5-2007

Algorithms and Complexity for Alliances and Weighted Alliances of Various Types

Lindsay Jamieson
Clemson University, lch@clemson.edu

Follow this and additional works at: https://tigerprints.clemson.edu/all_dissertations
Part of the Computer Sciences Commons

Recommended Citation
Jamieson, Lindsay, "Algorithms and Complexity for Alliances and Weighted Alliances of Various Types" (2007). All Dissertations. 66.
https://tigerprints.clemson.edu/all_dissertations/66

This Dissertation is brought to you for free and open access by the Dissertations at TigerPrints. It has been accepted for inclusion in All Dissertations by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.
ALGORITHMS AND COMPLEXITY FOR ALLIANCES AND WEIGHTED ALLIANCES OF VARIOUS TYPES

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
School of Computing

by
Lindsay Harris Jamieson
May 2007

Accepted by:
Dr. Stephen Hedetniemi, Committee Chair
Dr. Wayne Goddard
Dr. Sandra Hedetniemi
Dr. Roy Pargas
Dr. Alice McRae
ABSTRACT

The concept of alliances was introduced in 2002 in a paper by Kristiansen, Hedetniemi and Hedetniemi [13]. Although research has been published on the mathematical properties of various types of alliances, until recently, no research has been done to develop algorithms or establish the complexity of decision problems for alliances in graphs.

This thesis presents the first algorithmic study of alliances in graphs. We present linear algorithms for finding various alliance numbers in trees and series parallel graphs. These linear algorithms are designed using a new methodology based on the well-established Wimer methodology [26] for designing polynomial algorithms on k-terminal graphs. Linear algorithms on trees for minimum offensive, minimum powerful, minimum global defensive, minimum global offensive, and minimum global powerful alliances are presented. Also, a polynomial algorithm for finding the minimum defensive alliance number of a series parallel graph is presented. Additional linear algorithms are presented for minimum weighted defensive, minimum weighted offensive, minimum weighted powerful, minimum weighted global defensive, minimum weighted global offensive, and minimum weighted global powerful alliances.

We present the first complexity study of alliances in graphs. This study was developed concurrently with the work of Cami, Balakrishnan, Deo and Dutton [6]. Complexity results for defensive, powerful, global defensive, and global powerful alliances when restricted to bipartite and chordal graphs are presented. Also, complexity results for weighted defensive, weighted offensive, weighted powerful, weighted global defensive, weighted global offensive, and weighted global powerful alliances when restricted to stars are presented.

In addition to interesting open questions, we also include implementations of the minimum powerful alliance algorithm on trees and the minimum weighted global defensive alliances on paths.
ACKNOWLEDGMENTS

Thank you to Dr. Dean for his ideas and help with the weighted alliance NP-completeness proof.

Thank you to my committee: Dr. Pargas for making sure that I understood the relevance of my topic and could make it approachable to others; Dr. McRae for her assistance on the NP-completeness proofs; Dr. Goddard for finding additional cases to make the sure the algorithms are correct; Dr. S. M. Hedetniemi for your help in verifying the algorithms.

Thank you to my advisor, Dr. S. T. Hedetniemi, for challenging me and believing in me. I really appreciate the never ending supply of great ideas.

Thank you to my friends and family for your love and support. I could not be where I am without you. I don’t have the words to thank you properly.

And most importantly, thank you to my husband, Alan, for your encouragement, love and support. Thank you for being my biggest cheerleader.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE PAGE</td>
</tr>
<tr>
<td>ABSTRACT</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
</tr>
<tr>
<td>1 Introduction</td>
</tr>
<tr>
<td>2 Linear Algorithms for Alliances in Trees</td>
</tr>
<tr>
<td>2.1 Offensive Alliances</td>
</tr>
<tr>
<td>2.2 Powerful Alliances</td>
</tr>
<tr>
<td>2.3 Global Defensive Alliances</td>
</tr>
<tr>
<td>2.4 Global Offensive Alliances</td>
</tr>
<tr>
<td>2.5 Global Powerful Alliances</td>
</tr>
<tr>
<td>2.6 Strong Alliances</td>
</tr>
<tr>
<td>3 Alliance Complexity</td>
</tr>
<tr>
<td>3.1 Defensive Alliance Complexity</td>
</tr>
<tr>
<td>3.2 Powerful Alliance Complexity</td>
</tr>
<tr>
<td>3.3 Global Defensive Alliance Complexity</td>
</tr>
<tr>
<td>3.4 Global Powerful Alliance Complexity</td>
</tr>
<tr>
<td>4 Linear Algorithms for Alliances in Series-Parallel Graphs</td>
</tr>
<tr>
<td>5 Linear Algorithms for Weighted Alliances in Paths</td>
</tr>
<tr>
<td>5.1 Defensive Alliances</td>
</tr>
<tr>
<td>5.2 Offensive Alliances</td>
</tr>
<tr>
<td>5.3 Powerful Alliances</td>
</tr>
<tr>
<td>5.4 Global Defensive Alliances</td>
</tr>
<tr>
<td>5.5 Global Offensive Alliances</td>
</tr>
<tr>
<td>5.6 Global Powerful Alliances</td>
</tr>
<tr>
<td>6 Weighted Alliance Complexity</td>
</tr>
<tr>
<td>6.1 Weighted Defensive and Powerful Alliance Complexity</td>
</tr>
<tr>
<td>6.2 Weighted Offensive Alliance Complexity</td>
</tr>
<tr>
<td>6.3 Weighted Global Offensive and Global Powerful Alliance Complexity</td>
</tr>
<tr>
<td>7 Summary and Open Problems</td>
</tr>
<tr>
<td>APPENDICES</td>
</tr>
<tr>
<td>A Completed Work</td>
</tr>
<tr>
<td>B Powerful Alliance Implementation</td>
</tr>
<tr>
<td>C Weighted Global Defensive Alliances Implementation</td>
</tr>
</tbody>
</table>
Table of Contents (Continued)

<table>
<thead>
<tr>
<th>Bibilography</th>
<th>85</th>
</tr>
</thead>
</table>

viii
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Compositions for offensive alliances.</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Compositions for powerful alliances.</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Compositions for global defensive alliances.</td>
<td>15</td>
</tr>
<tr>
<td>2.4 Compositions for global offensive alliances.</td>
<td>17</td>
</tr>
<tr>
<td>2.5 Compositions for global powerful alliances.</td>
<td>18</td>
</tr>
<tr>
<td>2.6 Compositions for minimal strong defensive alliances.</td>
<td>20</td>
</tr>
<tr>
<td>4.1 Compositions for serial connections for defensive alliances.</td>
<td>41</td>
</tr>
<tr>
<td>4.2 Compositions for generalized serial connections for defensive alliances.</td>
<td>42</td>
</tr>
<tr>
<td>4.3 Compositions for parallel connections for defensive alliances.</td>
<td>43</td>
</tr>
<tr>
<td>5.1 Compositions for weighted offensive alliances.</td>
<td>47</td>
</tr>
<tr>
<td>5.2 Compositions for weighted powerful alliances.</td>
<td>49</td>
</tr>
<tr>
<td>5.3 Compositions for weighted global defensive alliances.</td>
<td>51</td>
</tr>
<tr>
<td>5.4 Compositions for weighted global offensive alliances.</td>
<td>52</td>
</tr>
<tr>
<td>5.5 Compositions for weighted global powerful alliances.</td>
<td>54</td>
</tr>
<tr>
<td>A.1 Alliance Algorithms</td>
<td>69</td>
</tr>
<tr>
<td>A.2 Alliance Complexity</td>
<td>69</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Defensive Alliance Example</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Defensive Alliance Example</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>Powerful Alliance Example</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>Global Defensive Alliance Example - Bipartite Graphs</td>
<td>30</td>
</tr>
<tr>
<td>3.5</td>
<td>Global Defensive Alliance Example - Chordal Graphs</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>Global Powerful Alliance Example - Bipartite Graphs</td>
<td>34</td>
</tr>
<tr>
<td>3.7</td>
<td>Global Powerful Alliance Subgraph non-valid configurations</td>
<td>35</td>
</tr>
<tr>
<td>3.8</td>
<td>Global Powerful Alliance Subgraph valid configurations</td>
<td>35</td>
</tr>
<tr>
<td>3.9</td>
<td>Global Powerful Alliance Example - Chordal Graphs</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Examples of Composition Types</td>
<td>39</td>
</tr>
<tr>
<td>6.1</td>
<td>Weighted Powerful Alliance Example</td>
<td>58</td>
</tr>
<tr>
<td>6.2</td>
<td>Weighted Offensive Alliance Example</td>
<td>61</td>
</tr>
<tr>
<td>6.3</td>
<td>Weighted Global Powerful Alliance Example</td>
<td>63</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

There are several definitions which are necessary for the study of alliances in graphs. Let $G=(V,E)$ be a graph. For any vertex $v \in V$, the open neighborhood of $v$ is the set $N(v) = \{ u : uv \in E \}$, while the closed neighborhood of $v$ is the set $N[v] = N(v) \cup \{v\}$. The open neighborhood $N(S)$ and closed neighborhood $N[S]$ of a set $S \subseteq V$ are defined as $N(S) = \cup_{v \in S} N(v)$ and $N[S] = N(S) \cup S$, respectively. The boundary of a set $S$ is the set $\partial(S) = \cup_{v \in S} (N(v) \cap V - S)$.

A non-empty set of vertices $S \subseteq V$ is called a defensive alliance if $\forall v \in S, |N[v] \cap S| \geq |N[v] - S|$. This is equivalent to saying that every vertex in $S$ has at least as many neighbors in $S$ (including itself) as it has neighbors not in $S$. The defensive alliance number of a graph $G$ equals the minimum cardinality of a defensive alliance in $G$ and is denoted $\alpha(G)$. A defensive alliance is called strong if this inequality is strict, i.e. $\forall v \in S, |N[v] \cap S| > |N[v] - S|$. An alliance $S$ is called critical if no proper subset of $S$ is an alliance of the same type. The upper defensive alliance number of a graph is the maximum cardinality of a critical defensive alliance.

An alliance that is both defensive and offensive is called a powerful alliance. The powerful alliance number of a graph $G$ $\gamma_p(G)$ equals the minimum cardinality of a powerful alliance while an alliance $S$ for which $\partial(S) = V - S$ is called a global alliance. The upper powerful alliance number $\Gamma_p(G)$ of a graph is the maximum cardinality of a critical powerful alliance. The global defensive, global offensive and global powerful alliance numbers are denoted $\gamma_a(G)$, $\Gamma_a(G)$, $\gamma_o(G)$ and $\Gamma_o(G)$, and $\gamma_{ap}(G)$ and $\Gamma_{ap}(G)$, respectively.

Let $G = (V,E)$ be a graph and let $W : V \rightarrow N$ be a non-negative integer weighting of the vertices in $V$. This leads to some additional types of alliances. A non-empty set of vertices $S \subseteq V$ is called a weighted defensive alliance if $\forall v \in S, \sum_{u \in N[v] \cap S} w(u) \geq \sum_{x \in N[v] - S} w(x)$. The weighted defensive
alliance number of a graph $G$ equals the minimum weight of a weighted defensive alliance in $G$ and is denoted $a_w(G)$. A weighted defensive alliance is called strong if this inequality is strict. A non-empty set $S \subseteq V$ is a weighted offensive alliance if $\forall v \in \partial(S), \sum_{u \in N(v) \cap S} w(u) \geq \sum_{x \in N(v) - S} w(x)$.

The weighted offensive alliance number $a_{wo}(G)$ equals the minimum weight of a weighted offensive alliance in $G$. A weighted offensive alliance is called strong if $\forall v \in \partial(S), \sum_{u \in N(v) \cap S} w(u) > \sum_{x \in N[v] - S} w(x)$. A weighted alliance which is both defensive and offensive is called a weighted powerful alliance, $(a_{wp}(G)$ denotes the weighted powerful alliance number of a graph $G$) while a weighted alliance $S$ for which $\partial(S) = V - S$ is called a weighted global alliance.

There are two ways of thinking about alliances in graphs, either the creation of alliances or the destruction of existing alliances. The first mention of alliances in graphs was in a 1979 paper by Lipman and Ringeisen [17] on finding the critical nodes in an alliance. This is the only paper regarding the destruction of existing alliances.

The creation of alliances in graphs was first investigated in [16] and [13] in papers that introduced the concepts of defensive and offensive alliances and looked at properties of alliance numbers and strong alliance numbers of cycles, wheels, grids, complete graphs, complete bipartite graphs and random graphs. The authors also made conjectures on bounds for alliance numbers. These were later proven in [10]. In [8] and [9], the authors concentrate on offensive alliances and calculate bounds for the offensive and strong offensive alliance numbers. This paper also looks at the effects of the union operation on alliances.

In [10] the authors prove two conjectures on sharp bounds for the defensive and strong defensive alliance numbers, introduced in [16]. In [2] the authors consider bounds for the upper defensive alliance number of a graph. Lower and upper bounds are given for regular graphs, but these bounds are not sharp. For arbitrary graphs, the authors explored the maximum size of a minimal (i.e. critical) defensive alliance and the maximum size of a minimal strong defensive alliance in an arbitrary graph on $n$ vertices. In [12] the authors determine the global defensive alliance numbers of graphs of specific types. Lower bounds are given for general graphs, bipartite graphs and trees and upper bounds were found for general graphs and trees.

In [3] the authors study properties of powerful alliances and global powerful alliances, including the powerful alliance number, the size of the boundary of a powerful alliance, and a correlation between the powerful alliance number and the domination number of a tree. Also, a sharp bound on the upper powerful alliance number of a tree is given and graphs with equal domination number and global powerful alliance number, and trees with equal powerful alliance and global powerful alliance numbers are characterized. In [4] the authors explore lower bounds on the powerful alliance
number of $m \times n$ grid graphs. Specifically, the authors find sharp bounds on the powerful alliance number of $C_m \times C_n$. They also give lower bounds for the powerful alliance number of the Cartesian product of two paths, and the Cartesian product of a cycle and a path.

In [5] the authors define and study properties of secure alliances. A secure alliance is one in which any simultaneous attack on all the vertices of any subset of the alliance can be defended; the formal definition of a secure alliance is complex and is omitted here. In [19] tight bounds for the defensive, offensive, powerful, global defensive, global offensive, and global powerful alliances are found using the algebraic complexity, the spectral radius, and the Laplacian spectral radius of a graph. In [18] bounds on the offensive alliance number of cubic graphs are found. In [20] bounds on the global offensive alliance number are found for general graphs.

Another paper on the creation of alliances was published by Shafique and Dutton in 2002 who investigated alliance-free and alliance-cover sets. In [21] the following are defined (i) $k$-alliances, where every vertex in the alliance has at least $k$ more neighbors in the alliance than not in the alliance, (ii) maximum alliance-free sets, that is, sets that do not have any $k$-alliances for a specific $k$, and (iii) minimum $k$-alliance-cover sets, that is sets which contain at least one vertex from each $k$-alliance. In addition to these definitions, basic properties and bounds on alliance-free and alliance-cover sets are given. In [22] they expand on the definitions in [21] and prove tight bounds on maximum alliance-free and minimum alliance-cover sets.

In [1] and in [23] the authors study the concept of unfriendly partitions. This is a concept similar to alliances. However, unfriendly partitions assign the integers between 0 and $n-1$ to the vertices in such a way that the number of neighboring vertices with the same value as a given vertex is less than or equal to the number of neighboring vertices whose value is not the same as a given vertex. In [1] the authors conjecture that every graph has an unfriendly 2-partition. In [23] they disprove this conjecture by introducing a graph which does not have an unfriendly 2-partition, but show that every graph has an unfriendly 3-partition. In [22] the authors look at satisfactory partitions of graphs. A satisfactory partition of a graph is a partition into “two or more non-empty cohesive sets”[22], where a cohesive set occurs when every vertex in the set has at least half of its neighbors in the set; this is the definition of a strong defensive alliance. The paper explores the relationship between satisfactory partitions and connectivity and proves that a graph has a satisfactory partition if and only if the graph has a critical cutset, that is “moving any vertex from one set to the other does not decrease the size of the resulting cutset”[22].

A recent PhD dissertation [27] provided a distributed self-stabilizing algorithm for finding global defensive and global offensive alliances. These algorithms were also discussed in [28].
recent works on alliances are [6] which looks at the complexity of global alliances and [7] which finds lower bounds for the sizes of global alliances on planar graphs. In [6] the authors show that the decision problems for global defensive, global offensive and global powerful alliances are all NP-complete for general graphs.

There are many possible applications of alliances. Most of these applications are based on the original definition of alliance used in [16] “in times of war, by nations for mutual support, usually defensive in nature, where allies are obligated to join forces if one or more of them are attacked, but also offensive, as a means of keeping the peace, e.g. NATO troops in a war-torn country.” With this definition in mind, alliances, especially global alliances, could be used in military modeling in order to plan responses to various alliances or where to place troops to create alliances. Also, in networking, alliances could be used to protect important nodes. Alliances can also be used to model sets of supply nodes in a network which, when working together, can meet the demands on resources received by a member of the alliance from its neighboring nodes not in the alliance.

An alliance is a very interesting concept which has at this point only been explored graph theoretically. Algorithms for finding minimum alliances in various types of graphs and complexity proofs are natural extensions of existing research on alliances and weighted alliances.

In this dissertation, we present the following research which is summarized in Appendix A Table A.1 and Table A.2:

In Chapter 2 we present linear algorithms for minimum cardinality offensive alliances on trees, minimum cardinality powerful alliances on trees, minimum cardinality global defensive alliances on trees, minimum cardinality global offensive alliances on trees, and minimum cardinality global powerful alliances on trees.

In Chapter 3 we prove that the decision problems for defensive alliances, powerful alliances, global defensive alliances, and global powerful alliances are NP-complete even when restricted to bipartite or chordal graphs.

In Chapter 4 we present a polynomial time algorithm for minimum cardinality defensive alliances on generalized series-parallel graphs.

In Chapter 5 we present linear algorithms for minimum weighted defensive alliances, offensive alliances, powerful alliances, global defensive alliances, global offensive alliances, and global powerful alliances on paths.

In Chapter 6 we prove that the weighted alliance decision problems are NP-complete even when restricted to trees.

In Chapter 7 we list some open questions that remain in the areas of alliances and weighted
alliances. Also included are possible extensions of the concept of alliances.

In Appendix A we summarize the algorithms and proofs in this dissertation. In Appendices B and C we present the code for implementing the algorithm for minimum cardinality powerful alliances in trees and code for implementing the algorithm for minimum cardinality weighted global defensive alliances on paths.
Chapter 2
Linear Algorithms for Alliances in Trees

In finding linear algorithms for alliances in trees, one avenue for exploration is the methodology for linear algorithms on trees described in [26]. This provides a framework for developing linear time, dynamic programming algorithms. However, the original methodology provides no mechanism for a variable that can keep track of the difference between the number of neighbors in an alliance and the number of neighbors not in an alliance. Thus, in order to design linear time algorithms for finding optimal alliances in trees, we will need to develop a variant of the Wimer methodology. In this variant we use such a variable and make decisions based on its values in the determination of the compositions of two subtree-subset pairs. This addition does not change the complexity of the algorithms but rather requires the use of conditionals in the composition tables. This particular variant was first published in our paper [14].

All of the algorithms in this dissertation follow the style set out by Wimer, Hedetniemi, and Laskar in [26]. Congruence classes are developed which completely describe all possible subtrees-subset pairs. Using the well-known fact that the class of rooted trees can be constructed recursively from individual vertices, there is only one rule of composition: two trees \((T_1, r_1)\) and \((T_2, r_2)\) are combined by adding an edge between \(r_1\) and \(r_2\), and \(r_1\) is the root of the combined tree. All the possible compositions of the classes are stored in a table. This table then allows the development of recurrence relations which relate the values of the classes after each composition. Due to the conditionals in the table, the recurrence relations in these algorithms only tell the reader where to check to determine possible values. For example, combine a tree \(T_1\) with root \(r_1\) of class 1 and balance 1 with a tree \(T_2\) with root \(r_2\) of class 2 and balance 1. If we are trying to develop the recurrence relation for class 1 and the table location for \([1,1]\) says "If the balance at \(r_1\) is 1, then if the balance at \(r_2\) is 1 then \([2:i+1]\), else \([1:i+1]\), else \(X\)”, then while \([1] \times [1]\) will occur in the recurrence relation for class 1, this particular composition is invalid. When running the algorithm, two vectors are needed: one to store the cardinality of the set, and one to store the balance at the root. For some classes, the balance will not be needed, so the balance for that class will be “-”.

An example of the pseudo-code for these types of algorithms is given in Algorithm ???. All of the algorithms in this chapter follow the same pattern, with the major differences being in the description of the classes and entries in the table.
### 2.1 Offensive Alliances

A variant of the Wimer method will be used to determine the offensive alliance number of a tree.

In order to compute the offensive alliance number of a tree, we must determine the possible types of subtree/subset pairs that could be created by a minimum cardinality offensive alliance, denoted o.a. Let $S$ be the set of vertices in a minimum offensive alliance. Let $T$ be the tree for which we are computing $a_o(T)$. Let $r$ be the root of the current subtree. Because this is an offensive alliance, the only place where it is important to know the balance between nodes in the set and not in the set is when $r \not\in S$. Thus, the integer $i = |N[r] \cap S| - |N[r] \cap (V-S)|$ will only be stored for those classes.

There are 6 classes:

- **[1]** = \{ $r \in S, S \cap T$ is an o. a. in $T$ \}
- **[2]** = \{ $r \in S, S \cap T$ is not an o. a. in $T$, risanisolatedvertex \}
- **[3:i]** = \{ $r \not\in S, S \cap T$ is an o. a. in $T$ and $N[r] \cap S = \emptyset$ \}
- **[4:i]** = \{ $r \not\in S, S \cap T$ is an o. a. in $T$ and $N[r] \cap S \neq \emptyset$ \}
- **[5:i]** = \{ $r \not\in S, S \cap T$ is not an o. a. in $T$ and $N[r] \cap S \neq \emptyset$, but adding neighbors in $S$ to $r$ can create an o.a. \}
- **[6:i]** = \{ $r \not\in S, S \cap T$ is not an o.a. in $T$ and $S \cap T = \emptyset$ \}

Once these classes have been defined, we can consider the composition of a tree $T$ from these subtrees subset pairs. At each point we combine a subtree $T_1$ rooted at $r_1$ and set $S_1$ with a subtree $T_2$ rooted at $r_2$ and set $S_2$ to produce a new tree $T' = T_1 \circ T_2$ with root $r_1$ and set $S_1 \cup S_2$. We must consider all possible compositions of classes [a(i)] and [b(j)] for $1 \leq a,b \leq 6$. If a particular composition cannot occur when producing a minimum cardinality offensive alliance, then this composition is marked with an 'X'.

<table>
<thead>
<tr>
<th></th>
<th>[1]</th>
<th>[2]</th>
<th>[3:i]</th>
<th>[4:i]</th>
<th>[5:i]</th>
<th>[6:i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[2]</td>
<td>[1]</td>
<td>X</td>
<td>X</td>
<td>if ( j = -1 ) then [1] else X</td>
<td>if ( j = -1 ) then [1] else X</td>
<td>X</td>
</tr>
<tr>
<td>[3:i]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>[4:i]</td>
<td>[4:i+1]</td>
<td>[4:i+1]</td>
<td>X</td>
<td>if ( j &gt; 0 ) then [4:i-1] else X</td>
<td>if ( j &gt; 0 ) then [4:i-1] else X</td>
<td>X</td>
</tr>
<tr>
<td>[5:i]</td>
<td>if ( i = -1 ) then [4:i+1] else [5:i+1]</td>
<td>if ( i = -1 ) then [4:i+1] else [5:i+1]</td>
<td>X</td>
<td>if ( j &gt; 0 ) then [5:i-1] else X</td>
<td>X</td>
<td>[5:i-1]</td>
</tr>
<tr>
<td>[6:i]</td>
<td>if ( i = -1 ) then [4:i+1] else [5:i+1]</td>
<td>if ( i = -1 ) then [4:i+1] else [5:i+1]</td>
<td>[3:i-1]</td>
<td>if ( j &gt; 0 ) then [3:i-1] else X</td>
<td>X</td>
<td>[6:i-1]</td>
</tr>
</tbody>
</table>

**Table 2.1:** Compositions for offensive alliances.

The initial values for each class combine to form the initial vector for the algorithm. We use
either the number of vertices in the set, or ∞ to represent a state which cannot logically exist. When combining subtrees, we want to minimize the number of vertices in the set. The only classes which logically can exist initially for an isolated vertex are class 2, with a value of 1, and class 6, with a value of 0. This means that the initial vector for all vertices is [∞, 1, ∞, ∞, ∞, 0]. As the last step to set up the tree for the algorithm, we create a parent array for the tree. This means that we have an array which stores, for each vertex, the parent of that vertex, with the parent of the root being ∞. After execution of the algorithm, we need to determine the answer. Only three classes can produce a minimum offensive alliance. These are [1] where the root is in, so there is no value, and [3:x] or [4:x] where x is any value. The minimum value from the final vector in these 3 classes is the size of a minimum offensive alliance for the tree T.

From Table 2.1, we obtain a set of recurrence relations as follows:


\[ [4:i+1] = \min ([4:i] \cup [4:i] \cup [5:i] \cup [5:i] \cup [6:i] \cup [6:i] \cup [2]) \]

\[ [4:i-1] = \min ([4:i] \cup [4:i] \cup [4:i] \cup [6:j]) \]

\[ [5:i+1] = \min ([5:i] \cup [5:i] \cup [6:i] \cup [6:i] \cup [2]) \]

\[ [5:i-1] = \min ([4:i] \cup [4:i] \cup [6:j] \cup [5:i] \cup [6:j]) \]

\[ [6:i-1] = [6:i] \cup [6:j] \]

We will discuss two of the more interesting compositions from Table 2.1: [4:i] \cup [6:j], [5:i] \cup [3:j].

Consider the composition of a subtree T₁ of class [4:i] with a subtree T₂ of class [6:j]. The root r₁ of subtree T₁ is adjacent to a vertex in an offensive alliance S, while the subtree T₂ does not contain vertices in S. Thus the class of the composition depends on the i value for T₁:

1. i > 0: In this case, the addition of a subtree which doesn’t contain any vertices in an offensive alliance will not destroy the existing offensive alliance, so the resulting T’ will be class [4:i-1].

2. i ≤ 0: In this case, the addition of a subtree which doesn’t contain any vertices in an offensive alliance will negatively balance the existing offensive alliance. However, if subtrees of types [1] or [2] are combined with the result, an offensive alliance could be created. Therefore the resulting T’ will be class [5:i-1]

Finally, consider the composition of a subtree T₁ of class [5:i] with a subtree T₂ of class [3:j]. In
this class, $T_1$ contains bits and pieces which may be put together into an offensive alliance if attached to subtrees whose roots are in $S$. However, $T_2$ already contains an offensive alliance and adding it to the bits and pieces in $T_1$ will create a bigger offensive alliance than the minimum cardinality we are seeking. Thus this location in Table 2.1 is marked with an ’$X$’. 
Procedure $a_\nu(T)$

for $i = 0$ to $p$ do

- Initialize the cardinality and balance vectors for every vertex in the tree

- Initialize $c[i, 1, \ldots, 6]$ to $[\infty, 1, \infty, \infty, 0]$

- Initialize $b[i, 1, \ldots, 6]$ to $[-, \infty, \infty, -\infty, -1]$

for $j = p$ downto 1 do

- Create 2 additional vectors for the new cardinality and new balance

- $k = \text{parent}[j]$, $nc[k, 1, \ldots, 6]$ to $[\infty, \infty, \infty, \infty, \infty, \infty]$; $nb[k, 1, \ldots, 6]$ to $[\infty, \infty, \infty, \infty, \infty, \infty]$

- \[1\] if $c[k,1]+c[j,1] < nc[k,1]$ then $nc[k,1]=c[k,1]+c[j,1]$

- \[1\] if $c[k,1]+c[j,2] < nc[k,1]$ then $nc[k,1]=c[k,1]+c[j,2]$

- \[1\] if $c[k,1]+c[j,4] < nc[k,1]$ then $nc[k,1]=c[k,1]+c[j,4]$

- \[1\] if $b[j,5]=-1$ & $c[k,1]+c[j,5] < nc[k,1]$ then $nc[k,1]=c[k,1]+c[j,5]$

- \[1\] if $b[j,6]=-1$ & $c[k,1]+c[j,6] < nc[k,1]$ then $nc[k,1]=c[k,1]+c[j,6]$

- \[2\] if $c[k,2]+c[j,1] < nc[k,1]$ then $nc[k,1]=c[k,2]+c[j,1]$

- \[2\] if $c[k,2]+c[j,4] < nc[k,1]$ then $nc[k,1]=c[k,2]+c[j,4]$

- \[2\] if $b[j,5]=-1$ & $c[k,2]+c[j,5] < nc[k,1]$ then $nc[k,1]=c[k,2]+c[j,5]$

- \[2\] if $b[j,6]=-1$ & $c[k,2]+c[j,6] < nc[k,1]$ then $nc[k,1]=c[k,2]+c[j,6]$

- \[3\] if $c[k,3]+c[j,3] < nc[k,3]$ then $nc[k,3]=c[k,3]+c[j,3]$ & $nb[k,3]=b[k,3]-1$


Algorithm 2.1: Offensive Alliances on Trees
if \( b[j,4] > 0 \) \& \( b[k,4] > 0 \) \& \( c[k,4]+c[j,4] < nc[k,4] \) then \( nc[k,4]=c[k,4]+c[j,4] \) \& \( nb[k,4]=b[k,4]-1 \) 

elseif \( b[j,4] > 0 \) \& \( b[k,4] \leq 0 \) \& \( c[k,4]+c[j,4] < nc[k,5] \) then 
\( nc[k,5]=c[k,4]+c[j,4] \) \& \( nb[k,5]=b[k,4]-1 \)

\begin{align*}
\text{if } b[j,4] > 0 \text{ and } c[k,4]+c[j,6] < nc[k,4] \text{ then }
& nc[k,4] = c[k,4]+c[j,6] \& nb[k,4] = b[k,4]-1 \\
\text{elseif } c[k,4]+c[j,6] < nc[k,5] \text{ then }
& nc[k,5] = c[k,4]+c[j,6] \& nb[k,5] = b[k,4]-1 \\
\end{align*}

\begin{align*}
\text{if } b[k,5] = -1 \text{ and } c[k,5]+c[j,1] < nc[k,4] \text{ then }
\text{elseif } c[k,5]+c[j,1] < nc[k,5] \text{ then }
\end{align*}

\begin{align*}
\text{if } b[k,5] = -1 \text{ and } c[k,5]+c[j,2] < nc[k,4] \text{ then }
\text{elseif } c[k,5]+c[j,2] < nc[k,5] \text{ then }
\end{align*}

\begin{align*}
\text{if } b[j,4] > 0 \text{ and } c[k,5]+c[j,4] < nc[k,5] \text{ then }
& nc[k,5] = c[k,5]+c[j,4] \& nb[k,5] = b[k,5]-1 \\
\text{elseif } c[k,5]+c[j,4] < nc[k,6] \text{ then }
\end{align*}

\begin{align*}
\text{if } b[k,6] = -1 \text{ and } c[k,6]+c[j,1] < nc[k,4] \text{ then }
\text{elseif } c[k,6]+c[j,1] < nc[k,5] \text{ then }
\end{align*}

\begin{align*}
\text{if } b[k,6] = -1 \text{ and } c[k,6]+c[j,2] < nc[k,4] \text{ then }
\text{elseif } c[k,6]+c[j,2] < nc[k,5] \text{ then }
\end{align*}

\begin{align*}
\text{if } c[k,6]+c[j,3] < nc[k,3] \text{ then }
& nc[k,3] = c[k,6]+c[j,3] \& nb[k,3] = b[k,6]-1 \\
\text{elseif } c[k,6]+c[j,3] < nc[k,4] \text{ then }
& nc[k,4] = c[k,6]+c[j,3] \& nb[k,3] = b[k,6]-1 \\
\end{align*}

\begin{align*}
\text{if } b[j,4] > 0 \text{ and } c[k,6]+c[j,4] < nc[k,3] \text{ then }
& nc[k,3] = c[k,6]+c[j,4] \& nb[k,3] = b[k,6]-1 \\
\text{elseif } c[k,6]+c[j,4] < nc[k,6] \text{ then }
& nc[k,6] = c[k,6]+c[j,4] \& nb[k,6] = b[k,6]-1 \\
\end{align*}

\begin{align*}
\text{if } c[k,6]+c[j,6] < nc[k,6] \text{ then }
& nc[k,6] = c[k,6]+c[j,6] \& nb[k,6] = b[k,6]-1 \\
\text{else if } c[k,6]+c[j,6] < nc[k,6] \text{ then }
& nc[k,6] = c[k,6]+c[j,6] \& nb[k,6] = b[k,6]-1 \\
\end{align*}

\begin{align*}
\text{if } c[k,1 \ldots 9] = \text{newcardinality}[k, 1 \ldots 9], \text{ b}[k, 1 \ldots 9] = \text{newbalance}[k, 1 \ldots 9] \\
\end{align*}

\begin{align*}
\alpha_o(G) = \min \{ c[0, 1], c[0, 3], c[0, 4] \}
\end{align*}
2.2 Powerful Alliances

The following algorithm was published in [14].

In order to compute the powerful alliance number of a tree, we must determine the possible types of subtree/subset pairs that could be created by a minimum cardinality powerful alliance, denoted \( p.a. \). Let \( S \) be the set of vertices in a minimum powerful alliance. Let \( T \) be the tree for which we are computing \( a_p(T) \) and let \( r \) be the root of the current subtree. There are 7 classes of possible subtree subset pairs at any point in the algorithm. The current difference between the number of neighbors that are in and not in \( S \) for each class being indicated by the value \( i \), that is \( i = |N[r] \cap S| - |N[r] \cap (V-S)| \).

\[
[1:i] = \{ r \in S, S \cap T \neq \emptyset \text{ is a p. a. in } T \} \quad \text{// } i \geq 0 \quad \text{//}
\]

\[
[2:i] = \{ r \in S, S \cap T \neq \emptyset \text{ is not a p. a. in } T, \text{ but adding neighbors in } S \text{ to } r \text{ can create a p. a.} \} \quad \text{// } i < 0 \quad \text{//}
\]

\[
[3:i] = \{ r \notin S, S \cap T \neq \emptyset \text{ is exactly a p. a. in } T \text{ (the balance is 0)}, S \cap N[r] \neq \emptyset \} \quad \text{// } i = 0 \quad \text{//}
\]

\[
[4:i] = \{ r \notin S, S \cap T \neq \emptyset \text{ is a p. a. in } T \text{ but the balance at } r \text{ is positive}, S \cap N[r] \neq \emptyset \} \quad \text{// } i > 1 \quad \text{//}
\]

\[
[5:i] = \{ r \notin S, S \cap T \neq \emptyset \text{ is a p. a. in } T, S \cap N[r] = \emptyset \} \quad \text{// } i < 0, \text{ but there is a p.a. below the root} \quad \text{//}
\]

\[
[6:i] = \{ r \notin S, S \cap T = \emptyset \} \quad \text{// } i < 0 \quad \text{//}
\]

\[
[7:i] = \{ r \notin S, S \cap T \neq \emptyset \text{ and } S \text{ is not a p. a. in } T, \text{ but adding neighbors in } S \text{ to } r \text{ can create a p. a.} \} \quad \text{// } i < 0 \quad \text{//}
\]

Once these classes have been defined, we can consider the composition of tree \( T \) from its subtrees. At each point we combine a subtree \( T_1, \) rooted at \( r_1, \) with set \( S_1, \) and balance \( i, \) with a subtree \( T_2, \) rooted at \( r_2, \) set \( S_2, \) and balance \( j, \) to produce a new subtree \( T = T_1 \diamond T_2 \) with root \( r_1 \) and set \( S_1 \cup S_2. \) We must consider all possible compositions of classes \( [a:i] \) and \( [b:j], \) for \( 1 \leq a,b \leq 7. \)

The recurrence relations are obtained as before. When combining subtrees, we want to minimize the number of vertices in the set. The only classes which logically can exist initially for an isolated vertex are class 1, with a value of 1, and class 6, with a value of 0. This means that the initial vector for all vertices is \([1, \infty, \infty, \infty, \infty, 0, \infty]\). After execution of the algorithm, we need to determine the answer. Only four classes can produce a minimum powerful alliance. These are \([3:x], [4:x], \) or \([5:x]\) where \( x \) can be any value, or \([1:(x>0)]\). The minimum value from the final vector in these 4 classes is the size of a minimum powerful alliance for the tree \( T. \)

We will discuss three of the more interesting entries in Table 2.2: \([2:i] \diamond [2:j], [4:i] \diamond [6:j], \) and \([6:i] \diamond [5:j]. \)
Table 2.2: Compositions for powerful alliances.

<table>
<thead>
<tr>
<th></th>
<th>[1:j]</th>
<th>[2:j]</th>
<th>[3:j]</th>
<th>[4:j]</th>
<th>[5:j]</th>
<th>[6:j]</th>
<th>[7:j]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1:i]</td>
<td>[1:i+1]</td>
<td>if (j = -1) then X</td>
<td>if (j &gt; 0) then [1:i-1] else [2:i-1]</td>
<td>if (i &gt; 0) then [1:i-1] else [2:i-1]</td>
<td>X</td>
<td>if (j = -1) then X</td>
<td>if (j &gt; 0) then [1:i-1] else [2:i-1]</td>
</tr>
<tr>
<td>[2:i]</td>
<td>if (i = -1) then [1:i+1] else [2:i+1]</td>
<td>if (j = -1) then X</td>
<td>if (i = -1) then X</td>
<td>X</td>
<td>if (j = -1) then [2:i-1] else X</td>
<td>if (j &gt; 0) then [1:i-1] else X</td>
<td></td>
</tr>
<tr>
<td>[3:i]</td>
<td>if (j &gt; 0) then [4:i+1] else X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>[7:i-1]</td>
<td>[7:i-1]</td>
<td></td>
</tr>
<tr>
<td>[4:i]</td>
<td>if (j &gt; 0) then [4:i+1] else X</td>
<td>X</td>
<td>if (j = 1) then [3:i-1] else [4:i-1]</td>
<td>X</td>
<td>if (i = -1) then [3:i-1] else [4:i-1]</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>[5:i]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>[5:i-1]</td>
<td>[5:i-1]</td>
<td>[6:i-1]</td>
</tr>
<tr>
<td>[6:i]</td>
<td>if (j &gt; 0) then (if (i = -1) then [3:i+1] else [7:i+1]) else X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>if (j = -1) then [3:i-1] else [4:i-1]</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>[7:i]</td>
<td>if (j &gt; 0) then (if (i = -1) then [3:i+1] else [7:i+1]) else X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>[7:i-1]</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

To start, consider the composition of a subtree $T_1$ of class [2:i] with a subtree $T_2$ of class [2:j]. If either $i$ or $j = -1$, then the root has 2 more children not in the set than in the set. If either $i$ or $j$ is less than -1, then root has 2 or more children not in the set than in the set. So, if we try to combine a tree with $i \leq -1$ with a tree with $j < -1$, we do not get a valid tree because the tree with $j < -1$ will never be combined with anything else and its root is not defined. If $j = -1$ there are two choices:

1. $i = -1$ : This forms a powerful alliance because the addition of a child $r2$ in the set $S$ creates a powerful alliance for the subtree $T_1 ([2:i])$ and the addition of a parent in the set $S$ creates a powerful alliance for the subtree $T_2 ([2:j])$. Thus, the new tree $T$ is [1:i+1]

2. $i < -1$: This does not form a powerful alliance in the subtree $T_1 ([2:i])$. However, the addition of a parent in the set $S$ does create a powerful alliance in the subtree $T_2 ([2:j])$, so the new tree $T$ is $[2:i+1]$ since the composition adds one more defender for $r_1$.

Consider next the composition of a subtree $T_1$ of class [4:i] with a subtree $T_2$ of class [6:j]. Because $T_2$ has no vertices in the set $S$, the value of $j$ has no bearing on the composition. This means that there are two possibilities for tree $T_1$:

1. $i = 1$ : Combining these two trees takes $i$ to 0. This means that currently we have a powerful alliance exactly in the subtree $T_1$. This is the definition of class 3, so the new tree is [3:i-1].
2. $i > 1$: Adding a child to $r_1$ which is not in $S$ will still leave a powerful alliance which is not minimal in $T_1$, so this composition is $[4:i-1]$.

Finally, consider the composition of a subtree $T_1$ of class $[6:i]$ with a subtree $T_2$ of class $[5:j]$. Because a powerful alliance exists somewhere in $T_2$, a powerful alliance will still exist in the combined tree at a level below the children of the root, so the new tree is $[5:i-1]$.

### 2.3 Global Defensive Alliances

In order to compute the global defensive alliance number of a tree, we must determine the possible types of subtree/subset pairs that could be created by a minimum cardinality global defensive alliance, denoted g.d.a. Let $S$ be the set of vertices in a minimum global defensive alliance. Let $T$ be the tree for which we are computing $\gamma_a(T)$. Let $r$ be the root of the current subtree. Because this is a defensive alliance, the only place where it is important to know the balance between nodes in the set and not in the set is when $r \in S$. Thus, the integer $i = |N[r] \cap S| - |N[r] \cap (V-S)|$ will only be stored for those classes. There are 4 classes:

- **[1:i]**: $\{r \in S, S \cap T \text{ is a g. d. a. in } T\}$ for $i \geq 0$
- **[2:i]**: $\{r \in S, S \cap T \text{ is not a g. d. a. in } T, \text{ but adding neighbors in } S \text{ to } r \text{ can create a g. d. a.}\}$ for $i < 0$
- **[3]**: $\{r \notin S, S \cap T \text{ is a g. d. a. in } T\}$
- **[4]**: $\{r \notin S, S \cap T \text{ is not a g. d. a. in } T, \text{ but adding a neighbor in } S \text{ to dominate } r \text{ can create a g. d. a.}\}$

Once these classes have been defined, we can consider the composition of a tree $T$ from these subtree/subset pairs. At each point we combine a subtree $T_1$ rooted at $r_1$ and set $S_1$ with a subtree $T_2$ rooted at $r_2$ and set $S_2$ to produce a new tree $T' = T_1 \diamond T_2$ with root $r_1$ and set $S_1 \cup S_2$. We must consider all possible compositions of classes $[a(i)]$ and $[b(j)]$ for $1 \leq a,b \leq 4$.

<table>
<thead>
<tr>
<th></th>
<th>[1:i]</th>
<th>[2:i]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1:i]</td>
<td>[1:i+1]</td>
<td>if $(j = -1)$ then [1:i+1] else X</td>
<td>if $(i &gt; 0)$ then [1:i-1] else [2:i-1]</td>
<td>if $(i &gt; 0)$ then [1:i-1] else [2:i-1]</td>
</tr>
<tr>
<td>[2:i]</td>
<td>if $(i = -1)$ then [1:i+1] else [2:i+1]</td>
<td>if $(j = -1)$ then [1:i+1] else [2:i+1]</td>
<td>[2:i-1]</td>
<td>[2:i-1]</td>
</tr>
</tbody>
</table>

**Table 2.3:** Compositions for global defensive alliances.

As before, we obtain the recurrence relations from Table 2.3. When combining subtrees, we want to minimize the number of vertices in the set. The only classes which logically can exist initially
for an isolated vertex are class 1, with a value of 1, and class 4, with a value of 0. This means that
the initial vector for all vertices is [1, ∞, ∞, 0]. Only two classes can produce a minimum global
defensive alliance. These are [3] where the root is not in, so there is no value, or [1:x] where x can
be any non-negative value. The minimum value from the final vector in these 2 classes is the size of
a minimum global defensive alliance for the tree $T$.

We will discuss two of the more interesting classes from the 16 entries in in Table 2.3: [1:i]⊙[3]
and [3]⊙[2:j].

To start, consider the composition of a subtree $T_1$ of class [1:i] with a subtree $T_2$ of class [3].
Because each subtree already contains a valid global defensive alliance, the only concern is that the
composition will destroy the global defensive alliance for one of the subtrees. However, only the
root of $T_1$ is in the set, so we can’t destroy the global defensive alliance for $T_2$. This means that if i
is greater than 0, the resulting $T'$ will be class [1:i-1], otherwise the resulting $T'$ will be class [2:i-1]
because adding more subtrees of class [1:x] or [2:x] could create a global defensive alliance again.

Next, consider the composition of a subtree $T_1$ of class [3] with a subtree $T_2$ of class [2:j].
Because $T_2$ does not contain a valid global defensive alliance and this is the last composition it will
be involved in, the fact that it is being combined with $T_1$ whose root $r_1$ is not in the set, means
that this composition cannot occur in a tree containing a valid global defensive alliance. Thus this
location in Table 2.3 is marked ‘X’.

### 2.4 Global Offensive Alliances

In order to compute the global offensive alliance number of a tree, we must determine the possible
types of subtrees that could be created by a minimum cardinality global offensive alliance, denoted
$\gamma_o(T)$. Let $r$ be the root of the current subtree. Because this is an offensive
alliance, the only place where it is important to know the balance between nodes in the set and not
in the set is when $r \notin S$. Thus, the integer $i = |N[r] \cap S| - |N[r] \cap (V-S)|$ will only be stored for those
classes. Again, $r$ is the root of the tree $T$. There are 4 classes:

- $[1] = \{ r \in S, S \cap T \text{ is not a g. o. a. in } T \}$
- $[2] = \{ r \in S, S \cap T \text{ is a g. o. a. in } T \}$
- $[3:i] = \{ r \notin S, S \cap T \text{ is a g. o. a. in } T \}$
- $[4:i] = \{ r \notin S, S \cap T \text{ is not a g. o. a. in } T, \text{ but adding neighbors in } S \text{ to } r \text{ can create a g. o. a.} \}$
Once these classes have been defined, we can consider the composition of a tree $T$ from these subtrees. At each point we combine a subtree $T_1$ rooted at $r_1$ and set $S_1$ with a subtree $T_2$ rooted at $r_2$ and set $S_2$ to produce a new tree $T' = T_1 \odot T_2$ with root $r_1$ and set $S_1 \cup S_2$. We must consider all possible compositions of classes $[a(i)]$ and $[b(j)]$ for $1 \leq a, b \leq 4$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3:j</th>
<th>4:j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>if (j = -1) then [2] else X</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>if (j = -1) then [2] else X</td>
</tr>
<tr>
<td>3:i</td>
<td>[3:i+1]</td>
<td>[3:i+1]</td>
<td>if (j &gt; 0) then (if (i = -1) then [3:i-1] else [4:i-1]) else X</td>
<td></td>
</tr>
<tr>
<td>4:i</td>
<td>if (i = -1) then [3:i+1] else [4:i+1]</td>
<td>if (i = -1) then [3:i+1] else [4:i+1]</td>
<td>if (j &gt; 0) then [4:i-1] else X</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.4: Compositions for global offensive alliances.

The recurrence relations are obtained from Table 2.4 as before. When combining subtrees, we want to minimize the number of vertices in the set. The only classes which logically can exist initially for an isolated vertex are class 1, with a value of 1, and class 4, with a value of 0. This means that the initial vector for all vertices is $[1, \infty, \infty, 0]$. Only two classes can produce a minimum global offensive alliance. These are [2] where the root is in, so there is no value, or [3:x] where x can be any value. The minimum value from the final vector in these 2 classes is the size of a minimum offensive alliance for the tree $T$.

We will discuss two of the more interesting classes of the 16 entries in Table 2.4: $[1] \odot [4:j]$ and $[4:i] \odot [2]$.

To start, consider the composition of a subtree $T_1$ of class [1] with a subtree $T_2$ of class [4:j]. Because $T_2$ does not contain a valid global offensive alliance, we are only concerned with whether we can create one in $T_2$. The only way to create one, because $T_2$ will not be combined with anything else, is for $j$ to be -1. If this is the case, then the addition of $r_1$ which is in the set, will successfully attack $r_2$, creating a powerful alliance. If $j$ is less than -1, then we cannot create a valid global offensive alliance out of this composition. Thus, if $j = -1$, the resulting $T'$ is class [1], otherwise the composition cannot occur and is marked with an 'X'.

Next consider the composition of a subtree $T_1$ of class [4:i] with a subtree $T_2$ of class [2]. Because $T_2$ already contains a global offensive alliance and $r_2$ is in the set, composition with $T_1$ cannot destroy the existing global offensive alliance. So, if $i = -1$, then the resulting $T'$ will be of class [3:i+1] because $r_2$, along with the other children of $r_1$ that are in the set, can attack $r_1$, but if $i < -1$, then the resulting $T'$ will be of class [4:i+1] because adding more neighbors in the set to $r_1$ can create a global offensive alliance, but a global offensive alliance does not currently exist.
2.5 Global Powerful Alliances

The following algorithm was published in [14].

In order to compute the global powerful alliance number of a tree, we must determine the possible types of subtrees that could be created by a minimum cardinality global powerful alliance, denoted g.p.a. Let S be the set of vertices in a minimum global powerful alliance. Let T be the tree for which we are computing \( \gamma_a(T) \). Let \( r \) be the root of the current subtree. There are only 4 classes:

\[
\begin{align*}
[1:i] &= \{ r \in S, S \cap T \text{ is a g. p. a. in } T \}/^* i \geq 0 */ \\
[2:i] &= \{ r \in S, S \cap T \text{ is not a g. p. a. in } T, \text{ but adding neighbors in } S \text{ to } r \text{ can create a g. p. a.} \}/^* i < 0 */ \\
[3:i] &= \{ r \notin S, S \cap T \text{ is a g. p. a. in } T \}/^* i \geq 0 */ \\
[4:i] &= \{ r \notin S, S \cap T \text{ is not a g. p. a. in } T, \text{ but adding neighbors in } S \text{ to } r \text{ can create a g. p. a.} \}/^* i < 0 */
\end{align*}
\]

Once these classes have been defined, we can consider the composition of a tree \( T \) from these subtrees. At each point we combine a subtree \( T_1 \) rooted at \( r_1 \) and set \( S_1 \) with a subtree \( T_2 \) rooted at \( r_2 \) and set \( S_2 \) to produce a new tree \( T' = T_1 \circ T_2 \) with root \( r_1 \) and set \( S_1 \cup S_2 \). We must consider all possible compositions of classes \([a:i]\) and \([b:j]\) for \( 1 \leq a, b \leq 4 \).

<table>
<thead>
<tr>
<th>([1:i])</th>
<th>([1:i+1])</th>
<th>(\text{if } (j &gt; 0) \text{ then } [1:i+1] \text{ else } X)</th>
<th>(\text{if } (i &gt; 0) \text{ then } [1:i-1] \text{ else } [2:i-1])</th>
</tr>
</thead>
<tbody>
<tr>
<td>([2:i])</td>
<td>(\text{if } (j &gt; 0) \text{ then } [1:i+1] \text{ else } X)</td>
<td>(\text{if } (i &gt; 0) \text{ then } [1:i-1] \text{ else } [2:i-1])</td>
<td>(\text{if } (j = -1) \text{ then } [2:i-1] \text{ else } X)</td>
</tr>
<tr>
<td>([3:i])</td>
<td>(\text{if } (j &gt; 0) \text{ then } [3:i+1] \text{ else } X)</td>
<td>(\text{if } (j &gt; 0) \text{ then } [3:i-1] \text{ else } [4:i-1] \text{ else } X)</td>
<td>(X)</td>
</tr>
<tr>
<td>([4:i])</td>
<td>(\text{if } (j &gt; 0) \text{ then } [3:i+1] \text{ else } X)</td>
<td>(\text{if } (j &gt; 0) \text{ then } [4:i-1] \text{ else } X)</td>
<td>(X)</td>
</tr>
</tbody>
</table>

Table 2.5: Compositions for global powerful alliances.

As before, we obtain the recurrence relations from Table 2.5. When combining subtrees, we want to minimize the number of vertices in the set. The only classes which logically can exist initially are class 1, with a value of 1, and class 4, with a value of 0. This means that the initial vector is \([1, \infty, \infty, 0]\). Two states can determine a global powerful alliance. These are \([3:x]\) where \(x \geq 0\), or \([1:(x>0)]\). The minimum value from the final vector in these two classes is the size of the minimum global powerful alliance for that tree.
As before, we will only discuss a couple of entries in Table 2.5. We will examine two entries which are different from those examined in the powerful alliances section: [3:i]○[2:j] and [4:i]○[1:j].

First, consider the composition of a subtree $T_1$ of class [3:i] with a subtree of class [2:j]. A subtree of class [3:i] will have $i \geq 0$ more children of the root in the set $S$ than not in $S$. A subtree of type [2:j] has a root with a negative balance. This composition is invalid because this is the last composition involving the [2:j] tree $T_2$ which will have $r_2$ with a negative balance. Adding another vertex not in $S$ adjacent to the root $r_2$ of $T_2$ is not going to make a global powerful alliance. This is therefore, an invalid composition, which is denoted with an X.

Next, consider the composition of a subtree $T_1$ of class [4:i] with a subtree $T_2$ of class [1:j]. A subtree of class [4:i] has a root with a negative balance, $i < 0$. A subtree of class [1:j] has a root with a non-negative balance, $j \geq 0$. For this composition, if $j = 0$, then combining with a subtree of type [4:i] will add a parent of [1:j] which is not in $S$, which means that a global powerful alliance will no longer exist. So we consider $j > 0$. There are 2 options:

1. $i = -1$: [1:j] will still be a global powerful alliance with [4:i] and [4:i] will now be a global powerful alliance, so it becomes a [3:i+1].

2. $i < -1$: [1:j] will still be a global powerful alliance when connected to [4:i]. However, [4:i] will not be a global powerful alliance, so it becomes [4:i+1].

### 2.6 Strong Alliances

There are very few changes needed to turn any of the above linear algorithms for finding the minimum alliance number of a given type into linear time algorithms for finding the minimum strong alliance number of a given type. In fact, the only changes would be in the conditionals given in the composition tables. For each conditional in the table increase the value being checked by 1. For instance, Table ?? for finding a minimal defensive alliance, if changed to find a minimal strong defensive alliance would be Table 2.6.

No other changes are necessary and this change to the table does not change the recurrence relations. These changes will find a minimal strong defensive alliance because the only difference between a minimal defensive alliance and a minimal strong defensive alliance is that it is necessary for a strong alliance to strictly outnumber the set of vertices not in the alliance at each vertex where the balance is verified.
Table 2.6: Compositions for minimal strong defensive alliances.

<table>
<thead>
<tr>
<th>[1:i]</th>
<th>[2:j]</th>
<th>[3:j]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1:i]</td>
<td>if (i = 2) then [ if (j=1) then [2:i+1] else X] else [1:i+1]</td>
<td>X</td>
<td>if (j=0) then { if i=1 then [1:i+1] else [2:i+1] else X } X</td>
<td>if (t &gt; 0) then [1:i-1] else [3:i-1]</td>
</tr>
<tr>
<td>[2:i]</td>
<td>if (j=1) then [2:i+1] else X</td>
<td>X</td>
<td>if (j = 0) then [2:i+1] else X</td>
<td>if (i=3) then [1:i-1] else [2:i-1]</td>
</tr>
<tr>
<td>[3:i]</td>
<td>if (i=0) then [ if (j=1) then [1:i+1] else X ] else [3:i+1]</td>
<td>X</td>
<td>if (j=0) then { if i=0 then [1:i+1] else [3:i+1] else X } X</td>
<td>[3:i-1]</td>
</tr>
</tbody>
</table>
Chapter 3

Alliance Complexity

When studying concepts in graph theory, it is useful to know the algorithmic complexity of a problem in question. In general, finding an optimal alliance of any type is thought to be NP-complete. In order to later more closely define the split between polynomial time solvable problems and NP-complete alliance problems, we would like to determine if alliance questions are NP-complete even when restricted to bipartite or chordal graphs. We will determine the complexity of the several alliance decision problems.

3.1 Defensive Alliance Complexity

DEFENSIVE ALLIANCE

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

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

The following proof was published in [14].

**Theorem 3.1** DEFENSIVE ALLIANCE is NP-complete, even when restricted to split or chordal graphs.

*Proof.* DEFENSIVE ALLIANCE is clearly in NP. A set \( S \) of size at most \( k \), could be given as a witness to a ‘yes’ instance and verified in \( O(E) \) time to be a defensive alliance. In order to show that DEFENSIVE ALLIANCE is NP-complete we construct a transformation from the following well-known NP-complete problem.

VERTEX COVER

INSTANCE: Set \( X = \{x_1, x_2, \ldots, x_n\} \), collection \( C = \{C_1, C_2, \ldots, C_m\} \) of subsets of \( X \) where \( |C_i| = 2 \), positive integer \( k < |X| \).

QUESTION: Does there exist a subset \( Y \subset X \) with \( |Y| \leq k \) such that \( Y \) contains at least one element from each subset in \( C \)?
Since we want each element to appear in at least two subsets, we can repeat any subset that contains an element that appears in only one subset, and if some element appears in no subset, then w.l.o.g. we can remove it from $X$.

Transformation: Let $X$, $C$, $k$ be an arbitrary instance of the VERTEX COVER problem.
Create an instance $\text{DA}(G, k)$ of DEFENSIVE ALLIANCE in this way:

1. For each $x_i \in X$, create a singleton vertex $x_i$.

2. For $|C| = m$, create $2m$ vertices labeled $c_1, c_2, \ldots, c_m, c_{m+1}, \ldots, c_{2m}$ and form a clique among these $2m$ vertices.

3. Add $2k+1$ independent vertices: $z_1, z_2, \ldots, z_{2k+1}$. Add all possible edges between the $z$ vertices and the second half of the $c$ vertices $c_{m+1}, c_{m+2}, \ldots, c_{2m}$.

4. For each vertex $c_i$, $1 \leq i \leq m$, if $C_i = \{x_r, x_s\}$ then add edges $c_i x_r$ and $c_i x_s$.

5. Set $k' = m+k$ (recall that $|C| = m$).

Example:
$X = \{x_1, x_2, x_3, x_4, x_5\}$
$C = \{\{x_1, x_4\}, \{x_1, x_3\}, \{x_2, x_5\}, \{x_4, x_5\}, \{x_2, x_1\}, \{x_3, x_5\}\}$
$k = 2$

![Diagram](image)

**Figure 3.1:** Defensive Alliance Example

Note that there is a clique defined on the set of $c$ vertices; all of these edges are not shown in Figure 3.1.
Claim: Given a vertex cover $Y$ of size at most $k$, the set $S = Y \cup \{c_1, c_2, \ldots, c_m\}$ is a defensive alliance of cardinality $\leq k + M = k'$.  

The x-vertices in $S$ will have $N[x] \cap S = N[x]$, so they satisfy the definition of a defensive alliance. Each of the vertices $c_1, c_2, \ldots, c_m$ has degree $2m+1 + 2 = 2m+2$, since all of the vertices $c_1, \ldots, c_m$ are in $S$ and at least one of the two x-vertices adjacent to each $c_i$ is in $S$ (from the vertex cover) $|N[c_i] \cap S| \geq m+1$ for each $c_i, 1 \leq i \leq m$. Thus, $S$ is a defensive alliance of cardinality $\leq k' = m+k$.  

Conversely, suppose we have a defensive alliance $S$ of size at most $m+k$.  

1. The c-vertices $c_{m+1}, \ldots, c_{2m}$ have degree $2m+1 + 2k + 1$. Therefore $|N[c_i]| = 2m+2k+1$, for $m+1 \leq i \leq 2m$. Therefore, no defensive alliance of size at most $m+k$ can contain a vertex in $\{c_{m+1}, \ldots, c_{2m}\}$.  

2. If a z vertex is in $S$, then some of its neighbors must be in $S$, but its only neighbors come from $c_{m+1}, \ldots, c_{2m}$, and we just ruled this out.  

3. It follows that only the x-vertices and the vertices $C_m = \{c_1, \ldots, c_m\}$ can be elements of $S$. Since each x-vertex is assumed to be in at least two subsets, if an x-vertex is in $S$, then some c vertex must also be in $S$ as well. But since the degree of each $c_i$ vertex is $2m+1$ at least $m$ of the x and $C_m$ neighbors of a c-vertex in $S$ must also be in $S$. Therefore, either all of the vertices $\{c_1, c_2, \ldots, c_m\} \subseteq S$, or all but one of these vertices are in $S$.  

   (a) $\{c_1, c_2, \ldots, c_m\} \subseteq S$. Then at most $k$ x-vertices are in $S$, since $|S| \leq k + m$. Each $c_i$ in $S$ must be adjacent to at least one x vertex, since at least $m+1$ of the vertices in $c_i$’s closed neighborhood must be in $S$. Therefore the x-vertices, of which there are at most $k$, in $S$ form a vertex cover.  

   (b) $m-1$ of the vertices $c_1, c_2, \ldots, c_m$ are in $S$. Each of the c-vertices in $S$ must have $m+1$ vertices chosen from their closed neighborhood. So both of the x-vertices they are adjacent to must be in $S$. There are at most $k+1$ x-vertices in $S$ ($|S| - (m-1) = k+1$) and these x-vertices “hit each subset twice”. Because each of these x-vertices appears in at least two subsets, selecting any $k$ of these x-vertices produces a vertex cover because each of the $c_1, c_2, \ldots, c_m$ will have at least one x-vertex in $S$.  

Note that the graph $G$ is a split graph, since the vertices can be partitioned into an independent set (the x and z vertices) and a clique (the c-vertices). Split graphs are also chordal. Thus DEFENSIVE ALLIANCE remains NP-complete even when restricted to split or chordal graphs. □
The NP-completeness of DEFENSIVE ALLIANCE when restricted to bipartite graphs can be shown by making a few other changes to the construction for split or chordal graphs.

**Theorem 3.2** DEFENSIVE ALLIANCE is NP-complete, even when restricted to bipartite graphs.

**Proof.** As above, DEFENSIVE ALLIANCE is clearly in NP. A set $S$ of size at most $k$, could be given as a witness to a ‘yes’ instance and verified in $O(E)$ time to be a defensive alliance. In order to show that DEFENSIVE ALLIANCE is NP-complete we construct a transformation from VERTEX COVER.

Transformation: Let $X, C, k$ be an arbitrary instance of the VERTEX COVER problem.

Create an instance $DA(G, k)$ of DEFENSIVE ALLIANCE in this way:

1. For each $x_i \in X$, create a singleton vertex $x_i$.
2. For $|C| = m$, create $2m$ vertices labeled $c_1, c_2, \ldots, c_m, c_{m+1}, \ldots, c_{2m}$ and connect each pair of vertices $c_i$ and $c_{2i}$.
3. Add $2k+1$ independent vertices: $z_1, z_2, \ldots, z_{2k}, z_{2k+1}$. Add all possible edges between the $z$ vertices and the second half of the $c$ vertices $c_{m+1}, c_{m+2}, \ldots, c_{2m}$.
4. For each vertex $c_i$, $1 \leq i \leq m$, if $C_i = \{x_r, x_s\}$ then add edges $c_ix_r$ and $c_ix_s$.
5. Set $k' = m+k$ (recall that $|C| = m$).

Claim: Given a vertex cover $Y$ of size at most $k$, the set $S = Y \cup \{c_1, c_2, \ldots, c_m\}$ is a defensive alliance of cardinality $\leq k$.

The $x$-vertices in $S$ will have $N[x] \cap S = N[x]$, so they satisfy the definition of a defensive alliance. Each of the vertices $c_1, c_2, \ldots, c_m$ has degree 3, so $|N[c_i]| = 4$. Since all of the vertices $c_1, \ldots, c_m$ are in $S$ and at least one of the two $x$-vertices adjacent to each $c_i$ is in $S$ (from the vertex cover) $|N[c_i]| \cap S = 2$ for each $c_i$, $1 \leq i \leq m$. Thus, $S$ is a defensive alliance of cardinality $\leq k' = m+k$.

Conversely, suppose we have a defensive alliance $S$ of size at most $m+k$. This defensive alliance cannot contain any of the $z_1, \ldots, z_{2k+1}$ and $c_{m+1}, \ldots, c_{2m}$ vertices because if $S$ contains any of the $c_{m+1}, \ldots, c_{2m}$ vertices, it will need to contain at least one more than half of the $z_i$ vertices, but this alliance would be more than $m+k$. Because this alliance is defensive and of cardinality $m+k$, it will contain all of the $c_1, \ldots, c_m$ vertices and $k$ of the $x_1, \ldots, x_n$ vertices. In order for this to be a defensive alliance, each of the $c_1, \ldots, c_m$ vertices need to have in its neighborhood at least one of the $x$-vertices in $S$. Therefore, the $k$ $x$-vertices will represent a vertex cover.  

\[\square\]
Example:

\[ X = \{x_1, x_2, x_3, x_4, x_5\} \]
\[ C = \{\{x_1, x_4\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_4, x_5\}, \{x_2, x_1\}, \{x_3, x_5\}\} \]
\[ k = 2 \]

\[ \text{Figure 3.2: Defensive Alliance Example} \]

Note that the graph \( G \) is a bipartite graph, since the vertices can be partitioned into two independent sets. Thus DEFENSIVE ALLIANCE remains NP-complete even when restricted to bipartite graphs.

### 3.2 Powerful Alliance Complexity

**POWERFUL ALLIANCE**

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

**QUESTION:** Does \( G \) have a powerful alliance of size at most \( k \)?

The following theorem was published in [14].

**Theorem 3.3** POWERFUL ALLIANCE is NP-complete, even when restricted to bipartite graphs.

**Proof.** POWERFUL ALLIANCE is in NP. A set \( S \) of size at most \( k \) can be given as a witness to a ‘yes’ instance and verified in \( O(E) \) time to be a powerful alliance (both offensive and defensive). We construct a polynomial-time transformation from the following, well known NP-complete problem.
DOMINATING SET

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

QUESTION: Does $G$ have a dominating set of cardinality $\leq k$?

Given an instance $G = (V, E), k$ of DOMINATING SET, we construct the following graph $H$.

1. We first construct what is called the $VV$-graph, with vertices labeled $a_1, \ldots, a_n$ (A) and $v_1, \ldots, v_n$ (V) where $v_i$ is adjacent to $a_j$ and $v_j$ is adjacent to $a_i$ if $i$ and $j$ are adjacent in $E$.

2. To this we add another set of $n$ independent vertices labeled $b_1, \ldots, b_n$ (B) and form a $VV+$-graph with the A vertices where $b_i$ is adjacent to $a_j$ and $v_j$ is adjacent to $a_i$ if $i$ and $j$ are adjacent in $E$ or $i = j$.

3. We then add an additional $n$ vertices $c_1, \ldots, c_n$ (C) that form a $VV$-graph with both the V and B vertices.

4. We add one additional vertex $d_1$, which is adjacent to each of the V vertices and add an additional $5n$ independent vertices, $e_1, \ldots, e_{5n}$ (E), that are only adjacent to vertex $d_1$.

Let $H$ denote the graph so constructed, see Figure 3.3 where not all $E$ vertices are shown.

Claim: Graph $G$ of order $n$ has a dominating set of size at most $k$ if and only if there is a powerful alliance of size at most $k + 2n$ in H. Highlighted is a dominating set of size $k = 2$ for the original graph $G$ and a powerful alliance of size $2 + 2 \times 5$ for the constructed graph $H$.

If $G$ has a dominating set of size $k$, then there is a powerful alliance $S$ of size $k + 2n$ as follows:

1. Let $S = B \cup C$ where B and C are the sets of vertices added in steps 1 and 2 of the transformation. This gives us $2n$ vertices.

2. Next, add to $S$ the vertices in A corresponding to the dominating set in the original graph $G$. This gives us an additional $k$ vertices.

The set $S$ is a defensive alliance because the chosen vertices in $S$ fall into one of three categories.

1. The B vertices have as neighbors the C vertices corresponding to those they were connected in the original graph and the A vertices to which they were connected in the original graph, plus its corresponding vertex in the original graph because $a_i$ is adjacent to $b_i$ for $1 \leq i \leq n$ in $VV+$. This means that, including itself, each of the vertices has $2\deg(i)+1$ neighbors in $H$ and at least $\deg(i)+1$ neighbors in $S$.  

26
2. The C vertices $c_i$ have $\text{deg}(i)$ neighbors in B and $\text{deg}(i)$ neighbors in V. This means that, including itself, each of the C vertices has $\text{deg}(i)$ neighbors in $S$ and at most $\text{deg}(i)$ neighbors not in $S$.

3. The A vertices in the dominating set $S$ have as neighbors the B vertices to which they were connected in $G$ plus its corresponding vertex in the original graph. This means they have $\text{deg}(i)+1$ neighbors that are in $S$ and $\text{deg}(i)$ neighbors not in $S$. Thus, including itself, each of the A vertices in $S$ has $\text{deg}(i)+2$ neighbors in $S$ and $\text{deg}(i)$ neighbors not in $S$.

Example: Let $n = 5$ and $k = 2$.

![Figure 3.3: Powerful Alliance Example](image)

Each of these cases combine to form a defensive alliance. The set $S$ is also an offensive alliance because the vertices in the boundary of $S$ fall into one of 2 categories.

1. The A vertices which are not part of the dominating set in the original graph have as neighbors the B vertices to which they were connected in the original graph, plus its corresponding vertex in the original graph, which means $\text{deg}(i)+1$ vertices that are in $S$, and the V vertices to which they were connected in the original graph, that is, $\text{deg}(i)$ neighbors not in $S$. This means that, including itself, each of the vertices has $\text{deg}(i)+1$ neighbors in $S$ and $\text{deg}(i)+1$ neighbors not
in $S$.

2. The V vertices have as neighbors the C vertices to which they were connected in the original graph, which means deg(i) vertices that are in $S$, and the A vertices to which they were connected in the original graph plus the d-vertex. Since the dominating set of the original graph is in $S$, at least one of the A-neighbors must be in $S$ and at most deg(i)-1 A-neighbors are not in $S$. Including itself, each of the vertices has at least deg(i)+1 neighbors in $S$ and deg(i)+1 neighbors not in $S$.

Because $S$ is both an offensive and a defensive alliance, by definition, it is a powerful alliance of size at most $k+2n$.

Conversely, if graph $H$ has a powerful alliance $S$ of size at most $k+2n$, then we must show that $G$ has a dominating set $D$ of size at most $k$:

1. If $S$ contains any of the E or V vertices or vertex $d_1$, then $S$ must contain at least half of them because the size of the closed neighborhood of $d_1$ is $6n+1$ and, in order to have a powerful alliance, if any of the vertices in $N[d_1]$ are in $S$, at least half of these vertices must be in $S$. However, half of the neighbors of $d_1$ would be $\lceil(6n+1)/2\rceil$ which is more than $k+2n$. Thus $S$ has no vertices in E or V and $d_1 \notin S$.

2. If $S$ is a powerful alliance of $H$ and $S \cap E = \emptyset$, $S \cap \{d\} = \emptyset$ and $S \cap V = \emptyset$, then it is easy to see that:

(a) $S \subseteq A$ is not possible, ($S$ would not dominate half of $N[v_i]$ and would not be a defensive alliance)

(b) $S \subseteq B$ is not possible, ($S$ would not dominate half of $N[c_j]$ and would not be a defensive alliance)

(c) $S \subseteq C$ is not possible ($S$ would not dominate half of $N[v_i]$ or $N[b_j]$ and would not be a defensive alliance)

(d) $S \subseteq A \cup B$ is not possible ($S$ would not dominate half of $N[c_i]$ or $N[v_j]$)

(e) $S \subseteq A \cup C$ is not possible ($S$ would not dominate half of $N[a_i]$ for $a_i \notin S$, and would not be a defensive alliance)

(f) $S \subseteq B \cup C$ is not possible ($S$ would not dominate half of $N[v_i]$)

Thus, $S \cap A \neq \emptyset$, $S \cap B \neq \emptyset$ and $S \cap C \neq \emptyset$. 

28
3. If we do not take a given C-vertex, then it cannot be dominated (only by B-vertices), thus we must take all C-vertices. This, in turn, means we must take all B-vertices, else some C-vertex cannot be defended. Therefore, in order to have a powerful alliance, we end up taking all of the C and B vertices.

4. Although $S$ cannot contain any of the V vertices, they affect the powerful alliance because they are in the neighborhood of the C vertices that are in $S$. Therefore, there must be an offensive alliance which can dominate the V vertices. These vertices have closed neighborhoods of size $2*\deg(i)+2$. Because $S$ contains all of the C vertices, there are $\deg(i)$ vertices in $S$ adjacent to each of the $v_i$ vertices. Therefore we must take at least one vertex from the A vertices that are connected to each of V vertices. If we take a dominating set of the graph $G$ and associate that set with the A vertices, then at least one of the vertices connected to each of the V vertices will be in $S$. This means each $v_i$ will have at least $\deg(i)+1$ vertices in $S$ and at most $\deg(i)+1$ vertices not in $S$ from its closed neighborhood.

Note that the constructed graph $H$ is bipartite, which means that POWERFUL ALLIANCE is NP-complete even when restricted to bipartite graphs. □

3.3 Global Defensive Alliance Complexity

GLOBAL DEFENSIVE ALLIANCE

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

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

Theorem 3.4 GLOBAL DEFENSIVE ALLIANCE is NP-complete, even when restricted to bipartite graphs.

Proof. GLOBAL DEFENSIVE ALLIANCE is clearly in NP. A set $S$ of size at most $k$, could be given as a witness to a ‘yes’ instance and verified in $O(E)$ time to be a global defensive alliance. In order to show that GLOBAL DEFENSIVE ALLIANCE is NP-complete we construct a transformation from the following well-known NP-complete problem from [11].

3SAT

INSTANCE: Collection $C = \{c_1, c_2, \ldots, c_m\}$ of clauses on a finite set $V = \{v_1, v_2, \ldots, v_n\}$ of variables such that $|c_i| = 3$ for $1 \leq i \leq m$.

QUESTION: Is there a truth assignment for $V$ such that each clause in $C$ has at least one true literal?
Transformation: Let $V$ and $C$ be an arbitrary instance of the 3-SAT problem. Create an instance $G(V, C)$ of GLOBAL DEFENSIVE ALLIANCE in this way:

1. For each $v_i \in V$, create a four cycle labeled in order $v_i, a_i, \bar{v}_i, b_i$, and then attach a leaf $g_i$ to each $a_i$-vertex.

2. For each $c_i \in C$ create a vertex labeled $c_i$ where the vertex is connected to those $v_x, v_y$, and $v_z$ in the clause $c_i$ and a tree, $T_i$, below the vertex $c_i$ thusly:

   (a) a leaf labeled $d_i$ and two vertices labeled $e_i$ and $f_i$.

   (b) adjacent to each of $e_i$ and $f_i$ are two leaves: $e_{i1}, e_{i2}, f_{i1}, f_{i2}$.

Let $k = 2n + 3m$. See Figure 3.5. Note that the constructed graph is bipartite.

Example: $m = 3; n = 4; c_1 = \{v_1, \bar{v}_2, v_3\}; c_2 = \{\bar{v}_1, v_2, \bar{v}_3\}; c_3 = \{\bar{v}_2, \bar{v}_3, v_4\}$

![Figure 3.4: Global Defensive Alliance Example - Bipartite Graphs](image)

Claim: Given a satisfying truth assignment $Y = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$ where $v_{ij} = v_j$ or $\bar{v}_j$, the set $S = Y \cup \{a_1 \ldots a_n\} \cup \{c_1 \ldots c_m\} \cup \{e_1 \ldots e_m\} \cup \{f_1 \ldots f_m\}$ is a global defensive alliance of cardinality $\leq 2n + 3m$. 

30
The set $S$ is a dominating set because every vertex in the graph is either in $S$ or adjacent to a vertex in $S$. $S$ is a defensive alliance because every $a_i$ has one other neighbor in $S$ and two neighbors not in $S$, every vertex in the truth assignment has one neighbor not in $S$, every clause has at least three neighbors in $S$ other than itself and at most 3 neighbors not in $S$, and each of $e_i$ and $f_i$ have two neighbors not in $S$ and one neighbor in $S$ other than itself.

Conversely, suppose there is a global defensive alliance $S$ of size at most $2n+3m$. Because this is a global alliance, it must contain two vertices in order to dominate $g_i$ and $b_i$, and these two vertices must be adjacent on the four cycle, one of which must be $a_i$, the other must be a v-vertex. Suppose not. In order for each tree $T_i$ to be dominated, at least 3 vertices must be in the global defensive alliance. Either $c_i$ or $d_i$ must be in or this will not be a global defensive alliance. Both $e_i$ and $f_i$ must be in the set in order to dominate the leaves $e_{i1}$, $e_{i2}$, $f_{i1}$, and $f_{i2}$. However, if $c_i$ is not in the set, then $S$ is not a defensive alliance because $e_i$ and $f_i$ will both be outnumbered. Thus all $c_i$ must be in the set. Therefore, the set of variable vertices must be a satisfying assignment because each $c_i$ needs an additional neighbor in the set in order for $S$ to be a global defensive alliance. □

**Theorem 3.5** GLOBAL DEFENSIVE ALLIANCE is NP-complete, even when restricted to chordal graphs.

**Proof.** GLOBAL DEFENSIVE ALLIANCE is clearly in NP. A set $S$ of size at most $k$, could be given as a witness to a ‘yes’ instance and verified in $O(E)$ time to be a global defensive alliance. In order to show that GLOBAL DEFENSIVE ALLIANCE is NP-complete when restricted to chordal graphs, we construct a transformation from 3SAT.

Transformation: Let $V$ and $C$ be an arbitrary instance of the 3-SAT problem. Create an instance $G(V,C)$ of GLOBAL DEFENSIVE ALLIANCE in this way:

1. For each $v_i \in V$, create a four cycle labeled in order $v_i, a_i, \bar{v}_i, b_i$.
2. Create a clique of the $v_i$ and $\bar{v}_i$ vertices.
3. For each $c_i \in C$ create a vertex labeled $c_i$ where the vertex is connected to those $v_x$, $v_y$, and $v_z$ in the clause $c_i$ and a tree, $T_i$, below the vertex $c_i$ thusly:
   (a) a leaf labeled $d_i$ and two vertices labeled $e_i$ and $f_i$.
   (b) adjacent to each of $e_i$ and $f_i$ are two leaves: $e_{i1}, e_{i2}, f_{i1}, f_{i2}$.
Let $k = 2n+3m$. See Figure 3.5. Note that the constructed graph is chordal.

Example: $m = 3; n = 4; c_1 = \{v_1, \overline{v}_2, v_3\}; c_2 = \{\overline{v}_1, v_2, \overline{v}_3\}; c_3 = \{\overline{v}_2, \overline{v}_3, v_4\}$

![Figure 3.5: Global Defensive Alliance Example - Chordal Graphs](image)

Claim: Given a satisfying truth assignment $Y = \{v_{i_1}, v_{i_2}, \ldots, v_{i_n}\}$ where $v_{i_j} = v_j$ or $\overline{v}_j$, the set $S = Y \cup \{a_1 \ldots a_n\} \cup \{c_1 \ldots c_m\} \cup \{e_1 \ldots e_m\} \cup \{f_1 \ldots f_m\}$ is a global defensive alliance of cardinality $\leq 2n + 3m$.

The argument is very similar to that given in the previous theorem. The set $S$ is a dominating set because every vertex in the graph is either in $S$ or next to a vertex in $S$. $S$ is a defensive alliance because every $a_i$ has one other neighbor in $S$ and two neighbors not in $S$, every vertex in the truth assignment has at most $n+2$ neighbors not in $S$ and at least $n+1$ neighbor in $S$, every clause has at least three neighbors in $S$ other than itself and at most 3 neighbors not in $S$ and each of $e_i$ and $f_i$ have two neighbors not in $S$ and one neighbor in $S$ other than itself.

Conversely, suppose there is a global defensive alliance $S$ of size at most $2n+3m$. Because this is a global alliance, it must contain two consecutive vertices from the four cycle, one to dominate $g_i$, the other to dominate $b_i$ (as in the previous proof). In order for each tree $T_i$ to be dominated at least 3 vertices must be in the global defensive alliance. Either $e_i$ or $d_i$ must be in or this will not be a global defensive alliance. Both $e_i$ and $f_i$ must be in the set in order to dominate the leaves $e_{i1}, e_{i2}, f_{i1},$ and $f_{i2}$. However, if $c_i$ is not in the set, then $S$ is not a defensive alliance because $e_i$ and $f_i$ will
both be outnumbered. Thus all $c_i$ must be in the set. Therefore, the set of variable vertices must be a satisfying assignment because each $c_i$ needs an additional neighbor in the set in order for $S$ to be a global defensive alliance.

\[ \Box \]

### 3.4 Global Powerful Alliance Complexity

**GLOBAL POWERFUL ALLIANCE**

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

**QUESTION:** Does $G$ have a powerful alliance of size at most $k$?

In [6], it is proven that GLOBAL POWERFUL ALLIANCE is NP-complete by reduction from DOMINATING SET. However, we can show that GLOBAL POWERFUL ALLIANCE is NP-complete even when restricted to bipartite or chordal graphs by a transformation from 3-SAT.

**Theorem 3.6** GLOBAL POWERFUL ALLIANCE is NP-complete, even when restricted to bipartite graphs.

**Proof.** GLOBAL POWERFUL ALLIANCE is clearly in NP. A set $S$ of size at most $k$, could be given as a witness to a ‘yes’ instance and verified in $O(E)$ time to be a global powerful alliance. In order to show that GLOBAL POWERFUL ALLIANCE is NP-complete we construct a transformation from 3SAT.

Transformation: Let $V$ and $C$ be an arbitrary instance of the 3-SAT problem. Create an instance $G(V, C)$ of GLOBAL POWERFUL ALLIANCE in this way:

1. For each $v_i \in V$, create a four cycle labeled in order $v_i, b_i, \bar{v}_i, d_i$.
2. For each $b_i$ that was just added, add another vertex $a_i$ which is only connected to that $b_i$ and similarly for each $d_i$, add another vertex $e_i$ which is only connected to that $d_i$.
3. For each $c_i \in C$ create a path of three vertices $c_{i1}, c_{i2}, c_{i3}$ where $c_{i1}$ is connected to those $v_x, v_y$, and $v_z$ in the clause $c_i$.

Let $k = 3n+2m$. See Figure 3.6. Note that the constructed graph is bipartite.

---

33
Claim: The given 3SAT instance has a solution if and only if the constructed bipartite graph has a global powerful alliance of size at most $3n + 2m$. Given a satisfying truth assignment $Y = \{v_1, v_2, \ldots, v_n\}$ where $v_i = v_j$ or $\bar{v}_j$, the set $S = Y \cup \{b_1 \ldots b_n\} \cup \{d_1 \ldots d_n\} \cup \{c_{11} \ldots c_{m1}\} \cup \{c_{12} \ldots c_{m2}\}$ is a global powerful alliance of cardinality $\leq 3n + 2m$.

This set $S$ is a global offensive alliance since the set $V - S$ is independent and $S$ is a dominating set. Also $S$ is a global defensive alliance since every vertex in $S$ has at most two neighbors in $V - S$ and at least one neighbor in $S$.

Conversely, suppose there is a global powerful alliance $S$ of size at most $3n + 2m$. Because this is a global offensive and defensive alliance, it must contain either the $b_i$ vertex or its associated $a_i$ vertex and similarly either the $d_i$ vertex or its associated $e_i$ vertex. If neither $a_i$ nor $b_i$ are taken then the alliance is not a global alliance because $a_i$ is neither in $S$ nor $V - S$. In fact, $S$ must contain at least 3 of the 6 vertices in the 6 vertex subgraph containing a four-cycle and the associated $a_i$ and $e_i$ vertices. Because for each $i$, either $b_i$ or $a_i$ and either $d_i$ or $e_i$ must be taken, then either $v_i$ or $\bar{v}_i$ must be in the set. In Figure 3.7, we see all four possible cases of taking only two of these vertices. Figure A has $b_i$ not defended, Figure B has $b_i$ and $d_i$ not attacked, Figure C has $b_i$ and $d_i$ not defended, and Figure D has $d_i$ not defended.
This means that there are 3 possible valid configurations for each subgraph as shown in Figure 3.8.

![Figure 3.7: Global Powerful Alliance Subgraph non-valid configurations](image)

Thus, for each variable subgraph, the variable and its complement will fall into one of three cases:

1. Figure A: $v_i, a_i, e_i \in S$. In this case, at least one more than half of the clauses containing $v_i$ must be in the set so that $v_i$ can be defended. Also, at least two more than half of the clauses containing $\overline{v}_i$ must be in the set so that $\overline{v}_i$ can be attacked.

2. Figure B: $v_j, a_j, d_j \in S$. In this case, at least half of the clauses containing $v_j$ must be in the set so that $v_j$ can be defended. Also, at least one more than half of the clauses containing $\overline{v}_j$ must be in the set so that $\overline{v}_j$ can be attacked.

3. Figure C: $v_k, b_k, d_k \in S$. In this case, at least one less than half of the clauses containing $v_k$ must be in the set so that $v_k$ can be defended. Also, at least half of the clauses containing $\overline{v}_k$ must be in the set so that $\overline{v}_k$ can be attacked.

![Figure 3.8: Global Powerful Alliance Subgraph valid configurations](image)

If $\overline{v}_i$ is in the set instead of $v_i$, then for each for each of the cases, switch $v_i$ with $\overline{v}_i$ and vice versa.

Now let’s examine the clauses. As with the $b_i, a_i, d_i, e_i$ vertices, for each 3 vertex clause path $c_{i1}, c_{i2}, c_{i3}$, either $c_{i2}$ or $c_{i3}$ must be in the global powerful alliance set. However, because of the $c_{i1}$
vertex, two of the three vertices must be in the set. If only $c_{i3}$ is in $S$, then this is not an offensive alliance because $c_{i2}$ is not in the set, has one neighbor not in the set ($c_{i1}$) and only one neighbor in the set ($c_{i3}$). If only $c_{i2}$ is in the set, then this is not a defensive alliance because $c_{i2}$ has two neighbors not in the set. If only one of the $v_x$, $v_y$, or $v_z$ variables in $c_i$ is in the global powerful alliance, then $c_{i1}$ must be in the powerful alliance. If more than one of the $v_x$, $v_y$, or $v_z$ variables in $c_i$ is in the global powerful alliance, then $c_{i1}$ may not be in the global powerful alliance, but both $c_{i2}$ and $c_{i3}$ must be in the global powerful alliance. Overall, this gives us $3n+2m$ vertices in the global powerful alliance set.

Now we need to show that these vertices must be distributed to create a 3-SAT solution. We find a truth assignment by assigning $v_i$ to be TRUE if it is in the set and $v_i$ to FALSE if it is not in the set. Assume that this is not a valid truth assignment. This would mean that there is a clause $c_i$ for which all the variables are FALSE. However, this would not happen in a powerful alliance because each clause vertex has 4 neighbors, only one of which is not a variable vertex. In order for this to be a powerful alliance, at least one of the neighbor variables must be true, meaning that the assumption that this is not a valid truth assignment is false.

□

**Theorem 3.7** GLOBAL POWERFUL ALLIANCE is NP-complete, even when restricted to chordal graphs.

**Proof.** GLOBAL POWERFUL ALLIANCE is clearly in NP. A set $S$ of size at most $k$, could be given as a witness to a ‘yes’ instance and verified in $O(E)$ time to be a global powerful alliance. In order to show that GLOBAL POWERFUL ALLIANCE is NP-complete when restricted to chordal graphs, we again construct a transformation from 3SAT. The transformation is the same as used in the proof of the previous theorem with the addition of making a clique of the $2n$ variable vertices (see Figure 3.9) in order to construct a chordal graph.

Example: $m = 4; n = 5; c_1 = \{v_1, v_2, v_3\}; c_2 = \{v_1, \bar{v}_2, v_3\}; c_3 = \{v_1, v_3, v_4\}; c_4 = \{v_1, \bar{v}_4, v_5\}$

Claim: As before, an instance of 3SAT has a solution if and only if the constructed chordal graph has a global powerful alliance of size at most $3n+2m$. Given a satisfying truth assignment $Y = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$ where $v_j = v_j$ or $\bar{v}_j$, the set $S = Y \cup \{b_1 \ldots b_n\} \cup \{d_1 \ldots d_n\} \cup \{c_{i1} \ldots c_{im}\} \cup \{c_{i12} \ldots c_{im2}\}$ is a global powerful alliance of cardinality $\leq 3n + 2m$.

The proof is virtually the same as the proof in the previous theorem and is omitted. □
Figure 3.9: Global Powerful Alliance Example - Chordal Graphs
Chapter 4

Linear Algorithms for Alliances in Series-Parallel Graphs

Because defensive and powerful alliances are NP-complete even when restricted to bipartite and chordal graphs, but both are solvable in linear time on trees, it is interesting to consider where the P-NP split is for alliances.

Consider series-parallel graphs. These are graphs with two terminals r and s which can be combined with other series-parallel graphs in three possible ways: parallel, series and generalized series (see Figure 4.1). A parallel combination occurs when the r terminal from the first graph is merged with the r terminal from the second and the s terminal from the first graph is merged with the s terminal from the second and these merged vertices become the terminals for the resultant graph. A series combination occurs when the s terminal from the first graph is merged with the r terminal from the second graph and the and the r terminal from the first graph and the s terminal from the second graph become the terminals in the resultant graph. A generalized series combination is similar to a series combination in that the s terminal from the first graph is merged with the r terminal from the second graph, however, the terminals for the resultant graph will be the r terminal from the first graph and the merged vertex.

Figure 4.1: Examples of Composition Types
Algorithms for series-parallel graphs were introduced in [15] which provided an algorithm for the domination number of a series-parallel graph. Other algorithms on series-parallel graphs include generalized matching in [24] and vertex cover in [25]. Here is a polynomial algorithm which finds the minimum defensive alliance number on a series-parallel graph.

When trying to create a set $S$ which is a minimal defensive alliance, m.d.a., in a series-parallel graph $G$, there are a few classes to consider which will represent all valid subgraph $G'$ subset $S'$ pair in $G$. Let $r$ and $s$ be the terminals of a subgraph $G'$, subset $S$ pair. Because this is part of a minimal defensive alliance, the only place where it is important to know the balance between nodes in the set and not in the set is when $r \in S$. Thus, the integer $i = |N[r] \cap S| - |N[r] \cap (V-S)|$ will only be stored for the root which is in $S$ in those classes.

There are 10 classes for the subgraph, subset pairs at any point in the algorithm:

1. $[1:i,j] = \{r, s \in S, S \cap G \text{ is a m. d. a. in } G\}$
2. $[2:i,j] = \{r, s \in S, S \cap G \text{ is not a m. d. a. in } G \text{ because } i < 0 \text{ but } j \geq 0\}$
3. $[3:i,j] = \{r, s \in S, S \cap G \text{ is not a m. d. a. in } G \text{ because } j < 0 \text{ but } i \geq 0\}$
4. $[4:i,j] = \{r, s \in S, S \cap G \text{ is not a m. d. a. in } G \text{ because } i,j < 0\}$
5. $[5:i] = \{r \in S, s \notin S, S \cap G \text{ is a m. d. a. in } G\}$
6. $[6:j] = \{r \notin S, s \in S, S \cap G \text{ is a m. d. a. in } G\}$
7. $[7:i] = \{r \in S, s \notin S, S \cap G \text{ is not a m. d. a. in } G\}$
8. $[8:j] = \{r \notin S, s \in S, S \cap G \text{ is not a m. d. a. in } G\}$
9. $[9] = \{r \notin S, s \notin S, S \cap G \text{ is a m. d. a. in } G\}$
10. $[10] = \{r \notin S, s \notin S, S \cap G \text{ is not a m. d. a. in } G\}$

**Algorithm 4.1: Defensive Alliances in Series Parallel Graphs**

Once these classes have been defined, we can consider the composition of a graph $G$ from these subgraphs. If we are connecting the two subgraphs in series, then we combine a subgraph $G_1$ with terminals $r_1$ and $s_1$ and subset $S_1$ with a subgraph $G_2$ with terminals $r_2$ and $s_2$ and subset $S_2$ to produce a new graph $G' = G_1 \circ G_2$ with terminals $r_1$ and $s_2$ and subset $S_1 \cup S_2$. If we are connecting the two subgraphs in parallel, then we combine a subgraph $G_1$ with terminals $r_1$ and $s_1$ and subset $S_1$ with subgraph $G_2$ with terminals $r_2$ and $s_2$ and subset $S_2$ to produce a new graph $G' = G_1 \ast G_2$ with terminals $r_1$ and $s_1$ and subset $S_1 \cup S_2$. If we are connecting the two subgraphs in generalized series, then we combine a subgraph $G_1$ with terminal $r_1$ and $s_1$ and subset $S_1$ with a subgraph $G_2$ with terminals $r_1$ and $s_1$ and subset $S_1 \cup S_2$. We must consider all possible compositions of classes $[a(i, j)]$ and $[b(k, l)]$ for
1 \leq a, b \leq 10. If a particular composition cannot occur when producing a minimal defensive alliance, then this composition is marked with an 'X'.

The initial values for each class combine to form the initial vector for the algorithm. We use either the number of vertices in the set, or $\infty$ to represent a state which cannot logically exist. When combining subgraphs, we want to minimize the number of vertices in the set. The only classes which logically can exist initially for an initial subgraph are class 1, with a value of 2, class 5, with a value of 1, class 6, with a value of 1, and class 10, with a value of 0. This means that the initial vector for all vertices is $[2, \infty, \infty, \infty, \infty, 1, 1, \infty, \infty, \infty, 0]$. After execution of the algorithm, we need to determine the answer. Only four classes can produce a minimum defensive alliance. These are $[1:(i,j>0)]$, $[5:(i>0)]$, $[6:(j>0)]$, and $[9]$ where neither terminal is in, so there is no value. The minimum value from the final vector in these 4 classes is the size of a minimum defensive alliance for the graph $G$.

<table>
<thead>
<tr>
<th>[1:k,l]</th>
<th>[2:k,l]</th>
<th>[3:k,l]</th>
<th>[4:k,l]</th>
<th>[5:k]</th>
<th>[6:l]</th>
<th>[7:k]</th>
<th>[8:l]</th>
<th>[9]</th>
<th>[10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1:i,j]</td>
<td>[1:i,l]</td>
<td>if (j+k \geq 0) then [1:i,l] else X</td>
<td>[3:i,l]</td>
<td>if (j+k \geq 0) then [3:i,l] else X</td>
<td>[5:i]</td>
<td>if(j+k \geq 0) then [5:i] else X</td>
<td>[7:i]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>[2:i,l]</td>
<td>[2:i,l]</td>
<td>if (j+k \geq 0) then [2:i,l] else X</td>
<td>[4:i,l]</td>
<td>if (j+k \geq 0) then [4:i,l] else X</td>
<td>[7:i]</td>
<td>if(j+k \geq 0) then [7:i] else X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>[3:i,l]</td>
<td>if (j+k \geq 0) then [1:i,l] else X</td>
<td>X</td>
<td>if(j+k \geq 0) then [3:i,l] else X</td>
<td>X</td>
<td>if(j+k \geq 0) then [5:i] else X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>[4:i,l]</td>
<td>if (j+k \geq 0) then [2:i,l] else X</td>
<td>X</td>
<td>if(j+k \geq 0) then [4:i,l] else X</td>
<td>X</td>
<td>if(j+k \geq 0) then [7:i] else X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>[6:i]</td>
<td>[6:i]</td>
<td>if(j+k \geq 0) then [6:i] else X</td>
<td>[8:i]</td>
<td>if(j+k \geq 0) then [8:i,l] else X</td>
<td>[9]</td>
<td>if(j+k \geq 0) then [9] else X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>[8:i]</td>
<td>if(j+k \geq 0) then [6:i] else X</td>
<td>X</td>
<td>if(j+k \geq 0) then [8:i] else X</td>
<td>X</td>
<td>if (j+k \geq 0) then [9] else X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 4.1: Compositions for serial connections for defensive alliances.

We will discuss one of the more interesting compositions: $[3:i,j] \circ [2:k,l]$ for all possible compositions.

First, consider a series composition of a subgraph $G_1$ with terminals $r_1$ and $s_1$ of class $[3:i,j]$, where both terminals are in the set, $i \geq 0$, and $j < 0$, with a subgraph $G_2$ with terminals $r_2$ and $s_2$ of class $[2:k,l]$, where both terminals are in the set, $k < 0$, and $l \geq 0$. In a series composition, we will merge $s_1$ and $r_2$ and have resultant terminals $r_1$ and $s_2$. Because $j$ and $k$ are both less than 0 and
Table 4.2: Compositions for generalized serial connections for defensive alliances.

<table>
<thead>
<tr>
<th></th>
<th>[1:k,l]</th>
<th>[2:k,l]</th>
<th>[3:k,l]</th>
<th>[4:k,l]</th>
<th>[5:k]</th>
<th>[6:l]</th>
<th>[7:k]</th>
<th>[8:l]</th>
<th>[9]</th>
<th>[10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1:i,j]</td>
<td>[1:i,j+k]</td>
<td>if (j+k ≥ 0) then [1:i,j+k] else [3:i,j+k]</td>
<td>X X</td>
<td>[1:i,j+k]</td>
<td>X</td>
<td>if (j+k ≥ 0) then [1:i,j+k] else [3:i,j+k]</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3:i,j]</td>
<td>if (j+k ≥ 0) then [1:i,j+k] else [3:i,j+k]</td>
<td>if (j+k ≥ 0) then [1:i,j+k] else [3:i,j+k]</td>
<td>X X</td>
<td>if (j+k ≥ 0) then [1:i,j+k] else [3:i,j+k]</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7:i]</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>[7:i]</td>
<td>X</td>
<td>X</td>
<td>[7:i]</td>
<td>[7:i]</td>
<td></td>
</tr>
<tr>
<td>[8:j]</td>
<td>[8:j+k]</td>
<td>if (j+k ≥ 0) then [6:j+k] else [8:j+k]</td>
<td>X X</td>
<td>[8:j+k]</td>
<td>X</td>
<td>X</td>
<td>[8:j+k]</td>
<td>X X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

this is the last composition for both \( s_1 \) and \( r_2 \), this composition can never occur in a valid minimum defensive alliance on a series-parallel graph.

Next, consider a generalized series composition of a subgraph \( G_1 \) with terminals \( r_1 \) and \( s_1 \) of class \([3:i,j]\), where both terminals are in the set, \( i ≥ 0 \), and \( j < 0 \), with a subgraph \( G_2 \) with terminals \( r_2 \) and \( s_2 \) of class \([2:k,l]\), where both terminals are in the set, \( k < 0 \), and \( l ≥ 0 \). In a generalized series composition, we will merge \( s_1 \) and \( r_2 \) and have resultant terminals \( r_1 \) and \( s_1 \). Because \( j \) and \( k \) are both less than 0, their summation will still be less than 0. Thus, this composition will result in a series parallel graph of class \([3:i,j+k]\) with terminals \( r_1 \) and \( s_1 \).

Finally, consider a parallel composition of a subgraph \( G_1 \) with terminals \( r_1 \) and \( s_1 \) of class \([3:i,j]\), where both terminals are in the set, \( i ≥ 0 \), and \( j < 0 \), with a subgraph \( G_2 \) with terminals \( r_2 \) and \( s_2 \) of class \([2:k,l]\), where both terminals are in the set, \( k < 0 \), and \( l ≥ 0 \). In a parallel composition, we will merge \( r_1 \) and \( r_2 \) and merge \( s_1 \) and \( s_2 \) and have resultant terminals \( r_1 \) and \( r_2 \) and \( s_1 \) and \( s_2 \). Because \( j \) and \( k \) are both less than 0, this composition is very complex. If \( i+k ≥ 0 \) and \( j+l ≥ 0 \) then the resultant series parallel graph will be of class \([1:i+k,j+l]\). If \( i+k < 0 \) and \( j+l ≥ 0 \) then the resultant series parallel graph will be of class \([2:i+k,j+l]\). If \( i+k ≥ 0 \) and \( j+l < 0 \) then the resultant series parallel graph will be of class \([3:i+k,j+l]\). If \( i+k < 0 \) and \( j+l < 0 \) then the resultant series parallel graph will be of class \([4:i+k,j+l]\).
Table 4.3: Compositions for parallel connections for defensive alliances.
Chapter 5

Linear Algorithms for Weighted Alliances in Paths

No research has been published to-date on integer weighted alliances in trees. We will show in Chapter 6 that weighted alliance decision problems in trees are NP-complete. However, it is possible to find minimum weighted alliances in paths and other specific types of trees.

5.1 Defensive Alliances

When attempting to find the minimum weighted defensive alliance number of a path, it is easy to see that no weighted defensive alliance will consist of more than 2 adjacent vertices.

CLAIM: A minimal weighted defensive alliance in a vertex weighted path will not have more than 2 adjacent vertices.

Proof. Let S be a minimal weighted defensive alliance. By the properties of defensive alliances, we know that S is connected. Suppose that S contains at least three consecutive vertices starting with A, B, and C whose corresponding weights are a, b, and c. Because this is a minimal defensive alliance, we cannot remove A because \( a > b + c \). However, we also cannot remove all but A and B because this is a minimal defensive alliance, which means that \( c > a + b \). Because all weights are assumed to be positive, this is a contradiction of the assumption that S is a minimal defensive alliance. Therefore, a minimal defensive alliance will contain no more than 2 adjacent vertices. □

In this section we present an algorithm for finding a minimum weighted defensive alliance (MWDA) on a path. Due to the claim above, the algorithm examines all possible individual vertices and adjacent pairs of vertices in the path. The algorithm works across the path from left to right with a 4-vertex window labeled consecutively \( a, b, c, \) and \( d \). The weights of the vertices are stored in an array \( w \). The current lowest defensive alliance for the path is called \( \text{low}_\text{weight} \) and begins the algorithm set to \( \infty \) with no vertices in the set. Note that this algorithm works for paths of length 4 or greater. For paths of length less than 4, simple inspection will be able to determine the minimum weighted defensive alliance. The rules here are executed immediately on verification that the rule may be applied. This means that in many instances no rules will be applied, and that in most
cases, only one rule will be applied per 4-vertex window. The obvious exception to this statement is $P_4$ for which 2 rules may be applied.

1. If $a$ is a leaf AND $(w(a) \geq w(b) \text{ AND } w(a) < low\_weight)$ then $S = \{a\}$ and $low\_weight = w(a)$.
2. If $d$ is a leaf AND $(w(d) \geq w(c) \text{ AND } w(d) < low\_weight)$ then $S = \{d\}$ and $low\_weight = w(d)$.
3. If $w(b) \geq w(a) + w(c) \text{ AND } w(b) < low\_weight$ then $S = \{b\}$ and $low\_weight = w(b)$.
4. If $w(c) \geq w(b) + w(d) \text{ AND } w(c) < low\_weight$ then $S = \{c\}$ and $low\_weight = w(c)$.
5. If $a$ is a leaf AND $(w(a) + w(b) \geq w(c) \text{ AND } w(a) + w(b) < low\_weight)$ then $S = \{a, b\}$ and $low\_weight = w(a) + w(b)$.
6. If $d$ is a leaf AND $(w(c) + w(d) \geq w(b) \text{ AND } w(c) + w(d) < low\_weight)$ then $S = \{c, d\}$ and $low\_weight = w(c) + w(d)$.
7. If $w(b) + w(c) \geq w(a) \text{ AND } w(b) + w(c) \geq w(d) \text{ AND } w(b) + w(c) < low\_weight$ then $S = \{b, c\}$ and $low\_weight = w(b) + w(c)$.

Algorithm 5.1: Weighted Defensive Alliances in Paths

As you can see, this algorithm deals with all possible variations of the 4-vertex window to find a smallest weighted defensive alliance. In many cases the weighted defensive alliance will consist of just one or the other of the end vertices.

5.2 Offensive Alliances

While a simple algorithm works for finding weighted defensive alliances on a path, weighted offensive alliances require more knowledge than we can find in a 4-vertex window. Because we are working on a path, the compositions for this algorithm will consist of combining a rooted path with a single vertex which will be the root of the new path.

In order to compute the weighted offensive alliance number of a path, we must determine the possible types of rooted sub-paths that could be created by a minimum weight offensive alliance, denoted WOA. Let $S$ be the set of vertices in a minimum weight offensive alliance. Let $P$ be the path
for which we are computing $a_{wo}(P)$. Let $r$ be the root of the current subpath. There are five classes which occur when finding a minimum weight offensive alliance in a path $P$. Because this alliance is only offensive, we only keep track of the balance when the root of the path is not in the alliance $S$. The classes are:

$[1] = \{ r \in S, S \cap P \text{ is a WOA in } P \}$

$[2:i] = \{ r \notin S, S \cap P \text{ is a WOA in } P \text{ and } N[r] \cap S = \emptyset \} /* \text{ all members of the offensive alliance are below the child of the root } */$

$[3:i] = \{ r \notin S, S \cap P \text{ is a WOA in } P \text{ and } N[r] \cap S \neq \emptyset \} /* \text{ the child of the root is a member of the offensive alliance and } i \geq 0 */$

$[4:i] = \{ r \notin S, S \cap P \text{ is not a WOA in } P \text{ and } S \neq \emptyset, \text{ but adding a neighbor in } S \text{ to } r \text{ can create a WOA} /* \text{ there is a member of the set directly below the root, but it does not successfully attack the root } */$

$[5:i] = \{ r \notin S, S \cap P \text{ is not a WOA in } P \text{ and } S = \emptyset, \text{ but adding a neighbor in } S \text{ to } r \text{ can create a WOA} /* \text{ there are no members of the set } */$

These classes produce Table 5.1, where we are combining a path $P_1$ consisting of a single vertex with path $P_2$ rooted at $s$. Due to the algorithm being run only on a path, rows 2, 3 and 4 consist of all 'X' because there is no way to have a sub-path of type 2, 3 or 4 on the left side. The $i$ value for this algorithm represents the sum of the weights of any neighbors in the set minus the sum of the weights of any neighbors not in the set, including itself. The weights of the vertices are stored in an array $w$ such that $w(s)$ refers to the weight of vertex $s$.

<table>
<thead>
<tr>
<th></th>
<th>[1]</th>
<th>[2:j]</th>
<th>[3:j]</th>
<th>[4:j]</th>
<th>[5:j]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>[1]</td>
<td>X</td>
<td>if (w(r) &gt; j) then [1] else X</td>
<td>if (w(r) &gt; j) then [1] else X</td>
<td>if (w(r) &gt; j) then [1] else X</td>
</tr>
<tr>
<td>[5:i]</td>
<td>if (w(s) &gt; w(r)) then [4:i+w(s)] else [3:i+w(s)]</td>
<td>[2:i+w(s)]</td>
<td>if (w(s) &lt; j) then [3:i+w(s)] else X</td>
<td>X</td>
<td>[5:i+w(s)]</td>
</tr>
</tbody>
</table>

Table 5.1: Compositions for weighted offensive alliances.

From Table 5.1 we obtain a set of recurrence relations as follows:


$[2:i+w(s)] = [5:i] \circ [2:j]$

$[3:i-w(s)] = [5:i] \circ [1] \cup [5:i] \circ [3:j]$

$[4:i+w(s)] = [5:i] \circ [1]$

$[5:i-w(s)] = [5:i] \circ [5:j]$

47
As with non-weighted alliances, the initial values for each class in weighted offensive alliances combine to form the initial vector for the algorithm. We use either the sum of the weights of the vertices in the set or \( \infty \) to represent a state which cannot logically exist. When combining sub-paths, we want to minimize the weight of the offensive alliance. The only classes which can logically exist initially are class 1, with a value of \( w(s) \), and class 5 with a value of 0. This means that the initial vector is \([w(s), \infty, \infty, \infty, 0]\). After execution of the algorithm, we need to determine the answer. Three states can determine a weighted offensive alliance. These are \([1]\), \([2:i]\) where \( i \) is any value, and \([3:i]\) where \( i \geq 0 \). The minimum value from the final vector for these three classes indicates the minimum weight of an offensive alliance for that path.

### 5.3 Powerful Alliances

While a simple algorithm works for finding weighted defensive alliances on a path, weighted powerful alliances require more knowledge than we can find in a 4-vertex window.

In order to compute the weighted powerful alliance number of a path, we must determine the possible types of subpaths that could be created by a minimum weight powerful alliance, denoted WPA. Let \( S \) be the set of vertices in a minimum weight powerful alliance. Let \( P \) be the path for which we are computing \( a_{wp}(P) \). Let \( r \) be the root of the current subpath. Because this alliance is powerful, we need to keep track of the balance for all classes. The 6 classes are:

- \([1:i]\) = \{ \( r \in S, S \cap P \) is a WPA in \( P \) \} / * \( i \geq 0 \) */
- \([2:i]\) = \{ \( r \in S, S \cap P \) is not a WPA in \( P \) \} / * \( i < 0 \) */
- \([3:i]\) = \{ \( r \not\in S, S \cap P \) is a WPA in \( P \) and \( N[r] \cap S = \emptyset \) \}
- \([4:i]\) = \{ \( r \not\in S, S \cap P \) is a WPA in \( P \) and \( N[r] \cap S \neq \emptyset \) / * \( i \geq 0 \) */
- \([5:i]\) = \{ \( r \not\in S, S \cap P \) is not a WPA in \( P \) and \( S \neq \emptyset \), but adding neighbors in \( S \) to \( r \) can create an WPA \}
- \([6:i]\) = \{ \( r \not\in S, S \cap P \) is not a WPA in \( P \) and \( S = \emptyset \), but adding neighbors in \( S \) to \( r \) can create a WPA \}

These classes produce Table 5.2, where we are combining path \( P_1 \) consisting of a single vertex \( r \) with path \( P_2 \) rooted at \( s \). Due to the algorithm being run only on a path, rows 2, 3, 4, and 5 consist of all ‘X’ because there is no way to have a sub-path of type 2, 3, 4, or 5 on the left side. The \( i \) value for this algorithm represents the sum of the weights of any neighbors in the set minus the sum of the weights of any neighbors not in the set, including itself. The weights of the vertices are stored in an array \( w \) such that \( w(s) \) refers to the weight of vertex \( s \).
Table 5.2: Compositions for weighted powerful alliances.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1:i</td>
<td>1:i+w(s)</td>
<td>if ((i+j) ≥ 0) then [1:i+w(s)] else X</td>
<td>X</td>
<td>if (i ≥ w(s)) then [1:i-w(s)] else [2:i-w(s)]</td>
<td>if ((i+j) ≥ 0) then { if (i ≥ w(s)) then [1:i-w(s)] else [2:i-w(s)] } else X</td>
</tr>
<tr>
<td>1:i-w(s)</td>
<td>if (i ≥ 0) then [1:i-w(s)] else X</td>
<td>X</td>
<td>if ((i+j) ≥ 0) then { if (i ≥ w(s)) then [1:i-w(s)] else [2:i-w(s)] } else X</td>
<td>X</td>
<td>[6:i-w(s)]</td>
</tr>
</tbody>
</table>

From Table 5.2 we obtain a set of recurrence relations as follows:

\[
[1:i+w(s)] = [1:i][1:j] \cup [1:i][2:j]
\]

\[
[1:i-w(s)] = [1:i][4:j] \cup [1:i][5:j] \cup [1:i][6:j]
\]

\[
[2:i-w(s)] = [1:i][4:j] \cup [1:i][6:j]
\]

\[
[3:i-w(s)] = [6:i][3:j] \cup [6:i][4:j]
\]

\[
[4:i+w(s)] = [6:i][1:j]
\]

\[
[5:i+w(s)] = [6:i][1:j]
\]

\[
[6:i-w(s)] = [6:i][6:j]
\]

As with non-weighted alliances, the initial values for each class in weighted powerful alliances combine to form the initial vector for the algorithm. We use either the sum of the weight of the vertices in the set or \(\infty\) to represent a state which cannot logically exist. When combining sub-paths, we want to minimize the total weight of the powerful alliance. The only classes which can logically exist initially are class 1, with a value of \(w(s)\), and class 6 with a value of 0. This means that the initial vector is \([w(s), \infty, \infty, \infty, \infty, 0]\). After execution of the algorithm, we need to determine the answer. Three states can determine a weighted powerful alliance. These are \([1], [3:i]\) where \(i\) is any value, and \([4:i]\) where \(i ≥ 0\). The minimum value from the final vector for these three classes indicates the minimum weight of an powerful alliance for that path.

As before, we will only discuss a few of the 12 entries from Table 5.2. We will discuss two entries, \([1:i][4:j]\) and \([6:i][2:j]\).

To begin, look at the composition of a path of class \([1:i]\), whose root \(r\) is in the set \(S\) and \(S\) is a weighted powerful alliance, and a path of class \([4:j]\), whose root \(s\) is not in the set \(S\) but \(S\) is a weighted alliance directly below \(s\). If connecting \(r\) and \(s\) will mean that \(s\) will still be an offensive alliance then we look at the effect of \(s\) on \(r\), other wise the connection cannot be made. If we are looking at the effect of \(s\) on \(r\) and the balance of \(r\) is greater than the weight of \(s\) then the resulting path is class \([1:i-w(s)]\) otherwise the resulting path is class \([2:i-w(s)]\) meaning that \(r\) will need a neighbor in \(S\) to be a weighted powerful alliance.
Next, look at the composition of a path of class [6:i], whose root r is not in the set S and S is an empty set, with a path of class [2:j], whose root s is in the set S, but S is not a weighted powerful alliance because it is not a weighted defensive alliance at s. Since r is not in S and this is the last composition to be made with s, this composition is not valid. Thus this location in Table 5.2 is filled with an 'X'.

5.4 Global Defensive Alliances

In order to compute the weighted global defensive alliance number of a path, we must determine the possible types of subpaths that could be created by a minimum weight global defensive alliance, denoted WGDA. Let S be the set of vertices in a minimum weight global defensive alliance. Let P be the path for which we are computing $\gamma_{aw}(P)$. Let r be the root of the current subpath. The weight of each node is stored in an array such that the weight of a node s is denoted $w(s)$. There are only 4 classes:

$[1:i] = \{ r \in S, S \cap P \text{ is a WGDA in } P \}/* i \geq 0 */$

$[2:i] = \{ r \in S, S \cap P \text{ is not a WGDA in } P, \text{ but adding neighbors in } S \text{ to } r \text{ can create a WGDA} */ i < 0 */$

$[3:i] = \{ r \not\in S, S \cap P \text{ is a WGDA in } P \}/* i \geq 0 */$

$[4:i] = \{ r \not\in S, S \cap T \text{ is not a WGDA in } P, \text{ but adding a neighbor in } S \text{ to } r \text{ can create a WGDA} */ i < 0 */$

These classes produce Table 5.3 where we are combining a path $P_1$ consisting of a single vertex r with a path $P_2$ rooted at s. Due to the algorithm being run only on a path, rows 2 and 3 consist of all 'X' because there is no way to have a sub-path of type 2, or 3 on the left side. The i value for this algorithm represents the sum of the weights of any neighbors in the set minus the sum of the weights of any neighbors not in the set, including itself. The weights of the vertices are stored in an array $w$ such that $w(s)$ refers to the weight of vertex s.

From Table 5.3, we obtain the following recurrence relations:

$[1:i+j] = [1:i] \circ [1:j] \cup [1:i] \circ [2:j]$


$[3] = [4] \circ [1:j]$

Table 5.3: Compositions for weighted global defensive alliances.

As with non-weighted alliances, the initial values for each class in weighted global defensive alliances combine to form the initial vector for the algorithm. We use either the sum of the weight of the vertices in the set, or $\infty$ to represent a state which cannot logically exist. When combining sub-paths, we want to minimize the total weight of the global defensive alliance. The only classes which can logically exist initially are class 1, with a value of $w(s)$, and class 4 with a value of 0. This means that the initial vector is $[w(s), \infty, \infty, 0]$. After execution of the algorithm, we need to determine the answer. Two states can determine a weighted global defensive alliance. These are [1] and [3:i] where $i \geq 0$. The minimum value from the final vector for these two classes indicates the minimum weight of an global defensive alliance for that path.

As before, we will only discuss a few of the 8 entries from Table 5.3. We will discuss two entries, [1:i]○[4] and [4]○[2:j].

To begin, look at the composition of a path of class [1:i] whose root $r$ is in the set $S$ and $S$ is a weighted global defensive alliance, and a path of class [4] whose root $s$ is not in the set $S$ and $S$ is not a weighted global defensive alliance. If the weight of $s$ is less than or equal to the current balance at $r$, then $S$ will still be a weighted global defensive alliance making this a class [1:i-w(s)]. However, if the weight of $s$ is more than the current balance at $r$, the $S$ will not be a weighted global defensive alliance, making this a class [2:i-w(s)].

Next look at the composition of a path of class [4], whose root $r$ is not in the set $S$ and $S$ is not a weighted global defensive alliance, and a path of class [2:j], whose root $s$ is in $S$, but $S$ is not a weighted global defensive alliance because it is not a defensive alliance. This composition is the last composition for $s$, so there will be no hope of creating a defensive alliance in $S$. Therefore this composition cannot occur in a weighted global defensive alliance on a path. Thus this entry in Table 5.3 is filled with an ’X’.

5.5 Global Offensive Alliances

In order to compute the weighted global offensive alliance number of a path, we must determine the possible types of sub-paths that could be created by a minimum weight global offensive alliance, denoted WGOA. Let $S$ be the set of vertices in a minimum weight global offensive alliance. Let $P$
be the path for which we are computing $\gamma_{w,\omega}(P)$. Let $r$ be the root of the current subpath. The weight of each node is stored in an array such that the weight of a node $s$ is denoted $w(s)$. Because we’re looking for a weighted global offensive alliance, only types 3 and 4 need $i$. There are only 4 classes:

1. $\{r \in S, S \cap P \text{ is a WGOA in } P\}$
2. $\{r \notin S, S \cap P \text{ is a WGOA in } P\}$
3. $\{r \notin S, S \cap P \text{ is not a WGOA in } P \text{ because } S \text{ is not an offensive alliance} \} \quad i \geq 0$
4. $\{r \notin S, S \cap T \text{ is not a WGOA in } P, \text{ because } r \text{ is not dominated} \} \quad i < 0$

These classes produce Table 5.4 where we are combining path $P_1$ rooted at $r$ with path $P_2$ rooted at $s$. Due to the algorithm being run only on a path, rows 2 and 3 consist of all ‘X’ because there is no way to have a sub-path of type 2 or 3 on the left side. The $i$ value for this algorithm represents the sum of the weights of any neighbors in the set minus the sum of the weights of any neighbors not in the set, including itself. The weights of the vertices are stored in an array $w$ such that $w(s)$ refers to the weight of the vertex $s$.

<table>
<thead>
<tr>
<th></th>
<th>[1]</th>
<th>[2:i]</th>
<th>[3:i]</th>
<th>[4:i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>if ($i+j \geq 0$) then [1] else X</td>
<td>if ($i+j \geq 0$) then [1] else X</td>
</tr>
<tr>
<td>[4:1] if ($w(s)+i \geq 0$) then [2:i+w(s)] else [3:i+w(s)]</td>
<td>if ($j+i \geq 0$) then [4:i+w(s)] else X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.4:** Compositions for weighted global offensive alliances.

From Table 5.4, we obtain the following recurrence relations:

2. $[2:i+w(s)] = [4:i] \circ [1]$
3. $[3:i+w(s)] = [4:i] \circ [1]$
4. $[4:i+w(s)] = [4:i] \circ [2:j]$

As with non-weighted alliances, the initial values for each class in weighted global offensive alliances combine to form the initial vector for the algorithm. We use either the sum of the weight of the vertices in the set or $\infty$ to represent a state which cannot logically exist. When combining sub-paths, we want to minimize the total weight of the global offensive alliance. The only classes which can logically exist initially are class 1, with a value of $w(s)$, and class 4 with a value of 0. This means that the initial vector is $[w(s), \infty, \infty, 0]$. After execution of the algorithm, we need to determine the answer. Two states can determine a weighted global offensive alliance. These are [1] and [2:x] where $x \geq 0$. The minimum value from the final vector for these two classes indicates the minimum weight of an global offensive alliance for that path.
As before, we will only discuss a few of the 8 entries from Table 5.4. We will discuss two entries, [1]◦[3:j] and [4:i]◦[2:j].

To begin, look at the composition of a path of class [1], whose root r is in the set S, with a path of class [3:j], whose root s is not in the set S, and S is not a weighted global offensive alliance because S is not an weighted offensive alliance. As long as the weight of r creates a weighted global offensive alliance, the resultant path will continue to be a weighted global offensive alliance, which means that if the weight of r plus the balance at s is greater than or equal to 0, r will be of class [1].

Next, look at the composition of a path of class [4], whose root r is not in the set S and S is not a weighted global offensive alliance with a path of class [2], whose root s is not in the set S and S is a weighted global offensive alliance. If the weight of r is less than the excess at s, then we still have a weighted offensive alliance, but it is not global, so this location in Table 5.4 will be a class [4:i-w(s)]. Otherwise, the composition can never occur in a minimum global weighted offensive alliance, which means that this location in Table 5.4 contains an ‘X’.

### 5.6 Global Powerful Alliances

In order to compute the weighted global powerful alliance number of a path, we must determine the possible types of sub-paths that could be created by a minimum cardinality weighted global powerful alliance, denoted WGPA. Let S be the set of vertices in a minimum weighted global powerful alliance. Let P be the path for which we are computing $\gamma_{gp}(T)$. Let r be the root of the current subpath. The weight of each node is stored in an array such that the weight of a node $s$ is denoted $w(s)$. Again, $r$ is the root of the path $P$. There are only 4 classes:

1. $[1] = \{r \in S, S \cap P \text{ is a WGPA in } P\}$
2. $[2] = \{r \in S, S \cap P \text{ is not a WGPA in } P, \text{ but adding neighbors in } S \text{ to } r \text{ can create a WGPA}\}$
3. $[3:i] = \{r \notin S, S \cap P \text{ is a WGPA in } P \text{ and } N[r] \neq \emptyset \text{ /i } i \geq 0 \text{/} \}$
4. $[4:i] = \{r \notin S, S \cap P \text{ is not a WGPA because } r \text{ is not dominated}\}$
5. $[5:i] = \{r \notin S, S \cap P \text{ is not a WGPA because } S \cap P \text{ is not an offensive alliance}\}$
6. $[6:i] = \{r \notin S, S \cap T \text{ is not a WGPA in } P \text{ because } S = \emptyset \text{ /i } i < 0 \text{/} \}$

These classes produce Table 5.5, where we are combining path $P_1$ rooted at $r$ with path $P_2$ rooted at $s$. Due to the algorithm being run only on a path, rows 2, 3, 4 and 5 consist of all ‘X’ because there is no way to have a sub-path of type 2, 3, 4 or 5 on the left side. The i value for this algorithm represents the sum of the weights of any neighbors in the set minus the sum of the weights of any
neighbors not in the set, including itself. The weights of the vertices are stored in an array \( w \) such that \( w(s) \) refers to the weight of the vertex \( s \).

From Table 5.5, we obtain the following recurrence relations:

\[
\begin{align*}
[1:i+w(s)] &= [1:i] \odot [1:j] \cup [1:i] \odot [2:j] \\
[3:i+w(s)] &= [6:i] \odot [1:j] \\
[3:i-w(s)] &= [6:i] \odot [3:j] \\
[5:i+w(s)] &= [6:i] \odot [1:j]
\end{align*}
\]

As with non-weighted alliances, the initial values for each class in weighted global powerful alliances combine to form the initial vector for the algorithm. We use either the sum of the weight of the vertices in the set or \( \infty \) to represent a state which cannot logically exist. When combining sub-paths, we want to minimize the total weight of the global powerful alliance. The only classes which can logically exist initially are class 1, with a value of \( w(s) \), and class 6 with a value of 0. This means that the initial vector is \([w(s), \infty, \infty, \infty, \infty, 0]\). After execution of the algorithm, we need to determine the answer. Two states can determine a weighted global powerful alliance. These are \([1] \) and \([3:x] \) where \( x \geq 0 \). The minimum value from the final vector for these two classes indicates the minimum weight of an global powerful alliance for that path.

As before, we will only discuss a few of the 12 entries from Table 5.5. We will discuss two entries, \([1:i] \odot [3:j] \) and \([6:i] \odot [4:j] \).

To begin, look at the composition of a path of class \([1:i] \), whose root \( r \) is in the set \( S \) which is a weighted global powerful alliance, with a path of class \([3:j] \), whose root \( s \) is not in the set \( S \) which is a weighted global powerful alliance. If the weight of \( r \) is greater than or equal to the weight of \( s \) then the resultant path will still be a weighted global powerful alliance and will be class \([1:i-w(s)] \), however if the weight of \( r \) is less than the weight of \( s \) then the resultant path will be class \([2:i-w(s)] \) indicating that the root needs a neighbor in \( S \) to be a weighted global powerful alliance.
Next, look at the composition of a path of class [6:i], whose root $r$ is not in the set $S$ and $S$ contains no vertices, with a path of class [4:j], whose root $s$ is not in the set $S$ which is not a weighted global powerful alliance because $s$ is not dominated. This composition would mean that $s$ will never be dominated, making the set $S$ not a global alliance. Thus this location in Table 5.5 indicates that the composition can never occur in a valid minimum weighted global powerful alliance set in a path.
Chapter 6

Weighted Alliance Complexity

As with general alliances, it is useful to know when the concept of weighted alliances becomes NP-complete. There are 6 relevant questions. 5 of these questions can be answered in three related proofs that the relevant decision problems are NP-complete even when restricted to stars. The remaining decision problem, WEIGHTED GLOBAL OFFENSIVE ALLIANCE, is actually polynomial on stars. It is interesting to note that all 6 decision problems are NP-complete on weighted complete graphs.

6.1 Weighted Defensive and Powerful Alliance Complexity

WEIGHTED DEFENSIVE ALLIANCE

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

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

WEIGHTED POWERFUL ALLIANCE

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

QUESTION: Does $G$ have a weighted powerful alliance of size at most $k$?

**Theorem 6.1** WEIGHTED POWERFUL ALLIANCE is NP-complete, even when restricted to stars.

**Proof.** WEIGHTED POWERFUL ALLIANCE is in NP. A set $S$ of weight at most $k$ can be given as a witness to a ‘yes’ instance and verified in $O(E)$ time to be a weighted powerful alliance. We construct a polynomial-time transformation from the following, well known NP-complete problem.

SUBSETSUM

INSTANCE: Finite set $A$, size $s(a) \in Z^+$ for each $a \in A$, positive integer $B$.

QUESTION: Is there a subset $A' \subset A$ such that the sum of the sizes of the elements in $A'$ is exactly $B$?
Let $n$ be the number of elements in $A$. Let $W$ be the sum of the elements in $A$. We know that SUBSETSUM remains NP-complete even when $B > W/2$, so let $B$ be greater than half of $W$. Given an instance of PARTITION, create a star consisting of $|A|+2$ nodes with the leaves labeled $l_1 \ldots l_{n+1}$ and center labeled $c$. Give $n$ leaves weights $s(a)$ from the set $A$. Give the remaining leaf the weight $W$ and $c$ weight $W$.

Example: $A = \{5, 4, 4, 4, 3\}$; $B = 13$; $W = 24$; $n = 6$

Figure 6.1: Weighted Powerful Alliance Example

Claim: A powerful alliance of weight at most $W+B$ will exist if and only if there is a ‘yes’ instance of SUBSETSUM.
Suppose some subset $A' \subset A$ of $l_1 \ldots l_{n(n)}$ has weights that sum to exactly $B$. Choose $c$ and those $l_1 \ldots l_{n(n)}$ which are not part of $A'$. This gives a subset $S$ whose weights sum to $W + B$. Those nodes not in $S$ are either part of $A'$ or have the weight $W$. This means that the weight of those nodes not in $S$ is less than $B + W$ since $B > W/2$. This means that $S$ is a defensive alliance. $S$ is also an weighted offensive alliance and therefore a weighted powerful alliance because $l_{n+1}$ has weight $W$. This alliance is because every node is either in $S$ or in the neighborhood of $S$.

Alternatively, choose a weighted powerful alliance $S$ of size $W + B$. This weighted powerful alliance must include $c$ because its weight is greater than or equal to any of its children. This means that the other nodes in the alliance must sum to $B$. The sum of all leaves is $2W$ which means that the sum of those nodes not in $S$ is going to be $2W - B$. The powerful alliance will not include the leaf with weight $W$ because its weight is too large. Thus there must be a subset of $l_1 \ldots l_n$ whose weights sum to $B$. Those nodes in the subset which sum to $B$ will be in the weighted powerful alliance. This means that there is a ‘yes’ instance of SUBSETSUM.

A simple variation of this construction can be used to show that WEIGHTED DEFENSIVE ALLIANCE is NP-complete even when restricted to stars.

Given a set $S$ of $n$ positive integer values $w_1 \ldots w_n$ and a positive integer $B$, is there a subset of $S$ whose sum is exactly $B$? As before, let $W$ be the sum of the $w_i$'s. Note that we can assume without loss of generality that $B > W/2$. Construct a star with $n+2$ vertices, where the center has weight $W$, one leaf has weight $2(W-B) < W$, and the remaining leaves have weights $w_1 \ldots w_n$. Any defensive alliance must include the center vertex, since it has larger weight than any leaf. We claim that the answer to this SUBSETSUM instance is ‘yes’ if and only if a defensive alliance of weight $2W-B$ exists in this star graph. For example, suppose there is a subset of $S$ of weight $B$. The leaves corresponding to this subset plus the leaf of weight $2(w-B)$ have total weight $B+2(W-B) = 2W -B$ and the set $S$ consisting of the remaining leaves plus the center vertex also has weight $W-B +W = 2W-B$. Thus $S$ is a weighted defensive alliance. □

### 6.2 Weighted Offensive Alliance Complexity

WEIGHTED OFFENSIVE ALLIANCE

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

QUESTION: Does $G$ have a weighted offensive alliance of size at most $k$?
**Theorem 6.2** WEIGHTED OFFENSIVE ALLIANCE is NP-complete, even when restricted to stars.

*Proof.* WEIGHTED OFFENSIVE ALLIANCE is in NP. A set $S$ of weight at most $k$ can be given as a witness to a ‘yes’ instance and verified in $O(E)$ time to be a weighted offensive alliance. We construct a polynomial-time transformation from the following, well known NP-complete problem.

**PARTITION**

INSTANCE: Finite set $A$, size $s(a) \in \mathbb{Z}^+$ for each $a \in A$.

QUESTION: Is there a subset $A' \subset A$ such that the sum of the sizes of the elements in $A'$ is exactly the sum of the sizes of the elements not in $A'$?

Let $|A| = n$ be the number of elements in $A$. Let $W$ be the sum of the elements in $A$ and let $B = W/2$ (we assume that $W$ is an even integer). Given an instance of PARTITION, create a star consisting of $|A|+1$ nodes with the leaves labeled $l_1 \ldots l_n$ and center labeled $c$. Give $n$ leaves weights $s(a_i)$ from the set $A$. Give the node labeled $c$ weight 0.

Claim: A weighted offensive alliance of order $B$ will exist if and only if there is a ‘yes’ instance of PARTITION.

Suppose some subset $A' \subset A$ of $l_1 \ldots l_{s(a)}$ has weights that sum to exactly $B$. Choose those $l_1 \ldots l_{s(a)}$ which are not part of $A'$. This gives a subset $S$ whose weights sum to $B$. Those nodes not in $S$ are part of $A'$. Because $B = W/2$, the weight of those nodes not in $S$ is $B$. This means that $S$ is an offensive alliance because the center node, which is the only node in the boundary has neighbors of combined weight $B$ in the set and neighbors of combined weight $B$ not in the set.

Alternatively, choose a weighted offensive alliance $S$ of size $B$. This weighted offensive alliance will not include $c$ because its weight is less than that of all of its neighbors. Thus there must be a subset of $l_1 \ldots l_n$ whose weights sum to $B$. Those nodes not in the subset which also sum to $B$ will be in the weighted offensive alliance. This means that there is a ‘yes’ instance of PARTITION. □
6.3 Weighted Global Defensive and Global Powerful Alliance

Complexity

WEIGHTED GLOBAL DEFENSIVE ALLIANCE
INSTANCE: Vertex Weighted Graph $G = (V, E)$, positive integer $k < |V|$.
QUESTION: Does $G$ have a weighted global defensive alliance of size at most $k$?

WEIGHTED GLOBAL POWERFUL ALLIANCE
INSTANCE: Vertex Weighted Graph $G = (V, E)$, positive integer $k < |V|$.
QUESTION: Does $G$ have a weighted global powerful alliance of size at most $k$?

Theorem 6.3 WEIGHTED GLOBAL POWERFUL ALLIANCE is NP-complete, even when restricted to stars.
Proof. WEIGHTED GLOBAL POWERFUL ALLIANCE is in NP. A set \( S \) of weight at most \( k \) can be given as a witness to a ‘yes’ instance and verified in \( O(E) \) time to be a weighted global powerful alliance. We construct a polynomial-time transformation from the well known NP-complete problem PARTITION (defined in the previous section).

Let \( n \) be the number of elements in \( A \). Let \( W \) be the sum of the elements in \( A \) and let \( B = W/2 \) (we assume that \( W \) is an even integer). Given an instance of PARTITION, create a star consisting of \(|A|+2\) nodes with the leaves labeled \( l_1 \ldots l_{n+1} \) and center labeled \( c \). Give \( n \) leaves weights \( s(a) \) from the set \( A \). Give the remaining leaf the weight \( W \) and \( c \) weight \( W \).

Claim: A weighted global powerful alliance of order \( W+B \) will exist if and only if there is a ‘yes’ instance of PARTITION.

Suppose some subset \( A' \subset A \) of \( l_1 \ldots l_{n(a)} \) has weights that sum to exactly \( B \). Choose \( c \) and those \( l_1 \ldots l_{n(a)} \) which are not part of \( A' \). This gives a subset \( S \) whose weights sum to \( W + B \). Those nodes not in \( S \) are either part of \( A' \) or have the weight \( W \). This means that the weight of those nodes not in \( S \) is \( B + W \). This means that \( S \) is a defensive alliance. \( S \) is also an weighted offensive alliance and therefore a weighted powerful alliance because \( l_{n+1} \) has weight \( W \). This alliance is global because every node is either in \( S \) or in the neighborhood of \( S \).

Alternatively, choose a weighted global powerful alliance \( S \) of size \( W + B \). This weighted global powerful alliance must include \( c \) because its weight is greater than or equal to any of its children. This means that the other nodes in the alliance must sum to \( B \). The sum of all leaves is \( 2W \) which means that the sum of those nodes not in \( S \) is going to be \( 2W - B \). The powerful alliance will not include the leaf with weight \( W \) because its weight is too large. Thus there must be a subset of \( l_1 \ldots l_n \) whose weights sum to \( B \). Those nodes not in the subset which sum to \( B \) will be in the weighted global powerful alliance. This means that there is a ‘yes’ instance of PARTITION.

A similar construction can be used to show that WEIGHTED GLOBAL DEFENSIVE ALLIANCE is NP-complete, even when restricted to stars. \( \square \)
Example: $A = \{4, 4, 4, 4, 4\}; B = 12; W = 24; n = 6$

**Figure 6.3:** Weighted Global Powerful Alliance Example
Chapter 7

Summary and Open Problems

In 2002, Kristiansen, Hedetniemi and Hedetniemi introduced the concept of alliances in graphs. In the following four years approximately 28 papers have been written on the graph theoretical properties of defensive, offensive, powerful and global alliances in graphs.

In this thesis we present the first algorithmic study of alliances in graphs. We present linear algorithms for finding various alliance numbers in trees and polynomial algorithms for finding the defensive alliance number of series parallel graphs using a newly developed algorithm design methodology based on the well-established Wimer methodology for designing polynomial algorithms on k-terminal graphs. A complete listing of algorithms developed in this thesis is given in Table A.1 in Appendix A.

Concurrently with Cami, Balakrishnan, Deo and Dutton, we present the first complexity study of alliances in graphs. Whereas Cami et. al. present NP-completeness results for global defensive, global offensive, and global powerful alliances in general graphs, we present NP-completeness results for defensive, powerful, global defensive, and global powerful alliances even when restricted to bipartite or chordal graphs. A complete listing of complexity results in this thesis is given in Table A.2 in Appendix A.

We also show that the decision problems for alliances in vertex weighted graphs are NP-completed even when restricted to weighted trees (and even weighted stars). We then develop several linear algorithms for finding minimum alliances in weighted paths.

A host of problems remain unsolved concerning alliances in graphs. We find the following to be of particular interest.

1. Can you develop polynomial alliance algorithms for other classes of graphs. In particular for:

   (a) 2-trees

   (b) unicyclic graphs

   (c) cacti

   (d) interval graphs

2. For which classes of weighted trees can you develop polynomial alliance algorithms? In
particular, for:

(a) coronas, \( P_n \circ K_1 \) of paths

(b) binary trees

3. Can you develop polynomial algorithms for finding minimum weighted alliances in \( 2 \times n \) grid graphs?

4. Are the decision problems for offensive and global offensive alliances NP-complete when restricted to bipartite or chordal graphs? Indeed, is the decision problem for offensive alliances NP-complete for general graphs?

5. Since we know that finding a weighted global offensive alliance is polynomial for stars, and the decision problems for all 6 types of weighted alliances are NP-complete for complete graphs, for what classes of graphs is the decision problem for weighted global offensive alliances NP-complete?
APPENDICES
Appendix A

Completed Work

<table>
<thead>
<tr>
<th>Alliances</th>
<th>Trees</th>
<th>Series-Parallel</th>
<th>Paths, Weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>defensive alliances</td>
<td>$a_d(T)$ §2.1</td>
<td>$a_p(T)$ §2.2</td>
<td>$a_v(T)$ §5.1</td>
</tr>
<tr>
<td>offensive alliances</td>
<td>$a_o(T)$ §2.2</td>
<td>$a_o(T)$ §2.3</td>
<td>$a_o(P)$ §5.2</td>
</tr>
<tr>
<td>powerful alliances</td>
<td>$\gamma_d(T)$ §2.4</td>
<td>$\gamma_o(T)$ §2.5</td>
<td>$\gamma_o(P)$ §5.3</td>
</tr>
<tr>
<td>global defensive alliances</td>
<td>$\gamma_d(T)$ §2.5</td>
<td>$\gamma_o(T)$ §2.5</td>
<td>$\gamma_o(P)$ §5.4</td>
</tr>
<tr>
<td>global offensive alliances</td>
<td>$\gamma_o(T)$ §2.5</td>
<td>$\gamma_o(T)$ §2.5</td>
<td>$\gamma_o(P)$ §5.4</td>
</tr>
<tr>
<td>global powerful alliances</td>
<td>$\gamma_o(T)$ §2.5</td>
<td>$\gamma_o(T)$ §2.5</td>
<td>$\gamma_o(P)$ §5.4</td>
</tr>
</tbody>
</table>

Table A.1: Alliance Algorithms

<table>
<thead>
<tr>
<th>Alliances</th>
<th>General</th>
<th>Bipartite</th>
<th>Chordal</th>
<th>Weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>defensive alliances</td>
<td>§3.1</td>
<td>§3.1</td>
<td>§3.1</td>
<td>§6.1</td>
</tr>
<tr>
<td>offensive alliances</td>
<td>§3.2</td>
<td>§3.2</td>
<td>§3.2</td>
<td>§6.2</td>
</tr>
<tr>
<td>powerful alliances</td>
<td>[6]</td>
<td>§3.3</td>
<td>§3.3</td>
<td>§6.1</td>
</tr>
<tr>
<td>global defensive alliances</td>
<td>[6]</td>
<td>§3.4</td>
<td>§3.4</td>
<td>[6]</td>
</tr>
<tr>
<td>global offensive alliances</td>
<td>[6]</td>
<td>§3.4</td>
<td>§3.4</td>
<td>[6]</td>
</tr>
</tbody>
</table>

Table A.2: Alliance Complexity
Appendix B

Powerful Alliance Implementation

This C++ code implements the algorithm for finding a powerful alliance in a tree.

```cpp
#include <iostream>
#include <fstream>
#include <stdlib.h>
#define MAXINT 1000000
#define NumNodes 500
using namespace std;

struct classes {
  int value;
  int cost;
};

struct node {
  classes myclass[7];
  int parent;
};

void powerful_alliance(node tree[], int p, int c) {
  node temp = tree[p];
  bool one=false, two=false, three=false,
      four=false, five=false, six=false, seven=false;

  // [1:i] with [1:j]
  if(one ||
      (temp.myclass[0].cost > (tree[p].myclass[0].cost+tree[c].myclass[0].cost))) {
    temp.myclass[0].value = tree[p].myclass[0].value+1;
    temp.myclass[0].cost = tree[p].myclass[0].cost+tree[c].myclass[0].cost;
    if(temp.myclass[0].cost>MAXINT) temp.myclass[0].cost = MAXINT;
    one = true;
  }

  // [1:i] with [2:j]
  if(tree[c].myclass[1].value == -1) {
    if(one ||
      (temp.myclass[0].cost > (tree[p].myclass[0].cost+tree[c].myclass[1].cost))) {
      temp.myclass[0].value = tree[p].myclass[0].value+1;
      temp.myclass[0].cost = tree[p].myclass[0].cost+tree[c].myclass[0].cost;
      if(temp.myclass[0].cost>MAXINT) temp.myclass[0].cost = MAXINT;
      one = true;
    }
  }

  // [1:i] with [3:j]
  if(tree[p].myclass[0].value > 0) {
    if(one ||
      (temp.myclass[0].cost > (tree[p].myclass[0].cost+tree[c].myclass[2].cost))) {
      temp.myclass[0].value = tree[p].myclass[0].value-1;
      temp.myclass[0].cost = tree[p].myclass[0].cost+tree[c].myclass[0].cost;
    }
  }
```

71
temp.myclass[0].cost = tree[p].myclass[0].cost + tree[c].myclass[2].cost;
if(temp.myclass[0].cost > MAXINT) temp.myclass[0].cost = MAXINT;
one = true;
}
} else {
if(!two ||
(temp.myclass[0].cost > (tree[p].myclass[0].cost + tree[c].myclass[2].cost))) {
temp.myclass[1].value = tree[p].myclass[0].value - 1;
temp.myclass[1].cost = tree[p].myclass[0].cost + tree[c].myclass[2].cost;
if(temp.myclass[1].cost > MAXINT) temp.myclass[1].cost = MAXINT;
two = true;
}
}

//[1:i] with [4:j]
if(tree[p].myclass[0].value > 0) {
if(!one ||
(temp.myclass[0].cost > (tree[p].myclass[0].cost + tree[c].myclass[3].cost))) {
temp.myclass[0].value = tree[p].myclass[0].value - 1;
temp.myclass[0].cost = tree[p].myclass[0].cost + tree[c].myclass[3].cost;
if(temp.myclass[0].cost > MAXINT) temp.myclass[0].cost = MAXINT;
one = true;
}
} else {
if(!two ||
(temp.myclass[1].cost > (tree[p].myclass[0].cost + tree[c].myclass[3].cost))) {
temp.myclass[1].value = tree[p].myclass[0].value - 1;
temp.myclass[1].cost = tree[p].myclass[0].cost + tree[c].myclass[3].cost;
if(temp.myclass[1].cost > MAXINT) temp.myclass[1].cost = MAXINT;
two = true;
}
}

//[1:i] with [6:j]
if(tree[c].myclass[5].value == -1) {
if(tree[p].myclass[0].value > 0) {
if(!one ||
(temp.myclass[0].cost > (tree[p].myclass[0].cost + tree[c].myclass[5].cost))) {
temp.myclass[0].value = tree[p].myclass[0].value - 1;
temp.myclass[0].cost = tree[p].myclass[0].cost + tree[c].myclass[5].cost;
if(temp.myclass[0].cost > MAXINT) temp.myclass[0].cost = MAXINT;
one = true;
}
} else {
if(!two ||
(temp.myclass[1].cost > (tree[p].myclass[0].cost + tree[c].myclass[5].cost))) {
temp.myclass[1].value = tree[p].myclass[0].value - 1;
temp.myclass[1].cost = tree[p].myclass[0].cost + tree[c].myclass[5].cost;
if(temp.myclass[1].cost > MAXINT) temp.myclass[1].cost = MAXINT;
two = true;
}
}
}

//[1:i] with [7:j]
if(tree[c].myclass[6].value == -1) {
if(tree[p].myclass[0].value > 0) {
  if(!one ||
    (temp.myclass[0].cost > (tree[p].myclass[0].cost + tree[c].myclass[6].cost))) {
    temp.myclass[0].value = tree[p].myclass[0].value - 1;
    temp.myclass[0].cost = tree[p].myclass[0].cost + tree[c].myclass[6].cost;
    if(temp.myclass[0].cost > MAXINT) temp.myclass[0].cost = MAXINT;
    one = true;
  } 
  else {
    if(!two ||
      (temp.myclass[1].cost > (tree[p].myclass[0].cost + tree[c].myclass[6].cost))) {
      temp.myclass[1].value = tree[p].myclass[1].value - 1;
      temp.myclass[1].cost = tree[p].myclass[0].cost + tree[c].myclass[6].cost;
      if(temp.myclass[1].cost > MAXINT) temp.myclass[1].cost = MAXINT;
      two = true;
    } 
  } 
  //[2:i] with [1:j]
  if(tree[p].myclass[1].value == -1) {
    if(!one ||
      (temp.myclass[0].cost > (tree[p].myclass[1].cost + tree[c].myclass[0].cost))) {
      temp.myclass[0].value = tree[p].myclass[1].value + 1;
      temp.myclass[0].cost = tree[p].myclass[1].cost + tree[c].myclass[0].cost;
      if(temp.myclass[0].cost > MAXINT) temp.myclass[0].cost = MAXINT;
      one = true;
    } 
    else {
      if(!two ||
        (temp.myclass[1].cost > (tree[p].myclass[1].cost + tree[c].myclass[0].cost))) {
        temp.myclass[1].value = tree[p].myclass[1].value + 1;
        temp.myclass[1].cost = tree[p].myclass[1].cost + tree[c].myclass[0].cost;
        if(temp.myclass[1].cost > MAXINT) temp.myclass[1].cost = MAXINT;
        two = true;
      } 
    } 
    //[2:i] with [2:j]
    if(tree[c].myclass[1].value == -1) {
      if(tree[p].myclass[1].value == -1) {
        if(!one ||
          (temp.myclass[0].cost > (tree[p].myclass[1].cost + tree[c].myclass[1].cost))) {
          temp.myclass[0].value = tree[p].myclass[1].value + 1;
          temp.myclass[0].cost = tree[p].myclass[1].cost + tree[c].myclass[1].cost;
          if(temp.myclass[0].cost > MAXINT) temp.myclass[0].cost = MAXINT;
          one = true;
        } 
        else {
          if(!two ||
            (temp.myclass[1].cost > (tree[p].myclass[1].cost + tree[c].myclass[1].cost))) {
            temp.myclass[1].value = tree[p].myclass[1].value + 1;
            temp.myclass[1].cost = tree[p].myclass[1].cost + tree[c].myclass[1].cost;
            if(temp.myclass[1].cost > MAXINT) temp.myclass[1].cost = MAXINT;
            two = true;
          } 
        } 
      } 

if(!two ||
(temp.myclass[1].cost > (tree[p].myclass[1].cost+tree[c].myclass[2].cost))) {
    temp.myclass[1].value = tree[p].myclass[1].value-1;
    if(temp.myclass[1].cost>MAXINT) temp.myclass[1].cost = MAXINT;
    two = true;
}
if(!two ||
(temp.myclass[1].cost > (tree[p].myclass[1].cost+tree[c].myclass[3].cost))) {
    temp.myclass[1].value = tree[p].myclass[1].value-1;
    temp.myclass[1].cost = tree[p].myclass[1].cost+tree[c].myclass[3].cost;
    if(temp.myclass[1].cost>MAXINT) temp.myclass[1].cost = MAXINT;
    two = true;
}
if(tree[c].myclass[5].value == -1) {
    if(!two ||
(temp.myclass[1].cost > (tree[p].myclass[1].cost+tree[c].myclass[5].cost))) {
        temp.myclass[1].value = tree[p].myclass[1].value-1;
        temp.myclass[1].cost = tree[p].myclass[1].cost+tree[c].myclass[5].cost;
        if(temp.myclass[1].cost>MAXINT) temp.myclass[1].cost = MAXINT;
        two = true;
    }
}
if(tree[c].myclass[6].value == -1) {
    if(!two ||
(temp.myclass[1].cost > (tree[p].myclass[1].cost+tree[c].myclass[6].cost))) {
        temp.myclass[1].value = tree[p].myclass[1].value-1;
        temp.myclass[1].cost = tree[p].myclass[1].cost+tree[c].myclass[6].cost;
        if(temp.myclass[1].cost>MAXINT) temp.myclass[1].cost = MAXINT;
        two = true;
    }
}
if(tree[c].myclass[0].value > 0) {
    if(!four ||
(temp.myclass[3].cost > (tree[p].myclass[2].cost+tree[c].myclass[0].cost))) {
        temp.myclass[3].value = tree[p].myclass[2].value+1;
        temp.myclass[3].cost = tree[p].myclass[2].cost+tree[c].myclass[0].cost;
        if(temp.myclass[3].cost>MAXINT) temp.myclass[3].cost = MAXINT;
        four = true;
    }
}
if(!seven ||
(temp.myclass[6].cost > (tree[p].myclass[2].cost+tree[c].myclass[3].cost))) {
    temp.myclass[6].value = tree[p].myclass[2].value-1;
if(temp.myclass[6].cost>MAXINT) temp.myclass[6].cost = MAXINT;
seven = true;
}

//[3:i] with [6:j]
if(!seven ||
(temp.myclass[6].cost > (tree[p].myclass[3].cost+tree[c].myclass[0].cost))) {
    temp.myclass[6].value = tree[p].myclass[2].value-1;
    if(temp.myclass[6].cost>MAXINT) temp.myclass[6].cost = MAXINT;
    seven = true;
}

//[4:i] with [1:j]
if(tree[c].myclass[0].value >0) {
    if(!four ||
    (temp.myclass[3].cost > (tree[p].myclass[3].cost+tree[c].myclass[0].cost)))){
        temp.myclass[3].value = tree[p].myclass[3].value+1;
        temp.myclass[3].cost = tree[p].myclass[3].cost+tree[c].myclass[0].cost;
        if(temp.myclass[3].cost>MAXINT) temp.myclass[3].cost = MAXINT;
        four = true;
    }
}

//[4:i] with [4:j]
if(tree[c].myclass[3].value == 1) {
    if(!three ||
        temp.myclass[2].value = tree[p].myclass[3].value-1;
        if(temp.myclass[2].cost>MAXINT) temp.myclass[2].cost = MAXINT;
        three=true;
    }
    else
    if(!four ||
    (temp.myclass[3].cost > (tree[p].myclass[3].cost+tree[c].myclass[5].cost)))) {
        temp.myclass[3].value = tree[p].myclass[3].value-1;
        if(temp.myclass[3].cost>MAXINT) temp.myclass[3].cost = MAXINT;
        four=true;
    }
}

//[4:i] with [6:j]
if(tree[p].myclass[3].value == 1) {
    if(!three ||
        temp.myclass[2].value = tree[p].myclass[3].value-1;
        if(temp.myclass[2].cost>MAXINT) temp.myclass[2].cost = MAXINT;
        three=true;
    }
    else
    if(!four ||
    (temp.myclass[3].cost > (tree[p].myclass[3].cost+tree[c].myclass[5].cost)))) {
        temp.myclass[3].value = tree[p].myclass[3].value-1;
if(temp.myclass[3].cost>MAXINT) temp.myclass[3].cost = MAXINT;
four=true;
}

//[5:i] with [1:j]
if(tree[c].myclass[0].value > 0) {
if(tree[p].myclass[4].value == -1) {
if(!three ||
(temp.myclass[2].cost > (tree[p].myclass[4].cost+tree[c].myclass[0].cost))) {
 temp.myclass[2].value = tree[p].myclass[4].value+1;
temp.myclass[2].cost = tree[p].myclass[4].cost+tree[c].myclass[0].cost;
if(temp.myclass[2].cost>MAXINT) temp.myclass[2].cost = MAXINT;
three=true;
}
}
}

//[5:i] with [5:j]
if(!five ||
(temp.myclass[5].cost > (tree[p].myclass[4].cost+tree[c].myclass[0].cost))) {
 temp.myclass[5].value = tree[p].myclass[4].value-1;
temp.myclass[5].cost = tree[p].myclass[4].cost+tree[c].myclass[0].cost;
if(temp.myclass[5].cost>MAXINT) temp.myclass[5].cost = MAXINT;
five = true;
}

//[6:i] with [1:j]
if(tree[c].myclass[0].value > 0) {
if(tree[p].myclass[5].value == -1) {
if(!three ||
(temp.myclass[2].cost > (tree[p].myclass[5].cost+tree[c].myclass[0].cost))) {
 temp.myclass[2].value = tree[p].myclass[5].value+1;
temp.myclass[2].cost = tree[p].myclass[5].cost+tree[c].myclass[0].cost;
if(temp.myclass[2].cost>MAXINT) temp.myclass[2].cost = MAXINT;
three = true;
}
}
}

//[6:i] with [4:j]
if(!five ||
(temp.myclass[4].cost > (tree[p].myclass[5].cost+tree[c].myclass[3].cost))) {
temp.myclass[4].value = tree[p].myclass[5].value-1;
if(temp.myclass[4].cost>MAXINT) temp.myclass[4].cost = MAXINT;
five = true;
}

//[6:i] with [5:j]
if(!five ||
(temp.myclass[4].cost > (tree[p].myclass[5].cost+tree[c].myclass[4].cost))) {
    temp.myclass[4].value = tree[p].myclass[5].value-1;
if(temp.myclass[4].cost>MAXINT) temp.myclass[4].cost = MAXINT;
five = true;
}

//[6:i] with [6:j]
if(!six ||
(temp.myclass[5].cost > (tree[p].myclass[5].cost+tree[c].myclass[5].cost))) {
    temp.myclass[5].value = tree[p].myclass[5].value-1;
temp.myclass[5].cost = tree[p].myclass[5].cost+tree[c].myclass[5].cost;
if(temp.myclass[5].cost>MAXINT) temp.myclass[5].cost = MAXINT;
six = true;
}

//[7:i] with [1:j]
if(tree[c].myclass[0].value >0) {
    if(tree[p].myclass[6].value = -1) {
        if(!three ||
            (temp.myclass[2].cost > (tree[p].myclass[6].cost+tree[c].myclass[0].cost))) {
            temp.myclass[2].value = tree[p].myclass[6].value+1;
temp.myclass[2].cost = tree[p].myclass[6].cost+tree[c].myclass[0].cost;
if(temp.myclass[2].cost>MAXINT) temp.myclass[2].cost = MAXINT;
        three = true;
    }
    } else {
        if(!seven ||
            (temp.myclass[6].cost > (tree[p].myclass[6].cost+tree[c].myclass[0].cost))) {
            temp.myclass[6].value = tree[p].myclass[6].value+1;
temp.myclass[6].cost = tree[p].myclass[6].cost+tree[c].myclass[0].cost;
if(temp.myclass[6].cost>MAXINT) temp.myclass[6].cost = MAXINT;
        seven = true;
    }
  }
}

//[7:i] with [6:j]
if(!seven ||
(temp.myclass[6].cost > (tree[p].myclass[6].cost+tree[c].myclass[5].cost))) {
    temp.myclass[6].value = tree[p].myclass[6].value-1;
temp.myclass[6].cost = tree[p].myclass[6].cost+tree[c].myclass[5].cost;
if(temp.myclass[6].cost>MAXINT) temp.myclass[6].cost = MAXINT;
seven = true;
}

//if a particular case can't occur by combining these two subtrees,
//then set the cost to MAXINT
if(!one)
temp.myclass[0].cost = MAXINT;
  if(!two)
temp.myclass[1].cost = MAXINT;
if(!three)
temp.myclass[2].cost = MAXINT;
if(!four)
temp.myclass[3].cost = MAXINT;
if(!five)
temp.myclass[4].cost = MAXINT;
if(!six)
temp.myclass[5].cost = MAXINT;
if(!seven)
temp.myclass[6].cost = MAXINT;
//p becomes the root of the new subtree, so copy costs and values
    tree[p].myclass[0] = temp.myclass[0];
    tree[p].myclass[1] = temp.myclass[1];
    tree[p].myclass[3] = temp.myclass[3];
tree[p].myclass[4] = temp.myclass[4];
    tree[p].myclass[5] = temp.myclass[5];
tree[p].myclass[6] = temp.myclass[6];
}

void init(node tree[]) {
    //sets each node in the tree to the initial vector
    for(int i =0; i<NumNodes; i++) {
        tree[i].myclass[0].value = 1; tree[i].myclass[0].cost = 1;
        tree[i].myclass[1].value = 1; tree[i].myclass[1].cost = MAXINT;
        tree[i].myclass[2].value = -1; tree[i].myclass[2].cost = MAXINT;
        tree[i].myclass[3].value = -1; tree[i].myclass[3].cost = MAXINT;
        tree[i].myclass[4].value = -1; tree[i].myclass[4].cost = MAXINT;
        tree[i].myclass[5].value = -1; tree[i].myclass[5].cost = 0;
        tree[i].myclass[6].value = -1; tree[i].myclass[6].cost = MAXINT;
    }
}

int main (int argc, const char * argv[]) {
    node tree[NumNodes];
    srand( time(NULL) );
    tree[0].parent = -1;
    // random tree generation
    for(int i = 1; i<NumNodes; i++) {
        tree[i].parent = rand()%i;
    }
    init(tree);

    for(int j = NumNodes-1; j>0; j--) {
        if(tree[j].parent > NumNodes) cout << "Parent out of range" << endl;
        else {
            powerful_alliance(tree, tree[j].parent, j);
            if(tree[j].parent > NumNodes) cout<<"Parent out of range"<<endl;
        }
    }
    cout << "Root values: " << endl;
    for(int i= 0; i<7; i++) {

cout << "Class " << i+1 << " : " << endl;
cout << " Value : " << tree[0].myclass[i].value << endl;
if(tree[0].myclass[i].cost == MAXINT)
cout << " Cost : MAXINT" << endl;
else
cout << " Cost : " << tree[0].myclass[i].cost << endl;
}

if(((tree[0].myclass[4].cost < tree[0].myclass[3].cost)
&& (tree[0].myclass[4].cost < tree[0].myclass[2].cost))
&& (tree[0].myclass[4].cost < tree[0].myclass[0].cost))
cout << "Choose class 5" << endl;

else if(((tree[0].myclass[3].cost < tree[0].myclass[4].cost)
&& (tree[0].myclass[3].cost < tree[0].myclass[2].cost))
&& (tree[0].myclass[3].cost < tree[0].myclass[0].cost))
cout << "Choose class 4" << endl;

else if(((tree[0].myclass[2].cost < tree[0].myclass[3].cost)
&& (tree[0].myclass[2].cost < tree[0].myclass[4].cost))
&& (tree[0].myclass[2].cost < tree[0].myclass[0].cost))
cout << "Choose class 3" << endl;

else if(((tree[0].myclass[0].cost < tree[0].myclass[3].cost)
&& (tree[0].myclass[0].cost < tree[0].myclass[2].cost))
&& (tree[0].myclass[0].cost < tree[0].myclass[4].cost))
cout << "Choose class 1" << endl;

else // we have a tie somewhere
  cout << "There are at least two equivalent classes" << endl;
return 0;
}
Appendix C

Weighted Global Defensive Alliances Implementation

This C++ code implements the algorithm for finding a weighted global defensive alliance in a tree.

```cpp
#include <iostream>
#include <fstream>
#include <stdlib.h>

#define MAXINT 100000
#define NumNodes 50
using namespace std;

struct classes {
    int value;
    int cost;
};

struct node {
    classes myclass[4];
    int weight;
    int end_class;
};

void init(node path[]) {
    for(int i =0; i<NumNodes; i++) {
        path[i].myclass[0].value = path[i].weight;
        path[i].myclass[0].cost = path[i].weight;
        path[i].myclass[1].value = path[i].weight;
        path[i].myclass[1].cost = MAXINT;
        //Classes 3 & 4 don't have values, so values are not set
        path[i].myclass[2].cost = MAXINT;
        path[i].myclass[3].cost = 0;
    }
}

void wgda(node path[], int p, int c) {
    node temp = path[p];
    bool one=false, two=false, three=false, four=false;

    //[1:i] with [1:j]
    temp.myclass[0].cost = path[p].myclass[0].cost + path[c].myclass[0].cost;
    temp.myclass[0].value = path[p].myclass[0].value + path[c].weight;
    one = true;

    //[1:i] with [2:j]
    if (one ||
        (temp.myclass[0].cost > (path[p].myclass[0].cost + path[c].myclass[1].cost)))) {
        if((path[p].myclass[0].value+path[c].myclass[1].value) >= 0) {
            temp.myclass[0].cost = path[p].myclass[0].cost + path[c].myclass[1].cost;
        }
    }
```

81
temp.myclass[0].value = path[p].myclass[0].value + path[c].weight;
one = true;
}

//[1:i] with [3]
if(path[c].weight <= path[p].myclass[0].value) {
  if(one ||
    (temp.myclass[0].cost > (path[p].myclass[0].cost+path[c].myclass[2].cost))) {
    temp.myclass[0].cost = path[p].myclass[0].cost + path[c].myclass[2].cost;
temp.myclass[0].value = path[p].myclass[0].value - path[c].weight;
one = true;
  } else
    if(!two ||
      (temp.myclass[1].cost > (path[p].myclass[0].cost+path[c].myclass[2].cost))) {
      temp.myclass[1].cost = path[p].myclass[0].cost + path[c].myclass[2].cost;
temp.myclass[1].value = path[p].myclass[0].value - path[c].weight;
two = true;
    }
  }
//[1:i] with [4]
if(path[c].weight <= path[p].myclass[0].value) {
  if(one ||
    (temp.myclass[0].cost > (path[p].myclass[0].cost+path[c].myclass[3].cost))) {
      temp.myclass[0].cost = path[p].myclass[0].cost + path[c].myclass[3].cost;
temp.myclass[0].value = path[p].myclass[0].value - path[c].weight;
one = true;
    } else
      if(!two ||
        (temp.myclass[1].cost > (path[p].myclass[0].cost+path[c].myclass[3].cost))) {
          temp.myclass[1].cost = path[p].myclass[0].cost + path[c].myclass[3].cost;
temp.myclass[1].value = path[p].myclass[0].value - path[c].weight;
two = true;
        }
    }
  //[4] with [1:j]
  if(path[p].weight <= path[c].myclass[0].value) {
    temp.myclass[2].cost = path[p].myclass[3].cost+path[c].myclass[0].cost;
three =true;
  }
//[4] with [3]
four = true;

//if no combination gets us a value for a case, set the cost to MAXINT
if(!one)
temp.myclass[0].cost = MAXINT;
  if(!two)
temp.myclass[1].cost = MAXINT;
    if(!three)
temp.myclass[2].cost = MAXINT;

82
if(!four)
temp.myclass[3].cost = MAXINT;
//now that we have the new costs, update the costs for the path
path[p].myclass[0] = temp.myclass[0];
path[p].myclass[1] = temp.myclass[1];
path[p].myclass[3] = temp.myclass[3];

cout << endl << endl;
}

int main (int argc, char * const argv[]) {
    node path[NumNodes];
srand( time(NULL) );
for(int i=0; i<NumNodes; i++) {
    path[i].weight = (rand()%(NumNodes-1))+1;
} init(path);
for(int j = NumNodes-1; j>0; j--) {
    wgda(path, j-1, j);
}
for(int i=0; i<NumNodes; i++) {
    cout << "Node " << i << ": " << path[i].weight << endl;
}

cout << "Root values: " << endl;
for(int i = 0; i<4; i++) {
    cout << "Case " << i+1 << ": " << endl;
    if(i<2) {
        cout << " Value : " << path[0].myclass[i].value << endl;
    }
    cout << " Cost : " << path[0].myclass[i].cost << endl;
}

cout << endl << endl;
if(path[0].myclass[0].cost < path[0].myclass[2].cost)
cout << "Choose case 1" << endl;
else
cout << "Choose case 3" << endl;
return 0;
}
Bibliography


