number of a BIBD$(v,k,1)$ using a threshold of 2. Again, most of the results in this section focus on the lazy version of the game. Are there similar results for the round-based version of the game?

Perhaps the most promising direction for future research is the study of more alternative propagation rules. One promising example is a non-deterministic propagation rule. Consider any edge e and suppose it contains m burned vertices at the end of round r. Then, in round $r+1$, each unburned vertex in e catches fire with probability $\frac{m}{|e|}$. Of course, when burning according to this propagation rule in the lazy model, one needs only to burn a single vertex

in each connected component. Therefore, the objective of this version of the game would be to determine the expected number of rounds the lazy burning game will take given a lazy burning set with one vertex in each connected component, and which of these lazy burning sets has the lowest expectation. In the round-based version of the game with this propagation rule, one may not use a pre-determined burning sequence since there is no way in general to tell if it will be valid due to the element of chance. Again, the objective would be to determine the expected number of rounds the game will take when playing optimally.

Bibliography

[1] A. Aazami. Hardness Results and Approximation Algorithms for some Problems on Graphs. PhD Thesis, University of Waterloo, 2008.

[2] AIM Minimum Rank-Special Graphs Work Group. Zero Forcing Sets and the Minimum Rank of Graphs. *Linear Algebra Appl.* Vol. 428(7) (2008) pp. 1628–1648.

[3] N. Alon. Transmitting in the n-Dimensional Cube. *Discrete Applied Mathematics* Vol. 37–38 (1992) pp. 9–11.

[4] M. Bahmanian, M. Šajna. Connection and Separation in Hypergraphs. *Theory and Applications of Graphs* Vol. 2 Iss. 2 Art. 5 (2015).

[5] J. Balogh, B. Bollobás, R. Morris, O. Riordan. Linear Algebra and Bootstrap Percolation. *Journal of Combinatorial Theory Series A* Vol. 119 (2012) pp. 1328–1335.

[6] F. Barioli, W. Barrett, S. Fallat, H. Hall, L. Hogben, B. Shader, P. van den Driessche, H. van der Holst. Zero Forcing Parameters and Minimum Rank Problems. *Linear Algebra Appl.* Vol. 433(2) (2010) pp. 401–411.

[7] F. Barioli, W. Barrett, S. Fallat, H. Hall, L. Hogben, B. Shader, P. van den Driessche, H. van der Holst. Parameters Related to Tree-Width, Zero Forcing, and Maximum Nullity of a Graph. *J. Graph Theory* Vol. 72(2) (2013) pp. 146–177.

[8] P. Bastide, M. Bonamy, A. Bonato, P. Charbit, S. Kamali, T. Pierron, M. Rabie. Improved Pyrotechnics: Closer to the Burning Number Conjecture. arXiv:2110.10530v2 (2022).

[9] S. Bessy, A. Bonato, J. Janssen, D. Rautenbach, E. Roshanbin. Burning a Graph is Hard. *Discrete Applied Mathematics* Vol. 232 (2017) pp. 73–87.

[10] S. Bessy, A. Bonato, J. Janssen, D. Rautenbach, E. Roshanbin. Bounds on the Burning Number. *Discrete Applied Mathematics* Vol. 235 (2018) pp. 16–22.

[11] B. Bollobás. Weakly k-Saturated Graphs. *Beiträge zur Graphentheorie* (Kolloquium, Manebach, 1967) Vol. 25 (1968) pp. 25-31.

[12] A. Bonato. A Survey of Graph Burning. *Contributions to Discrete Mathematics* Vol. 16 No. 1 (2021) pp. 185–197.

[13] A. Bonato, J. Janssen, E. Roshanbin. Burning a Graph as a Model of Social Contagion. Proceedings of WAW (2014) pp. 13–22.

[14] A. Bonato, J. Janssen, E. Roshanbin. How to Burn a Graph. *Internet Mathematics* 1–2 (2016) pp. 85–100.

[15] A. Bonato and T. Lidbetter. Bounds on the Burning Numbers of Spiders and Path-Forests. *Theoretical Computer Science* Vol. 794 (2019) pp. 12-19.

[16] A. Bretto. Hypergraph Theory: An Introduction. *Springer* (2013).

[17] B. Brimkov, I. Hicks. Complexity and Computation of Connected Zero Forcing. *Discrete Applied Mathematics* Vol. 229 (2017) pp. 31–45.

[18] C. Colbourn and J. Dinitz. Handbook of Combinatorial Designs 2nd ed. *Chapman & Hall/CRC* (2007).

[19] C. Colbourn and A. Rosa. Triple Systems. *Clarendon, Oxford University Press* (1999).

[20] S. Das, S. Dev, A. Sudhukhan, U. Sahoo, S. Sen. Burning Spiders. *Algorithms and Discrete Applied Mathematics* Vol. 10743 (2018) pp. 155-163.

[21] M. Dewar, D. Pike, J. Proos. Connectivity in Hypergraphs. *Canadian Mathematical Bulletin* Vol. 61 (2) (2018) pp. 252–271.

[22] J. Dinitz and D. Stinson. Contemporary Design Theory: A Collection of Surveys. *Wiley* (1992).

[23] J. Doyen. Sur la structure de certaines systèmes triples de Steiner. *Mathematische Zeitschrift* Vol. 111 (1969) pp. 289–300.

[24] S. Fallat, K. Meagher, B. Yang. On the Complexity of the Positive Semidefinite Zero Forcing Number. *Linear Algebra and its Applications* Vol. 491 (2016) pp. 101–122.

[25] F. Garbe, R. Mycroft, A. McDowell. Contagious Sets in a Degree-Proportional Bootstrap Percolation Process. *Random Structures and Algorithms* Vol. 53 Iss. 4 (2018) pp. 638-651.

[26] I. Hartarsky. $\mathfrak{U}$-Bootstrap Percolation: Critical Probability, Exponential Decay and Applications. *Annales de l'Institut Henri Poincairé Probabilités et Statistiques* Vol. 57 (2021) No. 3 pp. 1255–1280.

[27] L. Huang, G. Chang, H. Yeh. On Minimum Rank and Zero Forcing Sets of a Graph. *Linear Algebra Appl.* Vol. 432(11) (2010) pp. 2961-2973.

[28] S. Janson, T. Łuczak, T. Turova, T. Vallier. Bootstrap Percolation on the Random Graph $G_{n,p}$. *The Annals of Applied Probability* Vol. 22 No. 5 (2012) pp. 1989–2047.

[29] M. Kang, C. Koch, T. Makai. Bootstrap Percolation in Random k-Uniform Hypergraphs. *Electronic Notes in Discrete Mathematics, Elsevier* (2015).

[30] T. Kirkman. On a Problem in Combinations. *Cambridge and Dublin Mathematical Journal* Vol. 2 (1847) pp. 191-204.

[31] H. Liu, X. Hu, X. Hu. Burning Number of Caterpillars. *Discrete Applied Mathematics* Vol. 284 (2020) pp. 332-340.

[32] R. Mathon, K. Phelps, A. Rosa. Small Steiner Triple Systems and their Properties. *Ars Combinatoria* Vol. 15 (1983) pp. 3–110 (and errata: Vol. 16 (1983) pp. 286).

[33] N. Morrison, J. Noel. A Sharp Threshold for Bootstrap Percolation in a Random Hypergraph. *Electronic Journal of Probability* Vol. 26 Art. 97 (2021) pp. 1–85.

[34] N. Morrison, J. Noel. Extremal Bounds for Bootstrap Percolation in the Hypercube. *Journal of Combinatorial Theory Series A* Vol. 156 (2018) pp. 61–84.

[35] S. Norin and J. Turcotte. The Burning Number Conjecture Holds Asymptotically. arXiv:2207.04035v1 (2022).

[36] The On-Line Encyclopedia of Integer Sequences. https://oeis.org

[37] J. Singer. A Theorem in Finite Projective Geometry and Some Applications to Number Theory. *Transactions of the American Mathematical Society* Vol. 43 No. 3 (1938) pp. 377–385.

[38] D. Stinson. Combinatorial Designs: Constructions and Analysis. *Springer* (2004).

[39] D. West. Introduction to Graph Theory 2nd ed. *Pearson* (2001).

[40] B. Yang. Fast-Mixed Searching and Related Problems on Graphs. *Theoretical Computer Science* Vol. 507 (2013) pp. 100–113.

Appendix A

Source Code

A.1 Main Routine

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <limits.h>
#include "thesis_header.h"

int main(){

    time_t start_time;
    start_time = time(NULL);

    //the non-cyclic STS(13)

    int v=13;
    int B=26;
    int size=3*B;

    int *Blocks;

    if((Blocks = (int*) malloc(3*B*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
    }

    int array[]={1,2,7,1,3,6,1,4,8,1,5,9,1,10,12,1,11,13,2,3,8,2,
                 4,5,2,6,10,2,9,11,2,12,13,3,4,7,3,5,13,3,9,10,3,
```

```
                11,12,4,6,13,4,9,12,4,10,11,5,6,11,5,7,10,5,8,12,
                6,7,12,6,8,9,7,8,11,7,9,13,8,10,13};

    int x;

    printf("Blocks of the non-cyclic STS(13) are:\n\n");
    fflush(stdout);

    for(x=0;x<size;x++){
       Blocks[x]=array[x];
       printf("%d ",Blocks[x]);
       fflush(stdout);

       if(x%3==2){
         printf("\n");
         fflush(stdout);
       }
    }

    printf("\n");
    fflush(stdout);

/*

    //a cyclic STS(21)

    int v=21;
    int B=84;

    int *Blocks;

    if((Blocks = (int*) malloc(3*B*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
    }

    int x;

    int first=1;  //first base block is {1,2,6}
    int second=2;
    int third=6;

    for(x=0;x<21;x++){
```

```
   Blocks[3*x]=first;
   Blocks[3*x+1]=second;
   Blocks[3*x+2]=third;

   printf("Block %d: {%d,%d,%d}\n",x+1,first,second,third);

   first++;

   if(first>v)
     first-=v; //we dont use % here because we are writing our
               //points as 1,2,...,v, NOT as 0,1,...,v-1.
   second++;

   if(second>v)
     second-=v;

   third++;

   if(third>v)
     third-=v;
}

first=1;  //second base block is {1,3,11}
second=3;
third=11;

for(x=21;x<42;x++){

   Blocks[3*x]=first;
   Blocks[3*x+1]=second;
   Blocks[3*x+2]=third;

   printf("Block %d: {%d,%d,%d}\n",x+1,first,second,third);

   first++;

   if(first>v)
     first-=v; //we dont use % here because we are writing our
               //points as 1,2,...,v, NOT as 0,1,...,v-1.
   second++;

   if(second>v)
```

```
     second-=v;

   third++;

   if(third>v)
     third-=v;
}

first=1;  //third base block is {1,4,16}
second=4;
third=16;

for(x=42;x<63;x++){

   Blocks[3*x]=first;
   Blocks[3*x+1]=second;
   Blocks[3*x+2]=third;

   printf("Block %d: {%d,%d,%d}\n",x+1,first,second,third);

   first++;

   if(first>v)
     first-=v; //we dont use % here because we are writing our
               //points as 1,2,...,v, NOT as 0,1,...,v-1.
   second++;

   if(second>v)
     second-=v;

   third++;

   if(third>v)
     third-=v;
}

first=1;  //fourth base block is {1,8,15}
second=8;
third=15;

for(x=63;x<84;x++){

   Blocks[3*x]=first;
```

```
        Blocks[3*x+1]=second;
        Blocks[3*x+2]=third;

        printf("Block %d: {%d,%d,%d}\n",x+1,first,second,third);

        first++;

        if(first>v)
          first-=v; //we dont use % here because we are writing our
                    //points as 1,2,...,v, NOT as 0,1,...,v-1.
        second++;

        if(second>v)
          second-=v;

        third++;

        if(third>v)
          third-=v;
     }

     printf("\n");
     fflush(stdout);

*/

   char *general_adj_matrix; //only used in "general" and
                             //"fast" version of functions
   if((general_adj_matrix = (char*) malloc(v*B*sizeof(char)))==NULL){
     printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
     exit(1);
   }

   for(x=0;x<v*B;x++)
      general_adj_matrix[x]=0;

   convert_STS_to_general_adj_matrix(v, B, Blocks, general_adj_matrix);

   ////////////////////////////////////////////////////////////

   //STS_burning_number(v, Blocks, B);

   //general_burning_number(v, B, general_adj_matrix);
```

```
   fast_burning_number(v, B, general_adj_matrix);

   ////////////////////////////////////////////////////////

   //STS_lazy_burning_number(v, Blocks, B);

   general_lazy_burning_number(v, B, general_adj_matrix);

   //fast_lazy_burning_number(v, B, general_adj_matrix);

    time_t elapsed_time;
    elapsed_time = time(NULL) - start_time;
    printf("\nelapsed time: %ld \n", elapsed_time);

    return 0;

}
```

A.2 Subroutines

```
void STS_make_adj_matrix(int v, int *Blocks, int B, int *adj_matrix);
int STS_adj_matrix_what_is(int *A, int v, int i, int j);
void STS_adj_matrix_set_bit(int *A, int v, int i, int j, int value);
void STS_burning_number(int v, int *Blocks, int B); //finds the burning # of an STS
void STS_arsonist_move(int v, int *STS_adj_matrix, int round_count, int *burning_seq, int fire_size,
                       char *on_fire, int *burning_num_tally);
void STS_spread_fire(int v, int *STS_adj_matrix, int round_count, int *burning_seq, int fire_size,
                     char *on_fire, int *burning_num_tally);
void STS_lazy_burning_number(int v, int *Blocks, int B); //finds the lazy burning # of an STS
void STS_build_lazy_burning_sets(int *LBS, int current_size, int desired_size, int v,
                                 int *STS_adj_matrix, int *t_count, int *s_count);
int STS_lazily_burn(int *LBS, int size_of_LBS, int v, int *STS_adj_matrix, char *on_fire);
void convert_STS_to_general_adj_matrix(int v, int B, int *blocks, char *general_adj_matrix);
char general_adj_matrix_what_is(int v, int B, char *adj_martrix, int i, int j);
void general_adj_matrix_set_bit(int v, int B, char *adj_matrix, int i, int j, char x);
void general_burning_number(int v, int B, char *adj_matrix);//finds burning # of an arbitrary hypergraph
void general_arsonist_move(int v, int B, char *adj_matrix, int round_count, int *burn_seq,
                           int fire_size, char *on_fire, int *burn_num_tally);
void general_spread_fire(int v, int B, char *adj_matrix, int round_count, int *burning_seq,
                         int fire_size, char *on_fire, int *burning_num_tally);
void general_lazy_burning_number(int v, int B, char *adj_matrix);//finds lazy burning # of an arbitrary hypergraph
void general_build_lazy_burning_sets(int *LBS, int current_size, int desired_size, int v, int B,
                                     char *adj_matrix, int *t_count, int *s_count);
int general_lazily_burn(int *LBS, int size_of_LBS, int v, int B, char *adj_matrix, char *on_fire);
void convert_blocks_to_adj_matrix(int v, int B, int zero_or_one, int total_size_of_blocks,
                                  int *blocks, char *general_adj_matrix);
void print_adj_matrix(int v, int B, char *adj_matrix);
```

```
void double_STS(int v, int *Blocks, int B, int *new_blocks, int new_v, int new_B);
void fast_lazy_burning_number(int v, int B, char *adj_matrix); //always faster than general_lazy_burning_number()
int fast_build_lazy_burning_sets(int *LBS, int current_size, int desired_size, int v, int B,
                                 char *adj_matrix);
void fast_burning_number(int v, int B, char *adj_matrix); //NOT always faster than general_burning_number()
int fast_build_burning_seq(int *SEQ, int current_size, int desired_size, int v, int B,
                           char *adj_matrix);
int fast_test_burn_seq(int *SEQ, int seq_length, int v, int B, char *adj_matrix);



void STS_make_adj_matrix(int v, int *Blocks, int B, int *adj_matrix){

     //assumes inputs v, Blocks, and B are all valid
     //input: an STS(v) and a matrix of size ((v*v)-v)/2 initialized to all 0's (adj_matrix)
     //output: fills adj_matrix with info. If {x,y,z} is a block then the entry at row x
     //column y of adj_matrix is z, row x col z is y, etc.

     int x,y,z;

     int first, second, third; //hold the values of the corresponding entries in a block

     for(x=0; x<3*B; x+=3){ //x is the index of the first entry of each block

        first = Blocks[x];
        second = Blocks[x+1];
        third = Blocks[x+2];

        STS_adj_matrix_set_bit(adj_matrix, v, first, second, third);
        STS_adj_matrix_set_bit(adj_matrix, v, first, third, second);
        STS_adj_matrix_set_bit(adj_matrix, v, second, third, first);

     }

/*   printf("\nadjacency matrix of the STS(%d) is:\n", v);
     fflush(stdout);

     for(x=1;x<=v;x++){ //printing adj matrix will work nicely for any v<=9999

        printf("\n");
        fflush(stdout);

        for(y=1;y<=v;y++){

           if(x==y){
             printf("%d   ", 0);
             fflush(stdout);
           }

           else{

               z=STS_adj_matrix_what_is(adj_matrix, v, x, y);

               if(z<10){
                 printf("%d   ", z);
```

```
                fflush(stdout);
              }

              else if(z<100){
                     printf("%d  ", z);
                     fflush(stdout);
                   }

              else{
                    printf("%d ", z);
                    fflush(stdout);
                  }

          }//end else
       }
    }

    printf("\n\n");
    fflush(stdout);
*/

}



int STS_adj_matrix_what_is(int *A, int v, int i, int j){

 //returns entry at row i column j of the adjacency matrix representing our STS(v)

    if(i<1 || i>v || j<1 || j>v || i==j){
      printf("error, see line %d", __LINE__);
      exit(1);
    }

    if(i > j)
      return STS_adj_matrix_what_is(A, v, j, i);

    else
      return A[(i-1)*v - (i*(i-1))/2 + (j-i) - 1];

}



void STS_adj_matrix_set_bit(int *A, int v, int i, int j, int value){

 //sets row i column j of adjacency matrix representing our STS(v) to value

    if(i<1 || i>v || j<1 || j>v || i==j|| value<1 || value>v){ //entries in the blocks of our
      printf("error, see line %d", __LINE__);                  //STS(v) are in {1,2,...,v}
      exit(1);
    }

    if(i > j)
      STS_adj_matrix_set_bit(A, v, j, i, value);
```

```
      else
        A[(i-1)*v - (i*(i-1))/2 + (j-i) - 1] = value;

}



void STS_burning_number(int v, int *Blocks, int B){

     //input: an STS(v). B=# of blocks. Blocks is an array of 3B integers, each ranging from 1 to v.
     //output: a list of all possible burning times as well as the longest and shortest times
     //        i.e. the burning #
     int x,y,z;
     int p=v%6;
     //int q=sizeof(Blocks)/sizeof(Blocks[0]); //q is length of array Blocks
     //printf("%d",q); //for some reason this didnt work
     //add || q!=3*B to the below if

     if(v<1 || p==0 || p==2 || p==4 || p==5){
       printf("\n invalid input, see line %d\n", __LINE__);
       exit(1);
     }

     int *adj_matrix;
     int size_adj_matrix=((v*v)-v)/2; //doesnt need to be of size v*v since it is symmetrical
                                      //and the diagonal is all 0's
     if((adj_matrix = (int*) malloc(size_adj_matrix*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     for(x=0; x<size_adj_matrix; x++)
        adj_matrix[x]=0;

     STS_make_adj_matrix(v, Blocks, B, adj_matrix);

     char *on_fire; //keeps track of which vertices are currently burned...
                    //burned_vertices[x-1]=1 iff vertex x is burned, =0 otherwise
 if((on_fire = (char*) malloc(v*sizeof(char)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     int *burn_seq;//lists the vertices of the hypergraph in the order that the player wishes to burn them

 if((burn_seq = (int*) malloc(v*sizeof(int)))==NULL){//a burning sequence can be no longer than v
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     int *burn_num_tally;

     if((burn_num_tally = (int*) malloc(v*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
```

```
      exit(1);
    }

    for(x=0;x<v;x++){
       on_fire[x]=0;
       burn_seq[x]=0;
       burn_num_tally[x]=0;
    }

    int round_count=0;
    int fire_size=0;

    STS_arsonist_move(v, adj_matrix, round_count, burn_seq, fire_size, on_fire, burn_num_tally);

    printf("\n");
    fflush(stdout);

    printf("\nNote: we only consider those burning sequences in which");
    printf("\nredundant sources are avoided until possibly the last round.\n");
    printf("Also, sequences that differ only in the final redundant");
    printf("\nsource are counted seperately.\n\n");
    fflush(stdout);

    for(x=0;x<v;x++){
       printf("there were %d burning sequences of length %d\n", burn_num_tally[x], x+1);
       fflush(stdout);
    }

}



void STS_arsonist_move(int v, int *STS_adj_matrix, int round_count, int *burning_seq, int fire_size,
                       char *on_fire, int *burning_num_tally){

     //v is the number of points in the STS. STS_adj_matrix holds info about the blocks of the STS
     //and allows for faster look-ups than just using the blocks themselves. round_count is the
     //number of moves the arsonist has made so far i.e. the current length of the burning sequence.
     //burning_seq is an array of length v that holds the burning sequence, x_1,x_2,... where each
     //x_i is a vertex of the hypergraph i.e. an element of {1,2,..., v}. if currently the burning
     //sequence is of length k, then x_{k+1}=0. fire_size is the number of vertices that are currently
     //burning. on_fire is an array of length v where the jth entry is a 1 if point j is currently
     //on fire, 0 otherwise. burning_num_tally records the # of valid burning sequences of length k
     //in position k-1 of the array.

    int x;

    int arsonist_point=0; //delclared in each iteration of this function

    while(arsonist_point<v){ //arsonist can choose points 1,2,..., or v

         arsonist_point++;

         if(on_fire[arsonist_point-1]==0){ //arsonist cannot choose a vertex that is already on fire
```

```
            on_fire[arsonist_point-1]=1; //arsonist sets this point on fire as their move this round
            round_count++;

            //potentially add in something here that skips over testing any
            //seqs that are longer than the current shortest known seq

            burning_seq[round_count-1]=arsonist_point; //this point gets added to the burning sequence
            fire_size++; //asonist burned 1 unburned vertex so the fire size increased by 1

            if(fire_size<v) //if the hypergraph is not yet fully burned
              STS_spread_fire(v, STS_adj_matrix, round_count, burning_seq, fire_size, on_fire,
                              burning_num_tally);

            else{ //the hypergraph is fully burned... and the vertex just chosen was the last unburned
                  //vertex! So the arsonist did NOT choose a redundant source as the last source
                burning_num_tally[round_count-1]++;

                if(burning_num_tally[round_count-1]==1){ //if this is the first burning sequence
                                                         //discovered of this length
                  printf("first discovered burning sequence of length %d is:\n", round_count);
                  fflush(stdout);

                  for(x=0;x<round_count;x++){
                     printf("%d ", burning_seq[x]);
                     fflush(stdout);
                  }

                  printf("\n");
                  fflush(stdout);

                }
            }//end else

            on_fire[arsonist_point-1]=0;  //this function is semi-recursive... it calls STS_spread_fire,
            burning_seq[round_count-1]=0; //which calls STS_arsonist_move, etc. in order to go through
            fire_size--;                  //all valid burning sequences. once this iteration of
            round_count--;                //STS_arsonist_move gets to this point, all of the sequences
                                          //that start with the first (current) round_count entries
          } //end if                     //of the (current) burning_seq will have been tested... so
                                          //change the (current) most recent entry in burning_seq,
      }//end while                       //and test all the ways to finish the sequence.

}



void STS_spread_fire(int v, int *STS_adj_matrix, int round_count, int *burning_seq, int fire_size,
                     char *on_fire, int *burning_num_tally){

      int x,y,z;

      char *current_fire_snapshot;

      if((current_fire_snapshot = (char*) malloc(v*sizeof(char)))==NULL){
        printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
```

```
   exit(1);
 }

 char *newly_burned_vertices;

 if((newly_burned_vertices = (char*) malloc(v*sizeof(char)))==NULL){
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
 }

 for(x=0; x<v; x++){
    current_fire_snapshot[x]=on_fire[x];
    newly_burned_vertices[x]=0;
 }

for(x=0;x<v-1;x++){
   for(y=x+1;y<v;y++){

      if(on_fire[x]==1 && on_fire[y]==1){ //must spread fire to vertex z where {x,y,z} is a block

        z=STS_adj_matrix_what_is(STS_adj_matrix, v, x+1, y+1);//x & y are in {0,...,v-1} so input x+1 & y+1
                                              //note that a different x,y pair might give us the same z!
        if(on_fire[z-1]==0) //z is in {1,2,...,v}, so here input z-1 to on_fire[], NOT z
          newly_burned_vertices[z-1]=1; //CANNOT set on_fire[z-1] to 1 here bc then this new
                                       //fire would propogate to other vertices during this loop
      }
   }
}

for(x=0; x<v; x++){

   if(newly_burned_vertices[x]==1){ //dont need && on_fire[x]==0
     on_fire[x]=1;//be VERY CAREFUL about when/how we incriment fire_size!!! if one vertex is lit
     fire_size++; //on fire dure to its presence in multiple blocksthen we only want to incriment
   }              //fire_size by 1. for example, {x,y,z} and {u,v,z} where x,y,u,v are all on
                  //fire... then z gets set on fire due to 2 different blocks, but fire_size
}                 //only increased by 1

if(fire_size<v) //only continue with arsonists move if the graph is not yet fully burned
  STS_arsonist_move(v, STS_adj_matrix, round_count, burning_seq, fire_size, on_fire, burning_num_tally);

else{ //the hypergraph is fully burned after the most recent spread-fire step... remember in
      //this case the arsonist should pick a redundant source! but there is no need to call
      //arsonist_move() again
     if(burning_num_tally[round_count]==0){ //if this is the first burning sequence discovered of length
                                            //round_count+1 (including redundant source at the end)
        printf("first discovered burning sequence of length %d is:\n", round_count+1);
        fflush(stdout);         //NOT round_count (like it used to be in earlier version of code)

        for(x=0;x<v;x++){                //could replace v with round_count if we wanted
           printf("%d ", burning_seq[x]); //to not print the 0's at the end of burning_seq.
           fflush(stdout);               //burning_seq should have round_count many nonzero
        }                                //entries at the start, followed by v-round_count 0's

        printf(" plus any one of the following redundant sources at the end: ");
```

```
            fflush(stdout);

            for(x=0;x<v;x++){
               if(newly_burned_vertices[x]==1){

                 printf("%d ", x+1);
                 fflush(stdout);

               }
            }

          }//end if

           for(x=0;x<v;x++){ //we incriment burning_num_tally[round_count] once for each possible redundant
                             //source at the end, since each of them corresponds to a distinct burning sequence
              if(newly_burned_vertices[x]==1)
                 burning_num_tally[round_count]++;//NOT burning_num_tally[round_count-1] (like
                                                  // it used to be in earlier version of code)
           } //we incriment burning_num_tally[round_count] by the size of newly_burned_vertices[]

    }//end else

    free(newly_burned_vertices);
                                          //un-spread the fire. note that we do not have to change
    for(x=0; x<v; x++)                    //fire_size back to the value it started at, since fire_size
       on_fire[x]=current_fire_snapshot[x]; //was passed by value into this function i.e. the STS_arsonist_move
                                          //that called this function did not have its version of fire_size
    free(current_fire_snapshot);          //changed. We DO need to revert the array on_fire[] back to the
                                          //original since it was a pointer to an array i.e. this function
}                                         //altered the same on_fire array that is used in the
                                          //STS_arsonist_move function that called this function.


void STS_lazy_burning_number(int v, int *Blocks, int B){

 int t,x;        //t_count will hold the # of t-subsets that have been tested. Once we have tested
 int *t_count;   //them all, we will print t_count[0]... if it isnt v choose t then we know there
     int *s_count;   //is a mistake in the code! s_count is the # of successful tests i.e. the # of
                     //lazy burning sets for the STS(v)
     if((t_count = (int*) malloc(1*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     t_count[0]=0;

     if((s_count = (int*) malloc(1*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     s_count[0]=0;

     int *adj_matrix;
     int size_adj_matrix=((v*v)-v)/2; //doesnt need to be of size v*v since it is
```

```
                                    //symmetrical and the diagonal is all 0's
     if((adj_matrix = (int*) malloc(size_adj_matrix*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     for(x=0; x<size_adj_matrix; x++)
        adj_matrix[x]=0;

     STS_make_adj_matrix(v, Blocks, B, adj_matrix);

     int *LBS; //the lazy burning set. It is of size v. unused entries will be set to 0. ex. if we
               //are testing all 4-subsets of v=10 points then the last 6 entries of LBS will be 0.
     if((LBS = (int*) malloc(v*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     for(x=0; x<v; x++)
        LBS[x]=0;

     for(t=1; t<=v; t++){ //find all lazy burning sets of size t

        STS_build_lazy_burning_sets(LBS, 0, t, v, adj_matrix, t_count, s_count);

        printf("\n(%d choose %d) = %d. There are %d lazy burning sets of size %d\n", v, t,
               t_count[0], s_count[0], t);
        fflush(stdout);

        if(s_count[0]!=0){ //we just found the lazy burning number

          printf("\nlazy burning number is %d\n", t);
          fflush(stdout);
          break;

        }

        t_count[0]=0;
        //s_count[0]=0; //only need this if we want to keep testing
                        //after finding the lazy burning number
        for(x=0; x<v; x++)
           LBS[x]=0;

     }//end for

}



void STS_build_lazy_burning_sets(int *LBS, int current_size, int desired_size, int v,
                                 int *STS_adj_matrix, int *t_count, int *s_count){

     // recursively builds all unordered subsets of {1,2,...,v} of size desired_size
     // and calls lazily_burn() for each to see if they are indeed lazy burning sets.
```

```
//current_size is the # of cells of LBS that have been assigned a nonzero value so far
//remember that entries in the blocks f the sts must range from 1 to v

//t_count is the # of t-subsets of {1,2,...,v}... should be v choose t (where t=desired_size)
//s_count is the # of lazy burning sets of size desired_size for the STS(v)

int x,y,z;

if(current_size==desired_size){//potential lazy burning set is done being built...
                               //just have to test it with STS_lazily_burn
  t_count[0]++;

  //int fire_size=desired_size; //dont need to declare fire_size... just plug desired_size
                                //into STS_lazily_burn
  char *on_fire; //on_fire keeps track of which vertices are currently burned...
                 //on_fire[x-1]=1 iff vertex x is burned, =0 otherwise
  if((on_fire = (char*) malloc(v*sizeof(char)))==NULL){
    printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
    exit(1);
  }

  for(x=0;x<v;x++)
     on_fire[x]=0;

  for(x=0;x<desired_size;x++){

     y=LBS[x]; //y is in {1,2,...,v}. it is a point that is initially set on fire as part
               //of the lazy burning set
     on_fire[y-1]=1; //all points in the LBS are on fire initially

  }

  z=STS_lazily_burn(LBS, desired_size, v, STS_adj_matrix, on_fire);
                                    //note that second last input is fire_size=desired_size
  if(z){

    s_count[0]++;

    if(s_count[0]==1){

      printf("\n***** {");
      fflush(stdout);

      for(x=0;x<desired_size;x++){

         if(x==desired_size-1){
           printf("%d",LBS[x]);
           fflush(stdout);
         }

         else{
             printf("%d,",LBS[x]);
             fflush(stdout);
         }
```

```
      }//end for

      printf("} is the first discovered lazy burning set for the STS(%d) *****\n",v);
      fflush(stdout);

    }//end if
  }//end if

 /* else{

      printf("{");
      fflush(stdout);

      for(x=0;x<desired_size;x++){

         if(x==desired_size-1){
           printf("%d",LBS[x]);
           fflush(stdout);
         }

         else{
             printf("%d,",LBS[x]);
             fflush(stdout);
         }

      }//end for

      printf("} is NOT a lazy burning set for the STS(%d)\n",v);
      fflush(stdout);

  }//end else
*/

  free(on_fire);

 }//end if

 else if(current_size==0){

       for(x=1; x<=v-(desired_size-current_size)+1; x++){ //dont need to loop all the way up to
                                                          //v since t-subsets have elements
          LBS[0]=x; //current size is now 1               //listed in increasing order.
          STS_build_lazy_burning_sets(LBS, 1, desired_size, v, STS_adj_matrix, t_count, s_count);

       }
 }

 else{ //x starts at LBS[current_size-1]+1 since lazy burning sets must have points
       //listed in increasing order
    for(x=LBS[current_size-1]+1; x<=v-(desired_size-current_size)+1; x++){

       LBS[current_size]=x; //current size has increased by 1
       STS_build_lazy_burning_sets(LBS, current_size+1, desired_size, v, STS_adj_matrix,
                                   t_count, s_count);
```

```
        }
     }
}



int STS_lazily_burn(int *LBS, int size_of_LBS, int v, int *STS_adj_matrix, char *on_fire){

    //returns 1 if LBS was indeed a valid lazy burning set, 0 otherwise.
    //only use of LBS and desired_size=size_of_LBS as inputs is to print
    //them out when LBS is descovered to indeed be a lazy burning set.
    //entry in position i-1 of on_fire is 1 iff point i is on fore, 0 otherwise.

    int x,y,z;

    int fire_size=size_of_LBS;

    char *newly_burned_vertices;

    if((newly_burned_vertices = (char*) malloc(v*sizeof(char)))==NULL){
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
    }

    int was_there_change=1;

    while(was_there_change){

         was_there_change=0;

         for(x=0; x<v; x++)
            newly_burned_vertices[x]=0;

         for(x=0;x<v-1;x++){
            for(y=x+1;y<v;y++){

               if(on_fire[x]==1 && on_fire[y]==1){//must spread fire to vertex z where {x,y,z} is a block

                 z=STS_adj_matrix_what_is(STS_adj_matrix, v, x+1, y+1);//x & y are in {0,1,...,v-1} so input
                                       //x+1 & y+1. note that a different x,y pair might give us the same z!
                 if(on_fire[z-1]==0){//z is in {1,2,...,v}, so here input z-1 to on_fire[], NOT z

                   newly_burned_vertices[z-1]=1; //CANNOT set on_fire[z-1] to 1 here bc then this new
                   was_there_change=1;           //fire would propogate to other vertices during this loop

                 }//end if
               }//end if
            }//end inner for
         }//end outer for

         for(x=0; x<v; x++){

            if(newly_burned_vertices[x]==1){ //dont need && on_fire[x]==0
              on_fire[x]=1;
              fire_size++;//be VERY CAREFUL about when/how we incriment fire_size!!! if one vertex
```

```
            }             //due to its presence in multiple blocks is lit on fire then we only want
                          //to incriment fire_size by 1. for example, {x,y,z} and {u,v,z} where
         }                //x,y,u,v are all on fire... then z gets set on fire due to 2 different
                          //blocks, but fire_size only increased by 1
         if(fire_size==v)
           break;

    }//end while

    free(newly_burned_vertices);

    if(fire_size==v)
      return 1;

    else return 0;

}



void convert_STS_to_general_adj_matrix(int v, int B, int *blocks, char *general_adj_matrix){

     //given the blocks of an STS, creates the ‘‘general adjacency matrix’’ used in
     //general_burn_hypergraph and stores it in general_adj_matrix. general_adj_matrix
     //must already be a B row x v column matrix of all 0’s adj_matrix is a (B row) x (v column)
     //matrix where entry at row i, column j is a 1 iff point j is in block i, 0 otherwise
     //i.e. rows are blocks, columns are points

     int x,y,z;

     for(x=0; x<B; x++){ //traverse the blocks

        general_adj_matrix_set_bit(v, B, general_adj_matrix, x+1, blocks[3*x], 1);
        general_adj_matrix_set_bit(v, B, general_adj_matrix, x+1, blocks[3*x +1], 1);
        general_adj_matrix_set_bit(v, B, general_adj_matrix, x+1, blocks[3*x +2], 1);

     }

     printf("general adjacency matrix of the STS(%d) is:\n\n", v);
     fflush(stdout);

     for(x=1;x<=B;x++){

        for(y=1;y<=v;y++){

           printf("%d", general_adj_matrix_what_is(v,B,general_adj_matrix,x,y));
           fflush(stdout);
        }

        printf("\n");
        fflush(stdout);
     }

     printf("\n\n");
     fflush(stdout);
```

```
}



char general_adj_matrix_what_is(int v, int B, char *adj_matrix, int i, int j){

     //adj_matrix is a (B row) x (v column) matrix where entry at
     //row i, column j is a 1 iff point j is in block i, 0 otherwise
     //i.e. rows are blocks, columns are points

     if(i<1 || i>B || j<1 || j>v || v<1 || B<1){ //i is the block #, j is the index of the point
       printf("error, see line %d", __LINE__);
       exit(1);
     }

     return adj_matrix[(i-1)*v + j - 1];

}



void general_adj_matrix_set_bit(int v, int B, char *adj_matrix, int i, int j, char x){

     //adj_matrix is a (B row) x (v column) matrix where entry at
     //row i, column j is a 1 iff point j is in block i, 0 otherwise
     //i.e. rows are blocks, columns are points

     if(i<1 || i>B || j<1 || j>v || v<1 || B<1){ //i is the block #, j is the index of the point
       printf("error, see line %d", __LINE__);
       exit(1);
     }

     if(x!=0 && x!=1){
       printf("error, see line %d", __LINE__);
       exit(1);
     }

     adj_matrix[(i-1)*v + j - 1]=x;

}



void general_burning_number(int v, int B, char *adj_matrix){

     //adj_matrix is a (B row) x (v column) matrix where entry at
     //row i, column j is a 1 iff point j is in block i, 0 otherwise
     //i.e. rows are blocks, columns are points

     int x;

 char *on_fire; //keeps track of which vertices are currently burned...
                  //burned_vertices[x-1]=1 iff vertex x is burned, =0 otherwise
     if((on_fire = (char*) malloc(v*sizeof(char)))==NULL){
```

```
    printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
    exit(1);
  }

  int *burn_seq; //lists the vertices of the hypergraph in the
                 //order that the player wishes to burn them
if((burn_seq = (int*) malloc(v*sizeof(int)))==NULL){//a burning seq. can't be longer than v
    printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
    exit(1);
  }

  int *burn_num_tally;

  if((burn_num_tally = (int*) malloc(v*sizeof(int)))==NULL){
    printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
    exit(1);
  }

  for(x=0;x<v;x++){
     on_fire[x]=0;
     burn_seq[x]=0;
     burn_num_tally[x]=0;
  }

  int round_count=0;
  int fire_size=0;

  general_arsonist_move(v, B, adj_matrix, round_count, burn_seq,
                        fire_size, on_fire, burn_num_tally);

  printf("\n");
  fflush(stdout);

  ////////////////////////////////////////////////

  for(x=0;x<v;x++){

     if(burn_num_tally[x]!=0){

       printf("b(H) = %d\n", x+1);
       fflush(stdout);

       break;

     }
  }

  ////////////////////////////////////////////////

   //un-comment this for more output!

  for(x=0;x<v;x++){
     printf("there were %d burning sequences of length %d\n", burn_num_tally[x], x+1);
     fflush(stdout);
  }
```

```
}


void general_arsonist_move(int v, int B, char *adj_matrix, int round_count, int *burning_seq,
                           int fire_size, char *on_fire, int *burning_num_tally){

     //v is the number of points in the hypergraph. adj_matrix holds info about the blocks of the
     //hypergraph and allows for faster look-ups than just using the blocks themselves. round_count
     //is the number of moves the arsonist has made so far i.e. the current length of the burning
     //sequence. burning_seq is an array of length v that holds the burning sequence, x_1,x_2,...
     //where each x_i is a vetex of the hypergraph. if currently the burning sequence is of length k,
     //then x_{k+1}=0. fire_size is the number of vertices that are currently burning. on_fire is an
     //array of length v where the jth entry is a 1 if point j is currently on fire, 0 otherwise.
     //burning_num_tally records the # of valid burning sequences of length k in position k-1 of
     //the array.

     int x;

     int arsonist_point=0; //delclared in each iteration of this function

     while(arsonist_point<v){ //arsonist can choose points 1,2,..., or v

          arsonist_point++;

          if(on_fire[arsonist_point-1]==0){ //arsonist cannot choose a vertex that is already on fire

            on_fire[arsonist_point-1]=1; //arsonist sets this point on fire as their move this round
            round_count++;

            //potentially add in something here that skips over testing any
            //seqs that are longer than the current shortest known seq

            burning_seq[round_count-1]=arsonist_point; //this point gets added to the burning sequence
            fire_size++; //asonist burned 1 unburned vertex so the fire size increased by 1

            if(fire_size<v) //if the hypergraph is not yet fully burned
              general_spread_fire(v, B, adj_matrix, round_count, burning_seq, fire_size,
                                  on_fire, burning_num_tally);

            else{ //the hypergraph is fully burned... and the vertex
                  //just chosen was the last unburned vertex!
                burning_num_tally[round_count-1]++;

                if(burning_num_tally[round_count-1]==1){ //if this is the first burning
                                                         //sequence discovered of this length
                  printf("first discovered burning sequence of length %d is:\n", round_count);
                  fflush(stdout);

                  for(x=0;x<round_count;x++){
                     printf("%d ", burning_seq[x]);
                     fflush(stdout);
                  }
```

```
                printf("\nThere were no redundant sources\n");
                fflush(stdout);

              }



          }//end else

          on_fire[arsonist_point-1]=0; //this function is semi-recursive... it calls
          burning_seq[round_count-1]=0;//general_spread_fire, which calls STS_arsonist_move,
          fire_size--;                 //etc. in order to go through all valid burning
          round_count--;               //sequences. once this iteration of STS_arsonist_move
                                       //gets to this point, all of the sequences that start
        } //end if                     //with the first (current) round_count entries of the
                                       //(current) burning_seq will have been tested... so
    }//end while                       //change the (current) most recent entry in burning_seq,
                                       //and test all the ways to finish the sequence
}



void general_spread_fire(int v, int B, char *adj_matrix, int round_count, int *burning_seq,
                         int fire_size, char *on_fire, int *burning_num_tally){

    int x,y,z;

    int spread_to_x;

    char *current_fire_snapshot;

    if((current_fire_snapshot = (char*) malloc(v*sizeof(char)))==NULL){
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
    }

    char *newly_burned_vertices;

    if((newly_burned_vertices = (char*) malloc(v*sizeof(char)))==NULL){
      printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
      exit(1);
    }

    for(x=0; x<v; x++){
       current_fire_snapshot[x]=on_fire[x];
       newly_burned_vertices[x]=0;
    }

    for(x=1;x<=v;x++){ //check each point x

       if(on_fire[x-1]==0){//if x is not yet burned, check if the fire should spread to x this round

         for(y=1;y<=B;y++){ //check each block y
                                                                  //if point x is in block y, then check
            if(general_adj_matrix_what_is(v, B, adj_matrix, y, x)){//if fire spreads to point x due to
```

```
                                                                  //other burning points in block y.
          spread_to_x=1; //this variable determines if fire spreads to point x due to block y

          for(z=1;z<=v;z++){ //look at other points z in block y...
                            //check if x is the only non-burned point in block y
             if(z!=x && general_adj_matrix_what_is(v, B, adj_matrix, y, z) && on_fire[z-1]==0){
                //if z is a point other than x in block y that is not on fire

                spread_to_x=0; //found another un-burned point z different from x in
                break;         //block y. thus, fire does not spread to x due to block
                               //y... still need to check other blocks containing x though!
             }//end if         //note that blocks of size 1 are prohibited.
          }//end for

          if(spread_to_x){

            newly_burned_vertices[x-1]=1;
            break;  //breaks out of for loop that checks if point x is in block y, because we
                    //dont need to check if these blocks would cause fire to spread to x... it
          }//end if //already has! (due to the current block)
        }//end if
      }//end for
    }//end if
 }//end for

 for(x=0; x<v; x++){

   if(newly_burned_vertices[x]==1){ //dont need && on_fire[x]==0
     on_fire[x]=1;
     fire_size++;//be VERY CAREFUL about when/how we incriment fire_size!!! if one vertex is
   }             //lit on fire dure to its presence in multiple blocks then we only want to
                 //incriment fire_size by 1. for example, {x,y,z} and {u,v,z} where x,y,u,v
}                //are all on fire... then z gets set on fire due to 2 different blocks,
                 //but fire_size only increased by 1
if(fire_size<v)//only continue with arsonists move if the hypergraph is not yet fully burned
  general_arsonist_move(v, B, adj_matrix, round_count, burning_seq, fire_size,
                        on_fire, burning_num_tally);

else{ //the hypergraph is fully burned after the most recent spread-fire step...
      //remember in this case the arsonist should pick a redundant source!
      //but there is no need to call arsonist_move() again
     if(burning_num_tally[round_count]==0){//if this is the first burning sequence discovered of
                                  //length round_count+1 (including redundant source at the end)
        printf("first discovered burning sequence of length %d is:\n", round_count+1);
        fflush(stdout);        //NOT round_count (like it used to be in earlier version of code)

        for(x=0;x<v;x++){                //could replace v with round_count if we wanted
           printf("%d ", burning_seq[x]); //to not print the 0's at the end of burning_seq.
           fflush(stdout);               //burning_seq should have round_count many nonzero
        }                                //entries at the start, followed by v-round_count 0's

        printf(" plus any one of the following redundant sources at the end: ");
        fflush(stdout);

        for(x=0;x<v;x++){
```

```
            if(newly_burned_vertices[x]==1){

              printf("%d ", x+1);
              fflush(stdout);

            }
          }

        }//end if

         for(x=0;x<v;x++){//we incriment burning_num_tally[round_count] once for each possible redundant
                          //source at the end, since each of them corresponds to a distinct burning seq
             if(newly_burned_vertices[x]==1)
                burning_num_tally[round_count]++;//NOT burning_num_tally[round_count-1]
                                  //(like it used to be in earlier version of code)
         } //we incriment burning_num_tally[round_count] by the size of newly_burned_vertices[]

    }//end else
                                          //un-spread the fire. note that we dont have to change
    free(newly_burned_vertices);          //fire_size back to the value it started at since
                                          //fire_sizewas passed by value into this function i.e.
    for(x=0; x<v; x++)                    //the STS_arsonist_move that called this function did not
       on_fire[x]=current_fire_snapshot[x]; //have its version of fire_size changed. We DO need to
                                          //revert the array on_fire[] back to the original since
    free(current_fire_snapshot);          //it was a pointer to an array i.e. this function
                                          //altered the same on_fire array that is used in the
}                                         //STS_arsonist_move function that called this function.



void general_lazy_burning_number(int v, int B, char *adj_matrix){

     //adj_matrix is a (B row) x (v column) matrix where entry at row i, column j is a 1
     //iff point j is in block i, 0 otherwise i.e. rows are blocks, columns are points

     int t,x;       //t_count will hold the # of t-subsets that have been tested... once we have
                    //tested them all, we will print t_count[0]... if it isnt v choose t then we
     int *t_count;  //know there is a mistake in the code! s_count is the # of successful tests
     int *s_count;  //i.e. the # of lazy burning sets for the STS(v)

     if((t_count = (int*) malloc(1*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     t_count[0]=0;

     if((s_count = (int*) malloc(1*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     s_count[0]=0;

     int *LBS;//the lazy burning set. It is of size v. unused entries will be set to 0. ex. if we
```

```
            //are testing all 4-subsets of v=10 points then the last 6 entries of LBS will be 0.
     if((LBS = (int*) malloc(v*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     for(x=0; x<v; x++)
        LBS[x]=0;

     for(t=1; t<=v; t++){ //find all lazy burning sets of size t

        general_build_lazy_burning_sets(LBS, 0, t, v, B, adj_matrix, t_count, s_count);

        printf("\n(%d choose %d) = %d. There are %d lazy burning sets of size %d\n", v, t,
               t_count[0], s_count[0], t);
        fflush(stdout);

        if(s_count[0]!=0){

          printf("b_L(H) = %d", t);
          fflush(stdout);
          break;

        }

        t_count[0]=0;
        s_count[0]=0;

        for(x=0; x<v; x++)
           LBS[x]=0;

     }//end for
}



void general_build_lazy_burning_sets(int *LBS, int current_size, int desired_size, int v, int B,
                                     char *adj_matrix, int *t_count, int *s_count){

     // recursively builds all unordered subsets of {1,2,...,v} of size desired_size
     // and calls lazily_burn() for each to see if they are indeed lazy burning sets.

     //current_size is the # of cells of LBS that have been assigned a nonzero value so far
     //remember that entries in the blocks f the hypergraph must range from 1 to v

     //t_count is the # of t-subsets of {1,2,...,v}... should be v choose t (where t=desired_size)
     //s_count is the # of lazy burning sets of size desired_size for the hypergraph

     int x,y,z;

     if(current_size==desired_size){ //potential lazy burning set is done being built

       t_count[0]++;

       char *on_fire; //keeps track of which vertices are currently burned...
```

```
              //on_fire[x-1]=1 iff vertex x is burned, =0 otherwise
 if((on_fire = (char*) malloc(v*sizeof(char)))==NULL){
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
 }

 for(x=0;x<v;x++)
    on_fire[x]=0;

 for(x=0;x<desired_size;x++){

    y=LBS[x]; //y is in {1,2,...,v}. this point is initially set on fire as part of the LBS
    on_fire[y-1]=1; //all points in the LBS are on fire initially

 }

 z=general_lazily_burn(LBS, desired_size, v, B, adj_matrix, on_fire);
                      //note that second last input is fire_size=desired_size
 if(z){

   s_count[0]++;

   if(s_count[0]==1){

     printf("\n***** {");
     fflush(stdout);

     for(x=0;x<desired_size;x++){

        if(x==desired_size-1){
          printf("%d",LBS[x]);
          fflush(stdout);
        }

        else{
            printf("%d,",LBS[x]);
            fflush(stdout);
        }

     }//end for

     printf("} is the first discovered lazy burning set for the hypergraph *****\n");
     fflush(stdout);

   }//end if

 }//end if

/* else{

     printf("{");
     fflush(stdout);

     for(x=0;x<desired_size;x++){
```

```
            if(x==desired_size-1){
              printf("%d",LBS[x]);
              fflush(stdout);
            }

            else{
                printf("%d,",LBS[x]);
                fflush(stdout);
            }

          }//end for

          printf("} is NOT a lazy burning set for the hypergraph\n");
          fflush(stdout);

       }//end else
    */

       free(on_fire);

     }//end if

     else if(current_size==0){

            for(x=1; x<=v-(desired_size-current_size)+1; x++){//dont need to loop all the way up
                                                              //to v since t-subsets have elements
               LBS[0]=x; //current size is now 1              //listed in increasing order.
               general_build_lazy_burning_sets(LBS, 1, desired_size, v, B, adj_matrix,
                                               t_count, s_count);
            }
     }

     else{ //x starts at LBS[current_size-1]+1 since lazy burning sets must have points
           //listed in increasing order
         for(x=LBS[current_size-1]+1; x<=v-(desired_size-current_size)+1; x++){

            LBS[current_size]=x; //current size has increased by 1
            general_build_lazy_burning_sets(LBS, current_size+1, desired_size, v, B,
                                            adj_matrix, t_count, s_count);
         }
     }
}



int general_lazily_burn(int *LBS, int size_of_LBS, int v, int B, char *adj_matrix, char *on_fire){

    //returns 1 if LBS was indeed a valid lazy burning set, 0 otherwise.
    //only use of LBS and desired_size=size_of_LBS as inputs is to print
    //them out when LBS is descovered to indeed be a lazy burning set.
    //entry in position i-1 of on_fire is 1 iff point i is on fire, 0 otherwise.

    int x,y,z;

    int spread_to_x;
```

```
int fire_size=size_of_LBS;

char *newly_burned_vertices;

if((newly_burned_vertices = (char*) malloc(v*sizeof(char)))==NULL){
  printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
  exit(1);
}

int was_there_change=1;

while(was_there_change){

     was_there_change=0;

     for(x=0; x<v; x++)
        newly_burned_vertices[x]=0;

     for(x=1;x<=v;x++){ //check each point x

        if(on_fire[x-1]==0){   //if x is not yet burned, check if the
                               //fire should spread to x this round
          for(y=1;y<=B;y++){//check each block y

             if(general_adj_matrix_what_is(v, B, adj_matrix, y, x)){
                                 //if point x is in block y, then check if fire spreads
                                 //to point x due to other burning points in block y
               spread_to_x=1;//this variable determines if fire spreads to point x due to block y

               for(z=1;z<=v;z++){ //look at other points z in block y...
                                 //check if x is the only non-burned point in block y
                  if(z!=x && general_adj_matrix_what_is(v, B, adj_matrix, y, z) && on_fire[z-1]==0){
                    //if z is a point other than x in block y that is not on fire
                    spread_to_x=0;//found another un-burned point z different from x in
                    break;        //block y. thus, fire does not spread to x due to block
                                  //y... still need to check other blocks containing x though!
                  }//end if       //note that blocks of size 1 are prohibited.
               }//end for

               if(spread_to_x){

                 was_there_change=1;
                 newly_burned_vertices[x-1]=1;
                 break;  //breaks out of for loop that checks if point x is in block y, because
                         //we dont need to check if these blocks would cause fire to spread to
               }//end if //x... it already has! (due to the current block)
             }//end if
          }//end for
        }//end if
     }//end for

     for(x=0; x<v; x++){

        if(newly_burned_vertices[x]==1){ //dont need && on_fire[x]==0
```

```
            on_fire[x]=1;
            fire_size++; //be VERY CAREFUL about when/how we incriment fire_size!!! if one vertex
          }              //is lit on fire due to its presence in multiple blocks then we only want
                         //to inriment fire_size by 1. for example, {x,y,z} and {u,v,z} where
        }                //x,y,u,v are all on fire... then z gets set on fire due to 2 different
                         //blocks, but fire_size only increased by 1
        if(fire_size==v)
          break;

    }//end while

    free(newly_burned_vertices);

    if(fire_size==v)
      return 1;

    else return 0;

}



void convert_blocks_to_adj_matrix(int v, int B, int zero_or_one, int total_size_of_blocks,
                                  int *blocks, char *general_adj_matrix){

//v is the number of points/vertices, B is the number of blocks/hyperedges. zero_or_one is passed
//into this function as zero if the points are denoted {0,1,...,v-1}. If the points are denoted
//{1,2,...,v}, then pass zero_or_one in as 1. blocks[] represents the blocks/hyperedges of the
//design/hypergraph. the elemets of a hyperedge are listed next to each other, and different
//hyperedges are separated by a -1 in blocks[]. For example, for the STS(7),
//blocks[]={1,2,4,-1,2,3,5,-1,3,4,6,-1,4,5,7,-1,5,6,1,-1,6,7,2,-1,7,1,3}. there is no -1 at the
//very end. total_size_of_blocks is the number of entries in blocks[] that are not -1. Also,
//general_adj_matrix must be initialized to a B row x v column array of all zeros before being
//passed into this function.

//This function will fill general_adj_matrix with some 1's such that row i, column j is a 1 iff
//point j is in block i. That is, rows are blocks and columns are points. general_adj_matrix can
//then be input into the function general_burn_hypergraph.

     if(zero_or_one!=0 && zero_or_one!=1){
       printf("error, see line %d", __LINE__);
       exit(1);
     }

     if(B<v || v<=0 || B<=0){
       printf("error, see line %d", __LINE__);
       exit(1);
     }

     int block_counter=1;//blocks are listed left to right in blocks[]
     int x,y,z;

     if(zero_or_one==1){

       for(x=1; x<=total_size_of_blocks+B-1; x++){ //size_of_blocks+B-1 is the # of entries
```

```
                                                  //in blocks[] including -1's
          if(blocks[x-1]<0) //if blockks[x-1] is -1 then we have moved on to the next block
            block_counter++;

          else
            general_adj_matrix_set_bit(v, B, general_adj_matrix, block_counter, blocks[x-1], 1);
                                                                  //blocks[x-1] is 1,2,..., or v
        }
      }

      else{ //zero_or_one==0

          for(x=1; x<=total_size_of_blocks+B-1; x++){ //total_size_of_blocks+B-1 is the # of
                                                     //entries in blocks[] including -1's
             if(blocks[x-1]<0) //if blockks[x-1] is -1 then we have moved on to the next block
               block_counter++;

             else
               general_adj_matrix_set_bit(v, B, general_adj_matrix, block_counter, blocks[x-1]+1, 1);
     //blocks[x-1] is 0,1,..., or v-1 but columns of general_adj_matrix are from 1 to v, hence the +1
          }
      }

      if(block_counter!=B){
        printf("error, see line %d", __LINE__);
        exit(1);
      }

}



void print_adj_matrix(int v, int B, char *adj_matrix){

      int x,y;

      printf("\n\n");
      fflush(stdout);

      for(x=1;x<=B;x++){ //traverse rows/blocks
         for(y=1;y<=v;y++){ //traverse columns/points

           printf("%d ",general_adj_matrix_what_is(v, B, adj_matrix, x, y));
           fflush(stdout);

           if(y==v){

             printf("\n");
             fflush(stdout);

           }
         }
      }

      printf("\n\n");
```

```
     fflush(stdout);

}



void double_STS(int v, int *Blocks, int B, int *new_blocks, int new_v, int new_B){

//takes as input the STS(v) whose blocks are stored in array Blocks[] (first block is the first three
//consecutive entries, second block is the next three, etc). creates a new STS using the ‘‘double & add
//one’’ method & stores it in the same way in array new_blocks. new_v and new_B are inputs for the sole
//for the sole purpose of redundancy and returning an error if the input makes no sense. We assume the
//points in an STS(v) are labelled from 1 to v (NOT from 0 to v-1). Blocks contains 3*B entries.

     if(new_v != 2*v+1 || new_B != (new_v*(new_v-1))/6 || B != (v*(v-1))/6){
       printf("\n invalid input, see line %d\n", __LINE__);
       exit(1);
     }

     int i,x,y,z;
     int hold=3*B;

     for(i=0; i<3*B; i++)
        new_blocks[i]=Blocks[i]; //remember, input STS is a subsystem of the new STS

     for(i=1; i<=v; i++){ //for each point k in the original STS, there is a block {k,k',infinity}
                          //in the new STS. we write k':=k+v and infinity:=2*v+1.
        new_blocks[hold]=i;
        new_blocks[hold+1]=i+v;
        new_blocks[hold+2]=new_v;

        hold+=3;

     }//once this for loop ends, hold is the index of the next empty spot in new_blocks[]

     for(i=1; i<=B; i++){                  //for each block {x,y,z} in original STS, must add
                                           //blocks {x',y',z}, {x',y,z'}, and {x,y',z'} to new STS
        x=Blocks[3*(i-1)]; //first entry in i'th block of original STS
        y=Blocks[3*(i-1)+1]; //second entry in i'th block of original STS
        z=Blocks[3*(i-1)+2]; //third entry in i'th block of original STS

        new_blocks[hold]=x+v;    //add block {x',y',z}
        new_blocks[hold+1]=y+v;
        new_blocks[hold+2]=z;

        new_blocks[hold+3]=x+v;  //add block {x',y,z'}
        new_blocks[hold+4]=y;
        new_blocks[hold+5]=z+v;

        new_blocks[hold+6]=x;    //add block {x,y',z'}
        new_blocks[hold+7]=y+v;
        new_blocks[hold+8]=z+v;

        hold+=9;
```

```
     }//last "hold+8" should be last index in new_blocks[]

     if(hold!=3*new_B){ //if everything went right, this hold should end up at 3*new_B
       printf("\n error, see line %d\n", __LINE__);
       exit(1);
     }

}



void fast_lazy_burning_number(int v, int B, char *adj_matrix){
                                       //adj_matrix is a (B row) x (v column) matrix where entry
     int t,x;                          //at row i, column j is a 1 iff point j is in block i, 0
                                       //otherwisei.e. rows are blocks, columns are points
     int *LBS; //the lazy burning set. It is of size v. unused entries will be set to 0. ex. if we
               //are testing all 4-subsets of v=10 points then the last 6 entries of LBS will be 0.
     if((LBS = (int*) malloc(v*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     for(x=0; x<v; x++)
        LBS[x]=0;

     for(t=1; t<=v; t++){ //find all lazy burning sets of size t

        if(fast_build_lazy_burning_sets(LBS, 0, t, v, B, adj_matrix)){

          printf("b_L(H) = %d", t);
          fflush(stdout);
          break;

        }

        else{

            printf("no lazy burning sets of size %d\n", t);
            fflush(stdout);

        }

        for(x=0; x<v; x++)
           LBS[x]=0;

     }//end for

}



int fast_build_lazy_burning_sets(int *LBS, int current_size, int desired_size, int v,
                                 int B, char *adj_matrix){
    // recursively builds all unordered subsets of {1,2,...,v} of size desired_size
    // and calls lazily_burn() for each to see if they are indeed lazy burning sets.
```

```
//current_size is the # of cells of LBS that have been assigned a nonzero value so far
//remember that entries in the blocks of the hypergraph must range from 1 to v

//returns 1 if a lazy burning set was found... many of these funcions are nested inside
//one another and they will all return 1. it is just a way to stop building potential
//lazy burning sets once one has been found to save time

int x,y,z;

if(current_size==desired_size){ //potential lazy burning set is done being built

  char *on_fire; //keeps track of which vertices are currently burned... on_fire[x-1]=1
                 //iff vertex x is burned, =0 otherwise
  if((on_fire = (char*) malloc(v*sizeof(char)))==NULL){
    printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
    exit(1);
  }

  for(x=0;x<v;x++)
     on_fire[x]=0;

  for(x=0;x<desired_size;x++){

     y=LBS[x];//y is in {1,2,...,v}. this point is initially set on fire as part of the LBS
     on_fire[y-1]=1; //all points in the LBS are on fire initially

  }

  if(general_lazily_burn(LBS, desired_size, v, B, adj_matrix, on_fire)){
                 //note that second last input is fire_size=desired_size
    printf("\n***** {");
    fflush(stdout);

    for(x=0;x<desired_size;x++){

       if(x==desired_size-1){
         printf("%d",LBS[x]);
         fflush(stdout);
       }

       else{
           printf("%d,",LBS[x]);
           fflush(stdout);
       }

    }//end for

    printf("} is the first discovered lazy burning set for the hypergraph *****\n");
    fflush(stdout);

    return 1;

  }//end if
```

```
      free(on_fire);

    }//end if

    else if(current_size==0){
                                                          //dont need to loop all the way up to
       for(x=1; x<=v-(desired_size-current_size)+1; x++){//v since t-subsets have elements
                                                          //listed in increasing order.
          LBS[0]=x;//current size is now 1
          if(fast_build_lazy_burning_sets(LBS, 1, desired_size, v, B, adj_matrix))
            return 1;

       }
    }

    else{ //x starts at LBS[current_size-1]+1 since lazy burning
          //sets must have points listed in increasing order
        for(x=LBS[current_size-1]+1; x<=v-(desired_size-current_size)+1; x++){

          LBS[current_size]=x; //current size has increased by 1
          if(fast_build_lazy_burning_sets(LBS, current_size+1, desired_size, v, B, adj_matrix))
            return 1;

        }
    }

    return 0;
}



void fast_burning_number(int v, int B, char *adj_matrix){
                        //not always faster than general_burning_number()
     int t,x;

     int *SEQ; //the lazy burning set. It is of size v. unused entries will be set to 0. ex. if we
               //are testing all 4-subsets of v=10 points then the last 6 entries of LBS will be 0.
     if((SEQ = (int*) malloc(v*sizeof(int)))==NULL){
       printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
       exit(1);
     }

     for(x=0; x<v; x++)
        SEQ[x]=0;

     for(t=1; t<=v; t++){ //find all potential burning sequences of length t

        if(fast_build_burning_seq(SEQ, 0, t, v, B, adj_matrix)){

          printf("b(H) = %d", t);
          fflush(stdout);
          break;

        }
```

```
        else{

            printf("no burning sequences of length %d\n", t);
            fflush(stdout);

        }

        for(x=0; x<v; x++)
           SEQ[x]=0;

     }//end for

}



int fast_build_burning_seq(int *SEQ, int current_size, int desired_size, int v,
                           int B, char *adj_matrix){

    // recursively builds all ORDERED subsets of {1,2,...,v} of size desired_size
    // and calls fast_burn() for each to see if they are indeed burning sequences.

    //current_size is the # of cells of SEQ that have been assigned a nonzero value so far
    //remember that entries in the blocks of the hypergraph must range from 1 to v

    //returns 1 if a burning sequence was found... many of these funcions are nested inside
    //one another and they will all return 1. it is just a way to stop building potential
    //burning sequences once one has been found to save time

    int x,y,already_used;

    if(current_size==desired_size){ //potential burning sequence is done being built

      if(fast_test_burn_seq(SEQ, desired_size, v, B, adj_matrix)){

        printf("\n***** (");
        fflush(stdout);

        for(x=0;x<desired_size;x++){

           if(x==desired_size-1){
             printf("%d",SEQ[x]);
             fflush(stdout);
           }

           else{
               printf("%d,",SEQ[x]);
               fflush(stdout);
           }

        }//end for

        printf(") is the first discovered burning sequence for the hypergraph *****\n");
        fflush(stdout);
```

```
        return 1;

      }//end if

    }//end if

    else if(current_size==0){

           for(x=1; x<=v; x++){ //unlike the lazy version of this function, we have to loop all
                                //the way up to v since a burning sequence could start with v
              SEQ[0]=x; //current size is now 1
              if(fast_build_burning_seq(SEQ, 1, desired_size, v, B, adj_matrix))
                return 1;

           }
    }

    else{

        for(x=1; x<=v; x++){//again, unlike lazy version of this function we must loop from 1 to v

           already_used=0;

           for(y=0;y<current_size;y++){

              if(x==SEQ[y]){
                already_used=1;
                break;
              }

           }//end for

           if(already_used==0){//prevents SEQ from having multiple of the same number

             SEQ[current_size]=x; //current size has increased by 1

             if(fast_build_burning_seq(SEQ, current_size+1, desired_size, v, B, adj_matrix))
               return 1;

           }//end if

        }//end for
    }

    return 0;

}



int fast_test_burn_seq(int *SEQ, int seq_length, int v, int B, char *adj_matrix){

    int x,y,z,w,X; //X is just for if we want to print the progress of the fire spreading

    int spread_to_x;
```

```
 char *on_fire;

 if((on_fire = (char*) malloc(v*sizeof(char)))==NULL){
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
 }

 char *newly_burned_vertices;

 if((newly_burned_vertices = (char*) malloc(v*sizeof(char)))==NULL){
   printf("\n\nAllocation of memory failed...line %d\n", __LINE__);
   exit(1);
 }

 for(x=0; x<v; x++){
    on_fire[x]=0;
    newly_burned_vertices[x]=0;
 }

on_fire[SEQ[0]-1]=1;//burn the first source

 for(w=2; w<=seq_length; w++){//"skip" the first round to save time since we know
                              //all that happens is the first source catches fire
    if(on_fire[SEQ[w-1]-1]==1 ){   //if wth entry in the sequence in already on fire
      free(on_fire);               //then SEQ is not a valid burning sequence,
      free(newly_burned_vertices); //because we are supposed to burn wth entry as a
      return 0;                    //source but it is already on fire
    }

    //below: spread the fire once, then burn the source

    for(x=0; x<v; x++)
       newly_burned_vertices[x]=0;

    for(x=1;x<=v;x++){ //check each point x

       if(on_fire[x-1]==0){ //if x is not yet burned, check if the
                            //fire should spread to x this round
         for(y=1;y<=B;y++){ //check each block y
                                                              //if point x is in block y, then
            if(general_adj_matrix_what_is(v, B, adj_matrix, y, x)){//check if fire spreads to point
                                                    //x due to other burning points in block y
              spread_to_x=1; //this variable determines if fire spreads to point x due to block y

              for(z=1;z<=v;z++){ //look at other points z in block y... check if
                                 //x is the only non-burned point in block y

                 if(z!=x && general_adj_matrix_what_is(v, B, adj_matrix, y, z) && on_fire[z-1]==0){
                    //if z is a point other than x in block y that is not on fire
                   spread_to_x=0; //found another un-burned point z different from x in
                   break;         //block y. thus, fire does not spread to x due to block
                                  //y... still need to check other blocks containing x though!
                 }//end if        //note that blocks of size 1 are prohibited.
              }//end for
```

```
                if(spread_to_x){

                  newly_burned_vertices[x-1]=1;
                  break;  //breaks out of for loop that checks if point x is in block y, because
                          //we dont need to check if these blocks would cause fire to spread to
                }//end if //x... it already has! (due to the current block)
              }//end if
            }//end for
          }//end if
        }//end for

        for(x=0; x<v; x++){

          if(newly_burned_vertices[x]==1) //dont need && on_fire[x]==0
            on_fire[x]=1;

        }

        on_fire[SEQ[w-1]-1]=1; //burn wth source... its ok if it is redundant! i.e. if
                               //it caught fire in the most recent propagation step

       /* printf("\n on_fire after round %d:", w);
        fflush(stdout);

        for(X=0;X<v;X++){

           printf("%d ", on_fire[X]);
           fflush(stdout);

        } */

    }//end for

    for(x=1;x<=v;x++){           //if H is not fully burned after the final round
                                 //then SEQ wasnt a valid burning sequence
        if(on_fire[x-1]==0){
          free(on_fire);
          free(newly_burned_vertices);
          return 0;
        }

    }

    free(on_fire);                //if we get to this point then on_fire must be all 1's
    free(newly_burned_vertices);  //that is, H is fully burned after the final round.
    return 1;

}
```