Self-stabilization under Minimal Liar's Domination for Path Graphs

Rebecca Lynn Heinen

SENIOR HONORS THESIS
Submitted In Partial Fulfillment of Requirements of the College Scholars Program
North Central College

4/28/15

Approved:  
Thesis Director Signature
Dr. Neil Nicholson

Approved:  
Second Reader Signature
Dr. Caroline St. Clair
Abstract

Self-stabilizing algorithms ensure that a system that starts in an incorrect state will eventually come to a correct state in a finite number of moves and will remain in that correct state. The first part of this project focuses on creating a self-stabilizing algorithm for minimal liar’s domination for any path graph. The conditions of liar’s domination state that every vertex in the graph is at least doubly dominated and for any two unique vertices, there are at least three vertices in the set (a collection of vertices possessing a particular property) in the collective closed neighborhood. This project also aims to increase an undergraduate student’s understanding of graph theory within the context of self-stabilization. Minimal liar’s domination is important for maintaining the security of networks; therefore, this research can be used to ensure that an entire network of computers is secure under minimal liar’s domination at all times.
Acknowledgements

I owe much of my successes in mathematics to Dr. Neil Nicholson who has served as my academic and thesis advisor. If it was not for his open door policy, even at 7:00am, I would not have switched my major to mathematics. He enabled me to grow as a mathematician by challenging me through my coursework and my writings. Secondly, this thesis is largely composed of research that was completed at an NSF sponsored REU at St. Mary’s College of Maryland with Abiola Akanni (St. Mary’s College of Maryland), EmmaJulia Jones (Centre College), and Luis Urritia (University of Illinois-Urbana-Champaign). For that, I would like to thank our REU director Dr. Dave Kung. I owe a huge thank you to Dr. Alan Jamieson for being an awesome research advisor and mentor by providing us with guidance throughout the research process. Finally, I would like to thank my second reader Dr. Caroline St. Clair and my thesis librarian Emily Prather-Rodgers.
Preface

A common response I hear when I tell people that I am involved with mathematics research is, “Why do we need mathematics research? Don’t we already know all there is to know about math?” Prior to doing research, I might have agreed that we do know a good portion of mathematics already. However, I now know that is not the case at all. In our Nature of Proof in Mathematics course, Dr. Nicholson introduced us to a great analogy that compares our knowledge of mathematics to a tree. The trunk of the tree includes general mathematics knowledge such as arithmetic, algebra, and geometry. Branches represent different areas of concentrations such as graph theory, knot theory, abstract algebra, analysis, etc. These branches continue to branch off, forming specializations within various concentrations. The tree grows due to the lengthening of these smaller branches, which occurs through the development of “new mathematics.” Doing undergraduate research programs allows students to discover other fields of mathematics that we are not typically introduced to through our undergraduate courses. Initially, the concepts of graph theory, self-stabilization, and minimal liar’s domination were foreign to me. I attempted to read articles given to us by our advisor before attending the REU, but it was far above me. I am certainly not alone; many undergraduate students fail to understand mathematics concepts by just reading through a published paper. Throughout this thesis, I plan to introduce the concepts of graph theory, self-stabilization, and minimal liar’s domination in a way so that it is easily obtainable to an undergraduate student.
# Contents

1. Introduction to Graph Theory ........................................... 4
   1.1. Self-stabilization ................................................. 7
   1.2. Minimal Liar’s Dominating Set .................................. 10

2. Self-stabilizing Algorithm under Minimal Liar’s Domination
   for Path Graphs .......................................................... 11

3. Network Application ..................................................... 15

4. Conclusion ................................................................. 17
1 Introduction to Graph Theory

Imagine you are moving back to college and you suddenly realize that you have a few errands to run. Let’s assume that you need to go to the bookstore (B) to pick up your textbooks for the new year, school post office (P) to pick up the key to your mailbox, and lastly to the store (S) to buy some snacks for your room. For simplicity, let’s say you have a car on campus and you plan to drive to these various locations. We’ll also assume the starting and ending location is the dorm (D). We can use Figure 1 to help us plan your trip.

![Diagram of locations](image)

Figure 1: Diagram depicting the locations of the bookstore (B), school post office (P), store (S), and dorm (D).

In Figure 1, the empty circles each represent a particular location. The lines connecting the empty circles represent roads that connect one location to another. While planning your trip, you realize you want to get ice cream so you should probably go to the store last before returning to your dorm. Therefore, it should be pretty clear that the most efficient route is dorm, bookstore, post office, store, and back to the dorm. Although these types of scenarios are rarely ever envisioned using diagrams, they illustrate that we use graph theory throughout our lives without knowing it. Even things as simple as drawing a map for someone in need of directions requires graph theory.

At this particular point, we need to establish that these types of graphs are not the typical graphs of functions we often think of involving the $xy$-plane. We define a graph in the following way.

**Definition 1.1 ([7]).** A graph $G$ consists of a non-empty set of elements, called vertices, and a list of unordered pairs of these elements, called edges.
For example, the diagram we created earlier has four vertices, each representative of a destination point, and four edges, each representing roads connecting two destination points.

**Definition 1.2 ([7]).** The set of vertices of the graph $G$ is called the **vertex-set** of $G$, denoted $V(G)$.

In Figure 1, $V(G) = \{D, S, B, P\}$

**Definition 1.3 ([7]).** The list of edges is called the **edge-list** of $G$, denoted by $E(G)$.

In Figure 1, $E(G) = \{DS, DB, BP, SP\}$.

Thus, notice $G$ is completely determined by $V(G)$ and $E(G)$. We will then denote a graph $G$ as $G = (V, E)$. From just our short example of the college student, one can imagine that graphs can be very useful for mathematical modeling. In order to model specific situations, we often use various graphs of similar structures. The following figures are examples of the various types of graphs that are most often used in graph theory.

![Diagram 2: A $P_5$ graph with five vertices.](image)

Figure 2: This is a $P_5$ graph, which is a path graph with five vertices.

![Diagram 3: A $T_7$ graph with seven vertices.](image)

Figure 3: This is a $T_7$ graph, which is a tree graph with seven vertices.
Figure 4: This is a $C_5$ graph, which is a cycle graph with five vertices.

Figure 5: This is a $K_5$ graph, which is a complete graph with five vertices.

Figure 6: This is a $G_{3,3}$ graph, which is a grid graph with three rows and three columns.

This thesis will primarily focus on path graphs, denoted $P_n$, where $n$ is the number of vertices in the graph.
1.1 Self-stabilization

Before introducing the specific components necessary for self-stabilization, it is important to understand the overall concept of self-stabilization. To do so, we define self-stabilization in the following way.

**Definition 1.4.** _Self-stabilization_ is the process by which a system stabilizes under some condition in a finite number of actions, regardless of the initial state.

This property is essential to many different types of real world processes because a “system” can be modeled through mathematical graphs. For example, a computer network is an example of a system that can undergo self-stabilization. Furthermore, Dolev, author of _Self-Stabilization_, offers the example of a space shuttle to demonstrate the importance of such processes. He states, “If a space shuttle experiences a fault because of a momentary electrical power problem, it may malfunction for a while, but it will not be lost as long as it uses a self-stabilizing algorithm for its control, since this algorithm will cause the control to recover automatically and continue in its task,” [2].

While self-stabilization is applicable to many real world problems, that is not the only reason why people study self-stabilizing algorithms. Edsger Dijkstra introduced a new paradigm for self-stabilization with his paper _Self-stabilizing Systems in Spite of Distributed Control_ [1]. Because Dijkstra’s work was ahead of its time in 1974, this research prompted further questions on self-stabilization for various systems. Dolev also mentions that Dijkstra’s work sparked researchers to create new conditions and invent self-stabilizing algorithms under various conditions simply because it is fun and exciting [2].

Now that we have developed an overall understanding of the concept of self-stabilization, we will begin to describe the process in much more detail. To begin, we will first define a self-stabilizing algorithm because self-stabilization cannot occur without some self-stabilizing algorithm.

**Definition 1.5.** A _self-stabilizing algorithm_ is a set of rules, which, through a series of actions, allow a system to stabilize in a finite number of actions and remain stabilized.

**Definition 1.6.** A _rule_ is a statement that consists of a predicate and a particular action to be completed if the predicate is fulfilled.
Typically, rules are of the form of an “if, then” statement in which the “if” part is the predicate and is followed by the action that must be completed, which is defined in the “then” statement. Now we identify terms particular to self-stabilization under minimal liar’s domination.

**Definition 1.7** ([1, 4]). A vertex is privileged when it satisfies the predicate of a particular rule.

After becoming privileged, a vertex completes the action when prompted to do so by a serial daemon.

**Definition 1.8.** A serial daemon is typically thought of as a background program that allows privileged vertices to complete the specified action one at a time.

At this particular point, it is important to notice that we humans can be daemons when we choose which privileged vertices are allowed to complete the action.

**Definition 1.9.** Within $V(G)$ we are interested in a specific subset of vertices that possess a certain property. We will refer to this particular subset as the set and denote it by $S$. Visually, shaded vertices are in $S$ while unshaded vertices are not in $S$. In particular, we define a function $x : V \to \{0, 1\}$ by

$$x(i) = \begin{cases} 
0 & : i \notin S \\
1 & : i \in S
\end{cases}$$

Notice in Figure 1 none of the vertices are in the set. However, imagine that you have a coupon for the bookstore and the store. Because that is a particular quality that is unique to the bookstore and the store, we can denote that in our graph by shading in the vertices $B$ and $D$, which is depicted in Figure 7.
In Example 1.13 of the next section, vertex 1 is in $S$ and vertex 5 is not in $S$. If $i$ is vertex 1, then $x(i) = 1$. If $i$ is vertex 5, then $x(i) = 0$. In real world applications, a vertex $i$ is in $S$ if it has some specific property. In a computer network, for example, computers may be in the set if they have an antivirus program.

To summarize, vertices become privileged if they satisfy the predicate of a rule that is defined within the self-stabilizing algorithm. After becoming privileged, they are chosen one at a time to complete the desired action, which typically requires that the vertices move in or out of the set $S$. In other words, vertices once shaded may be unshaded or vertices once unshaded may be shaded in the final stabilized form. Whether or not a vertex is in the set in the stabilized form is not entirely dependent on the original state of the graph. It is however dependent on the order in which the daemon chooses to allow privileged vertices to complete the desired action.

The following two definitions are required to define minimal liar’s domination and for the proof of our main result.

**Definition 1.10** ([4]). The *open neighborhood of a vertex* $i$, denoted as $N(i)$, is the set of vertices adjacent to $i$.

**Definition 1.11** ([4]). The *closed neighborhood of a vertex* $i$, denoted as $N[i]$, is $N[i] = N(i) \cup \{i\}$.

In Figure 7, notice that $N(D) = \{S, B\}$ and $N[D] = \{S, B, D\}$. 

![Figure 7: Diagram depicting the locations of the bookstore ($B$), school post office ($P$), store ($S$), and dorm ($D$). Vertices $B$ and $D$ are shaded in to represent locations that have a coupon.](image)
1.2 Minimal Liar’s Dominating Set

Our primary goal is to create and prove correct a self-stabilizing algorithm for minimal liar’s domination for path graphs. Before we define minimal liar’s domination, we need to understand what domination means.

Definition 1.12 ([3]). Let $G = (V, E)$ be a graph with set $S$. Let $i \in S$. Then, $i$ is said to dominate all the vertices in $N[i]$. The collection of vertices in $S$ together form a subset of $G$ and are called a dominating set.

In less complex words, only vertices in the set dominate. When they dominate, they not only dominate themselves but also their neighboring vertices.

Example 1.13. Notice vertex 1 dominates itself and its only neighbor, vertex 2. Vertex 3 dominates itself and its two neighbors, 2 and 4. Lastly, vertex 4 dominates itself and its neighbors 3 and 5. We say vertices 3 and 4 are doubly dominated.

Now that we understand domination, we define liar’s domination.

Definition 1.14 ([5]). A set $D \subseteq V(G)$ is a liar’s dominating set if every vertex $i \in V(G)$ is at least doubly dominated and $|(N[i] \cup N[j]) \cap S| \geq 3$ for every $i, j \in V(G)$ where $i \neq j$.

In other words, the conditions for liar’s domination are that every vertex is at least doubly dominated and for any pair of unique vertices there must be at least three vertices in the set in their collective closed neighborhood.

We have defined what a liar’s dominating set is, but what does minimal mean? Why should we include minimal? If the set were just called a liar’s dominating set, we could just move all the vertices into the set because that certainly will fulfill the requirements for liar’s domination. However, it is very inefficient to simply put all the vertices in the set when we could use fewer vertices to create a stabilized graph under liar’s domination.

Definition 1.15. A set is minimal with respect to a certain parameter if a vertex that is in the set cannot be removed without violating the conditions for a certain parameter (liar’s domination for our purposes).

In the next section, Algorithm 2.1 is a self-stabilizing algorithm for path graphs under minimal liar’s domination.
2 Self-stabilizing Algorithm under Minimal Liar’s Domination for Path Graphs

To prove self-stabilizing algorithms are correct, researchers must show closure and convergence. Closure implies that once a system stabilizes, it will do so in a desired state. For this thesis, we want to show that a system will stabilize as a minimal liar’s dominating set. Convergence means that a system will stabilize in a finite number of actions and remain stabilized under our algorithm. This section includes our algorithm, a short example, and the results showing the algorithm exhibits both closure and convergence.

Algorithm 2.1. Let R1 and R2 denote the following two rules that are used to privilege and move vertices in and out of $S$. R1: If a vertex is in $S$ and has at least three immediate neighbors to both its left and right that are in $S$, then the vertex is moved out of $S$. R2: If a vertex is not in $S$ and has fewer than three immediate neighbors to both its left and right that are in $S$, then the vertex is moved into $S$.

Example 2.2. We will stabilize the following unstabilized $P_{12}$ graph, labeled $P1$ below, through a series of actions as defined by the rules outlined in Algorithm 2.1. Vertices labeled with a $P$ are privileged and have the potential to complete an action. When we refer to a vertex by its number, it assumed that we are numbering them from left to right in numerical order.

P1 is the unstabilized graph that we wish to stabilize. Notice vertex 3 is not doubly dominated. This is just one example of why this graph is not stabilized. The following sequence of steps show how P1 can be stabilized.

P1

This is the starting graph.

P2

Vertices 3, 4, and 6 become privileged by R2 of Algorithm 2.1.

P3

The daemon allows vertex 6 to move into $S$, but as that occurs vertices 8 and 9 become privileged by R1 of Algorithm 2.1.
When the daemon allows vertex 8 to leave \( S \), vertex 9 looses its privilege because while it is in \( S \) it no longer has three immediate neighbors to both its left and right that are in \( S \).

The daemon allows vertex 3 to move into \( S \). When vertex 3 moves into \( S \), vertex 4 looses its privilege because while it is not in \( S \) it has three immediate neighbors to both its left and right that are in \( S \). Now the graph is stabilized.

P1 can stabilize into a number of final forms as shown below.

![Figure 8: Other stabilized forms of P1.](image)

These are two other ways that this particular \( P_{12} \) can be stabilized. The reason graphs can be stabilized in multiple ways is because a daemon can choose a different order in which to move privileged vertices. A certain choice may yield different successive choices, ultimately resulting in a different stabilized graph. To prove that the use of Algorithm 2.1 stabilizes any sized path graph, regardless of its initial state, we will prove the following results. To do so, we will only refer to rules outlined in Algorithm 2.1.

**Lemma 2.3.** Once a vertex is privileged by \( R1 \) and moves out of the set, it will no longer become privileged again.

**Proof.** Let \( P_n \) be a path graph with \( S \subseteq V \) that has not stabilized to a minimal liar’s dominating set. Assume a vertex \( i \in S \) is privileged by \( R1 \). This implies that there are at least three vertices to \( i \)’s immediate left and right that are in the set. Next, assume a daemon chooses to allow \( i \) to move out of the set. Since \( i \) was privileged by \( R1 \), it has at least three immediate neighbors to both its left and right side that are in the set. Thus, \( i \) is out
of the set, while at least three immediate neighbors to the left and right of \( i \) are in the set.

Consider \( j \in S \) such that \( j \) is one of \( i \)'s three immediate neighbors to its left or right. Then, \( j \) will not be privileged by \( R1 \) since \( i \notin S \) and would be included in \( j \)'s three immediate neighbors on its left and right. Therefore, \( j \) will never be privileged by \( R1 \) since \( j \in S \), but \( j \) has less than three immediate neighbors in the set to its left and right. Therefore, \( i \) can no longer be privileged because \( R2 \) states \( i \) must be out of the set and have less than three immediate neighbors to the left and right of \( i \). Consequently, once a vertex moves out of the set by \( R1 \), it cannot be privileged again. \( \square \)

**Lemma 2.4.** Path graphs \( P_3, P_4, P_5, \) and \( P_6 \) will have all vertices in the set as a minimal liar’s dominating set.

**Proof.** Let \( P_n \) be a path graph with \( 3 \leq n \leq 6 \). We want to show that every vertex will need to be in the set to be a minimal liar’s dominating set. The parameter states that every vertex must be dominated at least twice and every pair of vertices must have \( |(N[a] \cup N[b]) \cap S| \geq 3 \).

First, the first and last vertices must be in the set in order to dominate themselves, since they only have one available neighbor to dominate them, and the first condition of liar’s domination must be satisfied. Secondly, since the first and last vertices are in the set, their only neighbors must also be in the set so that the first and last vertices are doubly dominated. Therefore, \( P_3 \) and \( P_4 \) have all of their vertices in the set by this observation alone. Additionally, in \( P_5 \) and \( P_6 \) the middle vertices will need to be in the set to satisfy the second half of the parameter for liar’s domination because only after these vertices are in the set will any given pair of vertices have \( |(N[a] \cup N[b]) \cap S| \geq 3 \).

Let \( i, j \in S \), where \( N(j) = \{i\} \). Then, there exists \( k \in N(i) - \{j\} \) such that \( |(N[i] \cup N[j]) \cap S| = 2 \). Thus, \( k \) must be in \( S \) so that \( |(N[i] \cup N[j]) \cap S| = 3 \). Thus, path graphs \( P_n \) with \( 3 \leq n \leq 6 \) will stabilize as a minimal liar’s dominating set with all of the vertices being in the set. \( \square \)

**Lemma 2.5.** Once stable, \( S \subseteq V(P_n) \) is a minimal liar’s dominating set.

**Proof.** We will proceed by contradiction, using three cases.

Case 1: Assume that a path graph \( P_n \) has stabilized under liar’s domination and \( P_n \) is not minimal. Then, every vertex is dominated at least
twice and for any pair of vertices, \( i \) and \( j \), \(|(N[i] \cup N[j]) \cap S| \geq 3\). Since \( P_n \) is not minimal, a vertex in the set may be removed without violating the parameter. Once \( P_n \) \((3 \leq n \leq 6)\) is stabilized, it is always minimal because a vertex cannot be removed without breaking a condition to liar’s dominating set as noted in Lemma 2.4. For path graphs \( P_n \) \((n \geq 7)\), we will consider removing a vertex that is in the set to form a minimal liar’s dominating set. Observe that we can remove any vertex \( k \) that is in the set if and only if there are at least three immediate neighbors in the set to the left and right of \( k \) because \( k \) would be still need to be doubly dominated in order to satisfy the conditions for liar’s dominating set. However, \( k \) would have been privileged by \( R1 \). Thus, a contradiction has been reached because we assumed \( P_n \) stabilized, but there is still a vertex that is privileged.

Case 2: Assume that \( P_n \) has stabilized under liar’s domination, but some vertex \( i \) is not doubly dominated. Assume the vertex \( i \) is not in the set and \(|N(i) \cap S| \leq 1\). Then, \( i \) should be privileged by \( R2 \) because \( i \) is not in the set and it has less than three immediate neighbors in the set to the left and right of it. Furthermore, we have reached a contradiction, so the system was never stabilized.

Now, assume a vertex \( j \) is in the set but isn’t doubly dominated. Since \( j \) dominates itself, \(|N(j) \cap S| = 0\). If at least one neighbor was in the set, \( j \) would be at least doubly dominated because \( j \) already dominates itself. Furthermore, we know \( N(j) = \{k, l\} \) will be privileged under \( R2 \) since the immediate neighbors to the left and right of \( k \) and \( l \) total to less than three. This is true because we know that \( k \in N(j) - \{l\} \), so since \( k \notin S \), \( l \) has less than three immediate neighbors to its left and right. Since \( l \in N(j) - \{k\} \) and \( l \notin S \), then \( k \) will also be privileged by \( R2 \) for the same as above. Hence, \( l \) and \( k \) are both privileged by \( R2 \) since their immediate neighbors in the set to the left and right of them total to less than three. Therefore, a contradiction has been reached because the system was never stabilized as originally assumed. Consequently, all vertices in \( P_n \) are at least doubly dominated when \( P_n \) is stabilized under liar’s domination.

Case 3: Assume \( P_n \) has stabilized as a minimal liar’s dominating set, but that for \( i, j \in V \), \(|(N[i] \cup N[j]) \cap S| < 3\) where \( i \neq j \). First, assume \( a, b \in S \) where \( a \in N[b] \) and \(|(N[a] \cup N[b]) \cap S| = 2\). Also, assume \( N(a) = \{b\} \). Then, there exists \( c \in N[b] \) such that \( c \notin S \) since \(|(N[a] \cup N[b]) \cap S| < 3\). However, \( c \) should have been privileged by \( R2 \) because \( c \notin S \) and there are fewer than
three immediate neighbors to its left and right. Thus, we have reached a contradiction to the assumption that \( P_n \) was stabilized. Similarly, assume that \( d, e \in S \) where \( d \in N(e) \) and \( |(N[d] \cup N[e]) \cap S| = 2 \). Then, there exists \( f, g \notin S \) such that \( f \in N(d) - \{e\} \) and \( g \in N(e) - \{d\} \). Since \( g \notin S \), \( f \) has less than three immediate neighbors in the set on one side, then \( f \) is privileged by \( R2 \). Similarly, since \( f \notin S \), \( g \) has less than three immediate in the set on one side, then \( g \) is also privileged by \( R2 \). Hence, this is a contradiction to the assumption that \( P_n \) was stabilized. Next, consider a case where \( k \in S \) and \( |N(k) \cap S| = 0 \). Then, there exists \( h \in N(k) \) such that \( h \notin S \) where \( |N[h] \cap S| = 2 \) by Lemma 2.3 so that \( h \) is doubly dominated. However, \( k \) is not doubly dominated since \( |N(k) \cap S| = 0 \). This would have been shown in Case 2.

Consider the case where \( l, m \in V \) and \( |(N[l] \cup N[m]) \cap S| = 1 \). This would imply that \( l \) and \( m \) are not doubly dominated, which we covered in Case 2. Next, consider the case where \( q, o \in V \), but \( q \notin N[o] \) and \( |(N[q] \cup N[o]) \cap S| = 2 \). Whether or not \( q \in S \) and \( o \in S \), either \( q \) or \( o \) is not being doubly dominated, which was covered in Case 2. Therefore, all stabilized minimal liar’s dominating sets will have for any pair of vertices \( i, j \in V |(N[i] \cup N[j]) \cap S| \geq 3 \), where \( i \neq j \). \( \square \)

**Theorem 2.6.** The system \( P_n \) will exhibit self-stabilization as a minimal liar’s dominating set in at most \( 2n \) moves.

**Proof.** Due to Lemma 2.3 the system \( P_n \) will stabilize in at most \( 2n \) moves. Due to Lemma 2.5, the system, once stabilized, \( P_n \) will be a minimal liar’s dominating set. \( \square \)

### 3 Network Application

After discussing the process of self-stabilization, minimal liar’s domination, and Algorithm 2.1, a natural question to ask is what are the applications of this research? The answer to this question is actually in the name of our condition, minimal liar’s domination.

The most practical application of this research is within the area of network security. Minimal liar’s domination can be used to correctly identify where an intrusion has occurred. However, this algorithm only works under the assumptions that there is only one intrusion at a given time and vertices, “computers,” in the set are the only ones that can indicate where an intrusion
occurred. To explain the relationship between minimal liar’s domination and network security, consider Figure 9 and Figure 10. Note the graphs shown in these figures are segments of a larger stabilized path graph since we know having all vertices in the set is the minimal liar’s dominating set for $P_4$.

In order to correctly locate the intrusion, we need two computers to confirm the same location for an intruder. Since vertices that are in the set can indicate where an intrusion occurs and a vertex in the set dominates its neighbors and itself, a truth (when two vertices indicate the same location) arises from the fact that every vertex is dominated at least twice. In Figure 9, vertices 1 and 3 both indicate that an intrusion happened at vertex 2. Since both vertices are indicating the same location, this is said to be a truth. Consequently, the intruder location is correctly identified.

![Figure 9: Path graph depicting the importance of double domination for minimal liar’s domination.](image)

However, what if one of the neighboring computers was not able to detect the location? This is why it is important to have at least three vertices in the collective closed neighborhood because one of the computers could be “lying.” In order to prevent this from happening, the collective closed neighborhood can reveal which vertex is lying. In doing so, it reveals where the actual intrusion occurred. At this particular point, we also assume that only one computer or vertex lied.

![Figure 10: Path graph depicting the importance of having at least three vertices in the set in the collective closed neighborhood for any two vertices.](image)

In Figure 10, notice that vertex 1 indicates that the intrusion happened at vertex 2; meanwhile, vertex 3 indicates that the intrusion happened at its location. Since we only assume that one vertex is lying, then vertex 4 should have indicated that the intrusion happened at vertex 3. However, vertex 4 is
not indicating an intrusion. Therefore, we know that vertex 1 is indicating the correct location.

4 Conclusion

My goal throughout this thesis was to present graph theory in such a way so that undergraduates with no experience in graph theory could easily follow the research that my colleagues and I completed at St. Mary’s College of Maryland with Dr. Alan Jamieson. Graph theory is often used for undergraduate research because undergraduates can jump into learning one particular subject of graph theory such as self-stabilization all the while learning about the broader field of graph theory. I discussed the real world applicability of this research in the previous section, but this research is limited to very simple networks that can be modeled by path graphs. Realistically, networks can be modeled by a variety and combination of the types of graphs seen in the introduction to graph theory section, Figures 2-6. During the summer, we attempted to create a self-stabilizing algorithm for minimal liar’s domination for tree graphs, but we quickly realized the algorithm becomes more involved as graphs increase in complexity. Future research is needed to create self-stabilizing algorithms for the various graphs outlined in the introduction. From this, the next step would be to create a self-stabilizing algorithm for minimal liar’s domination for arbitrary graphs. This would allow us to use self-stabilization and minimal liar’s domination for network security in real life. We have also been asked about the possibility of having multiple liars instead of just one as liar’s domination assumes. In this case, we would need to create a new condition that incorporates this new assumption into its requirements. From this, we can create self-stabilizing algorithms for specific types of graphs, working our way up to one for arbitrary graphs. The open questions I have suggested above are questions that undergraduates can tackle as a part of their own research.
References


