2011

Finding Dud Vertices In Defensive Alliances And Secure Sets Using Computational Tools

George Worley II
University of Central Florida

Finding dud vertices in defensive alliances and secure sets using computational tools

by

George Gordon Worley III
B.S. Computer Science, University of Central Florida, 2004
M.S. Computer Science, University of Central Florida, 2006

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Mathematics
in the College of Sciences
at the University of Central Florida
Orlando, Florida

Summer Term
2011

Major Professor:
Yue Zhao

ABSTRACT

Defensive alliances are a way of using graphs to model the defense of resources (people, buildings, countries, etc.) against attacks where the number of potential attackers against each resource is known. The initial study of defensive alliances focused on questions of minimal defensive alliances in a graph and the minimum possible size of a defensive alliance in a graph, but in order to apply defensive alliances in modeling real-world situations, additional considerations are important. In particular, since each vertex in a defensive alliance represents some real-world object that has a cost associated with remaining in the defensive alliance, it is important to consider the value each vertex adds to the defensive alliance. In this thesis we consider a method of assessing the efficiency of a defensive alliance, including the special case of secure sets.

TABLE OF CONTENTS

LIST OF FIGURES . . . . **iv**

CHAPTER 1 INTRODUCTION **1**

1.1 Defensive Alliances 3

1.2 Secure Sets 4

1.3 Algorithmic Complexity 6

CHAPTER 2 DUD VERTICES **8**

CHAPTER 3 COMPUTATIONAL TOOLS **15**

CHAPTER 4 CONCLUSION **23**

LIST OF REFERENCES **24**

LIST OF FIGURES

1.1 A graph with a defensive alliance that is not a secure set 4

2.1 A graph containing a minimal secure set with a dud vertex 9

2.2 A graph containing a minimum secure set with a dud vertex 13

CHAPTER 1

INTRODUCTION

A simple, undirected graph $G = (V, E)$ (hereafter referred to as a "graph") is composed of a set of vertices V and a set of edges E, where each edge is a set of two vertices. Two vertices are said to be adjacent of there is an edge in E containing the two vertices. The open neighborhood of a vertex $v \in V$, $N(v)$, is the set of all vertices adjacent to v in G. The closed neighborhood is defined as $N[v] = N(v) \cup \{v\}$. The open neighborhood of a set of vertices $U \subseteq V$ is $N(U) = \bigcup_{v \in U} N(v) - U$, and the closed neighborhood is $N[U] = N(U) \cup U$.

A defensive alliance (sometimes abbreviated to "alliance") is a set of vertices D in a graph G with the property that for all $v \in D$, $|N[v] \cap D| \geq |N(v) \cap \partial D|$, where $\partial D = \bigcup_{v \in D} N(v) - D$, the set of vertices in G neighboring D but not in D (referred to as the boundary of D). A strong defensive alliance requires that $|N[v] \cap D| > |N(v) \cap \partial D|$. [KHH04] A secure set is a defensive alliance with the added property that every subset S of D has the property that $|N[S] \cap D| \geq |N(S) \cap \partial D|$ (this property subsumes the defensive alliance property, though, and is sufficient for defining secure sets on their own). [BDH07]

A defensive alliance is minimal if it contains no proper subset that is a defensive alliance. Similarly, a secure set is minimal if it contains no proper subset that is a secure set. A defensive alliance or secure set is a minimum in a particular graph if it is of least cardinality among all defensive alliances or secure sets, respectively, in a graph.

Defensive alliances originated with the purpose of modeling neighboring countries defending themselves from attacks by assigning each country to a vertex and connecting vertices when the represented countries share a border. [KHH04] Other possible applications include modeling the defense of a military compound, the guarding of the crown jewels of a country, and the security of a museum. [Dut09] Although not explicitly considered in earlier works, each vertex added to an alliance has a cost associated with adding the object it represents in the real world, be it in terms of money, lives, or other valuable resources. Thus it is beneficial to find minimal defensive alliances. But because we may find more than one minimal defensive alliance meeting the requirements of the situation being modeled, we need a way of assessing which of several alliances is "best". To this end we want to consider the efficiency of defensive alliances, that is to measure how well the defenders are utilized.

In this thesis we will consider the existence of "dud" vertices in alliances (that is, vertices of at least distance 3 from the boundary of the alliance) and examine what they may tell us about the efficiency of an alliance.

Note: For all notation not explicitly introduced in the body of this thesis, we follow the terminology of West. [Wes01]

1.1 Defensive Alliances

In 2002 Kristiansen, Hedetniemi, and Hedetniemi introduced the concept of defensive alliances. [KHH04] In particular they studied minimal and minimum defensive alliances and defensive alliance number, defined as $a(G)$ to be the cardinality of a minimum alliance in G. They discovered several properties of defensive alliances that help us to understand their structure.

Theorem 1. *For any graph $G = (V, E)$,*

1. *For any minimal defensive alliance $D \subseteq V$, the subgraph induced by D in G is connected.*

2. *$a(G) = 1$ if and only if there exists a vertex $v \in V$ such that $deg(v) \leq 1$.*

3. *$a(G) = 2$ if and only if $\delta(G) \geq 2$ and G has two adjacent vertices of degree at most three.*

This tells us, for example, that for any tree or path graph, $a(G) = 1$, and for any cycle or wheel graph, $a(G) = 2$. [KHH04] But since determining the characteristics of graphs with $a(G) = k$ becomes harder as k increases, they also give bounds on $a(G)$.

Theorem 2. *For any graph G or order n, $1 \leq a(G) \leq \left\lfloor \frac{n}{2} \right\rfloor$.*

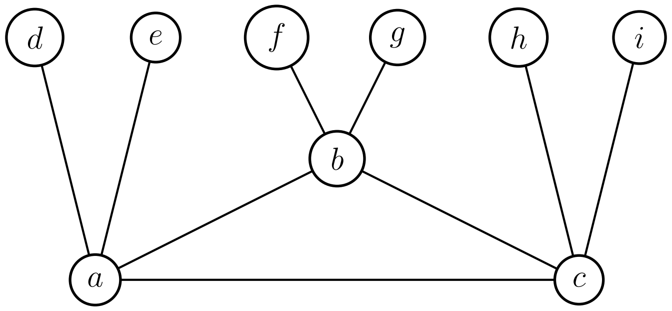

Figure 1.1: A graph with a defensive alliance that is not a secure set

This bound is known to be sharp, since $a(K_n) = \lfloor \frac{n}{2} \rfloor$, where K_n is the complete graph on n vertices.

1.2 Secure Sets

The work of Kristiansen, Hedetniemi, and Hedetniemi lead Brigham, Dutton, and Hedetniemi to further consider the graph theoretic modeling of real-world alliances and noted that defensive alliances might fail in a real-world defenses because the attackers could overwhelm the capacity of the defenders to help their neighbors. [BDH07] Consider the graph in Figure 1.1.

In it, the vertex set $\{a, b, c\}$ forms a defensive alliance, but does not form a secure set. In terms of the model, we could say that a defensive alliance can defend against an attack against any one vertex in the alliance, but not an attack against multiple vertices in the alliance. To address this issue Brigham, Dutton, and Hedetniemi formally defined the meaning of an attack on and a defense of an alliance and used these to construct a stronger type of alliance called a secure set. [BDH07]

Definition 3. *Let $G = (V, E)$ be a graph. For any $S = \{s_1, s_2, ..., s_k\} \subseteq V$, an attack on S is any k mutually disjoint sets $A = \{A_1, A_2, ..., A_k\}$ for which $A_i \subseteq N[s_i] - S, 1 \leq i \leq k$. A defense of S is any k mutually disjoint sets $D = \{D_1, D_2, ..., D_k\}$ for which $D_i \subseteq N[s_i] \cap S, 1 \leq i \leq k$. An attack A is defendable if there exists a defense D such that $|D_i| \geq |A_i|$ for $1 \leq i \leq k$.*

A set S is secure if and only if every attack on S is defendable.

A subset $X \subseteq S$ is S-secure if every attack on S in which $A_i = \emptyset$ whenever $s_i \notin X$ is defendable.

This definition of secure sets, while instructional in understanding the motivations behind the design of the concept, is not very useful in proofs because it is cumbersome. Instead, they give the following equivalent definition of secure sets. [BDH07]

Definition 4. *A set S of vertices in graph G is secure if and only if $|N[X] \cap S| \geq |N[X] - S|$ for all $X \subseteq S$.*

Because of the advantages of this definition of secure sets, in the remainder of this thesis we will refer to attacks and defenses only informally to make clear the scenarios being modeled using alliances.

Similar to the bounds on the defensive alliance number of a graph, bounds exist on the security number of a graph, defined as $s(G)$ to be the minimum cardinality of a minimal secure set in G. [DLB08] Much like in the case of defensive alliances, these bounds are especially useful since $s(G) = k$ becomes increasingly difficult to characterize as k increases.

Theorem 5. *Given a graph G of order n, $\left\lceil \frac{\delta(G)+1}{2} \right\rceil \leq s(G) \leq n - \left\lceil \frac{\delta(G)}{2} \right\rceil$.*

This bound is not sharp, but no better bound has yet been discovered. However it is known that there exist graphs for which $s(G) \geq \left\lceil \frac{n(G)}{2} \right\rceil$, so security number does not share an upper bound with defensive alliance number. [DLB08]

1.3 Algorithmic Complexity

Intuitively the problems of finding defensive alliances and secure sets should be in the set of NP problems, that is the set of decision problems where the affirmative instances can be identified in polynomial time by a nondeterministic Turing machine, because defensive alliances and secure sets are defined in terms of their subsets. For defensive alliances, the problem is formally stated as follows.

Problem 6. *DEFENSIVE ALLIANCE*

INSTANCE: Graph $G = (V, E)$, positive integer $k < |V|$.

QUESTION: Does G have a defensive alliance of size at most k?

DEFENSIVE ALLIANCE is not only in NP, but is in the set of NP-complete problems because there exists a transformation from the VERTEX COVER problem to DEFENSIVE ALLIANCE. [Jam07] This means it is among the hardest problems that can be answered in polynomial time by a nondeterministic Turing machine.

Not as much is known about the same problem for secure sets. First, we define it formally.

Problem 7. *SECURE SET*

INSTANCE: Graph $G = (V, E)$*, positive integer* $k < |V|$

QUESTION: Does G *have a secure set of size at most* k*?*

Currently nothing is known about the complexity classification of SECURE SET. We can currently only say that no deterministic or nondeterministic polynomial time algorithms are known for the problem, although special cases of the problem that restrict the structure of G are known to be in NP and $co - NP$. [Dut06] All known algorithms that answer SECURE SET require exponential time.

CHAPTER 2
DUD VERTICES

In a defensive alliance, a vertex is a dud if it is at least distance 3 from the alliance's boundary, which is to say that a dud never defends itself or one of its neighbors from an attack. There are two reasons we might include dud vertices in a defensive alliance. One is if the dud is a vertex that must be included in the defensive alliance, such as when using a defensive alliance to model the guards protecting the crown jewels, which we model by saying we have a vertex set J in a graph G that must be defended. The other reason we might be tempted to include a dud vertex is to keep it from attacking, thus out of the boundary. But, if our defensive alliance is minimal, there is no need for any dud vertices outside of any that might be contained in J, even if we require a strong defensive alliance.

The primary result of the paper follows.

Theorem 8. *Given $J \subseteq V(G)$, let D be a minimal defensive alliance that contains J. Then there are no duds in $D - J$.*

Proof. Assume to the contrary that v is a dud vertex in $D - J$. Then v must be at least distance 3 from ∂D. Then for all $u \in N(v)$, $N(u) \subseteq D$ since no u is adjacent to any vertex

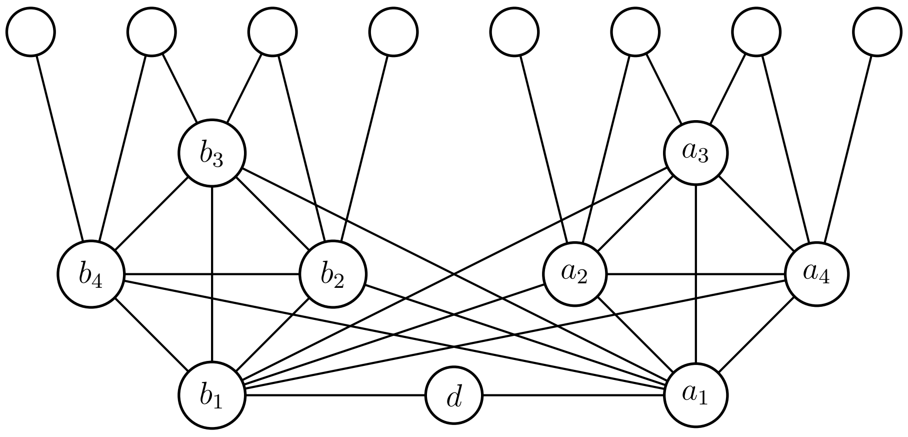

Figure 2.1: A graph containing a minimal secure set with a dud vertex

not in D, from which it also follows that $|N[u]| \geq 3$. Then in $D - v$, $|N[u] \cap (D - v)| \geq 2 > |N(u) \cap \partial(D - v)| = 1$, since $N(u) \cap \partial(D - v) = \{v\}$, and since for all $w \in D - N[v]$, $N[w] \cap D = N[w] \cap D - v$ and $N(w) \cap \partial D = N(w) \cap \partial D - v$, it follows that $D - v$ is a (strong) defensive alliance, hence D is not minimal, a contradiction to D being a minimal (strong) defensive alliance, thus v does not exist and the result is proved. $\square$

Since the only duds present in a minimal (strong) defensive alliance are those vertices that we have decided must be included, regardless of whether or not they were duds, duds do not provide a very good means of measuring the efficiency of defensive alliances.

We see, though, that a minimal secure set can contain a dud.

Observation 9. *Consider the graph in Figure 2.1. $S = \{d, a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4\}$ forms a minimal secure set (that is, no subset of S is also a secure set), and d is a dud. Thus the presence of dud vertices is relevant when considering the efficiency of minimal secure sets.*

This observation was verified computationally using the following Sage code, which is explained in further detail in Chapter 4.

```
G = Graph()

#arm 1
G.add_edge(0, 1)

G.add_edge(1, 2)
G.add_edge(1, 3)
G.add_edge(1, 4)

G.add_edge(2, 3)
G.add_edge(3, 4)
G.add_edge(4, 2)

G.add_edge(2, 5)
G.add_edge(2, 6)
G.add_edge(3, 6)
G.add_edge(3, 7)
G.add_edge(4, 7)
G.add_edge(4, 8)

#arm 2
G.add_edge(0, 11)

G.add_edge(11, 12)
G.add_edge(11, 13)
G.add_edge(11, 14)

G.add_edge(12, 13)
G.add_edge(13, 14)
G.add_edge(14, 12)

G.add_edge(12, 15)
G.add_edge(12, 16)
G.add_edge(13, 16)
G.add_edge(13, 17)
G.add_edge(14, 17)
G.add_edge(14, 18)
```

```
#between arms
G.add_edge(2, 11)
G.add_edge(3, 11)
G.add_edge(4, 11)

G.add_edge(12, 1)
G.add_edge(13, 1)
G.add_edge(14, 1)

S = Set([0, 1, 2, 3, 4, 11, 12, 13, 14])
print minimum_secure_sets_in_subset(G, S)
```

Which output that S did not contain any proper subsets which were themselves secure sets.

Let $\mathrm{dud}_G(S)$ be the number of dud vertices in a secure set S in G. If S contains a subset J that must be defended, J may contain dud vertices, so let $\mathrm{dud}_G^J(S)$ be the number of dud vertices in a secure set S in graph G where dud vertices in J are excluded from the count.

Although dud vertex count is a new concept, we can say at least a few useful things about it.

Observation 10. *Let S be a secure set in a graph G. If the diameter of $\langle S \rangle$ is less than 2 or the diameter of G is less than 3, then $dud(S) = 0$. Equivalently, if $dud(S) > 0$, then the diameter of $\langle S \rangle$ is at least 2 and the diameter of G is at least 3. Or, put another way, G must contain a vertex of at least eccentricity 3.*

This observation follows immediately from the fact that a dud vertex must be at least distance 3 away from any vertex in ∂S. This tells us something about the structure of a graph that contains a minimal secure set with a dud vertex.

We can also see that a minimum secure set may contain a dud (noting that in Observation 9, a minimal secure set contained a dud, but the graph contained a secure set of size 1).

Observation 11. *Consider the graph H, shown in Figure 2.2, formed as G from Figure 2.1 connected to the complete graph K_{17} with vertices in ∂S corresponding to any set of unique vertices in K_{17}. Then S is a minimum secure set of H that contains a dud vertex.*

We find the minimum secure sets of H using Sage, continuing on the code from our earlier example.

```
H = G
K = [5, 6, 7, 8, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]
i = 0
while i < len(K) - 1:
    j = i + 1
    while j < len(K):
        H.add_edge(K[i], K[j])
        j += 1
    i += 1

print minimum_secure_sets(H)
```

The output of this code tells us that any set of 9 vertices in the K_{17} subgraph of H form a secure set, and $|S| = 9$, so all are minimum secure sets. But since only S has a dud vertex count of 1, this demonstrates that even minimum secure sets may contain dud vertices.

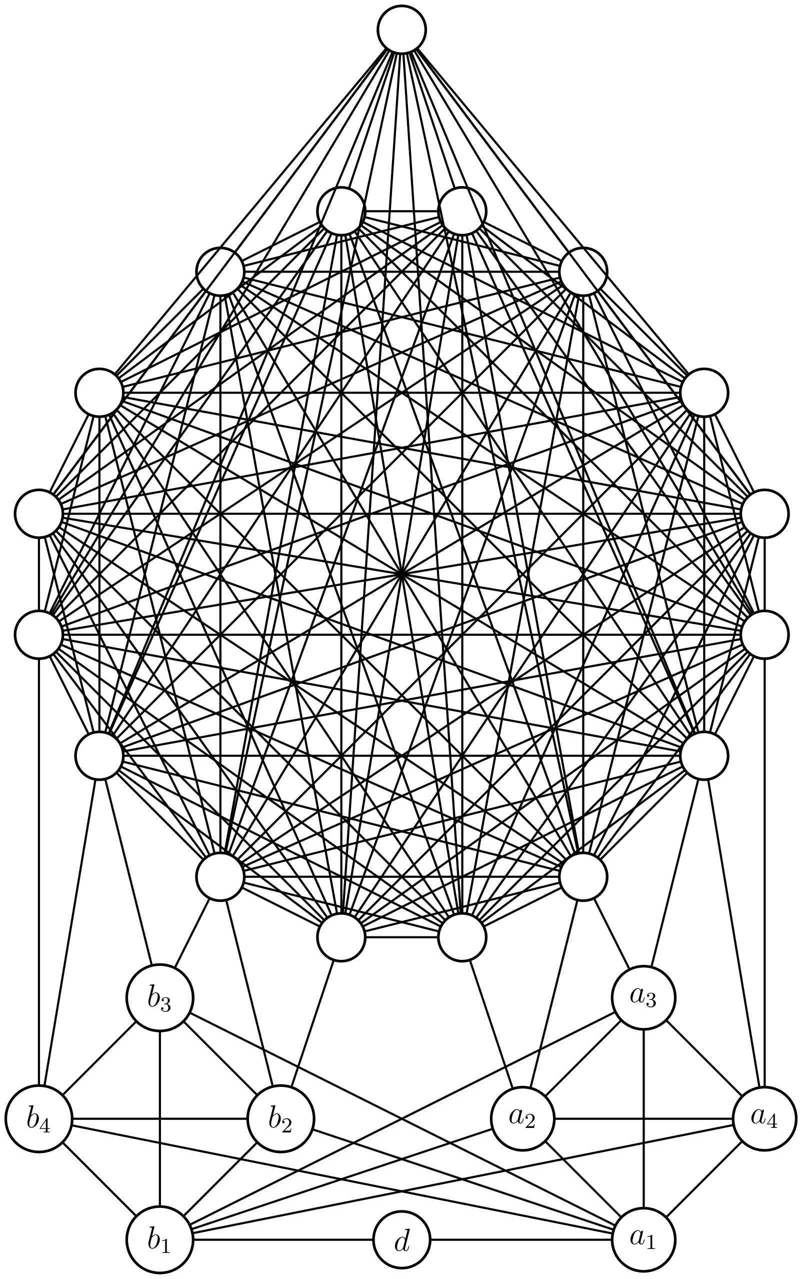

Figure 2.2: A graph containing a minimum secure set with a dud vertex

In terms of a model, consider the example where each vertex in H represents a position on a battlefield that a soldier might take up, and vertices are adjacent if they can attack each other. In this case, both S and the sets of 9 vertices in the K_{17} subgraph are minimum secure sets and can be defended with just 9 soldiers, but in S only 8 of those soldiers are actually involved in the defense whereas in the other minimum secure sets all 9 are. Depending on the situation being modeled, this might suggest S would be a better or worse choice than one of the other sets.

For example, if one soldier is wounded, S might be preferable because the wounded soldier could be place at the d position, but if no soldier is wounded, S might be a poor choice because it will require 8 soldiers to do the work of 9 while 1 soldier sits idle. However, the amount of work the soldiers have to do to defend each of the minimum secure sets in H is not clearly defined, and we have given no means of comparing the amount of work required to defend each secure set, so determining which of the minimum secure sets would be best for the soldiers is an open problem.

CHAPTER 3
COMPUTATIONAL TOOLS

In order to effectively study defensive alliances, and especially secure sets, computer assistance is needed to verify results that would take too long to do by hand since many of the problems posed by defensive alliances and secure sets require $O(2^n)$ operations to answer. [Jam07, Dut06] Building on the Sage mathematical library, we wrote the following functions to extend the existing graph theory functions available in Sage to support computations involving defensive alliances and secure sets. [Sag10]

Note that in many cases the functions could be optimized to generate results in less time using less memory, and indeed such algorithms do exist. In particular, it is possible to reduce the run time of the algorithms by applying known properties of defensive alliances and secure sets to skip processing portions of the search space. [Dut06] But since we did not need, in the course of this thesis, to deal with any graphs or alliances that were practically challenging to work with given the inefficiencies of the algorithms, we left the functions inefficient in favor of readability.

The first three functions do not deal directly with defensive alliances but instead provide convenience functions for obtaining information for which there is no built-in function in

Sage. In the future these will likely be rendered irrelevant as the Sage graph theory library expands, but they were necessary at the time of writing.

```
# Additional Graph Functions

def closed_neighborhood(G, X):
    """
     Input: a Graph G, a set of vertices X
    Output: the closed neighborhood of X in G as a set of vertices
    """
    Y = X
    for v in X:
        Y = Y.union(Set(G.neighbors(v)))
    return Y

def open_neighborhood(G, X):
    """
     Input: a Graph G, a set of vertices X
    Output: the open neighborhood of X in G as a set of vertices
    """
    Y = closed_neighborhood(G, X)
    Y = Y.difference(X)
    return Y

def min_distance(G, u, V):
    """
     Input: a Graph G, a vertex u, and a set of vertices V
    Output: minimum distance between u and any vertex in V in G,
            returns +Infinity if u is not connected to V
    """
    d = +Infinity
    for v in V:
        if G.distance(u,v) < d:
            d = G.distance(u,v)
    return d

# Defensive Alliance and Secure Set Functions

def is_defensive_alliance(G, S):
    """
```

```
     Input: a Graph G and a set of vertices S
    Output: true if S is a defensive alliance in G, false if not
    """
    for v in S:
        if (((Set(G.neighbors(v)).union(Set([v]))).intersection(S)
            ).cardinality() < ((Set(G.neighbors(v)).union(Set([v]))
            ).intersection((Set(G.vertices(v)).union(Set([v]))
            ).difference(S))).cardinality()):
                return false
    return true

def is_secure_set(G, S):
    """
     Input: a Graph G and a set of vertices S
    Output: true if S is a secure set in G, false if not

      Note: This iterates through every subset of S, so it runs
            in O(2^|S|).
    """
    P = S.subsets()
    for X in P:
        if (closed_neighborhood(G, X).intersection(S)).cardinality()
            < (closed_neighborhood(G, X).intersection(
            open_neighborhood(G, X).difference(S))).cardinality():
                return false
    return true

def is_minimal_secure_set(G, S):
    """
     Input: a Graph G and a set of vertices S that form a secure set
    Output: true if S is a minimal secure set in G

      Note: returns true if S does not form a secure set, so you should
            verify this separately.
    """
    P = S.subsets()
    for X in P:
        if is_secure_set(G, X):
            return false
    return true

def secure_sets_within_secure_set(G, S):
    """
     Input: a Graph G and a set of vertices S
    Output: a list of vertex sets forming secure sets within S,
              including S if S is a secure set in G
    """
    P = S.subsets()
    SS = [] # list of secure sets
    for X in P:
        if is_secure_set(G, X):
            SS.append(X)
    return SS

def minimum_defensive_alliance(G):
    """
     Input: a Graph G
    Output: a set of vertices forming a minimum defensive alliance in G
    """
    V = Set(G.vertices())
    VS = V.subsets()
    VS.next() # skip empty set at start
    for X in VS:
        if (is_defensive_alliance(G, X) & X.cardinality() > 0):
            return X

def minimum_secure_set(G):
    """
     Input: a Graph G
    Output: a set of vertices forming a minimum secure set in G
    """
    V = Set(G.vertices());
    VS = V.subsets()
    VS.next() # skip empty set at start
    for X in VS:
        if (is_secure_set(G, X)):
            return X

def minimum_defensive_alliances(G):
    """
     Input: a Graph G
    Output: a list of vertex sets, each a minimum defensive alliance in G
    """
    V = Set(G.vertices())
    Q = []

    VS = V.subsets()

    found_one = false
    card = 0

    VS.next() # skip the empty set at start
    for X in VS:
        if is_defensive_alliance(G, X):
            if found_one == false:
                Q.append(X)
                found_one = true
                card = X.cardinality()
            else:
                if card == X.cardinality():
                    Q.append(X)
                else:
                    break
    return Q

def minimum_secure_sets(G):
    """
     Input: a Graph G
    Output: a list of vertex sets, each a minimum secure set in G
    """
    V = Set(G.vertices())
    Q = []

    VS = V.subsets()

    found_one = false
    card = 0

    VS.next() # skip the empty set at start
    for X in VS:
        if is_secure_set(G, X):
            if found_one == false:
                Q.append(X)
                found_one = true
                card = X.cardinality()
            else:
```

```
                        if card == X.cardinality():
                            Q.append(X)
                        else:
                            break
        return Q

    def minimum_secure_sets_in_subset(G, S):
        """
          Input: a Graph G and a set of vertices S
         Output: a list of vertex sets containing minimum secure sets in G
                   that are subsets of S

           Note: Used to get faster output when a minimum secure set is known
                   to reside in a particular subset of the vertices of G.
        """
        Q = []

        VS = S.subsets()

        found_one = false
        card = 0

        VS.next() # skip the empty set at start
        for X in VS:
            X = VS[i]
            if is_secure_set(G, X):
                if found_one == false:
                    Q.append(X)
                    found_one = true
                    card = X.cardinality()
                else:
                    if card == X.cardinality():
                        Q.append(X)
                    else:
                        break
        return Q

    def all_defensive_alliances(G):
        """
            Input: a Graph G
           Output: a list of vertex sets, each forming a defensive alliance in G
        """
        V = Set(G.vertices())
        Q = []
        for X in V.subsets():
            if is_defensive_alliance(G, X):
                Q.append(X);
        return Q

    def all_secure_sets(G):
        """
            Input: a Graph G
           Output: a list of vertex sets, each forming a secure set in G
        """
        V = Set(G.vertices())
        Q = []
        for X in V.subsets():
            if is_secure_set(G, X):
                Q.append(X);
        return Q
```

The code was checked for correctness in two ways. First, we verified that the code matched the mathematical definitions, and second we further tested the functions experimentally using known results. In all cases the functions presented are believed to be correct.

In our usage of this code, described in Chapter 3, the total compute time was approximately three hours and required about 700 MB of process memory. In addition to techniques for speeding up the algorithms, if needed, could have reduced the amount of process memory used by changing the way Sage represents graphs. Currently, this code uses an incident list representation of graphs. [Sag10] Specifically, a graph is stored as an array of edges, each edge stored as a pair of integers, with integers used as vertex identifiers. If we switched to an adjacency matrix or incident matrix representation of graphs, though, we could potentially reduce memory usage significantly, allowing analysis of larger graphs with less computational resources. [Gib85]

CHAPTER 4

CONCLUSION

The presence of dud vertices, even in minimum secure sets, suggests that secure sets may contain inefficiencies that can be addressed. Additionally, dud vertices are only one possible way of considering the matter of efficiency in defensive alliances and secure sets, and since dud vertices do not appear in minimal defensive alliances, they do not even provide a comprehensive tool for assessing efficiency in alliances. To these points, we end with several open questions related to the efficiency of defensive alliances.

1. Does there exist a measure on defensive alliances (secure sets) that gives a useful definition of efficiency? If so:
   (a) Does the minimum defensive alliance in a graph also have maximum efficiency?
   (b) Does the maximum efficiency among all possible defensive alliances within a graph relate to other graphical invariants?
2. What is the smallest subgraph that can contain a secure set with at least one dud?
3. Does there exist a measure of the amount of "work" each vertex has to do to defend a defensive alliance (secure set)? How does this relate to efficiency?

LIST OF REFERENCES

[BDH07] Robert C. Brigham, Ronald D. Dutton, and Stephen T. Hedetniemi. "Security in graphs." *Discrete Applied Mathematics*, **155**(13):1708–1714, 2007.

[DLB08] Ronald D. Dutton, Robert Lee, and Robert C. Brigham. "Bounds on a graph's security number." *Discrete Applied Mathematics*, **156**(5), 2008.

[Dut06] Ronald D. Dutton. "Secure set algorithms and complexity." In *37th Southeastern International Conference on Combinatorics, Graph Theory, and Computing*, 2006.

[Dut09] Ronald D. Dutton. "On a graph's security number." *Discrete Mathematics*, **309**(13):4443–4447, 2009.

[Gib85] A.M. Gibbons. *Algorithmic graph theory.* Cambridge University Press, 1985.

[Jam07] Lindsay H. Jamieson. *Algorithms and Complexity for Alliances and Weighted Alliances of Various Types.* PhD thesis, Clemson University, 2007.

[KHH04] Petter Kristiansen, Sandra M. Hedetniemi, and Stephen T. Hedetniemi. "Alliances in graphs." *Journal of Combinatorial Mathematics and Combinatorial Computing*, **48**:157–177, 2004.

[Sag10] W. A. Stein et al. *Sage Mathematics Software (Version 4.6).* The Sage Development Team, 2010. `http://www.sagemath.org`.

[Wes01] Douglas B. West. *Introduction to Graph Theory.* Prentice Hall, 2nd edition, 2001.