

# CATCHING THE FIRE ON GRIDS

A Thesis Presented

by

Patricia Fogarty

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements  
for the Degree of Master of Science  
Specializing in Mathematics

May, 2003

Accepted by the Faculty of the Graduate College, The University of Vermont, in partial fulfillment of the requirements for the degree of Master of Science, specializing in Mathematics.

Thesis Examination Committee:

\_\_\_\_\_  
Jeff Dinitz, Ph.D. Co-Advisor

\_\_\_\_\_  
Dan S. Archdeacon, Ph.D. Co-Advisor

\_\_\_\_\_  
Alan Ling, Ph.D.

\_\_\_\_\_  
Jo Ellis-Monaghan, Ph.D. St. Michael's College

\_\_\_\_\_  
Barry Doolan, Ph.D. Chairperson

\_\_\_\_\_  
David Dummit, Ph.D. Special Assistant  
to the Provost  
for Graduate Education

Date: April 3, 2003

# Abstract

Firefighters respond to a fire that breaks out in the wilderness or some other area with the intent of ensuring that a minimum amount of damage is done. We utilize grids to model the area that needs to be protected when a fire ignites at a given vertex. We assume there are  $f$  firefighters available. At each stage a firefighter can protect one gridpoint, permanently, from catching on fire. However, in the meantime, the fire spreads from any point to any adjacent point.

In each of the grids considered in this thesis, we have the same objectives. First, we would like to determine the number of firefighters needed to contain the fire. Second, firefighters do not always respond to a fire immediately. Therefore, we determine whether the firefighters are still able to protect the grid if they arrive at time  $t \geq 0$ . Third, we obtain a strategy that guarantees a minimal number of vertices are burned in a minimal amount of time.

# Acknowledgments

Jeff Dinitz undertook the task of doing research with me these last two semesters. I spent countless hours in his office learning new ideas and presenting my new discoveries. He provided me with inspiration and enthusiasm when it was needed.

Jeff and Dan Archdeacon provided me with the needed support to complete this project. They gave up countless lunch hours to help me explore different research ideas and to go over ideas that I had. Both of you have greatly increased my love of math.

The graduate students in the basement helped me keep my sense of humor throughout this process.

The mathematics and computer science faculty at St. Michael's College helped me develop the necessary math skills to undertake a project like this. Without your support and help, I would not have made it this far.

My family offered me much support throughout this process and put up with my levels of stress. They listened to me while I would talk about the research I was doing even though they didn't quite understand it. They constantly ran upstairs to see small parts of my program running. And they provided much needed breaks when my head was a little too full of mathematics.

Thank you everyone for the source of motivation you've provided throughout this research process.

# Table of Contents

Acknowledgments . . . . .	ii
List of Figures . . . . .	iv
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Definitions . . . . .	1
1.3 Prior Results . . . . .	4
<b>2 Infinite Fires</b>	<b>6</b>
2.1 Introduction . . . . .	6
2.2 A Hall-Type Condition and Infinite Fires . . . . .	6
<b>3 Infinite Rectangular Grids</b>	<b>9</b>
3.1 Introduction . . . . .	9
3.2 Two Firefighters . . . . .	9
3.3 More than One Fire in the Plane . . . . .	16
3.4 JAVA Program for the Plane . . . . .	17
<b>4 Ribbon Grids</b>	<b>18</b>
4.1 Introduction . . . . .	18
4.2 Fires Starting on the First Level . . . . .	18
4.3 Fires Starting Anywhere . . . . .	23
4.4 JAVA Program for Rectangular Grids . . . . .	25
<b>5 The Quarter-Plane</b>	<b>27</b>
5.1 Introduction . . . . .	27
5.2 An Early Firefighter . . . . .	27

<b>6</b>	<b>Fractional Firefighters in the Plane</b>	<b>29</b>
6.1	Introduction . . . . .	29
6.2	One Fractional Firefighter . . . . .	30
<b>7</b>	<b>Hexagonal Grids</b>	<b>32</b>
7.1	Introduction . . . . .	32
7.2	Ribbon Hexagonal Grids . . . . .	33
7.3	Unrestricted Hexagonal Grids . . . . .	34
<b>8</b>	<b>Conclusion</b>	<b>40</b>
	<b>References</b>	<b>41</b>
	<b>Appendices</b>	<b>42</b>
<b>A</b>	<b>Java Source Code: Rectangular Grids</b>	<b>42</b>
<b>B</b>	<b>Java Source Code: Square Grids</b>	<b>55</b>

# List of Figures

1	The Key for Vertices. . . . .	3
2	The Grid is Unprotected through Time 2. . . . .	9
3	The Grid from Time $3k + 1$ until $9k$ for $k = 1$ . . . . .	11
4	The Grid from Time $9k + 1$ until $19k + 1$ for $k = 1$ . . . . .	12
5	The Grid Completely Protected for $k = 1$ . . . . .	13
6	The Pattern of Saving the Grid. . . . .	14
7	Pattern II. . . . .	15
8	Both Patterns of Saving the Grids. . . . .	16
9	How to Protect $n$ fires. . . . .	17
10	The Pattern to Protect the Rectangular Grid. . . . .	21
11	Ways to Protect the Grid. . . . .	23
12	Fire starting at the bottom for $r = 3$ . . . . .	23
13	The $7 \times \infty$ grid for $k = 4$ . . . . .	24
14	The General Pattern of Protecting Rectangular Grids. . . . .	25
15	An Early Firefighter. . . . .	28
16	Protecting the Quarter Plane with an Early Firefighter. . . . .	28
17	Fractional Firefighters. . . . .	29
18	A Hexagonal Grid. . . . .	32
19	The Lines in Hexagonal Grids. . . . .	35
20	Protecting a Hexagonal Grid with Pattern $J$ . . . . .	37
21	Pattern $P$ for Protecting Hexagonal Grids. . . . .	39

# 1 Introduction

## 1.1 Background

Each year fires destroy people's property, burn the wilderness, and take many lives. Firefighters responding to fires must determine where to fight the fire in order to ensure that a minimum amount of damage is done. They try to provide an immediate response to a fire; they organize together to provide the best possible coverage.

Most of the time the firefighters respond as quickly as possible; sometimes they get to the fire right away, but if the fire is in a remote area it may take several days before the firefighters can get to the fire. Firefighters attempt to construct fire walls which the fire cannot cross. Sometimes natural barriers help isolate the fire to a certain region. If the fire starts near a river, the firefighters can drive the fire towards the river so that they have less land to protect. Some other natural barriers that help minimize the firefighters job includes roads and lakes.

Firefighters try and put out all of the fire in a section of the land. However, sometimes a small ember of the fire might remain causing the fire to burn a little longer in that area and spread.

## 1.2 Definitions

Graphs provide a convenient way to model the spread of fires. Throughout this paper, we utilize the graph-theoretic notation of [8].

A *graph*  $G$  consists of a *vertex set*  $V(G) = \{v_1, v_2, \dots, v_n\}$  and an *edge set*  $E(G) = \{e_1, e_2, \dots, e_m\}$ , which is an unordered pair of vertices  $\{v_i, v_j\}$  called its *endpoints*. If  $v_i, v_j \in E(G)$  then we call  $v_i$  and  $v_j$  *adjacent vertices*. Let  $A$  be a subset of vertices where  $N(A)$  is the set of all vertices adjacent to any vertex in  $A$ . In our basic model, at each time period, the fire spreads from vertex  $v$  to all of its adjacent

vertices.

A firefighter *protects* a vertex if they are placed at a vertex that is in danger of going on fire. Assume that we have  $f$  firefighters to protect the grid. Once the fire starts, these  $f$  firefighters protect exactly  $f$  vertices. Once these vertices have been protected, they remain protected throughout the duration of the fire. At each time interval, these  $f$  firefighters change their position and protect another set of  $f$  vertices. They continue doing this until the fire can no longer spread to any other vertices.

The firefighters attempt to build a *fire wall* of protected vertices, a barrier from which the fire cannot escape. We say that the firefighters *surround* the fire where the fire is entirely within the fire walls built.

We define several possible states for the vertices. A *burned vertex* is a vertex that is on fire and stays on fire. The *original burned vertex* is the vertex that was the source of the fire. We use the terminology from Wang and Moeller [7] for a *protected vertex* and a *saved vertex*. A *protected vertex* is a vertex where a firefighter was present. A *saved vertex* is a vertex such that when the fire has been surrounded, the vertex did not burn in the end. An *unprotected vertex* is a vertex such that when the fire can still spread, the vertex is not burned or protected. Figure 1 shows how we will picture these types of vertices in rectangular grids.

A *path*  $P_n$  is a simple graph whose vertices can be ordered such that two vertices are adjacent if and only if they are consecutive in the list. Let  $P_\infty$  denote the two-way infinite path. Let  $v_i$  and  $v_j$  be two vertices in a graph  $G$ ; the *distance* between  $v_i$  and  $v_j$  is the length of the shortest path between them and is denoted  $d(v_i, v_j)$ . Paths model the spread of a fire from the original burned vertex to another vertex of distance  $k$ . A *cycle*  $C_n$  is a path with no repeated edges but with exactly one repeated vertex:  $v_n$  is adjacent to  $v_1$ .

## KEY

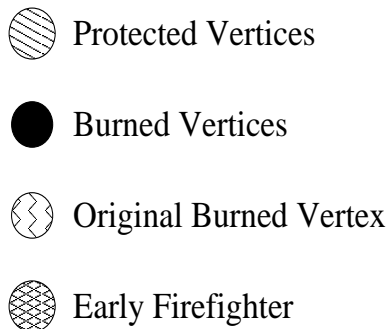


Figure 1: The Key for Vertices.

Three types of rectangular grids will be utilized to model the spread of fires. Using the notation of [8],  $P_n \square P_n$  denotes the Cartesian product of two paths of length  $n$ . We will call this the  $n \times n$  rectangular grid and will model it on the plane by letting the vertices be  $\{(x, y) \in \mathbb{N} \times \mathbb{N} \mid 1 \leq x, y \leq n\}$  with  $(x_1, y_1)$  adjacent to  $(x_2, y_2)$  if  $|x_2 - x_1| + |y_2 - y_1| = 1$ . The infinite rectangular grid ( $P_\infty \square P_\infty$ ) and the ribbon grid ( $P_n \square P_\infty$ ) are defined similarly. The vertices in the infinite grid are the points of  $\mathbb{Z} \times \mathbb{Z}$  while the vertices in the ribbon grid are the elements of  $\mathbb{Z} \times \{1, 2, \dots, n\}$ .

*Hexagonal grids* can model the fire spreading in a different pattern than rectangular grids. The original burned vertex in a hexagonal grid has six neighbors. We will study fires on these grids in Section 7.

We model our firefighting as follows. The fire begins at a vertex at time 0. At time 1, we place our firefighter(s) and then the fire spreads to each unprotected vertex at distance 1. In general, at time  $t$ , we first place our firefighter(s), then the fire spreads to all unprotected vertices adjacent to those on fire at time  $t - 1$ . Throughout this thesis, we assume that the number of firefighters available at each time interval is a constant  $f$ .

When protecting the grid from a fire, there may be several patterns which produce the same number of burned vertices. We say that pattern  $A$  is *equivalent*

to pattern  $B$  if it takes the same amount of time to surround the fire and the same number of vertices burn.

### 1.3 Prior Results

Wang and Moeller [7] determined the number of firefighters necessary to surround the fire in two dimensional grid graphs. They developed algorithms to save or protect the maximum number of vertices in finite grids. They determined the minimum number of time intervals it takes to protect the two-dimensional grid by establishing that it takes a minimum of eight time intervals where only eighteen vertices are burned.

A problem similar to firefighting is the containment of viruses in a network. Finbow and Hartnell [1] studied the reliability of communication networks. These are similar to a fire except that in this model, all vertices within distance two are destroyed instead of only the adjacent vertices. A vertex is within distance two if there is some path with two or fewer edges joining them or they are the same vertex. They assume that a random subset of vertices of a graph  $G$  are attacked, and a vertex  $v$  survives if there is no vertex  $u$  within distance two of  $v$ . They would like to design a connected graph with  $p$  vertices such that when a random subset of the vertices are attacked, the minimum number of vertices are destroyed. They then determine the optimal structure for such a graph.

Finbow, Hartnell et al. [2] researched viruses on a network with the aim of minimizing the damage incurred. They assume that viruses start at  $k$  vertices and there are  $d$  defenders, which defend  $d$  unprotected vertices. They first examine the case when a virus erupts at random. They want to determine the optimal graphs given  $p$  vertices,  $k$  viruses, and  $d$  defenders. An optimal graph  $G$  is a graph such that the number of infected vertices is minimum over all the graphs with the same number

of vertices. They also look at when the virus starts at a certain set of vertices which maximizes the damage. Given  $k$  vertices, they want to find which connected graphs minimize the damage.

Hartnell [4] originally proposed the problem of firefighting in terms of resistance movements. He wanted to know how to establish an underground communications network which minimizes the effects of treachery by a member of the network which is followed by the betrayal of other members. The resistance movement is modeled by a graph where the vertices represent the individual members and the edges represent the communication between members of the network.

Hartnell and Li [5] looked at the case of firefighting on trees. A fire starts at the root of a tree and one firefighter then protects any unprotected node. They study how effective the greedy algorithm is when there is one fire in the tree. They assume that given a tree  $T$  the fire starts at vertex  $v$  which is the root of the tree. They show that the greedy algorithm saves more than half of the vertices any other algorithm saves.

Other similar problems can be seen in [3] and [6].

## 2 Infinite Fires

### 2.1 Introduction

The job of firefighters is to surround the fire in any given situation. This may take a large number of firefighters and many time intervals. Each situation that a firefighter deals with is different, but their goal is the same, to surround and put out the fire.

There are many conditions that affect the spread of a fire; these include the number of firefighters, the type and dimension of the grid, and the number of time intervals before the firefighters start protecting the grid. We consider an *infinite fire* to be a fire that can never be surrounded when  $f$  firefighters start protecting the grid at time  $k$ . Sometimes a fire cannot be surrounded because there are not enough firefighters. Other times, the firefighters might not have come soon enough to put out the fire and had they arrived earlier they may have been able to put it out.

### 2.2 A Hall-Type Condition and Infinite Fires

In this section we discuss a Hall-Type Condition that will give information about the possible containment of a fire. We first develop the notation to be used in this section.

We assume that exactly one vertex catches on fire at time  $t = 0$ . Let  $D_k$  denote the set of vertices of distance  $k$  from the original burned vertex, and let  $B_k \subseteq D_k$  denote the set of burned vertices after  $k$  time intervals that are distance  $k$  from the original burned vertex.

Also let  $f$  denote the number of firefighters available at each iteration, and let  $r_k$  denote the number of firefighters in  $D_{k+1}, D_{k+2}, \dots$  after  $k$  time intervals. These are the firefighters in reserve. Let  $x$  denote the number of firefighters placed at distance

$k + 1$  at step  $k$  and  $y$  denote the number of firefighters placed at distance  $k + 1$  from the time periods  $1, 2, \dots, k$ . For  $A \subset D_k$  let  $|N^+(A)| = N(A) \cap D_{k+1}$ .

We are now in position to state our main theorem.

**Theorem 1** *Let  $G$  be a graph such that for each  $k$  if every  $A \subset D_k$  satisfies  $|N^+(A)| \geq |A| + f$ , then  $|B_n| \geq 1 + r_n$  for every  $n$ .*

**Proof:** Basis Step  $k = 0$ . In this case, it is obvious that  $D_0 = 1 \geq 1 + 0$  because there is one vertex on fire and no firefighters have been placed. Now assume that the inductive hypothesis holds for  $k$ . That is

$$|B_k| \geq 1 + r_k. \tag{1}$$

Then we will prove it is true for  $k + 1$ .

Consider step  $k + 1$ . Note that  $r_{k+1} = (r_k - y) + (f - x) = r_k + f - x - y$  because we have placed  $x$  firefighters out of  $f$  available firefighters in  $D_{k+1}$  at step  $k$ ,  $y$  firefighters in  $D_{k+1}$  from the earlier steps, and  $r_k$  reserve firefighters in  $D_{k+1}, D_{k+2}, \dots$ . Thus we have the following

$$\begin{aligned} |B_{k+1}| &= |N^+(B_k)| - x - y \\ &\geq |B_k| + f - x - y && \text{from hypothesis} \\ &\geq 1 + r_k + f - x - y && \text{from (1)} \\ &= 1 + r_{k+1}. \end{aligned}$$

Therefore the claim holds for all  $k$ . ■

This is a structural theorem based on the number of neighbors of a subset of vertices. We can use the theorem to show that  $f$  firefighters are not enough to surround the fire in a specific graph.

**Corollary 2** *It is not possible to surround a fire with one firefighter in the quarter plane.*

**Proof:** In the quarter plane with  $x, y \geq 0$ , if  $A \subset B_k$ , it is not difficult to observe that  $|N^+(A)| \geq |A| + 1$ . So if  $f = 1$ , we see from Theorem 1 that  $|B_k| \geq 1 + r_k \geq 1$ . Thus since one vertex is on fire at time zero, we have for every  $k$ ,  $|B_k| \geq 1$ . Hence the fire cannot be contained with one firefighter. ■

Theorem 4.2 in Wang and Moeller [7] proved that one firefighter is not sufficient to surround the fire in the infinite rectangular grid. This is an obvious corollary of Corollary 2.

**Corollary 3** [*Wang and Moeller*] *It is impossible to surround the fire with one firefighter in the infinite rectangular grid.*

**Proof:** If we could surround the fire with one firefighter, then we could surround the fire in the quarter plane with one firefighter. This contradicts Corollary 2. ■

In [7], it is stated that for the three-dimensional rectangular grid graph, at least two firefighters are necessary to surround a fire (starting at time zero). In fact, they conjecture that five are necessary. Our next corollary shows that at least three firefighters are necessary.

**Corollary 4** *In the three-dimensional rectangular grid, at least three firefighters are necessary to surround the fire.*

**Proof:** In the three-dimensional rectangular grid, if  $A \subset B_k$  then it is not difficult to observe that  $|N^+(A)| \geq |A| + 2$ . So if  $f = 2$ , we see from Theorem 1 that  $|B_k| \geq 1 + r_k \geq 1$ . Thus since one vertex is on fire at time zero, we have for every  $k$ ,  $|B_k| \geq 1$ . Hence this fire cannot be contained with two firefighters. ■

### 3 Infinite Rectangular Grids

#### 3.1 Introduction

Firefighters don't usually have any barriers that help them do their jobs better and more quickly. In many wildfires, the fire can spread in any direction. Therefore the first case we consider is when the grid is unrestricted in size. We would like to determine the number of firefighters that are needed to protect the plane.

In this section, we will assume that a single fire breaks out at time zero at the origin. Our  $f$  firefighters attempt to surround this fire.

#### 3.2 Two Firefighters

We see from Corollary 3 that one firefighter working alone cannot surround a fire in the plane. From Wang and Moeller [7], we have: In the infinite rectangular grid, two firefighters can surround a fire in 8 time intervals and not sooner. But what happens if the fire goes unprotected for  $k$  steps? How many firefighters are necessary to surround the fire? Can two firefighters surround such a fire or are more needed?

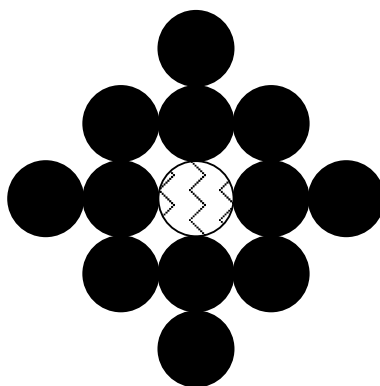


Figure 2: The Grid is Unprotected through Time 2.

Figure 2 shows what the grid looks like when the grid is not protected at time 1 and at time 2; if the firefighters protect the grid at time 3, then all vertices of distance

1 and 2 will remain on fire. If we assume the fire starts at time 0 at the origin, then the burned vertices at distance  $k$  are in a diamond shape and are bounded by the lines  $|x| + |y| = k$ .

The next theorem shows that two firefighters can still do the job no matter how late they arrive. We describe a way to place the two firefighters so that they will eventually surround the fire.

**Theorem 5** *For any  $k \in \mathbb{N}$ , if the plane goes unprotected through step  $k$ , then two firefighters suffice to surround the fire.*

**Proof:** Let the plane go unprotected through time  $k$ , and assume that the firefighters start protecting the grid at time  $k + 1$ . Assume without loss of generality that the fire starts at the origin; hence the following set of vertices are burned:  $\{(x, y) \mid |x| + |y| \leq k\}$ . At time  $k + 1$ , place the two firefighters, one at  $(-k - 1, 0)$  and the other at  $(-k, -1)$ .

At time  $k + 2$ , place the two firefighters, one at  $(-k, -2)$  and the other at  $(-k + 1, -3)$ . Next, we need to place a firefighter at  $(-k - 2, 1)$ . We also place a firefighter at  $(-k + 1, -4)$ . We continue this pattern until time  $3k + 1$  and protect vertices  $(-k + 1, -3)$ ,  $(-k + 1, -4)$ ;  $(-k - 3, 2)$ ,  $(-k + 1, -5)$ ;  $\dots$   $(-1, -3k - 1)$ , and  $(0, -3k)$ . At this point, we will have caught up to the bottom of the fire.

At time  $3k + 1$ , we protect  $(-2k - 1, k)$  and  $(0, -3k)$ . Next we place a firefighter at vertex  $(1, -3k)$  and  $(2, -3k + 1)$ . We continue this pattern of protecting the grid until time  $9k$  at which point we will have caught up with the fire on the left side. At this point, we will have placed the firefighters at  $(9k - 1, -1)$  and  $(9k, 0)$ . Figure 3 shows what the grid looks like up to time  $9k$  when  $k = 1$ .

We now alter the pattern of protecting. At time  $9k + 1$ , we place the firefighters at vertices  $(-5k - 1, 4k + 1)$  and  $(-5k, 4k + 2)$ . We then protect  $(-5k, 4k + 3)$  and  $(9k + 1, 1)$ . We continue this pattern of protecting the grid until time  $19k + 1$  at

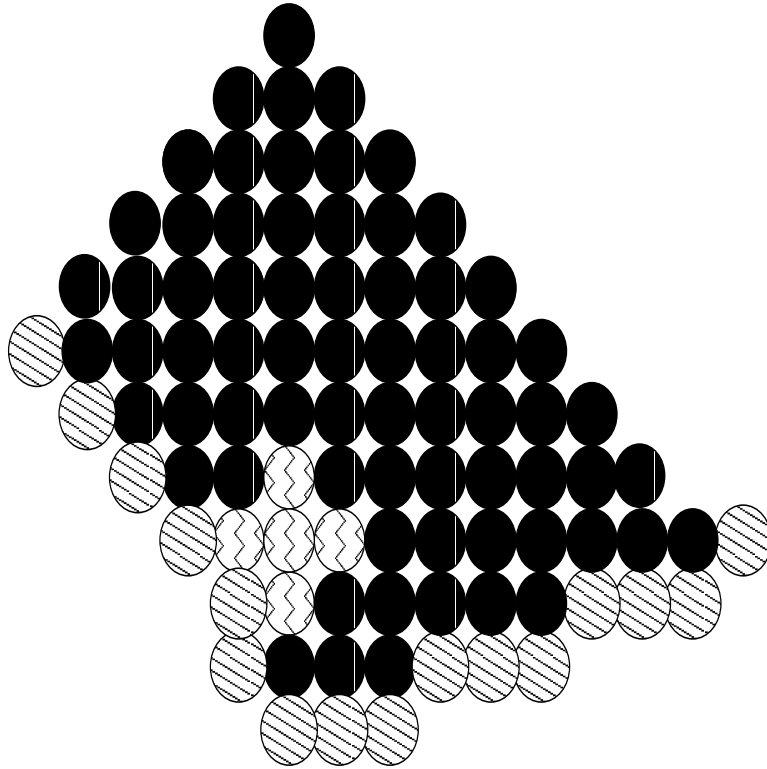


Figure 3: The Grid from Time  $3k + 1$  until  $9k$  for  $k = 1$ .

which point we will have reached the top of the fire. Figure 4 shows the grid up to time  $19k + 1$ .

At this point, we protect  $(-1, 19k)$ ,  $(0, 19k + 1)$ ;  $(1, 19k + 1)$ ,  $(14k + 1, 5k + 1)$ ;  $(2, 19k + 1)$  and  $(3, 19k)$ . We continue this pattern of protecting until time  $32k + 1$  at which point we place the firefighters at  $(21k - 1, 12k + 3)$  and  $(21k, 12k + 2)$ . Thus the fire has been surrounded. Figure 5 shows the fire completely surrounded. ■

We denote this way of protecting the grid as Pattern  $I$ . Now that we know that two firefighters are sufficient to surround the fire, we would like to know how many vertices are burned. By looking at the pattern of how the fire grows, we can find a quadratic function that gives the total number of burned vertices when the firefighters begin protecting the grid at time  $k + 1$  for  $k > 0$  using Pattern  $I$ .

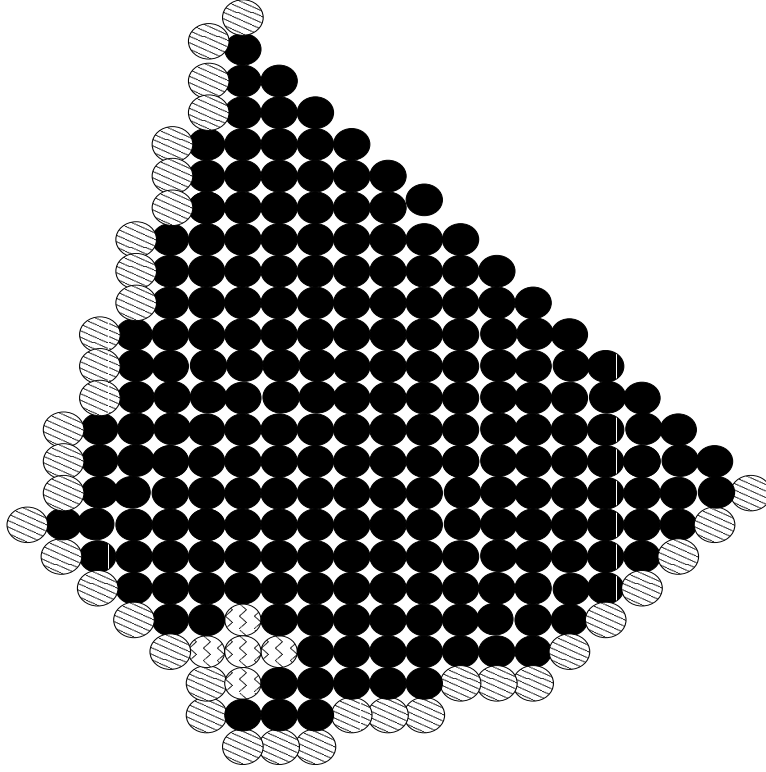


Figure 4: The Grid from Time  $9k + 1$  until  $19k + 1$  for  $k = 1$ .

**Theorem 6** *Assume that the grid goes unprotected through time  $k$  and two firefighters arrive at time  $k + 1$ . Then they can protect the grid in  $32k + 1$  steps and leave  $318k^2 + 14k + 1$  burned vertices.*

**Proof:** Split the grid into six parts as shown in Figure 6 where the vertices labeled are the last burned vertices in their respective row or column. By looking at each of these sections, we can determine the total number of burned vertices.

I. We have a triangle with vertices  $(0, 0)$ ,  $(0, -3k + 1)$ , and  $(9k - 1, 0)$ . An easy computation shows that there are  $\frac{27}{2}k^2 + \frac{9}{2}k$  burned vertices in this region.

II. We have a triangle with vertices  $(-k - 1, 0)$ ,  $(0, -3k + 1)$ , and  $(0, 0)$ . Therefore there are  $\frac{3}{2}k^2 - \frac{5}{2}k + 1$  burned vertices.

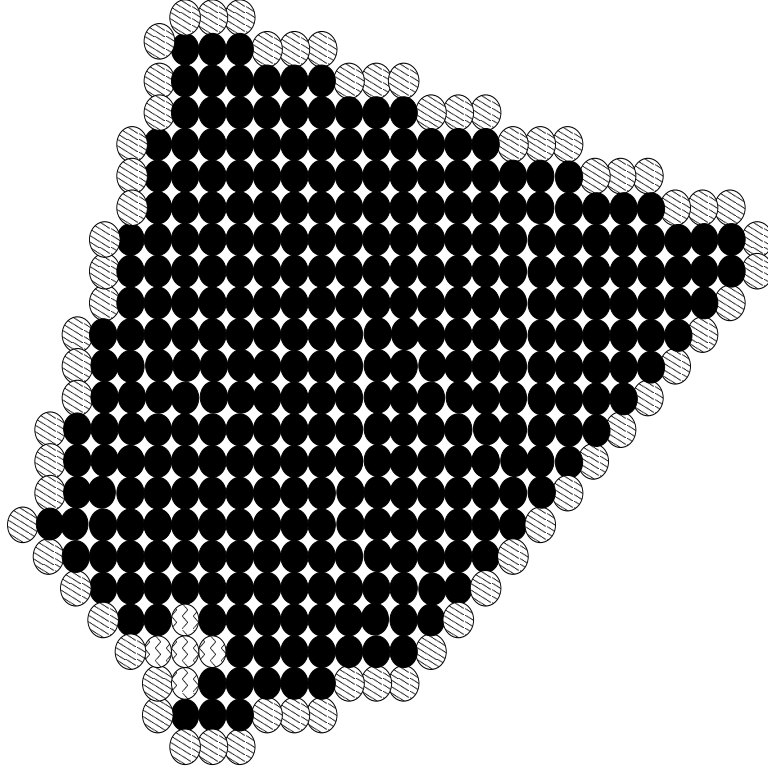


Figure 5: The Grid Completely Protected for  $k = 1$ .

III. This is a quadrilateral with vertices  $(-k - 1, 0)$ ,  $(0, 0)$ ,  $(-5k - 1, 4k + 1)$ , and  $(0, 4k + 1)$ . There are  $12k^2 + 3k$  burned vertices in this region.

IV. Here we have a triangle with vertices  $(0, 4k+1)$ ,  $(-5k-1, 4k+1)$ , and  $(0, 19k+1)$ . There  $\frac{75}{2}k^2 - \frac{15}{2}k$  burned vertices in region *IV*.

V. This section is a triangle with vertices  $(0, 19k+1)$ ,  $(0, 12k+1)$ , and  $(21k, 12k+1)$ . Hence there are  $\frac{147}{2}k^2 + \frac{21}{2}k$  burned vertices.

VI. This region is a quadrilateral with vertices  $(21k, 12k + 1)$ ,  $(0, 12k + 1)$ ,  $(0, 0)$ , and  $(9k - 1, 0)$ . Here there are  $180k^2 + 6k$  burned vertices in this region.

Therefore in total there are  $318k^2 + 14k + 1$  burned vertices as claimed. The number of steps is  $32k + 1$  by the proof of Theorem 5. ■

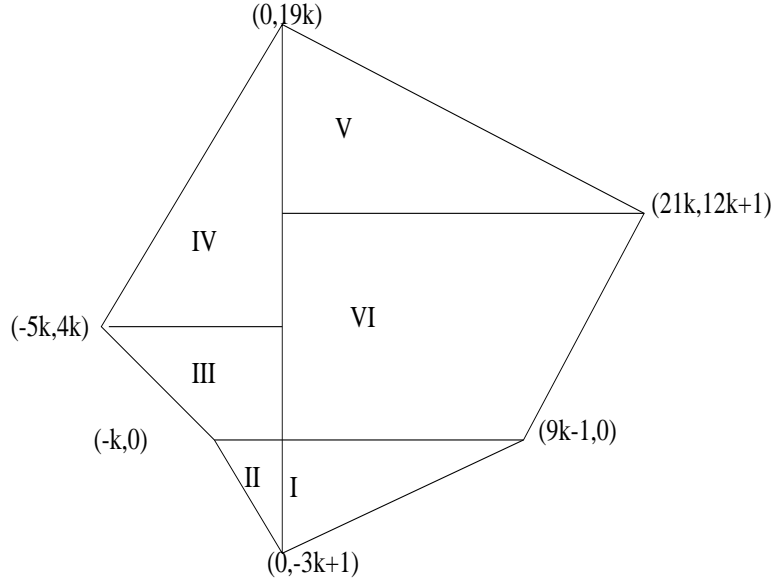


Figure 6: The Pattern of Saving the Grid.

We would like to know if it is possible to surround the fire in another way in order for fewer vertices to be burned. We give a different pattern below. We will see that this new pattern also surrounds the fire in  $32k + 1$  steps. Interestingly, it leaves  $318k^2 + 14k + 1$  burned vertices.

To construct this new pattern, we begin with Pattern *I*. At time  $19k + 2$ , place one firefighter at vertex  $(14k + 1, 5k + 1)$  and the other one at  $(14k, 5k + 2)$ . Then at time  $19k + 3$ , place one firefighter at  $(14k - 1, 5k + 3)$  and the other at  $(1, 19k + 2)$ . Continue this pattern until time  $32k + 1$  at which point we place our firefighters at  $(9k - 2, 27k)$  and  $(9k - 1, 27k - 1)$ . We call this Pattern *II*. Figure 7 shows the geometry of this pattern.

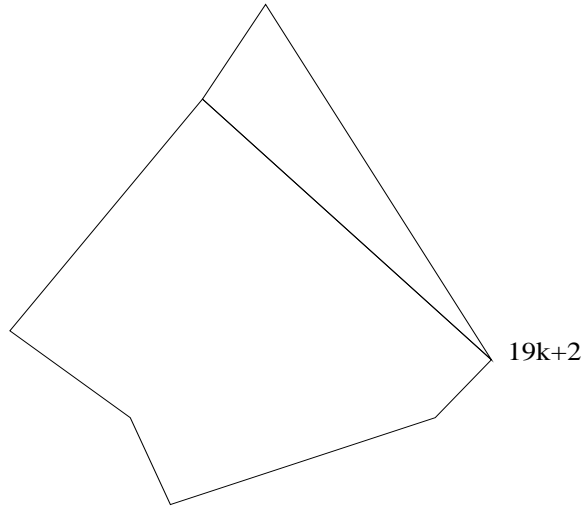


Figure 7: Pattern II.

**Theorem 7** *Pattern I and Pattern II are equivalent.*

**Proof:** The only parts of the grid that need to be checked are sections *I* of Pattern *A* and section *II* of the Pattern *B* since the other parts of the grid surround the exact same number of burned vertices. Figure 8 shows the geometry for both patterns.

By looking at the lengths of each of the sections, we see that the length of side *a* equals the length of side *f* since they are identical. We also have that the length of side *b* equals the length of side *e* and the length of side *c* equals the length of side *d*. Therefore by congruent triangles, section *I* of Pattern *I* is equivalent to section *II* of Pattern *II*. ■

We now have two patterns which produce the same number of burned vertices and which both take  $32k + 1$  steps to surround the fire. Compared with several other patterns which we found, we determined that Pattern *I* and Pattern *II* yielded fewer burned vertices. This leads us to the following conjecture.

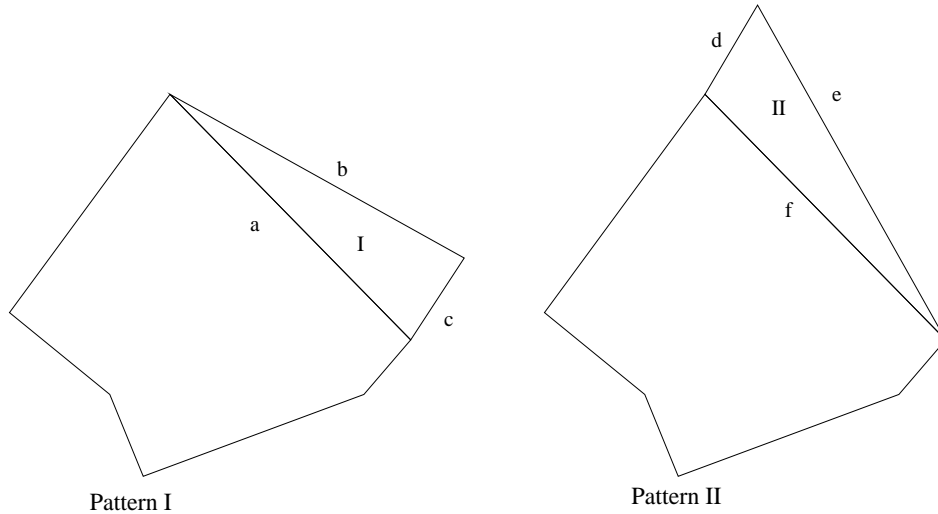


Figure 8: Both Patterns of Saving the Grids.

**Conjecture 8** *A minimum of  $318k^2 + 14k + 1$  vertices burn in  $32k + 1$  time intervals when two firefighters do not arrive until time  $k + 1 > 0$ .*

### 3.3 More than One Fire in the Plane

Sometimes more than one fire breaks out at a time, making the task of protecting the plane harder. Given  $n$  fires, we utilize a diamond shape to surround the fire. Figure 9 shows how we would like to protect the infinite rectangular grid.

**Corollary 9** *If  $n$  fires start anywhere in the infinite rectangular grid, two firefighters suffice to surround the fire.*

**Proof:** We first construct a diamond of side  $k$  for some  $k$  surrounding all the fires and proceed using Pattern I. Thus we are able to surround the fire. ■

We note that this pattern of protecting  $n$  fires probably does not protect the minimum number of fires.

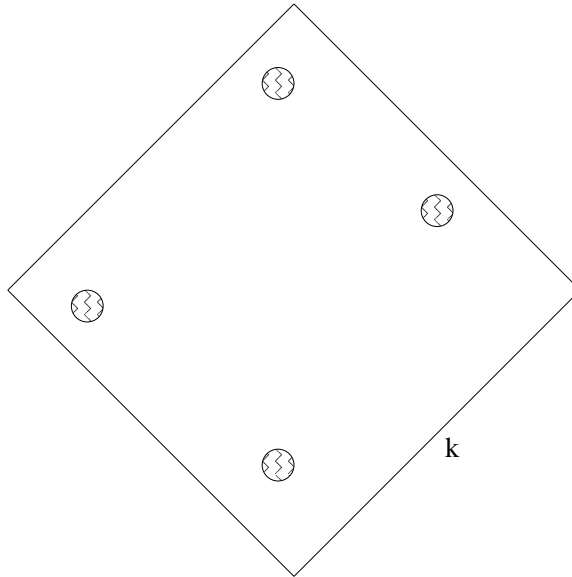


Figure 9: How to Protect  $n$  fires.

### 3.4 JAVA Program for the Plane

To illustrate Pattern  $I$ , we wrote a JAVA program that shows the growth of the fire when the firefighter starts protecting the grid at time  $k$ . The program is in Appendix B, and a disk is included that contains this program.

## 4 Ribbon Grids

### 4.1 Introduction

In this section, we restrict the plane to have two borders, an upper and a lower one, and also assume that exactly one firefighter is available at each time period. This could model a fire which begins between two rivers. We will look at the case where the fire starts at the top level of the grid then further consider the case where the fire starts at the  $k^{\text{th}}$  level.

A *ribbon grid* is a  $P_n \square P_\infty$  grid with the points  $\{(x, y) \mid -\infty < x < \infty, 1 \leq y \leq n\}$ . We will call this the  $n \times \infty$  grid. The fire starts at a specific *level*, which is the  $y$  coordinate of the original burned vertex,  $(0, y)$ . Clearly one firefighter alone can surround this fire by merely constructing two fire walls, one at  $x = -m$  and the other at  $x = m$  for any value of  $m \geq 2n$ .

Obviously an important question is where the firefighter should be placed at each step in order to minimize the number of burned vertices. We would like to find a general pattern which minimizes the total number of burned vertices.

### 4.2 Fires Starting on the First Level

In order to understand the properties of rectangular grids and how the fire spreads, we look at fires starting in the first level. Note that this case is equivalent to a fire starting in the top level.

As before, we let  $D_k$  be the set of vertices at distance  $k$  from the original burned vertex  $(0, 1)$ . Let  $B_k$  be the set of burned vertices at distance  $k$ , and let  $f_k$  be the number of firefighters placed in  $D_k$  at or prior to time  $k$ . Note that  $\sum_{i=1}^k f_i \leq k$ .

The following two lemmas give lower bounds on  $B_k$  for  $1 \leq k \leq 2n$ .

**Lemma 10**  $|B_k| \geq k + 1$  for  $k \leq n - 1$ .

**Proof:** We prove this by induction.

Basis:  $|B_0| = 1 \geq 1$ .  $|B_1| \geq 2$  because we can place a firefighter in  $D_2, D_3, \dots$  or we place the firefighter in  $D_1$ . Note this is the first firefighter placed.

Now we assume for all  $k < s \leq n - 1$  that  $B_k \geq k + 1$ . Let  $k \leq s - 1$ . Since  $B_k \geq k + 1$ , it is easy to see that  $B_{k+1} \geq B_k + 2 - f_{k+1}$  as each burned vertex will burn one vertex above it and the leftmost and rightmost burned vertices will also burn one additional vertex to the left and right, respectively.

Now,

$$\begin{aligned}
 \sum_{k=0}^s |B_k| &= |B_0| + \sum_{k=1}^s |B_k| \\
 &\geq |B_0| + \sum_{k=1}^s |B_{k-1}| + 2 - f_k \\
 &= |B_0| + \sum_{k=0}^{s-1} |B_k| + 2 - f_{k+1} \\
 &= |B_0| + \left( \sum_{k=0}^{s-1} |B_k| \right) + 2s - \sum_{k=0}^{s-1} f_{k+1}.
 \end{aligned}$$

Hence  $|B_s| \geq |B_0| + 2s - s$ . So  $|B_s| \geq 1 + 2s - s = s + 1$ . ■

Next we consider the case of  $B_k$  for  $n \leq k \leq 2n$ .

**Lemma 11**  $|B_k| \geq 2n - k$  for  $n \leq k \leq 2n$ .

**Proof:** We prove this by induction.

Basis:  $k = n$ . We have

$$\begin{aligned} \sum_{k=0}^n |B_k| &= \sum_{k=0}^{n-1} |B_k| + |B_n| \\ &\geq |B_0| + \sum_{k=0}^{n-2} (|B_k| + 2 - f_{k+1}) + (|B_{n-1}| + 1 - f_n) \\ |B_n| &\geq 1 + 2(n-1) - \sum_{k=1}^n f_k + 1 \geq 2n - n = n \end{aligned}$$

as desired. Note, we used the fact that  $|B_n| \geq |B_{n-1}| + 1 - f_n$ . In what follows, we have  $|B_k| \geq |B_{k-1}| - f_k$  for  $k \geq n+1$ .

Let  $n+1 \leq s \leq 2n$ . Thus

$$\begin{aligned} \sum_{k=0}^s |B_k| &= \sum_{k=0}^{n-1} |B_k| + |B_n| + \sum_{k=n+1}^s |B_k| \\ &\geq 1 + \sum_{k=0}^{n-2} (|B_k| + 2 - f_{k+1}) + (|B_{n-1}| + 1 - f_n) + \sum_{k=n}^{s-1} (|B_k| - f_{k+1}). \end{aligned}$$

Therefore we have

$$\begin{aligned} |B_s| &\geq 1 + 2(n-1) + 1 - \sum_{k=1}^s f_k \\ &\geq 2n - s \end{aligned}$$

as desired. ■

Now we can provide a lower bound for the total number of burned vertices no matter what pattern is used.

**Lemma 12** *If the fire starts in the first level, the minimum number of burned vertices is at least  $n(n+1)$ .*

**Proof:** From Lemma 10 and Lemma 11 we have

$$\begin{aligned} \sum_{k=0}^{2n} |B_k| &\geq \sum_{k=0}^{n-1} (k+1) + \sum_{k=n}^{2n} (2n-k) \\ &= \frac{n(n+1)}{2} + 2n(n+1) - \frac{3n(n+1)}{2} \\ &= n(n+1) \end{aligned}$$

as desired. ■

**Theorem 13** *If the fire starts in the first level of an  $n \times \infty$  grid, the minimum number of vertices burned is  $n^2 + n$ .*

**Proof:** We give a pattern that yields  $n^2 + n$  burned vertices. The theorem then follows from Lemma 12.

Assume that we protect the left side of the grid until time  $r \leq n$  and start protecting the right side at time  $r + 1$ . We place the firefighter at vertices  $(-1, 1)$ ,  $(-1, 2)$ ,  $(-1, 3)$ ,  $\dots$ ,  $(-1, r)$  which are all on the left side. We now protect vertex  $(r + 2, 1)$ . Now let the placement of the firefighter alternate between the left and right side. It is easy to check the the following pattern in Figure 10 is obtained.

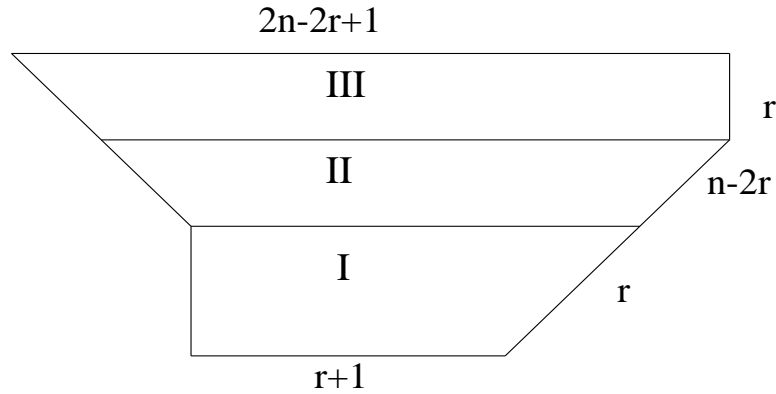


Figure 10: The Pattern to Protect the Rectangular Grid.

In order to count the number of burned vertices, we consider the three regions of the grid.

- I. Here we have height  $r$  and base of  $r + 1$ . The width increases by one as the height increases by one so the number of burned vertices in this region is  $(r + 1) + (r + 2) + \cdots + (r + r)$ . Hence in this region a total of  $r^2 + \frac{r(r+1)}{2}$  vertices burn.
- II. Here we have height of  $n - 2r$  and base of  $2r + 2$ . The width increases by two as the height increases by one. Hence the number of burned vertices in this region is  $(2r + 2) + (2r + 4) + \cdots + (2n - 2r)$ ; a total of  $(n + 1)(n - 2r)$  vertices burn.
- III. Here we have height of  $r$  and base of  $2n - 2r + 2$ . The width increases by one as the height increases by one. Thus the number of burned vertices is  $(2n - 2r + 2) + (2n - 2r + 1) + \cdots + (2n - 2r + r)$ ; there is a total of  $(2n - 2r)r + \frac{r(r+1)}{2} + r$  burned vertices in this region.

If we add the burned vertices from Region *I* and Region *III*, there are  $2r(n + 1)$  burned vertices. Now adding the number of burned vertices from Region *II* we get that there are  $n^2 + n$  burned vertices in total. ■

Figure 11A shows one way of protecting the  $4 \times \infty$  grid if we alternate between the left side and right side from the beginning,  $r = 1$ . Figure 11B shows another way of protecting the grid if we protect the left side first and then proceed to protect the right side,  $r = n$ . Note that if the fire starts at level  $n$ , a total of  $n^2 + n$  vertices burn also.

Figure 12 shows how the grid is protected when we start protecting the right side at time 4 in a  $10 \times \infty$  grid.

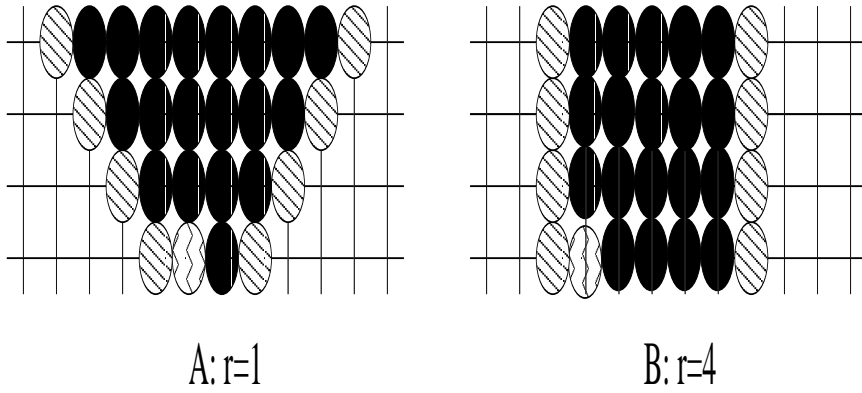


Figure 11: Ways to Protect the Grid.

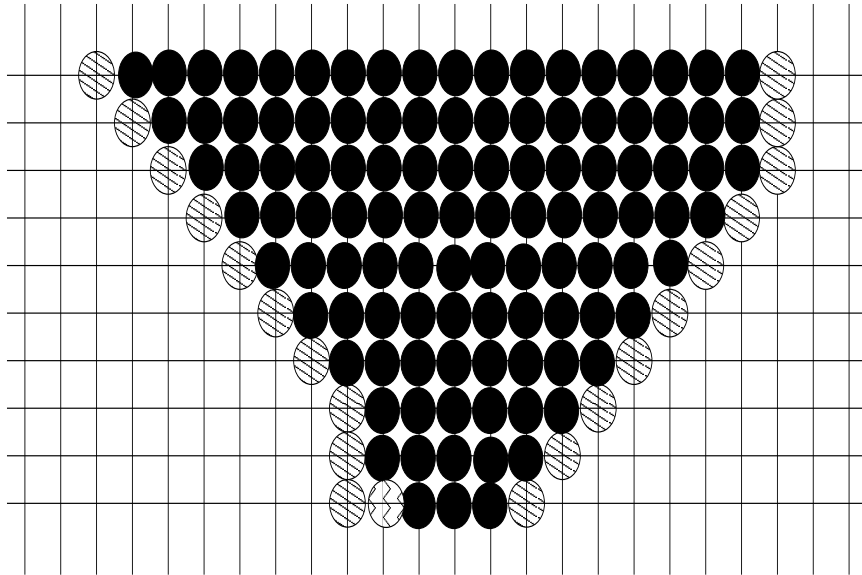


Figure 12: Fire starting at the bottom for  $r = 3$ .

### 4.3 Fires Starting Anywhere

Using the case where the fire starts at the top as a guide, we can determine how many vertices will burn when the fire starts at any level in the ribbon grid.

Figure 13 shows how we protect a fire starting at level  $k = 4$  in the  $7 \times \infty$  grid.

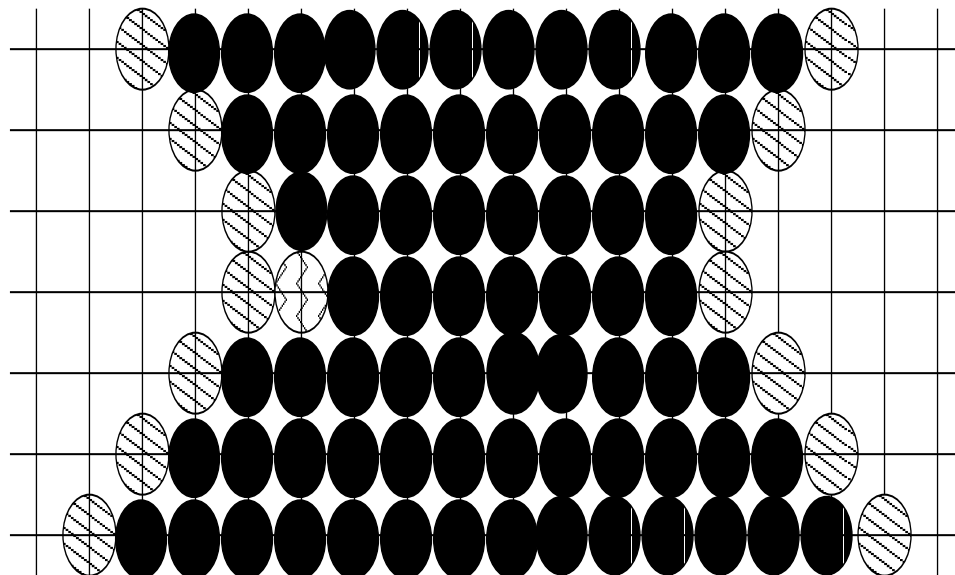


Figure 13: The  $7 \times \infty$  grid for  $k = 4$ .

In a manner similar to Lemma 10, Lemma 11, and Lemma 12, one can count the minimum number of burned vertices in the case where the fire begins on the  $k^{\text{th}}$  level. We will not include the calculations here.

**Lemma 14** *Given an  $n \times \infty$  grid where the fire starts at level  $k$ , the minimum number of vertices burned is at least  $n^2 - n - 2k^2 + 2k + 2nk$ .*

**Theorem 15** *Given an  $n \times \infty$  grid where the fire starts at level  $k$ , the minimum number of vertices burned is  $n^2 - n - 2k^2 + 2k + 2nk$ .*

**Proof:** First note that we the case  $k = 1$  follows from Theorem 13. We give a pattern that yields  $n^2 - n - 2k^2 + 2k + 2nk$  burned vertices. The theorem then follows from Lemma 14. Therefore we need to prove the claim holds for  $1 < k < n$ . Assume that the fire breaks out at  $(0, k)$ .

As in the proof of Theorem 13 the firefighter protects the entire left side of the grid and then the right side of the grid. Figure 14 shows the pattern of surrounding the fire when the fire starts at level  $k$ .

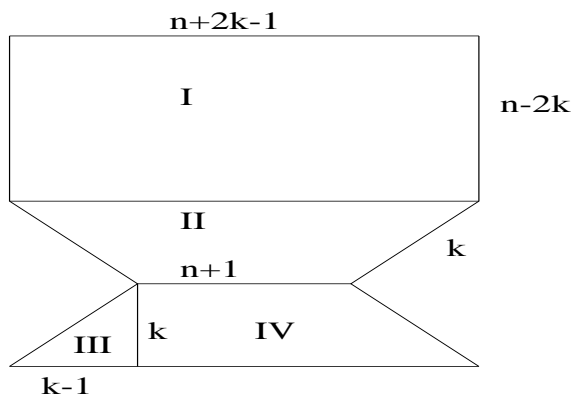


Figure 14: The General Pattern of Protecting Rectangular Grids.

By looking at each of the regions, it is not hard to count that there are  $n(2n + 1) - (n - k)(n - k + 1) - k(k - 1) - n = n^2 - n - 2k^2 + 2k + 2nk$  burned vertices. ■

We have seen that one firefighter is sufficient to surround the rectangular grid. How much of a difference do two firefighters make when the fire starts at level one?

**Proposition 16** *If the fire starts at the first level in a  $P_n \square P_\infty$  grid and two firefighters are available, there are only two burned vertices.*

**Proof:** Let the fire start at  $(0, 1)$ . Assume the firefighters protect vertices  $(1, 1)$  and  $(0, 2)$ . Then the fire can only spread to vertex  $(-1, 0)$ . Therefore at the next step the firefighters protect vertices  $(-1, 2)$  and  $(-2, 0)$  and the fire cannot spread any further. ■

#### 4.4 JAVA Program for Rectangular Grids

To illustrate the algorithms used to surround the fire in rectangular grids, we wrote a program simulating the growth of the fire. The program makes the most efficient move at each point for the firefighters. The code is in Appendix A and a disk is included that contains this program.

If the fire starts in the first or last level of the grid, the program creates two parallel walls to surround the fire. However, if there is not room on one side of the fire to do so, the firefighters will protect only one side of the grid. In most cases, it is possible to save both sides of the grid.

The second part of the algorithm protects the grid if the fire starts at level  $1 < k < n$ . The first move will always be adjacent to the fire. The second move is always directly above this protected vertex, i.e. the firefighter protects a vertex on the left side on level  $k + 1$ . The third move is below the first protected vertex, i.e. the firefighter protects the vertex on the left side on level  $k - 1$ . The program then alternates between protecting above and below level  $k$ . Again if there is not room to surround the fire on one side of the grid, the firefighter picks the other side to protect.

## 5 The Quarter-Plane

### 5.1 Introduction

In some cases, there are two perpendicular barriers that the firefighters can use to trap their fire. We model this by letting the fire be in the first quadrant of the Cartesian Plane and assume it cannot escape from that quadrant.

### 5.2 An Early Firefighter

From Corollary 2, we know that one firefighter can not alone surround a fire in the quarter plane. However, the next theorem shows that it is nearly possible. We assume the quarter plane consists of the lattice points  $\{(x, y) | x, y \geq 0\}$  and that the fire starts at the origin.

**Theorem 17** *If one firefighter comes to the grid before the fire starts and protects any vertex in the first quadrant, it is possible to surround the fire in the quarter plane with one firefighter.*

Figure 15 shows what it looks like when we have an early firefighter at vertex  $(5, 3)$ .

**Proof:** If the firefighter is placed anywhere along the  $x$ -axis, say  $(n, 0)$ , we can place this same firefighter successively at  $(0, 1), (1, 1), (2, 1), (3, 1), \dots, (n - 1, 1)$  to eventually surround the fire.

Now assume that the early firefighter protected the vertex  $(i, j)$  where  $i > 0$ . Then at time 1, we place a firefighter at  $(0, j)$ . We then protect  $(1, j), (2, j), (3, j), \dots, (i - 1, j)$ . We then place the firefighter successively at  $(i + 1, j - 1), (i + 2, j - 2), (i + 3, j - 3), \dots, (i + j, 0)$  and have successfully surrounded the fire. ■

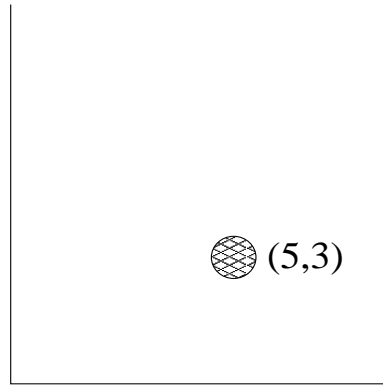


Figure 15: An Early Firefighter.

Figure 16 shows the fire surrounded when an early firefighter arrives at vertex  $(5, 3)$ .

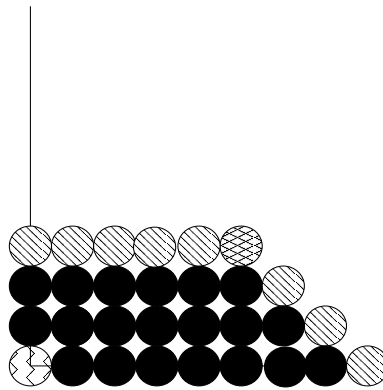


Figure 16: Protecting the Quarter Plane with an Early Firefighter.

This may not be an optimal strategy to surround the fire, but it shows how effective an early firefighter can be in the quarter plane.



## 6.2 One Fractional Firefighter

From Corollary 3, we know that one firefighter does not suffice to surround the fire in the plane. However, we show that when one firefighter protects only portions of each vertex, the one fractional firefighter can surround the fire.

**Theorem 18** *One fractional firefighter can surround a fire in the plane in thirty-one time intervals when the firefighter protects the fire from time 1.*

**Proof:** At time 0, one vertex is on fire. At time 1, the firefighter protects  $\frac{1}{4}$  of each vertex in  $D_1$ ; thus  $\frac{3}{4}$  of each vertex in  $D_1$  is on fire at the end of time 1.

At time 2, the firefighter now protects  $\frac{1}{8}$  of each vertex in  $D_2$  and thus  $1 - \frac{1}{4} - \frac{1}{8} = \frac{5}{8}$  of each vertex in  $D_2$  is on fire.

We continue this pattern of fire protecting and firefighter spreading. At time  $n$ , the firefighter protects  $\frac{1}{4n}$  of each vertex of distance  $D_n$ . Thus in  $D_n$

$$1 - \frac{1}{4} - \frac{1}{8} - \dots - \frac{1}{4n}$$

of each vertex is on fire. Clearly the fire will be contained if for some  $n$ ,  $\sum_{k=1}^n \frac{1}{4k} > 1$ . But  $\sum_{k=1}^n \frac{1}{4k} = \frac{1}{4} \sum_{k=1}^n \frac{1}{k}$ . Since  $\sum \frac{1}{k}$  is a divergent series, we have that the fire is contained at step  $n$  for some  $n$ .

We now note that

$$\frac{1}{4} \sum_{k=1}^{30} \frac{1}{k} = .9987467827 < 1$$

but

$$\frac{1}{4} \sum_{k=1}^{31} \frac{1}{k} = 1.006811299 > 1.$$

Hence the fire will be surrounded at time 31. ■

Now as in Section 3.2 assume that the firefighter does not show up until time  $k$ . We show that it is still possible to surround the fire with one fractional firefighter.

**Corollary 19** *If one fractional firefighter shows up at time  $t$ , the fire can be surrounded.*

**Proof:** Let the firefighter begin protecting the grid at time  $t$ . At time  $n$ , the strength of the fire at  $D_n$  is  $1 - \sum_{k=t}^n \frac{1}{4k}$ . Clearly the fire will be contained if for some  $n$ ,  $\sum_{k=1}^n \frac{1}{4k} > 1$ . But  $\sum_{k=t}^n \frac{1}{4k} = \frac{1}{4} \sum_{k=t}^n \frac{1}{k}$ . Since  $\sum \frac{1}{k}$  is a divergent series, we have that the fire is contained at step  $n$  for some  $n$ . ■

## 7 Hexagonal Grids

### 7.1 Introduction

After exploring the different possibilities in the plane, quarter plane, and in ribbon grids, we look at a different grid: the hexagonal grid. One might expect the results from rectangular grids to hold for hexagonal grids also, but the layout of these grids does not permit application of the rectangular results. In rectangular grids, a vertex at distance  $i$  is not adjacent to any other vertex at distance  $i$ . However, in hexagonal grids, each vertex in  $D_k$  is adjacent to a neighbor on both sides of it. In this section we will restrict our attention to the case where the firefighter arrives at time 1. Figure 18 illustrates a hexagonal grid.

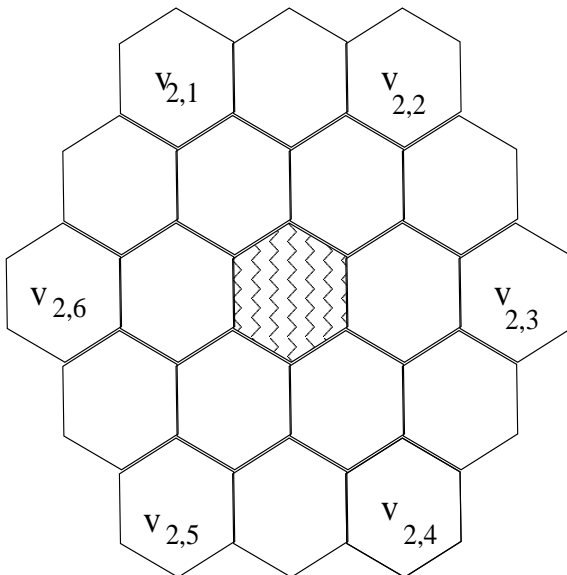


Figure 18: A Hexagonal Grid.

At each time interval  $m$  there are  $6m$  vertices in  $D_m$ . There are exactly six *corner vertices*. We label each of these  $v_{i,j}$  where  $i$  denotes the distance from the original burned vertex and  $j \in \{1, 2, \dots, 6\}$  denotes the specific corner vertex starting in the upper left. Each of the corner vertices at distance  $i$  is adjacent to three vertices at

distance  $i + 1$ , i.e.  $N^+(v_{i,j}) = 3$ . In Figure 18, vertices  $v_{2,j}$  where  $j \in \{1, 2, \dots, 6\}$  are corner vertices of distance 2.

There are  $6i - 6$  *border vertices*, where a border vertex is between the corner vertices; we denote them by  $u_{k,l}$  where  $k$  denotes the distance from the original burned vertex and  $l \in \{1, 2, \dots, 6i - 6\}$  denotes the specific border vertex starting from the upper left side. We see that for each of the border vertices at distance  $i$ ,  $N^+(u_{i,j}) = 2$ . In Figure 18 the vertices between  $v_{2,1}, v_{2,2}, \dots, v_{2,6}$  are border vertices.

## 7.2 Ribbon Hexagonal Grids

We first consider hexagonal grids that are restricted to a given height and where the fire starts on the first level. An  $n \times \infty$  hexagonal grid is a subgraph of the hexagonal grid which has height  $n$ . We also assume that there is one firefighter. Note that in Figure 18 the height of the hexagonal grid is 5.

**Theorem 20** *If the fire breaks out on the bottom level of a hexagonal grid of height  $n$ , the minimum number of vertices burned is at most  $\frac{3}{2}n^2 + \frac{1}{2}n$ .*

**Proof:** Assume that we are given an  $n \times \infty$  rectangular hexagonal grid.

We start protecting the grid right next to the original burned vertex at level 1. We then protect the vertex above and to the left of the last protected vertex and continue doing this until the left side is protected. Then we start protecting the right side beginning in row 1. When protecting the right side, we protect the vertex above and to the right of the last protected vertex.

Then on the first level there are  $n + 1$  burned vertices, and on the second level there are  $n + 2$  burned vertices. Thus on level  $i$ , there are  $n + i$  burned vertices. Hence there a total of

$$\sum_{i=1}^n (n + i) = n^2 + \frac{n^2 + n}{2} = \frac{3}{2}n^2 + \frac{1}{2}n$$

burned vertices. ■

### 7.3 Unrestricted Hexagonal Grids

After looking at rectangular hexagonal grids, we look at hexagonal grids unrestricted in height. First we show that 2 firefighters cannot surround the fire.

**Theorem 21** *In a hexagonal grid, two firefighters do not suffice to protect the grid.*

**Proof:** In the hexagonal grid, if  $A \subset B_k$ , then  $N^+(A) \geq |A| + 2$ . So if  $f = 2$ , we see from Theorem 1 that  $|B_k| \geq 1 + r_k \geq 1$ . Thus since one vertex is on fire at time zero, we have for every  $k$ ,  $|B_k| \geq 1$ . Hence this fire cannot be contained with two firefighters. ■

We now show that three firefighters can surround a fire in the hexagonal grid even if they do not begin protecting the grid until step  $k$ . We will find a pattern that will allow us to protect the grid and denote the pattern we use as Pattern  $J$ .

**Theorem 22** *For any  $k \in \mathbb{N}$ , if the plane goes unprotected through time  $k$ , three firefighters suffice to surround the fire in  $44k - 7$  steps.*

**Proof:** Figure 19 shows the lines that the fire wall will follow when protecting the hexagonal grid as well as showing the burned vertices at time  $k$ .

At time  $k + 1$ , one firefighter is placed at the left border vertex  $2k$  cells away from the highest vertex on fire on line  $e$ . The other two firefighters are placed below the first firefighter on line  $e$ , each firefighter on a different level. From time  $k + 2$  until time  $2k$ , place all three firefighters underneath the original three firefighters on line  $e$ .

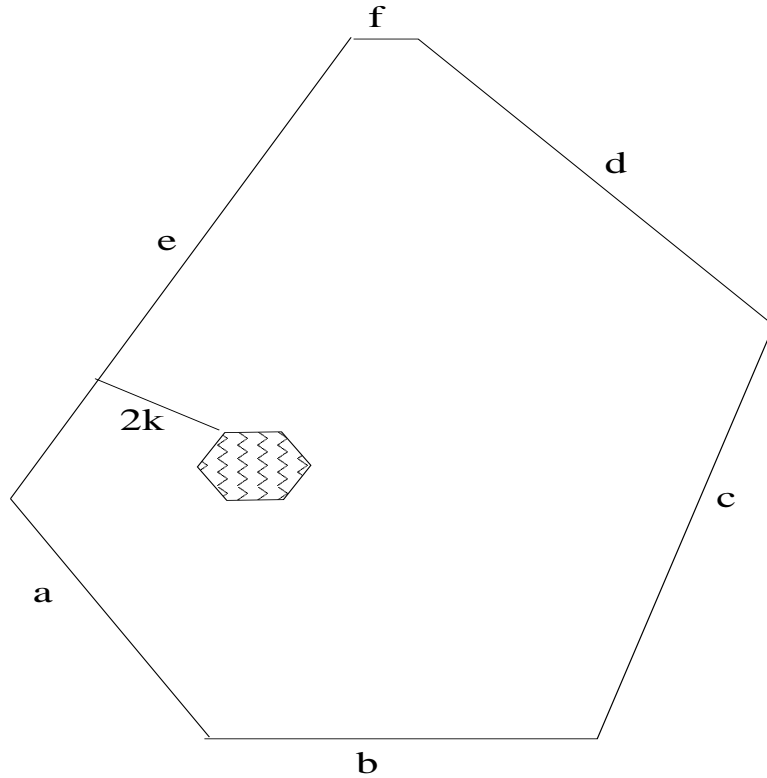


Figure 19: The Lines in Hexagonal Grids.

Continue placing the one firefighter on line  $e$  moving northeast from where the first protected vertex was placed. We do this from step  $2k + 1$  until step  $44k - 7$ . This firefighter maintains an equal height with the burned vertices.

From time  $2k + 1$  until time  $7k - 2$ , we place the other two firefighters in the lower portion of the grid along line  $a$ . At this point, we will have protected the entire southwest portion of the grid.

At time  $7k - 1$ , we begin to place the two firefighters horizontally in the easterly direction along line  $b$ . We continue this strategy with the two firefighters until time  $14k - 1$ . We now have protected the southern part of the grid.

From time  $14k$  until time  $32k - 5$ , place the two firefighters along line  $c$  moving northeast. We will now have protected the southeast section of the grid at time  $32k - 5$ .

From time  $32k - 4$  to  $43k - 7$ , we continue northwest along line  $d$ . This will allow the firefighters to protect the northeastern part of the grid.

Finally beginning at time  $43k - 6$ , we place the two firefighters horizontally in a westerly direction on line  $f$ . We will then have saved the grid after only  $k$  more time intervals since in  $k$  intervals the two firefighters met up with the one firefighter who was moving northeast on line  $f$  beginning at step  $2k + 1$ . We will have surrounded the fire at step  $44k - 7$ . ■

**Theorem 23** *If the grid goes unprotected through time  $k$  and three firefighters arrive at time  $k + 1$ , they can surround the fire in less than  $44k - 7$  steps with  $1100k^2 - 372k + 27$  burned vertices using Pattern J.*

**Proof:** Using the algorithm described above we can count the vertices burned following the lines in the grid as shown in Figure 20.

**Region 1:** Here we have a triangle that adds up the numbers from 1 to  $7k$ . Hence we have a total of  $98k^2 + 7k$  vertices burned.

**Region 2:** This is a rectangle that has width  $36k - 9$  and height  $14k + 1$ . Therefore there are  $504k^2 - 90k - 9$  burned vertices.

**Region 3:** This region is also a rectangle with width of  $8k - 4$  and height of  $50k - 9$ . Therefore  $400k^2 - 272k + 36$  vertices burn in this region.

**Region 4:** Here we have a rectangle, which has width  $2k$  and height  $48k - 9$ . Therefore there are  $96k^2 - 18k$  burned vertices in this region.

**Region 5:** This is a triangle that adds up the numbers 1 to  $2k$ ; therefore the number of vertices burned is  $2k^2 + k$ .

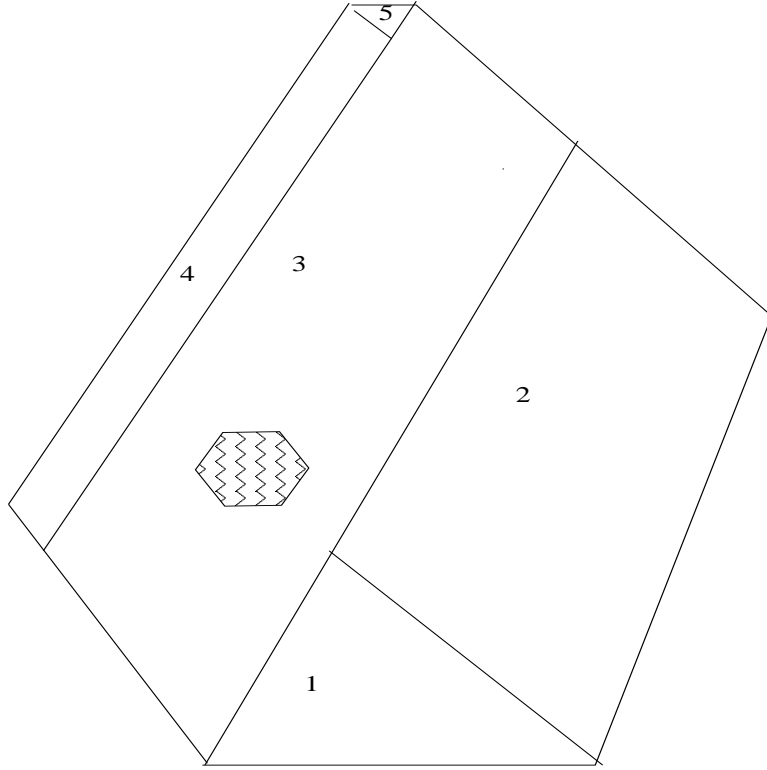


Figure 20: Protecting a Hexagonal Grid with Pattern  $J$ .

Hence there are a total of  $1100k^2 - 372k + 27$  burned vertices. ■

Pattern  $J$  shows a grid can be contained with three firefighters. But clearly this is not optimal. Is there a better way to do this? One can surround the fire with a similar outline as in the infinite rectangular grid. This time however we have three firefighters and one must always be used in the upper half of the grid. We give a pattern that allows three firefighters to protect the grid much faster with fewer burned vertices and call this Pattern  $P$ . Let the fire begin at level 0. At each step, we place the firefighters next to a burned vertex. (This will create a smaller number of burned vertices than in Theorem 23.)

1. At time  $k + 1$ , we place each firefighter on level 0 down to level  $-2$  by placing them on the left side of the fire.
2. We continue placing two firefighters below level 0 and adjacent to a burned vertex until time  $6k + 1$ . At this point, we will be back at level 0.
3. We place the third firefighter on the left of the burned vertices and on levels  $1, 2, \dots, 6k + 1$ .
4. From time  $6k + 1$  until time  $15k + 3$ , we take one of the two firefighters working together and put it with the lone firefighter, who is at level  $6k + 1$  from the last time interval. These two firefighters will now be placed next to the burned vertices on levels  $6k + 2$  and  $6k + 3$  on the left side of the burned vertices. They continue protecting vertices on the left side until time  $15k + 3$  when they will have reached the top of the fire.
5. The lone firefighter protects the right side of the grid and is placed directly to the right of the burned vertices on level  $1, 2, \dots$  until time  $18k + 5$  when the three firefighters will meet.
6. From time  $15k + 4$  until time  $18k + 5$ , the two firefighters are placed to the right of the burned vertices. They work from level  $16k + 4$  down to level of the third firefighter.

From Pattern  $P$ , we are able to determine the number of burned vertices and the number of time intervals it takes to protect the grid. Compared with other various patterns which also protect the grid, we found that Pattern  $P$  minimized the number of burned vertices. This leads to the following conjecture.

**Conjecture 24** *The hexagonal grid can be protected in a minimum of  $18k + 5$  time intervals with a minimum of  $193k^2 + 47k + 20$  burned vertices (using Pattern  $P$ ).*

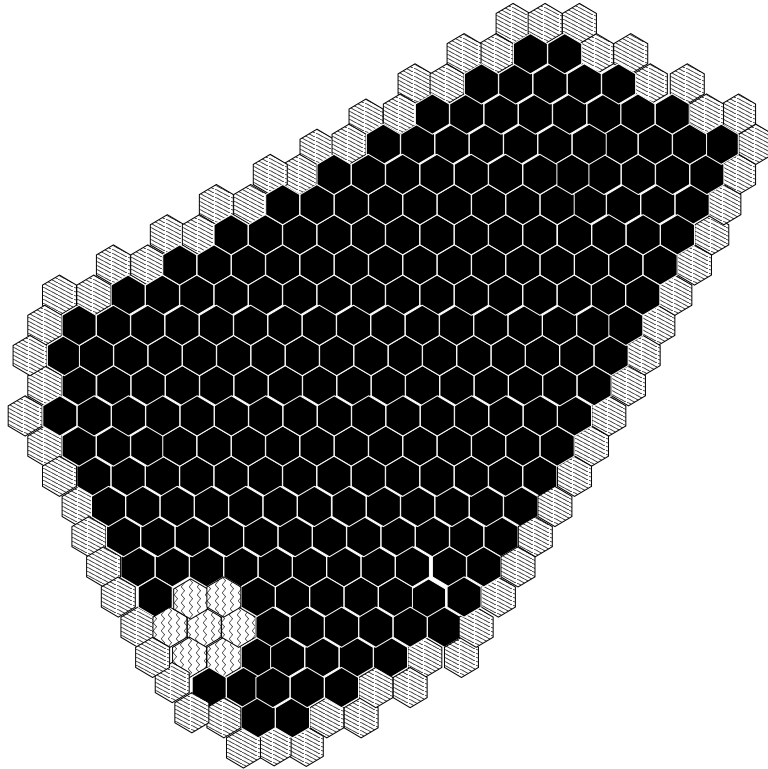


Figure 21: Pattern  $P$  for Protecting Hexagonal Grids.

Figure 21 shows the hexagonal grid when it has been protected using Pattern  $P$ .

## 8 Conclusion

The ability to surround a fire depends on the number of firefighters present, the number of beginning fires, how much of each vertex the firefighter saved, when the firefighters started protecting the grid, and the structure of the grid. In this thesis, we have looked at these cases for four types of grids: infinite rectangular grids, ribbon grids, the quarter plane grid, and the hexagonal grids.

We began with infinite rectangular grids in order to determine an optimal pattern. From this pattern, we found the minimum number of vertices burned but were unable to prove this assertion. We leave this as an open question.

From infinite rectangular grids, we looked at a subgraph of it, namely the ribbon grid. Here we restricted the  $y$  component. We determined that one firefighter was sufficient to surround the fire no matter where the fire started and found the minimum number of vertices burned.

Another subcase of infinite rectangular grids considered is the quarter plane. Here we looked at the first quadrant of a grid and wanted to know whether we could utilize only one firefighter as in the case of the ribbon grid. This assertion is not true unless a firefighter arrives on the scene early.

The idea of an early firefighter led to looking at fractional firefighters. These firefighters only protect certain portions of each vertex. Surprisingly, we showed that only one fractional firefighter is necessary to surround any fire in the infinite rectangular grid no matter when the firefighter arrives.

After gaining an understanding of infinite rectangular grids and their subgraphs, we next looked at a different type of grid, namely the hexagonal grid. The properties of this grid were different than the others that we considered. However, we conjectured that the optimal pattern we found in infinite rectangular grids also applied to these new grids.

## References

- [1] A.S. Finbow and B.L. Hartnell, On Designing a Network to Defend against Random Attacks of Radius Two, *Networks* **19** (1989) 771-792.
- [2] A.S. Finbow, B.L. Hartnell, Q. Li, K. Schmeisser, On Minimizing the Effects of Fire on a Virus on a Network, *J. Combin. Math. Combin. Comput.* **33** (2000) 311-322.
- [3] G. Gunther and B.L. Hartnell, Graphs with Isomorphic Survivor Graphs, *Congressus Numerantium* **79** (1990) 69-77.
- [4] B.L. Hartnell, The optimum defense against random subversions in a network, *Congressus Numerantium* **24** (1979) 493-499.
- [5] B.L. Harntell and Qiyan Li, Firefighting on Trees: How Bad is the Greedy Algorithm?, *Congressus Numerantium* **145** (2000) 187-192.
- [6] Dennis E. Shasha, Wildfires, *Dr. Dobbs Journal* (January 2001) 193-194.
- [7] P. Wang and S.A. Moeller, Fire Control on Graphs, *J. Combin. Math. Combin. Comput.* **41** (2000) 19-34.
- [8] D. West, "Introduction to Graph Theory", Prentice Hall, New Jersey (1996).

# A Java Source Code: Rectangular Grids

```
/******  
* AUTHOR : Patty Fogarty  
* SECTION : Thesis Research  
* DATE : September 2002- May 2003  
* TITLE : Catching the Fire on Grids  
*****/  
  
import java.awt.*;  
  
/******  
* Purpose: we want to protect the ribbon grid  
*  
* Methods  
* burnFire(): the main algorithm which decides whether protect or burnVertex is called  
* burnVertex(): decides which vertices are burned and sets the boolean matrix appropriately  
* firstRow(): goes and protects if the first burned vertex is on the first row  
* getBurned(): gives the burnedBefore matrix  
* getGrid(): gives the grid of the vertices as to whether they are burned, protected, or the first  
* burned vertex  
* protectVertex(): protects the vertices depending on where the vertex was  
* left(): if the fire started in the first row, use this to protect left side  
* right(): if the fire started in the first row, use this to protect right side  
* otherRow(): if the fire started somewhere else in the grid besides first row  
* returnTotal(): returns total number of vertices burned  
* leftSide(): if fire didn't start in first row, this protects left side  
* rightSide(): if fire didn't start in first row, this protects right side  
*  
* Private Variables  
* int burnX: the x value for the grid we are in for point 1  
* burnY: the x value for the grid we are in for point 2  
* int[][] gridPoints: this represents the grid and indicates which points on the grid  
* have been selected  
* boolean bothPoints: whether two points have been selected  
* burn: are we burning the vertices or are we protecting them  
* burnedBefore: the boolean vertex as to whether it was burned before  
* columns: how many columns the grid has  
* rows: how many rows the grid has
```

```

* anyBurn: the number of vertices that burned, or maybe we have burned all possible
*   but doesn't take the whole time
* below: have we protected below the vertex
* doneRow: the number of times we need to protect the grid
* first: we haven't protected any vertices and the fire starts in column 0
* total: the number of vertices burned at a specific time
*****/

```

```

public class FireAlgorithm {
    private int rows,columns,doneRow, burn, burnX,burnY,total,
        anyBurn;
    private int[][] gridPoints;
    private boolean[][] burnedBefore;
    private boolean below,first;

    public FireAlgorithm(int gridX, int gridY, int[][] grid,
        boolean[][] burned,int row, int column){
        burnX = gridX; burnY = gridY; gridPoints = grid;
        burnedBefore = burned;rows = row; columns = column;

        doneRow = 0; anyBurn = 0; total = 0;burn = 1;

        first = false; below = false;
    } end DrawingArea

    public boolean[][] getBurned(){
        return burnedBefore;
    }

    public int[][] getGrid(){
        return gridPoints;
    }

    public void firstRow(){
        int i, j;
        i = 0; j = 0;

        // trying to find where the fire started
        for(int row = 0; row< rows; row ++){
            for(int column=0; column < columns; column ++){

```

```

        if(gridPoints[row][column]==3&& first ==false){
            i = row;
            j = column;
        }

// protect the left side and then the right
if(doneRow<rows&& j!=0)
    left();
else
    right();
}

public void left(){
    boolean protect= false;
    for(int row = 0, place = 0; row < rows && protect == false;row ++){
        for (int column = 0; column < columns && protect == false;column ++){
            // trying to find where the fire started
            if(gridPoints[row][column] ==3){
                if((column-1)<0)
                    column = column +1;
                //check columns to the left to see if it was protected
                if(gridPoints[row][column -1] !=1){
                    gridPoints[row][column-1]=1;
                    protect = true;
                }
            }
            else if((row+1)<rows){
                column --;
                // check row below to see if it was protected
                for(int r = row; r<rows && protect == false; r ++){
                    if(gridPoints[r][column] !=2 && gridPoints[r][column]!=1 &&
                    gridPoints[r][column]!=3){
                        gridPoints[r][column]=1;
                        protect = true;
                    }
                }
            }
            else if((row+1)>=rows){
                column --;
                // check row above
                for(int r = row; r<rows && protect == false; r --){

```

```

        if(gridPoints[r][column] !=2 && gridPoints[r][column]!=1&&
        gridPoints[r][column]!=3){
            gridPoints[r][column]=1;
            protect = true;
        }
    }
}
}
}
}

public void right(){
    boolean protect= false;
    for(int row = 0, place = 0; row < rows && protect == false;row ++){
        for (int column = 0; column < columns && protect == false;column ++){
            // trying to find the original burned vertex
            if(gridPoints[row][column] ==3){
                if(doneRow==rows){
                    for(int col = column; col <columns && protect == false; col ++){
                        if(col+1 >=columns){
                            col --;
                        }
                    }
                }
                // trying to find the first column to protect
                else if(gridPoints[row][col]==2 && gridPoints[row][col+1]!=2 &&
                gridPoints[row][col+1]!=1){
                    gridPoints[row][col+1]=1;
                    protect= true;
                }
            }
        }
    }

    else if((row+1)<rows){
        column ++;
        for(int c = column; c <columns && protect == false;c ++){
            // finding the column to protect
            if(gridPoints[row][c]==2 && gridPoints[row][c+1]==1)
                column = c;
        }
        for(int r = row; r<rows && protect == false; r ++){
            // find the next column to protect

```



```

        //here we don't have enough columns to protect the whole right side
        else if(j < rows)
            doneRow = rows;
    }
// first protect the left side of the grid
if(j< rows)
    rightSide();
else if(doneRow<rows&& j!=0)
    leftSide();

// protect the right side of the grid
else
    rightSide();

} // end row()

public void leftSide(){
    boolean protect = false;
    int checking = 0;
    // first we want to protect right next to the fire
    for(int row = 0, place = 0; row < rows && protect == false; row ++){
        for (int column = 0; column < columns && protect == false; column ++){
            if(gridPoints[row][column] ==3){
                column --;
                if(gridPoints[row][column]!=2 && gridPoints[row][column] !=1){
                    gridPoints[row][column]=1;
                    checking++;
                    protect = true;
                }
            }
            else if(gridPoints[row][column]==1 && below == true){
                // have already protected next to it either protect below
                for(int r=row ; r < rows && protect == false ; r++){
                    for(int c = column; c >= 0 && protect == false;
                        c --){
                        if(c-1<0)
                            c --;
                        else if(gridPoints[r][c]==2 && gridPoints[r-1][c] ==1 &&
                            gridPoints[r][c-1]!=2 && gridPoints[r][c-1] !=1){
                            gridPoints[r][c-1] = 1;
                            protect = true;
                        }
                    }
                }
            }
        }
    }
}

```

```

        checking++;
    }

}

// or above
for(int ro=row; ro >=0 && protect == false ; ro--){
    for(int co = column; co >= 0 && protect == false; co --){
        if(co-1<0)
            co --;
        else if(gridPoints[ro][co]==2 && gridPoints[ro+1][co] ==1 &&
            gridPoints[ro][co-1]!=2 && gridPoints[ro][co-1] !=1){
            gridPoints[ro][co-1] = 1;
            protect = true;
            checking++;
        }
    }
}

// we haven't protected two vertices
if(protect == false){
    for(int r1=row; r1>=0&& protect== false; r1--){
        for(int c1=column; c1>=0&& protect==false;c1--){
            r1--;
            // looking from original row to top, original
            // column to left
            if(r1>=0)
                if(gridPoints[r1][c1] !=2 && gridPoints[r1+1][c1] ==1
                    && gridPoints[r1][c1+1]==2 && gridPoints[r1][c1]!=1){
                    gridPoints[r1][c1]=1;
                    protect=true;
                    checking++;
                }
            r1++;
        }
    }
    for(int r2=row; r2<rows&& protect== false; r2++){
        for(int c2=column; c2>=0&& protect==false;c2--){
            r2++;
            // looking from original row to top and
            // original column to left
            if(r2<rows)
                if(gridPoints[r2][c2] !=2 && gridPoints[r2-1][c2] ==1
                    && gridPoints[r2][c2+1]==2 && gridPoints[r2][c2]!=1){

```

```

        gridPoints[r2][c2]=1;
        protect=true;
        checking++;
    }
    r2--;
}
}
}
else if(below == false){
    gridPoints[row-1][column]=1;
    protect = true;
    below = true;
    checking++;
}
}

} //end for
} // end Leftside()

public void rightSide(){
    boolean protect = false;
    // first we want to protect to the right of the fire
    for(int row = 0, place = 0; row < rows && protect == false; row ++){
        for (int column = 0; column < columns && protect ==false;column ++){
            if(gridPoints[row][column] ==3){
                column ++;
                if(doneRow==rows){
                    for(int col = column; col<columns &&protect==false;col ++){
                        if((col+1)>= columns)
                            col ++;
                    }
                }
                else if(gridPoints[row][col-1]==3&&
                    gridPoints[row][col]!=2
                    && gridPoints[row][col]!=1 && first == false){
                    gridPoints[row][col]=1;
                    protect = true;
                    first = true;
                }
                else if(gridPoints[row][col]==2&&
                    gridPoints[row][col+1]!=2&
                    &gridPoints[row][col+1]!=1){

```

```

        gridPoints[row][col+1]=1;
        protect= true;
    }
}
else if(protect== false ){
// have already protected next to it
//either protect below
for(int r=row ; r < rows && protect == false ; r++)
    for(int c = column; c < columns && protect== false;c ++){
        if(gridPoints[r][c]==2 & gridPoints[r-1][c] ==1
            && gridPoints[r][c+1]!=2 && gridPoints[r][c+1] !=1){
            gridPoints[r][c+1] = 1;
            protect = true;
        }
// or above
for(int ro=row; ro >=0 && protect == false ; ro--){
    for(int co=column; co<columns && protect == false; co ++){
        if(gridPoints[ro][co]==2 && gridPoints[ro+1][co] ==1
            && gridPoints[ro][co+1]!=2 && gridPoints[ro][co+1] !=1){
            gridPoints[ro][co+1] = 1;
            protect = true;
        }
        else if(gridPoints[ro][co-1]==3 &&gridPoints[ro][co]==1
            && gridPoints[ro-1][co-1]==2 && gridPoints[ro-1][co]!=1
            && gridPoints[ro-1][co]!=2){
            gridPoints[ro-1][co]=1;
            protect= true;
        }
    }
}
if(protect == false){
    for(int r1=row; r1>=0&& protect== false; r1--){
        for(int c1=column; c1<columns &&protect==false;c1++){
            r1--;
            // finding the right vertex to
            //protect looking up in the rows and
            //to the right in columns
            if(r1>=0){
                if(c1+1 >= columns)
                    c1 ++;
            }
        }
    }
}

```

```

        else if(gridPoints[r1][c1] ==2 && gridPoints[r1+1][c1+1] ==1
        && gridPoints[r1][c1+1]!=2&& gridPoints[r1][c1+1]!=1){
            gridPoints[r1][c1+1]=1;
            protect=true;
        }
    }
    r1++;
}
for(int r2=row; r2<rows&& protect== false; r2++)
for(int c2=column;c2<columns &&protect==false;c2++){
    r2++;
    // finding the right vertex to
    //protect looking down in the rows and
    //to the right in columns
    if(r2<rows){
        if(c2+1>=columns)
            c2 ++;
        else if(gridPoints[r2][c2] ==2 && gridPoints[r2-1][c2+1] ==1
        && gridPoints[r2][c2+1]!=2&& gridPoints[r2][c2+1]!=1){
            gridPoints[r2][c2+1]=1;
            protect=true;
        }
        else if(gridPoints[r2][c2]==2 && gridPoints[r2][c2+1]!=2
        && gridPoints[r2][c2+1]!=1){
            gridPoints[r2][c2+1]=1;
            protect = true;
        }
    }
    r2--;
}
}
}
}
}

}

public void protectVertex(){

    if(burnY == 0 || burnY == rows -1)

```

```

        firstRow();
    else
        otherRow();
}

public void burnVertex(){
    int checkC,
        checkR;

    checkC = 0;
    checkR = 0;

    // here we burn more vertices
    for(int row= 0, place = 0; row < rows; row++){
        checkR = row;
        for(int column = 0; column < columns; column++){
            checkC = column;
            if(gridPoints[row][column] == 3 || gridPoints[row][column] == 2){
                if(burnedBefore[row][column] == true){

                    // want to check up
                    checkR --;
                    if(checkR < 0)
                        checkR ++;
                    else if (gridPoints[checkR][checkC] != 1 ) {
                        if(gridPoints[checkR][checkC] != 3 && gridPoints[checkR][checkC] !=2){
                            gridPoints[checkR][checkC] = 2;
                            checkR ++;
                            anyBurn ++;
                        }
                    }

                    // want to check down
                    checkR = row;
                    checkR ++;
                    if(checkR >= rows)
                        checkR --;
                    else if (gridPoints[checkR][checkC] != 1 ){
                        if(gridPoints[checkR][checkC] != 3 && [checkR][checkC] !=2){
                            gridPoints[checkR][checkC] = 2;

```

```

        checkR --;
        anyBurn ++;
    }
}

// want to check right
checkC ++;
checkR = row;
if(checkC >= columns)
    checkC --;
else if (gridPoints[checkR][checkC] != 1 ){
    if(gridPoints[checkR][checkC] != 3 && gridPoints[checkR][checkC] !=2 ){
        gridPoints[checkR][checkC] = 2;
        checkC --;
        anyBurn ++;
    }
}

// want to check left
checkC = column;
checkC --;
if(checkC < 0)
    checkC ++;
else if (gridPoints[checkR][checkC] != 1 ){
    if(gridPoints[checkR][checkC] != 3 && gridPoints[checkR][checkC] !=2 ){
        gridPoints[checkR][checkC] = 2;
        checkC ++;
        anyBurn ++;
    }
}
} // end if
} // end if
}

} // end for which goes through burned vertices

// now we have set the new burned vertices
for(int row= 0; row < rows; row ++){
    for(int column = 0; column < columns; column ++){
        if (gridPoints[row][column] == 2)
            burnedBefore[row][column] = true;
    }
}

```

```

    }
}
if(anyBurn ==0)
    doneRow = 2*rows+1;
}

public void burnFire(){
    int i;
    i = 2*rows;

    if(doneRow < i ){
        if(burn == 1) {
            protectVertex();
            burn = 0;
            doneRow ++;
            burnVertex();
            burn = 1;
            anyBurn = 0;
        }
    }
} // end burnFire()

public int returnTime(){
    if(doneRow<2*rows)
        return doneRow;
    else
        return doneRow-1;
}

public int returnTotal(){
    total = 0;
    for(int row= 0; row < rows; row ++){
        for(int column = 0; column < columns; column ++){
            if (gridPoints[row][column] == 2||
                gridPoints[row][column]==3)
                total ++;
        }
    }
} // end FireAlgorithm

```

## B Java Source Code: Square Grids

```
/******  
* AUTHOR : Patty Fogarty  
* SECTION : Thesis Research  
* DATE : September 2002- May 2003  
* TITLE : Catching the Fire on Grids  
*****/  
  
import java.awt.*;  
  
/******  
* Purpose: we want to protect the infinite rectangular grid  
*  
* Methods  
* burnFire(): the main algorithm which decides whether  
* protect or burnVertex is called  
* burnVertex(): decides which vertices are burned and sets  
* the boolean matrix appropriately  
* firstRow(): goes and protects if the first burned vertex is on the first row  
* getBurned(): gives the burnedBefore matrix  
* getGrid(): gives the grid of the vertices as to whether  
* they are burned, protected, or the first burned vertex  
* protectVertex(): protects the vertices depending on where the vertex was  
* afterNine(): time >= (9*where)-1 && time <= 19*where -1  
* endTime(): time > 19*where -1 && time <= 32*where+1  
* findHeight(): gives how many rows should be in the matrix  
* findWidth(): gives how many columns should be in the matrix  
* leftSide(): when the fire starts at time 0, how to protect left  
* makeGrid(): makes the matrix up  
* returnTime(): tells how long the fire has been burning  
* returnTotal(): gives the number of vertices burned at a certain time  
* rightSide(): when fire starts at time 0, how to protect right  
* xTime(): algorithm when we protect after x steps  
* zeroTime(): this gives the algorithm when we protect right away  
*  
* Private Variables  
* int burnX: the x value for the grid we are in for point 1  
* burnY: the x value for the grid we are in for point 2  
* int[][] gridPoints: this represents the grid and indicates
```

```

*   which points on the grid have been selected
*   boolean bothPoints: whether two points have been selected
*   burnedBefore: the boolean vertex as to whether it was burned before
*   columns: how many columns the grid has
*   rows: how many rows the grid has
*   below: are we done protecting below
*   both: have we protected both on the same side
*   burn: whether we protect or burn
*   diffCol, diffRow: the last row and column we protected on a certain side
*   doneRow: how long have we been protecting
*   first: we've protected the first place
*   lastCol, lastRow: the last row and column we protected on a certain side
*   protect: we've protected once
*   time: what time interval are we at
*   where: how many times for the grid to go unprotected
*****/

```

```

public class FireSquare {
    private int rows, columns,doneRow,diffRow,diffCol,
        burn,burnX,burnY,where,protect,twice,time,
        lastRow,lastCol;
    private int[][] gridPoints;
    private boolean[][] burnedBefore;
    private boolean below,both,first,running,upper;

    public FireSquare(int start){
        rows = 0; columns = 0; doneRow = 0; lastRow = 0; lastCol = 0;
        where = start; burnX=0; diffRow= 0; diffCol = 0;
        burnY=0; twice = 0; burn = 0; protect = 0; time = 0;
        both = false;running = false; first = false; upper = true;
    } // end DrawingArea

    public int findWidth(){
        columns = 26*where+10 ;
        burnX = 5*where+3;
        return columns;
    }

    public int findHeight(){

```

```

    rows = 24*where+10    ;
    burnY = 19* where+4 ;
    return rows;
}

public void makeGrid(){
    gridPoints = new int[rows][columns];
    burnedBefore = new boolean[rows][columns];
    for(int i = 0; i < rows; i ++){
        for(int j = 0; j < columns; j ++){
            gridPoints[i][j] = 0;
            burnedBefore[i][j] = false;
        }

        gridPoints[burnY][burnX] = 3;
        burnedBefore[burnY][burnX] = true;
    }

public void burnGraph(){
    if(where >0)
        makeGrid();
    else{
        rows = 15;
        columns = 15;
        burnY = 9;
        burnX = 5;
        makeGrid();
        burn = 1;
    }
}

public void burnFire(boolean run1){
    int i;
    running = run1;
    if(where !=0)
        i = 32*where +1;
    else
        i = 8;
    if(running == true && doneRow <=i){
        while(doneRow <= i){

```

```

        while(time < where-1){
            burnVertex();
            time ++;
        }
        if(burn == 1) {
            protectVertex();
            burn = 0;
            doneRow ++;
        }
        else {
            burnVertex();
            burn = 1;
        }
    }
}
else if(running == false &&doneRow <= i ){
    while(time < where-1){
        burnVertex();
        time ++;
    }
    if(burn == 1) {
        protectVertex();
        burn = 0;
        doneRow ++;
    }
    else {
        burnVertex();
        burn = 1;
    }
}
else
    return;
} // end burnFire()

public void burnVertex(){
    int checkC,
        checkR;

    checkC = 0;
    checkR = 0;

```

```

// here we burn more vertices
for(int row= 0, place = 0; row < rows; row++){
    checkR = row;
    for(int column = 0; column < columns; column++){
        checkC = column;
        if(gridPoints[row][column] == 3 || gridPoints[row][column] == 2){
            if(burnedBefore[row][column] == true){

                // want to check up
                checkR --;
                if(checkR < 0)
                    checkR ++;
                else if (gridPoints[checkR][checkC] != 1 ) {
                    if(gridPoints[checkR][checkC] != 3 && gridPoints[checkR][checkC] !=2){
                        gridPoints[checkR][checkC] = 2;
                        checkR ++;
                    }
                }

                // want to check down
                checkR = row;
                checkR ++;
                if(checkR >= rows)
                    checkR --;
                else if (gridPoints[checkR][checkC] != 1 ){
                    if(gridPoints[checkR][checkC] != 3&&gridPoints[checkR][checkC] !=2){
                        gridPoints[checkR][checkC] = 2;
                        checkR --;
                    }
                }
            }

            // want to check right
            checkC ++;
            checkR = row;
            if(checkR >= columns)
                checkC --;
            else if (gridPoints[checkR][checkC] != 1 ){
                if(gridPoints[checkR][checkC] != 3 && gridPoints[checkR][checkC] !=2){
                    gridPoints[checkR][checkC] = 2;
                }
            }
        }
    }
}

```

```

        checkC --;
    }
}

// want to check left
checkC = column;
checkC --;
if(checkR < 0)
    checkC ++;
else if (gridPoints[checkR][checkC] != 1 ){
    if(gridPoints[checkR][checkC] != 3&& gridPoints[checkR][checkC] !=2){
        gridPoints[checkR][checkC] = 2;
        checkC ++;
    }
}
} // end if
} // end if
}
} // end for which goes through burned vertices

// now we have set the new burned vertices
for(int row= 0, place = 0; row < rows; row++){
    for(int column = 0; column < columns; column++){
        if (gridPoints[row][column] == 2)
            burnedBefore[row][column] = true;
    }
}
}

public void protectVertex(){
    if(where == 0)
        zeroTime();
    else {
        xTime();
    }
}

public boolean[][] getBurned(){
    return burnedBefore;
}
}

```

```

public int[] [] getGrid(){
    return gridPoints;
}

public void zeroTime(){
    if(time ==0){
        for(int r= burnY; r < rows && protect !=2; r ++){
            for(int c = burnX; c > 0 && protect !=2; c --){
                // protecting the vertex to the left of the
                // original burned vertex
                if(gridPoints[r][c]==3 && gridPoints[r][c-1]!=1 && gridPoints[r][c-1]!=2){
                    gridPoints[r][c-1]=1;
                    protect ++;
                    diffRow = r;
                    diffCol = c-1;
                }
                // protecting the vertex below the original burned
                // vertex
                if(gridPoints[r][c]==3 && gridPoints[r+1][c]!=1 && gridPoints[r+1][c]!=2){
                    gridPoints[r+1][c]=1;
                    protect ++;
                    lastRow = r+1;
                    lastCol = c;
                }
            }
        }
        if(protect == 2){
            protect = 0;
            time ++;
            both = false;
            below = false;
        }
    }

    else if(below == false){
        // finding the right next two vertices to protect...
        // we want to protect the bottom part twice first and
        // then start alternating
        if(gridPoints[lastRow][lastCol] == 1&& gridPoints[lastRow-1][lastCol+1] == 2

```

```

    && gridPoints[lastRow][lastCol+1] != 1 && gridPoints[lastRow][lastCol+1] != 2){
        gridPoints[lastRow][lastCol+1]=1;
        protect ++;
    }
    if(gridPoints[lastRow-1][lastCol+1] == 2&& gridPoints[lastRow-1][lastCol+2] != 1
    && gridPoints[lastRow-1][lastCol+2] != 2){
        gridPoints[lastRow-1][lastCol+2]=1;
        protect ++;
        lastRow = lastRow - 1;
        lastCol = lastCol +2;
    }
    if(protect ==2){
        time ++;
        below = true;
        both = true;
        protect = 0;
    }
}
else if(time < 5){
    leftSide();
}
else if(time >=5 && time < 8)
    rightSide();
}

public void leftSide(){
    if(both == true){
        for(int row = lastRow; protect !=2 && row < rows-2; row --){
            for(int col = lastCol; protect !=2 && col >0; col --){
                if(time == 2){
                    // we don't want to alternate yet
                    if(gridPoints[row][col]==1 && gridPoints[row-1][col-1]!=1&&
                    gridPoints[row-1][col-1]!=2){
                        gridPoints[row-1][col-1]=1;
                        protect ++;
                    }
                    if(gridPoints[row-1][col+1]==2 && gridPoints[row-1][col]!=1&&
                    gridPoints[row-1][col]!=2){
                        gridPoints[row-1][col]=1;
                        protect ++;
                    }
                }
            }
        }
    }
}

```

```

    lastRow = row -1;
    lastCol = col;

}
}
// protecting the bottom portion first before
// starting to move up
else if(time <4) {
    if(gridPoints[row][col]==1 && gridPoints[row-1][col+1]==2&&
gridPoints[row-1][col]!=1 && gridPoints[row-1][col]!=2){
        gridPoints[row-1][col]=1;
        protect ++;
    }
    if(gridPoints[row-1][col]==1 && gridPoints[row-2][col+2]==2&&
gridPoints[row-2][col+1]!=1 &&gridPoints[row-2][col+1]!=2){
        gridPoints[row-2][col+1]=1;
        protect ++;
        lastRow = row -2;
        lastCol = col+1;
    }
}
}
else if(time == 4){
    if(gridPoints[row][col]==1&& gridPoints[row-1][col+1]==2&&
gridPoints[row-1][col]!=1 && gridPoints[row-1][col]!=2){
        gridPoints[row-1][col]=1;
        protect ++;
    }
    if(gridPoints[row][col]==1 && gridPoints[row-1][col+1]==2&&
gridPoints[row-2][col+1]!=1 && gridPoints[row-2][col+1]!=2){
        gridPoints[row-2][col+1]=1;
        protect ++;
        lastRow = row-2;
        lastCol = col+1;
    }
}
}
}
if(protect == 2){
    both = false;
    protect = 0;
}

```

```

        time ++;
    }
} // end if
else{
    if(gridPoints[lastRow][lastCol]==1 && gridPoints[lastRow-1][lastCol]!=1
    && gridPoints[lastRow-1][lastCol]!=2){
        gridPoints[lastRow-1][lastCol]=1;
        protect ++;
        lastRow = lastRow-1;
        lastCol = lastCol;
    }
    for(int r1 = diffRow; protect !=2 && r1 >0; r1 --){
        for(int c1 = diffCol; protect !=2 && c1 < columns -1; c1 ++){
            if(gridPoints[r1-1][c1]==2 && gridPoints[r1-1][c1+1]!=1&&
            gridPoints[r1-1][c1+1]!=2){
                gridPoints[r1-1][c1+1]=1;
                protect ++;
                diffRow= r1-1;
                diffCol = c1+1;
            }
        }
    }
    if(protect == 2){
        both = true;
        protect = 0;
        time ++;
    }
} // end else
}

public void rightSide(){
    if(both == true){
        for(int row = lastRow; protect !=2&& row < rows-1; row --){
            for(int col = lastCol; protect !=2 && col < columns-2; col ++){
                if(gridPoints[row][col]==1 && gridPoints[row][col+1]!=1&&
                gridPoints[row][col+1]!=2){
                    gridPoints[row][col+1]=1;
                    protect ++;
                }
                if(gridPoints[row][col+1]==1 && gridPoints[row+1][col+1]==2&&

```

```

        gridPoints[row+1][col+2]!=1 &&gridPoints[row+1][col+2]!=2){
            gridPoints[row+1][col+2]=1;
            protect ++;
            lastRow = row +1;
            lastCol = col+2;
        }
    }
}
if(protect == 2){
    both = false;
    protect = 0;
    time ++;
}
} // end if
else{
    if(gridPoints[lastRow][lastCol]==1 && gridPoints[lastRow][lastCol+1]!=1
    && gridPoints[lastRow][lastCol+1]!=2){
        gridPoints[lastRow][lastCol+1]=1;
        protect ++;
        lastCol = lastCol +1;
    }
    for(int r1 = diffRow; protect !=2&& r1 >0; r1 --){
        for(int c1 = diffCol; protect !=2 && c1 < columns -1; c1 ++){
            if(gridPoints[r1-1][c1]==2 && gridPoints[r1-1][c1+1]!=1&&
            gridPoints[r1-1][c1+1]!=2){
                gridPoints[r1-1][c1+1]=1;
                protect ++;
                diffRow= r1-1;
                diffCol = c1+1;
            }
        }
    }
    if(protect == 2){
        both = true;
        protect = 0;
        time ++;
    }
} // end else
}

```

```

public void xTime(){
    // here we are protecting the bottom portion and want to do
    // this twice
    if(protect < 2 && below == false){
        for(int r = burnY; r < rows && protect !=2 && first == false; r ++){
            for(int c = burnX; c >= 0 && below != true && first == false; c --){
                if(gridPoints[r][c] == 2 && gridPoints[r][c-1] !=2 &&
                    gridPoints[r][c-1] !=1 ){
                    gridPoints[r][c-1] = 1;
                    protect ++;
                    time ++;
                    first = true;
                    // remember the column and row last protected
                    // when alternating
                    if(protect == 1){
                        lastRow = r;
                        lastCol = c-1;
                    }
                    gridPoints[r+1][c] = 1;
                }
                if(protect ==2){
                    below = true;
                    protect=0;
                }
            }
            first = false;
        }
    }

    // here we want to protect two together
    else if(time < (9*where)-1&& both == true){
        for(int row = burnY; protect !=2 && row < rows && time <(9*where)&&
            both == true; row ++){
            for(int col = burnX; protect !=2 && col >= 0 && both == true
                && time < 3*where; col --){
                if(gridPoints[row][col] == 2 && gridPoints[row][col-1] !=1 &&
                    gridPoints[row][col-1] !=2 ){
                    gridPoints[row][col-1] = 1;
                    protect ++;
                }
            }
        }
    }
}

```

```

    if(protect == 2){
        both = false;
        protect = 0;
        time ++;
    } // end if
} // end for
for(int row1 = diffRow; protect !=2 && row1 >=0 && time <(9*where-1)
    && both == true; row1 --){
    for(int col1 = diffCol; protect !=2 && col1 >= 0 && both == true &&
        time < 9*where; col1 ++){
        if(gridPoints[row1][col1] == 1 && gridPoints[row1][col1+1] !=1
            &&gridPoints[row1][col1+1] !=2 ){
            gridPoints[row1][col1+1] = 1;
            protect ++;
        } // end if
        if(gridPoints[row1-1][col1+1]==2 &&gridPoints[row1-1][col1+2]!=2
            && gridPoints[row1-1][col1+2]!=1){
            gridPoints[row1-1][col1+2]=1;
            protect ++;
            diffRow = row1 -1;
            diffCol = col1 +2;
        } // end if
        if(time < 3*where){
            if(gridPoints[row1][col1+1] ==2 &&gridPoints[row1+1][col1] !=1
                &&gridPoints[row1+1][col1]!=2 &&gridPoints[row1+1][col1+1]==2){
                gridPoints[row1+1][col1] =1;
                protect ++;
                gridPoints[row1+2][col1+1]=1;
                protect++;
            } // end if
        }
    } // end for
} // end for
    if(protect == 2){
        both = false;
        time ++;
        protect = 0;
    } // end if
}

```

```

}
// here we want to protect one in the upper portion and one
// in the lower portion
else if(both == false && time < 9*where-1){
    if(gridPoints[lastRow-1][lastCol ] ==2 && gridPoints[lastRow-1][lastCol-1] !=2
    && gridPoints[lastRow-1][lastCol-1] !=1){
        gridPoints[lastRow-1][lastCol-1] =1;
        protect ++;
        time ++;
        lastRow = lastRow - 1;
        lastCol = lastCol -1;
    }
    for(int i = burnY; i < rows -1 && protect !=2; i ++){
        for(int j = burnX - where; j < columns -1 && protect !=2; j ++){
            if(time < 3*where){
                if(gridPoints[i][j+1] ==2 && gridPoints[i+1][j] !=1&&
                gridPoints[i+1][j] !=2 && gridPoints[i+1][j+1] ==2 ){
                    gridPoints[i+1][j] =1 ;
                    protect ++;
                    diffRow = i+1;
                    diffCol = j;
                }
            }
            else{
                if(gridPoints[i][j] ==1 && gridPoints[i][j+1] !=1&&
                gridPoints[i][j+1] !=2 ){
                    gridPoints[i][j+1] =1 ;
                    protect ++;
                    diffRow = i;
                    diffCol = j+1;
                }
            }
        }
    }

    if (protect == 2){
        both = true;
        protect = 0;
    }
}
else if(time >= (9*where)-1 && time <= 19*where -1){

```

```

        if(time == 9*where -1)
            both = true;
        afterNine();
    }
    else if( time> 19*where -1&& time <=32*where+(where-1))
        endTime();
}

public void afterNine(){
    // we want to protect two together
    if(both == true){
        for(int row = lastRow; protect !=2 && row < rows-2; row --){
            for(int col = lastCol; protect !=2&& col >0; col --){
                if(time == 9*where-1){
                    if(gridPoints[row][col]==1 && gridPoints[row-1][col-1]!=1&&
                    gridPoints[row-1][col-1]!=2){
                        gridPoints[row-1][col-1]=1;
                        protect ++;
                    }
                    if(gridPoints[row-1][col+1]==2 && gridPoints[row-1][col]!=1&&
                    gridPoints[row-1][col]!=2){
                        gridPoints[row-1][col]=1;
                        protect ++;
                        lastRow = row -1;
                        lastCol = col;
                    }
                }
            }
            else if(time <19*where-1) {
                if(gridPoints[row][col]==1 && gridPoints[row-1][col+1]==2&&
                gridPoints[row-1][col]!=1 && gridPoints[row-1][col]!=2){
                    gridPoints[row-1][col]=1;
                    protect ++;
                }
                if(gridPoints[row-1][col]==1 && gridPoints[row-2][col+2]==2&&
                gridPoints[row-2][col+1]!=1 &&gridPoints[row-2][col+1]!=2){
                    gridPoints[row-2][col+1]=1;
                    protect ++;
                    lastRow = row -2;
                    lastCol = col+1;
                }
            }
        }
    }
}

```

```

    }
}

// we know which vertex to protect at this time
else if(time == 19*where -1){
    if(gridPoints[row][col]==1 && gridPoints[row-1][col+1]==2&&
        gridPoints[row-1][col]!=1 && gridPoints[row-1][col]!=2){
        gridPoints[row-1][col]=1;
        protect ++;
    }
    if(gridPoints[row][col]==1 && gridPoints[row-1][col+1]==2&&
        gridPoints[row-2][col+1]!=1 && gridPoints[row-2][col+1]!=2){
        gridPoints[row-2][col+1]=1;
        protect ++;
        lastRow = row-2;
        lastCol = col+1;
    }
}
}
}
if(protect == 2){
    both = false;
    protect = 0;
    time ++;
}
} // end if

// we want to protect using the alternating strategy
else{

    // place one in the last row we protected
    if(gridPoints[lastRow][lastCol]==1 &&gridPoints[lastRow-1][lastCol]!=1
        && gridPoints[lastRow-1][lastCol]!=2){
        gridPoints[lastRow-1][lastCol]=1;
        protect ++;
        lastRow = lastRow-1;
        lastCol = lastCol;
    }

    // find the first vertex to protect by looking one row

```

```

// back and one column to right
for(int r1 = diffRow; protect !=2 && r1 >0; r1 --){
    for(int c1 = diffCol; protect !=2 && c1 < columns -1; c1 ++){
        if(gridPoints[r1-1][c1]==2 && gridPoints[r1-1][c1+1]!=1&&
        gridPoints[r1-1][c1+1]!=2){
            gridPoints[r1-1][c1+1]=1;
            protect ++;
            diffRow= r1-1;
            diffCol = c1+1;
        }
    }
}
if(protect == 2){
    both = true;
    protect = 0;
    time ++;
}
} // end else
}

public void endTime(){
    // want to protect two together
    if(both == true){
        // find the last row we protected: place one in that
        // row, one in the row below
        for(int row = lastRow; protect !=2&& row < rows-1; row --){
            for(int col = lastCol; protect !=2 && col < columns-2; col ++){
                if(gridPoints[row][col]==1 && gridPoints[row][col+1]!=1&&
                gridPoints[row][col+1]!=2){
                    gridPoints[row][col+1]=1;
                    protect ++;
                }
                if(gridPoints[row][col+1]==1&& gridPoints[row+1][col+1]==2&&
                gridPoints[row+1][col+2]!=1&& gridPoints[row+1][col+2]!=2){
                    gridPoints[row+1][col+2]=1;
                    protect ++;
                    lastRow = row +1;
                    lastCol = col+2;
                }
            }
        }
    }
}

```

```

    }
    if(protect == 2){
        both = false;
        protect = 0;
        time ++;
    }
} // end if

// want to protect using alternating strategy
else{
    // protect the right hand side by moving up and over one
    if(gridPoints[lastRow][lastCol]==1 &&gridPoints[lastRow][lastCol+1]!=1
    && gridPoints[lastRow][lastCol+1]!=2){
        gridPoints[lastRow][lastCol+1]=1;
        protect ++;
        lastCol = lastCol +1;
    }

    // find the last two protected and move down one and
    // over by one column
    for(int r1 = diffRow; protect !=2 && r1 >0; r1 --){
        for(int c1 = diffCol; protect !=2 && c1 < columns -1; c1 ++){
            if(gridPoints[r1-1][c1]==2&& gridPoints[r1-1][c1+1]!=1&&
            gridPoints[r1-1][c1+1]!=2){
                gridPoints[r1-1][c1+1]=1;
                protect ++;
                diffRow= r1-1;
                diffCol = c1+1;
            }
        }
    }
    if(protect == 2){
        both = true;
        protect = 0;
        time ++;
    }
} // end else
}

public int returnTime(){
    return time;
}

```

```
}  
public int returnTotal(){  
    int total = 0;  
    for(int row= 0; row < rows; row ++)  
        for(int column = 0; column < columns; column ++){  
            if (gridPoints[row][column] == 2||  
                gridPoints[row][column]==3)  
                total ++;  
        }  
    return total;  
}  
  
} // end FireSquare
```