Parameterized Complexity Of Defensive Alliances

B.A.A Horn

October 2022

Contents

1 Introduction

This thesis aims to tackle two broad areas of work surrounding Defensive Alliance and its related problems. The first is to develop parameterized algorithms for finding *Defensive Alliances* in graphs, and for variants of this problem. The second is to consider the practical aspects of finding them in graphs, through experimental work.

A Defensive Alliance is a subset of vertices in a graph, such that each vertex in the subset has more neighbours in the subset than outside it. Parameterized Algorithms are ones that measure their total complexity based on multiple parameters, beyond just input size.

Defensive Alliances were originally introduced by Kristiansen et al. [1], to model the concept of military alliances. The reason for the definition is that every vertex in the alliance has enough neighbours for backup if it is attacked by a vertex outside the set. This also leads to the related problems of Offensive and Powerful alliances, where a vertex can attack others because it will have sufficient neighbours to back it up. Powerful Alliances extend this further, by defining a set that is both Defensive and Offensive.

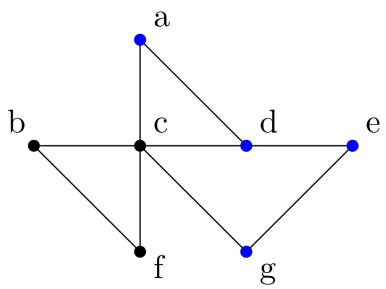


Figure 1: An example of a non-minimal Defensive Alliance, consisting of vertices {a, d, e, g} (labeled in blue).

An example of a Defensive Alliance is shown in Figure 1, where each vertex included has at least half of its neighbours in the alliance. There are several other Defensive Alliances in this graph, such as the smaller subset of $\{b, f\}$ or $\{e, g\}$. It is worth noting that if all the vertices are included in the subset, then that is also a Defensive Alliance.

Defensive Alliances have been used in a variety of fields to model a diverse set of problems. These include uses in bio-informatics, modelling communities and clusters, distributed computing, fault tolerant networking and more [2]. They have also been used as part of an Alliance Polynomial, which is a polynomial where the coefficients are the number of exact Defensive Alliances of that size, to study properties for some types of graphs [3]. Variants of the problem can be used to extend the uses even further, such as building transport networks that are protected.

1.1 Overview

This thesis will focus on three areas that form the research aims. The first area focused on is how existing and known algorithms can be applied to new variants. The second area considers the application of new graph parameters, and how they can be applied to the existing problems and their variants. Finally computational experiments that are focused on finding alliances in graphs. There is a secondary focus on improving the bounds of algorithms, but there are only a limited set of results focused in that aspect.

The next section discusses the prior work in more detail, going over the problems that have been studied and their known results for various parameters and variants. After that, new variants of Defensive Alliance are discussed, along with the application of various parameters found in the literature concerning them. New Parameters are then discussed in detail, primarily focused on Defensive Alliance as the core problem but with some consideration given to variants. Next, the experiments are discussed which review the practical side of various algorithms. Finally, concluding thoughts are given.

2 Preliminaries and Prior Work

Defensive Alliances, along with the sister problems of Offensive and Powerful alliances, have been studied in detail since they were first proposed by Kristiansen et al. [1] This section covers the known results regarding the various problems general complexity, along with the results for different graph parameters and graph types, in order to narrow the scope. It is worth noting that only a few variants have parameterized results outside these core three problems. There is an extensive amount of work regarding their general properties, which have been covered in surveys by Ouazine et al. [2] and Yero et al. [4]

2.1 Definitions

All graphs discussed are of the form $G = (V, E)$. They are all simple graphs, where no pair of vertices have multiple direct edges between them, with no self loops. $N_G(v)$ referrers to the open neighbour of a vertex v in G, which is the neighbours of v. $N_G[v]$ is the closed neighbourhood of the vertex v in G, which is the open neighbourhood and the vertex together.

For a graph $G = (V, E)$ and a non-empty subset $S \subseteq V$, a vertex $v \in S$ is considered r-protected if $N_S(v) \geq N_{V/S}(v) + r$, where $N_G(v)$. [5]. A vertex is called strongly protected when it remains protected even if any one of its neighbours are removed from the alliance.

A subset is called an r-Defensive Alliance if every vertex inside it is r-protected. r is taken to be -1 if it is left out.

The decision variant of this problem is defined as:

r-Defensive Alliance
Input: A graph $G = (V, E)$, an integer r and an integer k
Question: Is there a subset $S \subseteq V$ of size at most k where every vertex is r-protected?

Defensive Alliances are commonly discussed with two related problems, which are the Offensive and Powerful Alliances. An r-Offensive Alliance is one where every vertex outside of S has at least r more neighbours inside S than outside it. A r-Powerful Alliance is itself both an r-Defensive Alliance and an r-Offensive Alliance. It is worth noting that if all the vertices are included in the subset, then it is a Defensive, Offensive, and Powerful Alliance. This does not apply for larger values of r, but does in the case of -1.

2.1.1 ILP Formulation of Defensive Alliance

An Integer Linear Programming (ILP) formulation is useful for understanding the problem, and it can also act as a baseline for solving instances. An example formulation is given below:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{|V|} x_j \\
\text{subject to} \quad & \sum_{j=1}^{|V|} x_j \geq 1, \\
& \sum_{i \in N(j)} x_i \geq x_j t_j, \quad j = 1, \ldots, |V|, \\
& x_j \in \{0, 1\}, \quad j = 1, \ldots, |V|
\end{aligned}
$$

This formulation is for finding the smallest Defensive Alliance in the graph, while ensuring it has at least one vertex inside the set. $N(v)$ which refers to the set of neighbours of vertex v (its open neighbourhood, so excluding v itself.) and x_v is a binary variable indicating if vertex v is included in the alliance. t_v refers to the threshold of vertex, sometimes called the demand, which is the number of adjacent vertices that must be in the alliance for a vertex to be included. It is defined as $t_v = \lceil \frac{|N(v)|+r}{2} \rceil$.

2.1.2 ILP Results

Many of the later algorithms rely on results showing two cases of Integer Linear Programming (ILP) are in $\mathcal{FPT}$. The first is the initial feasibility result, which seems to determine a satisfying set of values, and the second is the optimization variant which aims to minimize an objective function.

Theorem 1 ([6]). *p*-Variable Integer Linear Programming Feasibility *can be solved using* $O(p^{2.5p+o(p)} \cdot L)$ *arithmetic operations and space polynomial in L. Here L is the number of bits in the input.*

Theorem 2 ([7]). *p*-Opt-ILP *can be solved using* $O(p^{2.5p+o(p)} \cdot L \cdot log(MN))$ *arithmetic operations and space polynomial in L. Here, L is the number of bits in the input, N is the maximum of the absolute values any variable can take, and M is an upper bound on the absolute value of the minimum taken by the objective function.*

2.2 Problems

Before going into the complexity, it is worth discussing what problems and variants have already been considered in the literature.

Defensive Alliance, Offensive Alliance and Powerful Alliance have been considered in detail in the literature. Their definitions are given below:

r-Defensive Alliance
Input: A graph $G = (V, E)$, an integer r and an integer k
Question: Is there a set $S \subseteq V$ of size at most k where every vertex is r-protected?

r-Offensive Alliance
Input: A graph $G = (V, E)$, an integer r and an integer k
Question: Is there a set $S \subseteq V$ of size at most k where every vertex outside of S is r-protected by S?

r-Powerful Alliance
Input: A graph $G = (V, E)$, an integer r and an integer k
Question: Is there a set $S \subseteq V$ of size at most k where every vertex inside and outside of S is r-protected by S?

Offensive and Powerful Alliance will not be considered in further detail, but they are an important part of the literature and their results will be noted. As with the above examples the alliances are prefixed with r to represent the level of protection required for S to be an alliance, which is otherwise assumed to be -1 for the majority of problems.

One of the popular problems in the literature was the Global variants of the core problems which involve finding an alliance that is also a dominating set, where every vertex not in the set is adjacent to at least one member of the set.

Global Defensive Alliance
Input: A graph $G = (V, E)$ and an integer k
Question: Is there a dominating set $S \subseteq V$ of size at most k where every vertex is r-protected?

Two variants have been discussed in detail over several recent papers. These are Globally Minimal Defensive Alliance (not to be confused with Global Defensive Alliance) and Locally Minimal Defensive Alliance. Globally Minimal Defensive Alliance (GMDA), or Critical Defensive Alliances in some of the literature, is a Defensive Alliance that does not contain any smaller Defensive Alliances. Locally Minimal Defensive Alliances are similar but only require that removing a single vertex does not result in a smaller Defensive Alliance.

While not immediately obvious, it is possible to test if an alliance is globally minimal or not by attempting to remove each vertex and checking which of the remaining become unprotected. If all the vertices result in a empty set after unprotected vertices are repeatedly removed, it is minimal. Otherwise, it is not minimal as a smaller alliance will have been found [8].

Globally Minimal Defensive Alliance
Input: A graph $G = (V, E)$, an integer r and an integer k
Question: Is there a set $S \subseteq V$ of size at most k where every vertex is r-protected and where no proper subset is also alliance?

Locally Minimal Defensive Alliance
Input: A graph $G = (V, E)$, an integer r and an integer k
Question: Is there a set $S \subseteq V$ of size at most k where every vertex is r-protected and where removing a single vertex from S does not result in a smaller alliance?

Another type of variant studied in the literature are the weighted variants. These use a function $\rho_S(v)$ that takes a vertex, which must be larger than 0.5 for a vertex to be included in the alliance. $\rho_S(v)$ is the local density of S around a vertex v. In the weighted cases, $\rho_S(v)$ is be defined as $w(N_[v] \cap S)/w(N_[v])$, where $w : V \to \mathbb{R}$ is an arbitrary weighting function. [9]

Weighted Defensive Alliance
Input: A graph $G = (V, E)$, a weighting function $w : V \to \mathbb{R}$ and an integer k
Question: Is there a set $S \subseteq V$ of size at most k where every vertex $v \in S$ satisfies $\rho_S(v) \geq 0.5$?

Beyond these variants, a few specific modifiers to them have been discussed. These include forcing the alliances to be connected, including a specific vertex, or being an exact size.

Finally, while not discussed further, the existence of a similar problem called Secure Sets are worth noting. See Brigham et al. [10] for an introduction to Secure Sets.

2.3 General Complexity

When reviewing the literature, it was found that most studied graph types resulted in Alliance problems being $\mathcal{NP}$-Complete with rare exceptions. The notable exceptions are those of trees and series-parallel graphs. Defensive Alliance is known to be $\mathcal{NP}$-Complete on general graphs [4], circle graphs [11] and planar graphs [12]. For the rooted case, the minimum size of a Defensive Alliance containing a specific vertex is NP-hard [13]. Rooted Globally Minimal Defensive Alliance is NP-complete on general graphs as well [14]. Global Defensive Alliance is NP-Complete on General Graphs, Chordal and Bipartite graphs [15] [4]. This further extends to Rooted Powerful Alliance [16].

Trees and Series parallel graphs seem to be an exception to the rule of $\mathcal{NP}$-Completeness. Global Defensive Alliance can be solved in linear time on both [17], with a $O(|V| \log D)$ algorithm on trees (where D is the maximum degree) [15]. Further, Connected Locally Minimal Defensive Alliance and Connected Globally Minimal Defensive Alliance can both be found in $O(n \log n)$ time on trees [14] [18].

Finding a minimum-cost weighted Defensive Alliance of a specific size is $\mathcal{NP}$-complete on stars. However, it can be found on paths, cycles, trees of bounded degree, and graphs of bounded pathwidth in linear time. Minimum cardinally weighted alliances can also be solved in polynomial time on a tree [9]. These weighted results directly carry over to normal Defensive Alliances, as it is possible to model them with a weight of one.

2.4 Parameterized Complexity

Parameterized Complexity studies how different parameters (beyond just the number of vertices or edges) influence the running time of a problem. Problems being studied exist in several complexity classes, such as Fixed Parameter Tractable (FPT), part of the the W hierarchy, or in XP. This study is primarily focused on FPT algorithms which are ones that have a running time of the form $f(k) \cdot |(x, k)|^{O(1)}$, where k is the parameter. The primary focus on these is due to their general tractability for small values of k [19]. It is worth noting that different parameters may be better choices for some problems, so studying a variety of parameters is important.

In this section, the discussion of parameters, will be done in general terms without the exact definitions. The exact definitions will be given in the later sections when they are used.

2.4.1 Known FPT Parameters

Solution Size Solution Size is a common parameter and was found to readily apply to many alliance problems. It describes the maximum size of the alliance. It is known to parameterize Defensive Alliance, along with both the Offensive Alliance and Powerful Alliance, including the global variants [20]. For Defensive Alliance, it uses a branching technique that considers the possible vertices that can be reached from a connected subset, which manages to bound the branching, as no vertex with a degree larger the $2k + 1$ has to be considered as it can not be part of the solution.

There is also a randomized algorithm for Exact Connected Locally Minimal Defensive Alliance, using a color-coding approach [21]. This works by assigning a random color to each vertex, checking if one of the colors has connected components of size at least k and is a Locally Minimal Defensive Alliance. Also for Locally Minimal Defensive Alliance is a result that shows it can be parameterized by solution size, in a graph with a minimum degree of 2 [21].

Vertex Cover Vertex Cover is a common parameter in Parameterized Complexity, which applies to many problems. A vertex cover is a set of vertices that make up at least one endpoint of every edge of the graph. This can then be used to bound the unique neighbourhoods outside the vertex cover, which is how this is commonly used for parameterized algorithms. Because every vertex outside the vertex cover only has edges to vertices inside the vertex cover, there are at most 2^{vc} unique edge patterns they can use. With this, only counts need to be assigned to each of the unique edge patterns to determine an alliance.

Vertex Cover has been used to designed $\mathcal{FPT}$ algorithms for both global and non-global variants of Defensive Alliance, Offensive Alliance and Powerful Alliance. These algorithms branch up to 3^k times based on if a vertex in the cover is part of the solution, some of its neighbours are, or neither of those cases. Each branch is then used to determine the starting conditions for an ILP with up to 2^k variables, which gets solved once for each branch [22].

Finally, Globally Minimal Defensive Alliance when parameterized by vertex cover is known not to admit a polynomial time kernel based on a reduction from Red Blue Dominating Set, which also has this property [14].

Neighbourhood Diversity A neighbourhood in a graph refers to the set of adjacent vertices a vertex has. Neighbourhood Diversity refers to the number of unique neighbourhoods in a graph. Two vertices a and b are said to share the same neighbourhood if their neighbourhoods are the same, with the exception of each other. That is, they satisfy $N(a)/\{b\} = N(b)/\{a\}$. This parameter allows these unique neighbourhoods to be treated as a single entity, considering only the number of vertices taken from a neighbourhood, which allows it be a viable parameter in many problems.

Defensive Alliance, Offensive Alliance, Locally Minimal Defensive Alliance and Globally Minimal Defensive Alliance are known to be FPT when parameterized by Neighbourhood Diversity. [23] [18] [14] Each of these uses an approach based around vertices in each neighbourhood being interchangeable with each other, and solving an ILP repeatedly based on this property. The number of times it needs to solve this ILP in the worst case varies per problem, but for Defensive Alliance this was 2^k times, Globally Minimal Defensive Alliance $3^k k^{3^k}$ times and for Locally Minimal Defensive Alliance 5^k times.

Domino Treewidth A tree-decomposition of a graph concerns a decomposition of a graph into sets, where the decomposition can be thought of as having a tree-like structure. A vertex can appear in multiple sets, and the size of the largest set is said to be the Tree-Width of the graph. Domino Tree Decomposition considers a variation of this where a vertex can appear in at most two of the sets at once, with the domino tree-width also being the size of the largest set [24]. It is worth noting that the Domino Treewidth of a graph is at most $(9k + 7)\Delta(\Delta + 1) - 1$, where k is the treewidth and Δ is the maximum degree [25].

It commonly occurs as a parameter throughout the literature and has been used to parameterized Defensive Alliance, Global Defensive Alliance [12] and Offensive Alliance [23]. For Global Defensive Alliance, it resulted in a $O(8^{dtw(G)}n)$ run time, where $dtw(G)$ is the domino treewidth of a graph [12]. The algorithm used for Defensive Alliance and Global Defensive Alliance was designed for graphs of bounded treewidth.

Domino Pathwidth Similar to Domino Treewidth, there are also results for Domino Pathwidth where For Global Defensive Alliance this gave an $O(4^{dpw(G)}n)$ run time, where $dpw(G)$ is the domino pathwidth of the graph [12].

Combinations of Parameters One successful tactic has been to apply multiple parameters, which works in cases where one might otherwise be in $W[1]$ or XP on its own. This has been successfully applied by Gaikwad et al. [21] for Locally Minimal Defensive Alliance where both the solution size and maximum degree were the parameters.

2.4.2 XP and W[1]-hard Parameters

Finally, outside of the previously discussed parameters, the remainder that have been studied either ended up being proven to be W[1]-hard or have had an algorithm in XP constructed. XP contains the W-Hierarchy, included $\mathcal{FPT}$, so there is a potential for improvements unless strict proofs were given.

Vertex Deletion Set An unusual parameter, but both Defensive Alliance and Globally Minimal Defensive Alliance have been shown that they are W[1]-hard by the size of the subset of vertices in a graph that need to be removed to leave only trees of specific sizes. In the case of Defensive Alliance this was shown to be trees of height at most 2 [11] and in Globally Minimal Defensive Alliance this was shown to be trees of height at most 3 [14].

Feedback Vertex Set Number The Feedback Vertex Set Number is the number of vertices that need to be removed from a graph such that the graph has no cycles. For it, three W[1]-hardness results have been obtained, covering Exact Defensive Alliance [16], Offensive Alliance and Exact Offensive Alliance [26].

Tree-depth When parameterized by Tree-depth both Defensive Alliance [16] and Offensive Alliance [26] are W[1]-hard, in their exact and non-exact variations. Further, Globally Minimal Defensive Alliance is also W[1]-hard [14].

Path-Width Like Tree-Width is a tree-decomposition, Path-Width is a decomposition in to a path. It has been shown that Defensive Alliance, Exact Defensive Alliance [16], Offensive Alliance, Exact Offensive Alliance [26] and Globally Minimal Defensive Alliance [14] are W[1]-hard with Path-Width.

Tree-width As previous mentioned, Tree-Width is a graph parameter that measures how similar a graph is to a tree. It has been used with Defensive Alliance and its exact variant, which were both shown to be W[1]-hard by Bliem et al. [27], along with Globally Minimal Defensive Alliance [14] and Exact Connected Locally Minimal Defensive Alliance [18] which were also W[1]-hard.

Outside of these, there were also XP algorithms proposed for graphs of bounded-treewidth in the case of Defensive Alliance and Offensive Alliance [23]. These algorithms have a running time of $O(2^k n^{2k+4})$ and $O(2^k n^{2k+6})$ respectively.

Clique-Width Clique-Width refers to the number of labels used when constructing a graph using several operations, and has been used as a parameter for several XP algorithms for the various problems. Both the global and the non-global variants of Defensive Alliance, Offensive Alliance and Powerful Alliance were shown to all have a running time of $O(n^{2^{3c+3}-7})$, where c is the clique-width [22]. These running times puts the algorithms into XP.

2.5 Open Problems

From reviewing the literature, several open areas were noticed, which informs the later sections and the research directions taken.

First, it was observed that there were only a limited number of variants considered, which informed the decision to study variants of the problem. There is a large body of work focused on specifically Defensive Alliance, Global Defensive Alliance and the Minimal Defensive Alliance variants. Outside of theses, the rest were focused on just minor properties such as being exactly a specified size or being connected. This led to the decision to focus on new variants of the problem, as an attempt to catalog when the existing approaches become viable for different problems.

A wide variety of Graph Parameters have been considered, but some authors mentioned various other novel parameters. These include Twin-Cover, Shrub-Depth and Modular Width as potentially interesting research directions, and the study of these form a significant part of this thesis.

Finally, many of the algorithms have complexities that make them impractical for real world usage, so there is interest in improving the bounds. Through this work, it was found that this was often difficult with limited success achieved, except in a few instances.

Continuing with the trend of practical applications, there was also a limited body of worked considering the experimental aspects of finding alliances. This is of interest as there are many areas where they have applications, as discussed in the motivation of this thesis.

3 Variants

In this section, variants of Defensive Alliance will be considered, with attempts to apply parameters that have been previously successful applied in literature. This is to give a general framework for working with a wide variety of problems that may be encountered in different fields where there is interest in a protected subset, for example logistics problems. To do this the existing algorithms for the parameters being used, or simpler formulations of them, will be discussed in detail, with attention paid towards specific properties they have and how they work with the variants being discussed.

This section is structured to first introduce the variants and their general complexity. After going through the variants, the different parameters will be worked through with results given. The focus of these results is to set out the general areas in which they can be applied, and also to give specific algorithms.

3.1 Problems

Several variants of Defensive Alliance will be discussed, with various forms of additional constraints applied.

3.1.1 Threshold Alliance

In this variant, the concept of a vertex being protected is extended in a way that is similar to the weighted variants. Instead of having more neighbours inside the Alliance than outside it, it is replaced with a unique per vertex threshold defined by the function $t : V \to \{0, ..., |V|\}$, such that the new requirement for being protected is now satisfying $|N_S(v)| \geq t(v)$. This is an extension of the normal definition of a defensive alliance because a defensive alliance can be directly modelled with this by defining the threshold for each vertex the minimal value of which would satisfy $|N_S(v)| \geq |N_{V/S}(v)| + r$. This also motivates the threshold definition of the ILP formulation.

An example of a instance of this problem is shown in Figure 2, which has an alliance consisting of four vertices.

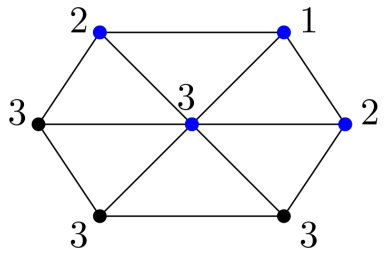


Figure 2: An example of Threshold Defensive Alliance, with the labels being the thresholds, and the blue vertices being part of a Alliance.

The problem can be defined as a decision problem, which will be used to discuss its general complexity.

Threshold Defensive Alliance
Input: A graph $G = (V, E)$, an integer k and a threshold function $t : V \to \{0, ..., |V|\}$
Question: Is there a set $S \subseteq V$ of size at most k, in which every vertex $v \in S$ is threshold-protected?

Theorem 3. Threshold Defensive Alliance *is $\mathcal{NP}$-complete on general graphs.*

Proof. This can be shown through a reduction from Defensive Alliance, which is known to be $\mathcal{NP}$-Complete. Threshold Defensive Alliance replaces the ratios with specific numbers for the number of neighbours that have to be in the alliance for a vertex to be included. For Defensive Alliance, the threshold for each vertex can be directly computed and used, reducing it to Threshold Defensive Alliance. The threshold of a vertex is defined as $t_v = \lceil \frac{|N(v)|-1}{2} \rceil$. □

Theorem 4. Threshold Defensive Alliance *is solvable in linear on complete graphs.*

Proof. The algorithm works by sorting the vertices by their thresholds and adding them sequentially until a protected subset is found or more than k vertices have been added. This is correct, as when

finding a solution of minimal size the vertices with the smallest thresholds will need to be in the solution as the vertices are otherwise interchangeable due to each vertex having the same neighbourhood.

Sorting can be done by placing each vertex into a bucket based on its threshold, of which there are at most n buckets. After this the algorithm sequentially goes through each bucket in order, taking vertices until satisfied. Hence, it can be solved in linear time. □

Theorem 5. Threshold Defensive Alliance *is solvable in $O(n)$ time on paths.*

Proof. In a path, there are three possible threshold values for a vertex that can be part of the solutions. These are zero, one and two. With these, there are three cases of solutions for the problem.

If the graph contains a single vertex with a threshold of zero, then this is the solution, which can be found in linear time.

Given this, it can be assumed that the graph consists of either vertices with a threshold of one or two. If two vertices with a threshold of one are adjacent, this will also be a solution and it can be found in linear time.

So, the only remaining case is regions of vertices with a threshold of two surrounded by ones with a threshold of one. In this case, the solution is the smallest region of vertices of threshold two, as the vertices with a threshold of one enclose these. Finding this is also a linear time operation thorugh a sequential scan by keeping note of the smallest region. The solution is the smallest region including the bounding vertices of degree 1.

□

Theorem 6. Threshold Defensive Alliance *is solvable in linear time on trees.*

Proof. The algorithm considers the cost of including a section of the tree depending on if the parent is included or not. To compute this, the algorithm starts at the leaves, progressing upwards up the tree until the root.

For a leaf to be included in the alliance, it must have a threshold of at most one, else it can not be protected. If it has a threshold of 0, it forms an alliance on its own and a solution has been found. So the only case for leaves that needs to be considered is if their parent is included.

For the rest of the vertices higher up in the tree, they need to decide which of the children get included. This can be computed by traversing down and taking the cost of them that includes the parent. The cost of this part of the tree without the parent vertex is found by taking the t_v children and summing their cost. And the cost for including the parent vertex is $t_v - 1$. If there are not enough children to protect this vertex, then this section of the tree can not form a solution without its parent.

At the root of the tree, there is no parent so only the lowest cost of including the root vertex is considered.

This process is then repeated over the whole tree, checking if any component of the tree has a cost below k when not including the parent, which results in each vertex being visited once. □

3.1.2 Rooted Alliances

The first variant of Rooted Alliances involves finding a connected Alliance that includes everything in a fixed subset.

Connected l-Rooted Threshold Defensive Alliance

Input: A graph $G = (V, E)$, an integer k, a fixed subset $L \subseteq V$ and a threshold function $t : V \to \{0, ..., |V|\}$

Question: Is there a connected set $S \subseteq V$ of size at most k, where all $L \subseteq S$, in which every vertex $v \in S$ protected?

The next variant is similar, but lacking the connectivity constraint.

l-Rooted Threshold Defensive Alliance

Input: A graph $G = (V, E)$, an integer k, a fixed subset $L \subseteq V$ and a threshold function $t : V \to \{0, ..., |V|\}$

Question: Is there a set $S \subseteq V$ of size at most k, where all $L \subseteq S$, in which every vertex $v \in S$ protected?

These problems can be thought of as finding the minimal set of external vertices that protects the rooted set and is protected by itself.

Theorem 7. *l*-Rooted Threshold Defensive Alliance *can be solved in* $O(n \log n)$ *time on complete graphs*

Proof. The algorithm works the same way as the algorithm used in the proof for Theorem 4, just with the rooted set added in initially. Vertices are sorted based on their threshold, and are repeatedly added until the problem is satisfied as they are interchangeable beyond their threshold. □

One property of l-Rooted Alliances, that is useful in the disconnected cases, is that a solution can be found by considering the smallest alliances that protect each of the 2^l combinations of the l vertices. These subsets can be considered together to check if there is a combination of them that stays below k without overlapping these subsets.

3.1.3 J-Subset Rooted Alliances

J-Subset Rooted Alliances describe an Alliance where there are a set of constraints. These constraints each consist of a set of vertices in the graph and a number j, the number of vertices that must be included from this set for the alliance to be valid. It often turns up as a sub-problem when developing algorithms for a parameter like the distance to a base graph. This is because it allows branching on vertices outside a core graph type and representing the requirements to protect these external components through the constraints. This can allow it to be used as part of a algorithm working with multiple parameters. Outside of this, it can be used as the base for problems with a location planning aspect.

Figure 3 shows an example with three constraints. If the thresholds (omitted in the diagram) are otherwise met, a sample solution could consist of the vertices $\{a, b, d, f\}$, or another subset that satisfies all the constraints. This is because it includes at least three from the first constraint ($\{a, b, d\}$), and one from the second and third ($\{d\}$ and $\{f\}$).

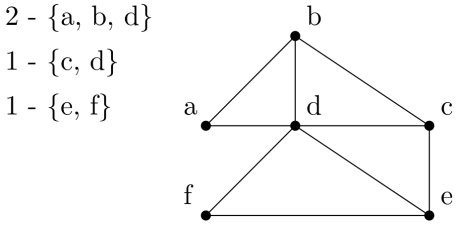


Figure 3: An example of a graph, and the respective constraints that make up an instance of j-Subset Rooted Threshold Alliance.

J-Subset constraints work trivially with the existing ILP formulation (see section 2.1.1, as they can be written down as requiring the sum of the variables representing the vertices in the constraint to exceed the j value. When using this formulation, care needs to be taken with the total number of constraints added if there is intent to use the algorithm alongside the existing ILP $\mathcal{FPT}$ formulations.

J-Subset Rooted Threshold Defensive Alliance

Input: A graph $G = (V, E)$, a set of constraints J, an integer k and a threshold function $t : V \to \{0, ..., |V|\}$

Question: Is there a set $S \subseteq V$ of size at most k, in which every vertex $v \in S$ is threshold-protected and the constraints in J are satisfied?

These reductions rely on using zero as the threshold for each vertex and using the subset aspect of the problem to implement Set Cover. Constraints are written in the form $j|\{a, b, c\}$, where j is the number of vertices from the subset that need to be included for it to be satisfied.

Theorem 8. J-Subset Rooted Threshold Defensive Alliances *can be verified in polynomial Time.*

Proof. This is trivially done by first verifying that for every $v \in S$ that they are threshold protected and iterating through each constraint to check if they are satisfied. □

Definition 9. *Set Cover involves a set of n elements (the universe) which must be covered using m sets in S. Each of member of S consist of some members of the elements, whose union covers the whole universe.*

The problem is to find the smallest subset of S such that covers the universe.

An example is shown in Figure 4, represented as a bipartite graph. Each of the top vertices the represent the sets, and the bottom vertices represents the universe.

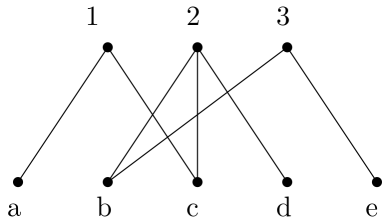


Figure 4: An example of Set Cover, represented as bipartite graph.

Lemma 10. *J-Subset Rooted Threshold Defensive Alliance is $\mathcal{NP}$-complete on graphs consisting of just disjoint single vertices.*

Proof. Set Cover can reduced into an instance of J-Subset Threshold Defensive Alliance by using a graph consisting of a disjoint set of single vertices. This is done by adding a vertex to the new graph for each set in S, and giving each of them a threshold of zero. Further, the universe is covered through the use of constraints of the form $1|\{a, .., n\}$ where only one in the subset has to be included for the constraint to be satisfied. For example, in the diagram the vertex labeled b has edges to vertices 2 and 3, so the constraint will be $1|\{2, 3\}$.

The correctness relies on several properties. First, every vertex that needs to be covered in the original problem is included in a constraint. Each constraint only needs one vertex it includes to be the alliance for the constraint to be satisfied, which directly models the original behaviour of Set Cover. The problem still consists of selecting vertices (that directly map to the original problem) to satisfy each of these constraints. As this is a disjoint set of vertices, each vertex must have a threshold of 0 for it to be included. Because of this, each vertex included can not influence if another vertex needs to be included for a valid solution to exist. So any solution that satisfies all the constraints can be directly translated to a solution to the original Set Cover instance.

Given Theorem 8, and the above reduction, the problem is $\mathcal{NP}$-Complete.

□

Theorem 11. *J-Subset Rooted Threshold Defensive Alliance is $\mathcal{NP}$-complete on any class of graph.*

Proof. This can be shown by reducing the problem for disjoint single vertices to any graph class, which due to Lemma 10 is known to be NP-complete.

In a graph of disjoint single vertices, the only valid threshold is zero as otherwise a vertex could not be included. This can be directly translated to general graphs, as with a threshold of zero the connections between vertices do not have any importance to the solution.

So any graph of size k can be used to represent an instance of Set Cover by giving each vertex a threshold of zero and by constructing the constraints in the same way. □

There is a useful result regarding J-Subset Rooted Threshold Defensive Alliance in that it can be parameterized by the amount of overlap between the constraints. In cases where l-Rooted Threshold Defensive Alliance is easy to solve, it is possible to use this to help solve instances of J-Subset Rooted Threshold Defensive Alliance. This applies in complete graphs, and disjoint sets of single vertices as these are both cases where l-Rooted Threshold Defensive Alliance is solvable in polynomial time.

Lemma 12. *J-Subset Rooted Threshold Defensive Alliance is equivalent to l-Rooted Threshold Defensive Alliance when the constraints share no overlapping vertices, on complete graphs.*

Proof. In cases where the vertices do not overlap on complete graphs, each constraint can be satisfied by just selecting the j vertices with the lowest threshold from the constraint. As there is no overlap, only the smallest set of vertices will be in the solution, which becomes the rooted set in a l-Rooted Threshold Defensive Alliance instance. □

Theorem 13. *There exists an $\mathcal{FPT}$ algorithm for J-Subset Rooted Threshold Alliance when parameterized by the number overlapping vertices in constraints m, on complete graphs.*

Proof. All 2^m instances of l-Rooted Threshold Defensive Alliances that include valid combinations of vertices from the constraints can be obtained. Each instance can be solved in polynomial time as shown in Lemma 12 for complete graphs.

The problem is discovering if there is a solution of size at most k, which this does correctly. This is due to there being 2^m possible solutions involving the overlapping vertices (including using none of them), of which all are tested. These cover every possible solution of size at most k, as each of the instances of l-Rooted Threshold Defensive Alliance will discover the smallest solution that can satisfy the constraints given the fixed vertices. □

3.1.4 Vertex and Edge Weights

Edge and Vertex weights are useful to consider as they can cover the cost of certain alliances. For example, a problem where a well defended set of bases are required where cost of building each of the bases is a concern.

Vertex-Budgeted Defensive Alliance
Input: A graph $G = (V, E)$ with vertex weights and an integer l
Question: Is there a defensive alliance $S \subseteq V$ with a cost of at most l?

Edge weights are worthwhile considering as they can help with problems that include things such as time to travel between locations, fuel costs or the maintenance costs of the edge.

Edge-Budgeted Defensive Alliance
Input: A graph $G = (V, E)$ with edge weights and an integer l
Question: Is there a defensive alliance $S \subseteq V$ where the internal edges do not exceed l?

Further interesting variants of the edge-budgeted alliances are ones that consider the spanning tree, or the cost of the shortest routes between each of the vertices in the alliance. These will problems will be discussed later in regards to the parameters.

3.1.5 Specific Properties

Finding a Defensive Alliance that also satisfies a secondary property is a useful basis for many problems, especially ones with a logistics focus. For example, consider an organisation that is trying to move a certain amount of cargo from one location to another but has concerns about the safety of the terrain and wants to be get support quickly from neighbouring locations. This can be modelled as a Defensive Alliance problem, for ensuring nearby support, with the additional requirement that it also forms a flow network to ensure there is enough capacity on the roads to transport the cargo.

Enumeration of Specific Properties It is often difficult to merge specific property checking into the algorithms, so if the algorithm has the capacity to support enumeration of all solutions the following result becomes useful:

Theorem 14. *Given an algorithm that enumerates all alliances up to size k in $\mathcal{FPT}$, and a polynomial time function $f(s)$ that checks if a property is satisfied, is is possible to combine these to find all alliances up to size k that satisfy the property in $\mathcal{FPT}$ time*

Proof. For each enumerated alliance s, they are evaluated with $f(s)$. If this is a yes-instance, the property will be true and a valid solution has been found. If no valid solutions are found, none of the alliances of up to size k satisfy the property and this is a no-instance. □

3.2 Solution Size

The known algorithms for Defensive Alliance that are parameterized by Solution Size rely on being able to limit how many vertices are explored due to protection requirements. If a vertex has more than $2k+1$ neighbours, it can not be part of the alliance as it would need more than the specific solution size to be protected.

Theorem 15. *[20]* Defensive Alliance *is $\mathcal{FPT}$ when parameterized by Solution Size.*

Remark 16. *The algorithm, which will be called Fernau's algorithm, starts from one vertex in the graph (and is repeated for each vertex in the graph). If this vertex has more than $2k+1$ neighbours, it can not be part of the alliance and thus needs to terminate. If the vertex has less than $2k+1$ neighbours, it*

branches on which of the neighbours are included in the alliance a recursively visiting them and reducing k by 1. This continues until either all the possible alliances of size k including the initial vertex are explored or an alliance is found.

During the recursion step, it considers all the neighbours of the constructed alliance not just the newly added one.

3.2.1 Thresholds

Two cases for thresholds will be considered that have different results associated with them, as they introduce complexity to the known cases because they break the connection between solution size and the maximum allowed degree of a vertex in the alliance.

Related Thresholds Thresholds that have a relation to the degree of the vertex are possible in certain cases, as with the default case, or something like having thresholds defined as the square root of the degree. The primary requirement is that the threshold has a maximum degree that will result in a specific value, to give a bound on the degree of a vertex that can exist in a solution for a given solution size.

Theorem 17. *Given a graph $G = (V, E)$, k the maximum the solution size, and a threshold function $t(v) \to \{0, ..., |V|\}$. If the threshold function $t(v)$ is related to the degree of v, such that it never returns a threshold above k for degrees larger than a constant value h, it is is possible to find a solution if one exists, in $\mathcal{FPT}$ time when parameterized by k.*

Proof. If all vertices with a threshold above k have a degree larger than h, this provides an upper bound on the maximum degree of a vertex than can be included in an alliance of at most size k. Given this, it is possible to branch and ignore any vertex with a degree larger than h as with the original algorithm (where h would be $2k + 1$). Thus placing it in $\mathcal{FPT}$. □

Arbitrary Thresholds Arbitrary thresholds, with no relation to the degree, do not directly work with this algorithm. This is due to the vertices potentially having a large number of neighbours but a low threshold, which could theoretically be part of a small solution.

In these cases, secondary parameters need to be considered, such as the maximum degree.

Theorem 18. Threshold Alliance *is in $\mathcal{FPT}$ when parameterized by both the maximum degree Δ and k the solution size.*

Proof. By parameterizing on both the maximum degree and the solution size, this bounds the number of vertices that can be explored from a single starting point in the graph. Given the bounds, the algorithm used in Theorem 15 can be directly applied, with potentially a lower bound on the complexity of the problem. As $2k + 1$ represents the maximum degree of a vertex that needs to be visited in the original algorithm, Δ directly replaces this. □

3.2.2 Rooted Alliances

An l-Rooted Connected Defensive Alliance can be dealt with using the basic algorithm, by only branching from one of the rooted vertices.

Theorem 19. l-Rooted Connected Defensive Alliance *paramaterized by solution size is in $\mathcal{FPT}$.*

Proof. In a connected l-Rooted Alliance, all vertices must be reachable from each other. So, by branching from one vertex using Fernau's Algorithm, it will discover the solution if one exists. □

Handling the case that allows disconnected solutions in $\mathcal{FPT}$ is harder, but is possible through a algorithm with a significantly higher complexity that depends on the size of the l-Rooted set. Depending on the definition of the problem used, each of the l vertices may be considered part of the k vertices in the solution or not. However, l is considered as a separate parameter here for the sake of clarity. For this, there are two approaches that are both included as they can both potentially be used as a starting point for further algorithms. The first finds the possible combinations that can be obtained for each combination of the 2^l members of the l-Rooted set, using this to form a larger solution. The second directly tackles the enumeration from the starting point.

Enumerating l-Rooted set combinations

Theorem 20. *l-Rooted Defensive Alliance is in $\mathcal{FPT}$ when parameterized by k the solution size and l the number of vertices in the rooted alliance.*

Proof. This algorithm discovers the lowest cost, in terms of extra vertices, that are needed to included combinations from the l-Rooted set. This is used to construct a table for each of the 2^l combinations of the l-Rooted set, which are searched to build the completed alliance at the end. This is done in this way to handle disconnected case.

Initially, the algorithm starts from each member of the l-Rooted set. As these enumerate all defensive alliances that include each of the l members of the set, the minimal size of an alliance required to protect all of the 2^l combinations is recorded in the table. So if one branch can reach vertices a and b, both of which are members of the the l-Rooted set, the table entry for those is updated if this is a lower cost solution.

After these have been fully enumerated, a solution that uses these to cover all l members for the minimal number of vertices outside of l needs to be found. It can be assumed that none of these enumerated solutions overlap, because if they did, they would exist as the optimal solution for the combination of these two solutions which would be the minimal combination for including them together, in which case it would be more optimal to include the combination set.

So given these, the algorithm then branches on which of the 2^l combinations are used. Invalid solutions are ones involving overlap or use more than k vertices in total. After the enumeration is complete, the optimal l-Rooted Defensive Alliance is found.

In total this algorithm will consider 2^{2^l} combinations, and perform l enumerations of all alliances starting from a specific vertex, which is an operation that is $\mathcal{FPT}$ purely by solution size. This places the algorithm in $\mathcal{FPT}$ when parameterized by both l and k. □

The approach used to apply the 2^l generated combinations is similar to another problem called Exact Cover but with added costs to each covering. Exact Cover will be discussed later on in more detail, with a focus on the parameterized verion of the problem.

Larger Initial Neighbourhood It is possible to consider the entire neighbourhood of the l-Rooted set as the starting point, instead of starting from each vertex individually.

Theorem 21. *l-Rooted Defensive Alliance is in $\mathcal{FPT}$ when parameterized by k the Solution size and l the number of vertices in the l-rooted set.*

Proof. Using the algorithm from Theorem 15, the algorithm can be modified to included the l-Rooted set from the start. In the worst case, each of the l vertices in the l-rooted set would cover completely different sections of the graph with no overlap with any of the others, resulting in at most l times as many possible solutions being considered. □

3.2.3 Property Checking

To check various properties, Theorem 14 can be relied on to find solutions for any problem where there is a polynomial time algorithm for checking it. This can apply towards Vertex and Edge weights, j-rooted connected subsets, spanning trees and various other properties. As an example, this will be applied to j-Rooted Connected Defensive Alliance.

Theorem 22. *J-Subset Connected Defensive Alliance is in $\mathcal{FPT}$ when parameterized by Solution Size.*

Proof. Given the algorithm used in the proof of Theorem 15 it is known that it is possible to enumerate all Defensive Alliances starting from a specific vertex. Further, in Theorem 8 it was shown that J-Subset Rooted Threshold Alliance could be verified in polynomial time. As J-Subset Rooted Threshold Alliance is a generalization of J-Subset Connected Defensive Alliance, with an algorithm that verifies only if the alliance is threshold protected and if all the J-subsets are satisfied, it can be directly applied to verify solutions for this problem. Finally, given Theorem 14, its known that J-Subset Connect Defensive Alliance is in $\mathcal{FPT}$ as it can both enumerate all solutions up to size k and verify them in polynomial time. □

For vertex and edge weights, there is a secondary concern if you are parameterizing purely on the number of vertices in the solution, or just the cost of the path. In the case of Vertex-Budgeted Defensive Alliance, as long each vertex has weights above 1 it can be directly solved using this approach. For Edge-Budgeted Defensive Alliance the same would apply as long as each edge also has weights above 1, as extending the alliance will always cause the cost to go up and force it to stop before exceeding k.

3.3 Vertex Cover

The vertex cover number is a common parameter, and has been previously used for Defensive Alliance and other problems, as discussed in the literature review.

Definition 23. *A vertex cover C of a graph G is a set of vertices where at at least one endpoint of every edge is included in C.*

Theorem 24. *Given a vertex cover C, then there are at most $|C| + 2^{|C|}$ neighbourhoods in the graph.*

Proof. Each member of C can have a unique neighbourhood, as it can be connected to any number of vertices inside or outside C.

For the vertices outside C, they can be connected only to members of C as they are only adjacent to members of the vertex cover. This limits the number of potential unique neighbourhoods for those outside C to $2^{|C|}$.

Combining these, there are at most $|C|+2^{|C|}$ neighbourhoods in a graph with a vertex cover of C. □

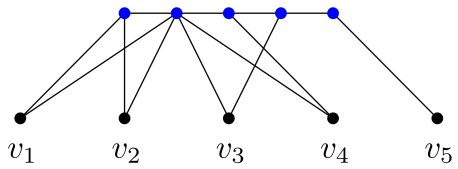


Figure 5: The blue vertices form vertex cover for the vertices in black, with v_1 and v_2 sharing the same neighbourhood.

3.3.1 Improvements to the Existing Algorithm

It is possible to reduce the complexity of the known algorithm by only requiring $2^{|C|}$ ILPs to be solved compared to $3^{|C|}$ ILPs used by Kiyomi et al. [22].

Lemma 25. *Given r-Defensive Alliance D, where $|D| > 1$, it must involve the vertex cover C.*

Proof. As at least one endpoint of every edge is in C, so every vertex must already be a part of C or have all of its neighbours in C. Thus, if a vertex is not protected by itself and it is not part of C, it must obtain protection from C to be part of the alliance. □

In this step, the objective is to determine a set of vertices outside the vertex cover that are needed to protect a given subset of the vertex cover.

Lemma 26. *Given a graph $G = (V, E)$, its vertex cover C and a subset $K \subset C$, it is possible to compute the set of vertices that are protected by K in polynomial time.*

Proof. This can be done via iterating through every $v \in V$, computing the threshold t_v for every vertex and counting how many neighbours v has in K. If $t_v \leq N_v(K)$, add to the list to return. As K can contain up to $|V|$ nodes, this algorithm has a running time of $O(|V|^2)$. □

Theorem 27. *Given a graph G and a vertex cover C of G, it is possible to decided if a Defensive Alliance of size at most k in $\mathcal{FPT}$ with at most $2^{|C|}$ evaluations of an ILP.*

Proof. The algorithm is based around handling two cases; one if the Alliance is of size one, and the other is if the alliance is larger than that.

Given Lemma 25 it is known that all Defensive Alliances must involve the vertex cover, or be a single vertex. The single vertex case can be handled by a linear search that computes the threshold of every

vertex. If one is found with a threshold of zero or less, this is a yes-instance. After performing this initial search, it is known that there are no vertices that could form an alliance on their own, and all alliances must now involve the vertex cover.

To handle Alliances larger than one, the algorithm will then branch on deciding if each of the vertices $v \in C$ belong to a fixed set K. This fixed set K is then used to build a list of vertices outside it, called D, that are protected by K. This was shown to be computable in polynomial time in Lemma 26.

Now the goal is to make sure K is protected using a subset of D. This done by solving an optimization instance of ILP is defined, which is to be solved for all possible values of K, with the objective of minimising the sum of all the variables used. This ILP has a variable for each unique neighbourhood in D, of which there are at most $2^{|K|}$ unique neighbourhoods. The bound on the number of unique neighbourhoods in D is due to each member being connected only to vertices in the vertex cover. These variables are constrained to be between 0 and the number of vertices in D sharing this neighbourhood. The ILP then has constraints added for each member of K that sums the neighbouring members in D, checking if they exceed the threshold. If members of K are adjacent, the respective thresholds are updated by subtracting one from their thresholds. This ILP is then solved for each of the branches, and if there is a solution with an objective value less than $k - |K|$ this is a yes-instance.

The branching will explore and construct up to $2^{|C|}$ candidates for K, involving equally as many ILPs. Given Theorem 2, ILP is known to be $\mathcal{FPT}$ when parameterized by the number of variables. The number of variables is fixed based on the vertex cover, the value of all variables and the objective function is fixed to the range $\{0, ..., n\}$ and the number of bits to represent the constraints being bounded on $|C|$.

The branching explores all possible combinations of how the vertex cover could interact with the alliance, as cases where the alliance does not involve C are handled seperately. The ILP will only be satisfied if all the members of the chosen set K are protected and as everything in its respective set D is already protected, this ensures only valid solutions can be returned by the ILP. □

3.3.2 Threshold Defensive Alliances

It is possible to apply the vertex cover algorithm to finding Threshold Defensive Alliances, with some small modifications. The primary modification is using the threshold directly in the algorithm for determining how many outside the vertex cover need to be included to protect one member of the vertex cover. The secondary modification looks at which vertices can be protected by the chosen members of the vertex cover, as that remains possible.

Theorem 28. Threshold Defensive Alliance *is in $\mathcal{FPT}$ when parameterized by the vertex cover number.*

Proof. Two modifications are made to the algorithm in Theorem 27. The generated ILP is modified to use the threshold attached to each member of the vertex cover, and not the computed threshold that comes from the alliance.

The second modification considers computing which vertices are adjacent to the chosen members of the vertex cover. To implement this, each vertex outside the vertex cover is enumerated and checked to see if it can be protected by the given vertex cover. Each neighbourhood is then bounded to the range between zero and the number of vertices in that neighbourhood that can be protected by the vertex cover. This allows the ILP to be used as there are still vertices that are interchangeable. So with these modifications, the algorithm can then proceed as normal. □

3.3.3 l-Rooted vertex cover

When considering an l-Rooted subset with vertex cover, the primary concern is if each vertex in the l-Rooted set is either in the vertex cover or outside it. When inside the vertex cover, it can be forced to include each of the l-Rooted sets when performing the branching step. Outside the vertex cover, there is a requirement that the shared neighbourhoods they are in include at least one of them, which can be adapted into the ILP.

Theorem 29. l-Rooted Defensive Alliance *is in $\mathcal{FPT}$ when parameterized by the vertex cover number.*

Proof. Using the algorithm from Theorem 27, vertices in the l-Rooted set that are also in the vertex cover can be forced to be included in the set K. This is allowed, as any valid solution using them would only occur for branches including them.

The branching can then continue as before, just with the requirement that the ILP is only solved when the remainder of the l-Rooted set appears in D. If they do not appear in D, this can not be a solution so there is no reason to solve the ILP. The ILP can then be solved with a few extra constraints on the variables generated from D. The constraints needed set lower bounds on the number of vertices included from the neighbourhood to be the number of vertices with that neighbourhood in the l-Rooted set. □

3.3.4 Other Problems

J-Subset Defensive Alliance Extending this to J-Subset Defensive Alliance has some difficulties, as it becomes harder to treat the vertices outside the vertex cover interchangeably with the constraints. As vertices contained in the constraints may be also in the same neighbourhood outside the vertex cover, it becomes no longer possible to represent them through simple constraints in the ILP.

Vertex and Edge Weights It is not directly possible to apply the known algorithm for vertex cover to cases involving Vertex or Edge weights. In both of these cases, each vertex outside of the vertex cover with the same neighbourhood can no longer be treated as interchangeable due to the possibility of each member having a different weight. If only the vertex cover is considered, it is possible.

Specific Properties The algorithm does not enable enumeration, and without an alternative parameter it is not possible to adapt it to perform enumeration and enable the checking of arbitrary properties.

Combinations l-Rooted Defensive Alliance and Threshold Defensive Alliance can be both be parameterized by vertex cover, and the combination of the two is a viable variant.

3.4 Neighbourhood Diversity

Definition 30. *Given a graph $G = (V, E)$, its* neighbourhood diversity *is the number of unique neighbourhoods in the G.*

Theorem 31 ([23], Theorem 1))**.** Defensive Alliance *is in $\mathcal{FPT}$ when parameterized by neighbourhood diversity.*

Theorem 32. *([28], Theorem 7) There exists an algorithm which runs in polynomial time and given a graph $G = (V, E)$ finds a neighborhood partition of the graph with minimum size.*

The algorithms for neighbourhood diversity treat each neighbourhood as something that has a count assigned to it, as each vertex in the neighbourhood is interchangeable. With this, it is then possible to branch on if vertices from a neighbourhood are included or not and assign counts by solving an ILP. This is how the algorithm provided by Gaikwad et al. [23] works for Defensive Alliance.

3.4.1 Thresholds

The first observation to note when applying neighbourhood diversity to Threshold Defensive Alliance is that it becomes difficult to treat vertices in the same neighbourhood as interchangeable. This is because vertices sharing the same neighbourhood can also have different thresholds. It is possible to treat vertices with the same neighbourhood and threshold as interchangeable, which is an option if these are closely related.

3.4.2 Rooted

l-Rooted constraints become easy to implement using neighbourhood diversity, as the only requirements needing to be added are those that enforce that one or more vertices are included from the neighbourhoods where one or more of the rooted vertices are. This can be implemented by assuming the neighbourhoods where the rooted vertices are in are always included, with the ILP being modified to enforce their counts as extra constraints.

Theorem 33. l-Rooted Defensive Alliance *is in $\mathcal{FPT}$ when parameterized by neighbourhood diversity.*

Proof. The algorithm in Theorem 31 can be extend to l-Rooted Defensive Alliance by adding additional constraints. In a neighbourhood that has p members in the l-Rooted set, then the new constraint requires that this neighbourhood will have at least p members included. If the branching finds an ILP that has a solution that satisfies this, this must be a yes-instance as all the members of the l-Rooted set are included and it is an alliance below the maximum required size. □

3.4.3 Other Problems

Vertex and Edge Weights Vertex and Edge Weights can not be directly solved using neighbourhood diversity as this removes the ability to treat the different vertices in the same neighbourhood as interchangeable.

Evaluating Properties This neighbourhood diversity algorithm does not directly provide a way of enumeration all alliances of a specific size, which makes the results relying on enumeration not feasible without using a secondary parameter. Using an algorithm that is parameterized by both solution size and neighbourhood diversity is an option and is tackled in Section 4.4.3.

Combinations As only the rooted variants were solvable directly, combining parts of each of the problems is not feasible.

4 New Parameters

In this section, various Graph Parameters that have not been previously used for Defensive Alliance problems will be studied. This includes some combinations of known parameters, a tactic that led to results in several cases.

The parameters being discussed include Distance to Clique, Solution Size, Modular Width and some problem specific parameters. With these parameters, some turned out to viable on their own for $\mathcal{FPT}$ algorithms, while others required the use of secondary parameters to enable the complexity to be bounded. A notable example of these is the section on Modular Width, which was found to be able to be naively placed in $\mathcal{XP}$ (and the further work lowers the upper bound on complexity to $w[2]$) but considering the target solution size as well brings the problem into $\mathcal{FPT}$. Secondary Parameters often allow the removal of an ILP instance, and through that it can be adapted into enumeration algorithms which can be used for property checking in certain circumstances.

Solution Size was discussed in the previous section, but in this case it was applied to show Globally Minimal Defensive Alliance is in fact para-$\mathcal{NP}$. Given the known result that Locally Minimal Defensive Alliance was $\mathcal{FPT}$ for graphs with a degree larger than 2, this was unexpected. The reduction for Globally Minimal Defensive Alliance does require vertices of degree one, which may partially explain this and give ideas for future work.

4.1 Distance To Clique

Defensive Alliance can be shown to be parameterized by its *Distance to Clique* due to this parameter giving a bound on the graphs Neighbourhood Diversity. As Defensive Alliance when parameterized by Neighbourhood Diversity is known to be in $\mathcal{FPT}$ [23], this results in this parameter also being in $\mathcal{FPT}$.

4.1.1 Computing Distance to Clique

Computing the distance to clique can be initially thought of as being difficult, as the parameter involves finding a clique of size $n-k$. However, it is a special case of Weighted d-Cluster Vertex Deletion [29], where the goal is to delete k vertices such that the graph results in a d-cluster graph. In this case, a 1-cluster graph is wanted. Weighted d-Cluster Vertex Deletion is in $\mathcal{FPT}$, so this case is handled.

4.1.2 Core Algorithm

Definition 34. *Distance to Clique[[30]] Given a graph G, its distance to clique is the minimum number of vertices that need to be removed from G to obtain a clique.*

Lemma 35. *A graph that is at most k vertices from a complete graphs has at most $k + 2^k$ unique neighbourhoods.*

Proof. There are two cases that need to be considered, which are if a vertex is outside the clique or if it is otherwise inside the clique. In the first case, each of the k vertices outside the clique can each have a unique neighbourhood.

In the second case, there is a limited number of unique neighbourhoods bounded on k. Each vertex in the clique is connected to each other, with some connections to the k vertices outside. A vertex can have at most k edges outside the clique, with the total combination of possible neighbourhoods being 2^k. If there are more than 2^k vertices inside the clique, then there have to be repeated neighbourhoods, resulting in the bounds.

Given this, there can be at most $k + 2^k$ unique neighbourhoods in a graph. □

Theorem 36. Defensive Alliance *is in $\mathcal{FPT}$ when parameterized by distance to clique.*

Proof. Given Theorem 31 it is known that the problem is bounded by the number of neighbourhoods in a graph. It was shown in Lemma 35 that there can be at most $k + 2^k$ neighbourhoods based on the graphs distance to clique, and it is possible for it to compute the real value in polynomial time given Theorem 32.

Given this, Defensive Alliance must have an FPT algorithm for distance to clique as the number of neighbourhoods is bounded on the parameter and the total number of neighbourhoods can be computed in Polynomial time, and a parameterized algorithm exists for neighbourhood diversity. □

4.1.3 Bounded Degree

A faster algorithm is possible when bounding the maximum degree for vertices outside the clique. If the degree of the clique of members of the clique are bounded, this just results in a smaller clique, but the result still applies if only the vertices outside the clique have their degree bounded.

This relies on the idea that vertices can be stripped from a graph to obtain an instance of J-Subset Rooted Threshold Defensive Alliance, which is easy to solve if the original graph has a bounded degree for the vertices outside the clique. k will refer to the distance to clique, and d for the maximum degree of the graph.

Lemma 37. *Given a graph with a distance to clique of k, fixing whether each vertex outside the clique is included in the Defensive Alliance or not gives an instance of J-Subset Rooted Threshold Alliance on a complete graph.*

Proof. The clique will act as the complete graph, with the vertices outside informing constraints.

If every vertex outside the clique is fixed, it will need a certain amount of connections in the clique (if possible) for that vertex to be included in an alliance. Given this, it can be modelled by a constraint consisting of the number of neighbours needed to make it protected with each of the neighbours. If this constraint is not satisfied, the vertex can not be part of the alliance which violates the requirement that it is included in the solution. □

Theorem 38. Defensive Alliance *parameterized by k, the distance to clique, and d, the maximum degree of a vertices outside the clique, admits an $\mathcal{FPT}$ algorithm.*

Proof. An instance of j-Subset Rooted Threshold Alliance can be obtained by fixing vertices outside the Clique due to Lemma 37. By branching on which vertices are fixed, this gives 2^k instances of j-Subset Rooted Threshold Alliance.

As the degree is bounded, possible overlap is limited in the clique, as each of the k vertices outside the clique can only be connected to d vertices in the clique, which is the maximum overlap a single vertex can add. Given Theorem 13, this must be in $\mathcal{FPT}$ as the maximum number of overlapping vertices is dk. This results in a total $O(2^k 2^{dk} n \log n)$ running time.

□

4.2 GMDA and Solution Size

Solution size has already been discussed in depth for some variants, but this section covers it when applied to to a problem called Globally Minimal Defensive Alliance, where the goal is to find an alliance that has no smaller alliances contained inside it, with this alliance having at least k vertices. This problem is different from the previously discussed variants due to the "at least" constraint, which significantly affects the complexity.

4.2.1 Exactly k

Before considering the at least constraint, it is worthwhile to note that the existing approach for enumerating Alliances with Solution Size can be applied to find Globally Minimal Defensive Alliances of exactly k vertices. This is done by using an existing algorithm that checks if a Defensive Alliance is minimal or not, provided by Bazgan et al. [8].

Proposition 39 ([8])**.** *There is a polynomial-time algorithm to determine whether a Defensive Alliance S is globally minimal.*

Remark 40. *The algorithm given by Bazgan et al. [8] written as psuedo-code is as follows:*

```
for v ∈ S do
    S' ← S \ v
    J ← unprotected vertices ∈ S'
    while |J| > 0 do
        S' ← S' \ J
        J ← unprotected vertices ∈ S'
    end while
    if |S'| > 0 then
        return False
```

end if
end for
return *true*

The idea behind this algorithm is that after removing one vertex, if any others become unprotected they can not continue to be part of a solution, and if by performing this process you get an empty graph for every vertex this is minimal. There should not be a case where removing two or more vertices initially is required to discover if a defensive alliance is contained inside S.

To give a worst case upper bound for the above process, as one was not previously given, it is assumed that checking if a vertex is in S and checking the degree of a vertex are both $O(1)$ time operations. Given those, it can be seen that:

- *The procedure contained inside the for-loop will run at most $|S|$ times.*
- *As the inner while loop will either remove at least one vertex or terminate, it is known that it is only necessary to search for unprotected vertices at most $|S|$ times.*
- *Searching for unprotected vertices involves visiting each vertex, and checking if each of the neighbors are in S. For each vertex, at most $2|S|-1$ other vertices need to be considered (otherwise it can be immediately rejected). So this step can consider at most $2|S|^2-|S|$ vertices.*

Considering this, it can be seen that the algorithm in this form will take at most $O(|S|^4)$ operations.

As has been confirmed it is possible to verify that an alliance has been found, two other checks are needed. These are confirming that all possible Alliances are connected, and that the degree constraint is satisfied.

Lemma 41. *A Globally Minimal Defensive Alliance is always connected.*

Proof. For the disconnected components to be part of a defensive alliance, they must be defensive alliances by themselves. This is the case as each of the disconnect components will have no connections to the rest of S and be entirely self sufficient in their components.

As the definition of a Globally Minimal Defensive Alliance excludes any proper subsets from being defensive alliances, allowing disconnected components would be a contradiction. □

Lemma 42. *A Globally Minimal Defensive Alliance of size k can not include a vertex with a degree greater than $2k$.*

Proof. If a vertex with a degree higher than $2k$ is included in the Defensive Alliance, then there are not enough vertices inside the defensive alliance for this vertex to be included. E.g, including a vertex of degree $2k+1$ would result in it being impossible to protect the vertex. □

Finally, this can all be brought together to share that searching for a Globally Minimal Alliance of a specific size is in $\mathcal{FPT}$, which will be called Exact Globally Minimal Defensive Alliance.

Theorem 43. *The* Exact Globally Minimal Defensive Alliance *is fixed-parameter tractable when parameterized by the solution size k.*

Proof. By using the algorithm used by Fernau et al. [20], the validity check can be replaced with the one provided by Bazgan et al. [8]. Fernau's algorithm was discussed in section 3.2, where it is explores connected sections of a graph by visiting vertices that do not require more than $2k+1$ neighbours to be part of the alliance.

It is possible to make this adaption due to several properties. All Globally Minimal Defensive Alliances are connected, as shown in Lemma 41, and the degree requirement of Fernau's algorithm holds as shown in Lemma 42. The algorithm will enumerate all alliances up to a specific size k, and the algorithm by Bazgan given in Proposition 39 can be used to validate each possilbe alliance. □

4.2.2 Globally Minimal Defensive Alliance is para-$\mathcal{NP}$

This will be shown using a reduction from X3SAT which is a known $\mathcal{NP}$-Complete problem.

Definition 44. *X3SAT, or Exact-1-3 SAT, involves finding a set of truth assignments for literals used in clauses of size 3. Each clause has one and only one true literal.*

Theorem 45 ([31])**.** *X3SAT is $\mathcal{NP}$-Complete.*

Theorem 46. *For $k = 1$, there is always a* Globally Minimal Defensive Alliance *of size at least k.*

Proof. As there is always an Defensive Alliance for $r = -1$, one that includes every vertex in the graph, there must always be a Globally Minimal Defensive Alliance of size 1 or more. □

Theorem 47. *For $k = 2$,* Globally Minimal Defensive Alliance *of size at least k can be solved in polynomial time.*

Proof. If a minimal alliance of size larger than 1 is to exist, it must not include any vertices which form an alliance on their own, as then it would no longer be a minimal alliance. Alliances of size 1 can be identified by searching the graph for vertices that are protected by themselves, which is every vertex with a degree of at most 1.

By starting with an alliance that includes all vertices, with the exception of the ones which already form an alliance on their own, and repeatedly removing all unprotected vertices, it can be seen that if a subset remains protected after this process then there must be an minimal alliance of at least size 2. So, if after removing all unprotected vertices some still remain, this is a yes instance, otherwise it is a no instance. □

Theorem 48. Globally Minimal Defensive Alliance *when parameterized by solution size is para-$\mathcal{NP}$-Complete for any fixed $k \geq 3$*

Proof. This is shown by using a reduction from X3SAT, where a minimal alliance S of size $|S| \geq k$ exists if and only if the core X3SAT problem has a solution. X3SAT is converted into an instance of Globally Minimum Defensive Alliance by mapping clauses and variables directly into gadgets representing them. Given Theorem 45, X3SAT is known to be $\mathcal{NP}$-Complete so Globally Minimal Defensive Alliance must be as well given a reduction of X3SAT to it.

For this reduction, a combination of gadgets are used. These gadgets have a set of defined points (referred to as *inputs*) where they can be connected to other parts of the graph, and if these inputs form part of an alliance, then parts of this component can be included as part of the alliance subject to certain gadget specific constraints. These gadgets are constructed by using alliances of sizes one and two in them, which are used to enforce these properties by making violations introduce a minimal alliance of those sizes, thus making the whole alliance non-minimal. Further, they are designed to be combined without violating their properties.

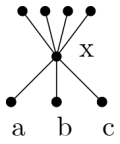


Figure 6: AND Gadget

The first and most fundamental gadget is the AND gadget, shown in Figure 6, which has a certain number of vertices that form alliances on their own to balance out the inputs. The key property is that all the inputs must be included for the vertex labeled x to be included in the alliance without including one of the balancing vertices. A n input AND gadget can be constructed by adding $n + 1$ balancing vertices for $n \geq 1$.

The second gadget, called the XOR gadget and shown in Figure 7, keeps two paths from existing together. This is implemented by introducing the condition that if x and y are included in together in an alliance, they form a minimal alliance of size two. This leaves the only allowed minimum alliances of size larger than two to be $\{a, b, c, d, x\}$ and $\{e, f, g, h, y\}$. The middle vertices can be included as long as the ends are included. So $\{a, d\}$ and $\{e, h\}$ are the two pairs of inputs that can be included at the same time. As an example, including $\{a, b\}$ in an alliance requires $\{x, c\}$ to also be included, as otherwise b is unprotected. This continues further for c, which requires $\{b, d, x\}$ to be included for it to be protected.

The base gadget, shown in Figure 8, is designed to only be included if both C and V are included, which is done by a sequence of AND gadgets. Each of the unlabeled vertices need at least two neighbours to be in the alliance, else they would be including a non-minimal alliance, which they can only get from C and V.

The variable gadget, shown in Figure 9, splits each variable into two valid paths. It is implemented with an inner XOR gadget, which keeps both a and $\neg a$ from both being part of the alliance at the same

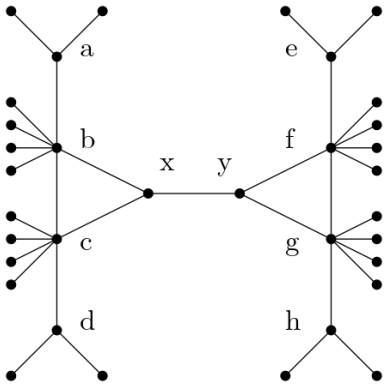


Figure 7: XOR Gadget

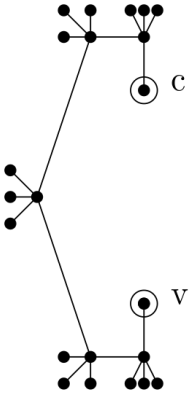


Figure 8: Base Gadget

time. Further, there is the input i which must be included in a solution for it to be part of an alliance. So this results in the only valid paths that do not include a smaller alliance involving either $\{a, i\}$ or $\{\neg a, i\}$ but not both. This gadget is connected to the next one, to implement the clauses, and i connected to the base to enforce the variables inclusion in any solution.

The clause gadget, shown in Figure 10, combines multiple XOR gadgets in a sequence to ensure that the only minimal alliances that can exist include either $\{a, o\}$, $\{b, o\}$ or $\{c, o\}$ but never combinations of them.

Finally, the reduction algorithm uses these gadgets to convert an X3SAT instance by first adding one base gadget. Then, it creates a copy of the variable gadget for each variable, and a clause gadget for each clause. Each variable is then connected to a clause gadget it is used in, with balancing vertices being added to ensure all the connections are enforced. If a variable or its negation are not used, vertices are added to just balance it. Next, each clause gadget is connected from their input labeled o to the base at the input labeled C, with an AND gadget to ensure every clause and the base is required for C to be included. Finally, each variable gadget is connected at their vertex labeled i to V in the base gadget, using the AND gadget to balance out and ensure each variable is included. This final structure is shown in Figure 11.

This reduction is clearly in polynomial time.

The correctness of this relies on how each gadget propagates an alliance through it, which is used as a loop to create the minimal alliance. Starting from the base, it is known that C and V must both be included for it to be part of an alliance. Then, as each variable exposes i, which must be included for each variable for V to be included. Given i for each variable must be included, either their negation or non-negation must be included. For C to be included, each of the clauses must have at least one true input literal as otherwise a smaller alliance was included via one of its balancing vertices. This finally connects the loop, ensuring a loop around the base and that every variable and clause is included. □

A example of the reduction is shown in Figure 11

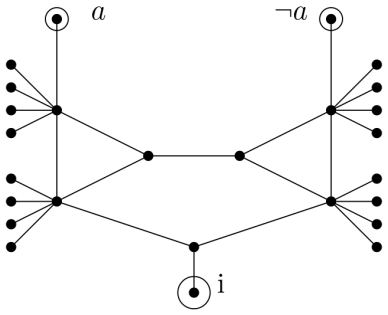


Figure 9: Variable Gadget

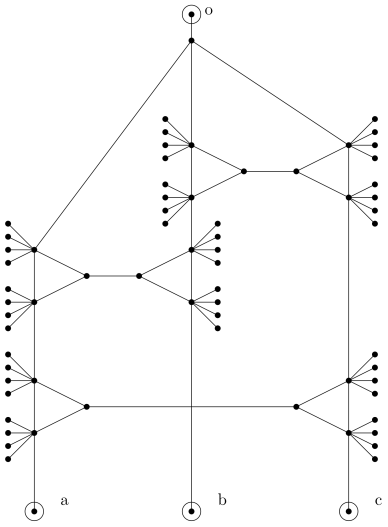


Figure 10: Clause Gadget

4.3 Modular Width

4.3.1 Definitions

Definition 49 ([32]). *A* modular decomposition *of a Graph is one formed from the following operations:*

- *O1 - Create an isolated vertex.*
- *O2 - The disjoint union of 2 graphs.*
- *O3 - The complete join of 2 graphs.*
- *O4 - The substituting of the vertices of one template graph G, where each vertex is substituted with one of the operand graphs. If the template graph had an edge between two vertices, the replaced operand graphs are subject to a complete join between their vertices.*

Definition 50 ([32]). *The* modular width *of a graph G is the largest number of operands graphs used by an operation in the modular decomposition.*

The modular decomposition can be constructed in linear time [33], and each operation with the exception of O1 can be emulated through O4.

4.3.2 Algorithm

The algorithm for modular width works by solving a sub-problem, which in this instance will be called *Table-Merge*, for each component of the Modular Decomposition. Table-Merge is used to keep track of how many external vertices are needed to protect a module in the decomposition. This subproblem is solved from the leafs upwards in the decomposition, which allows the construction of tables which state how many vertices can be taken from parts of the decomposition and how much protection they need for other sections to form part of an alliance.

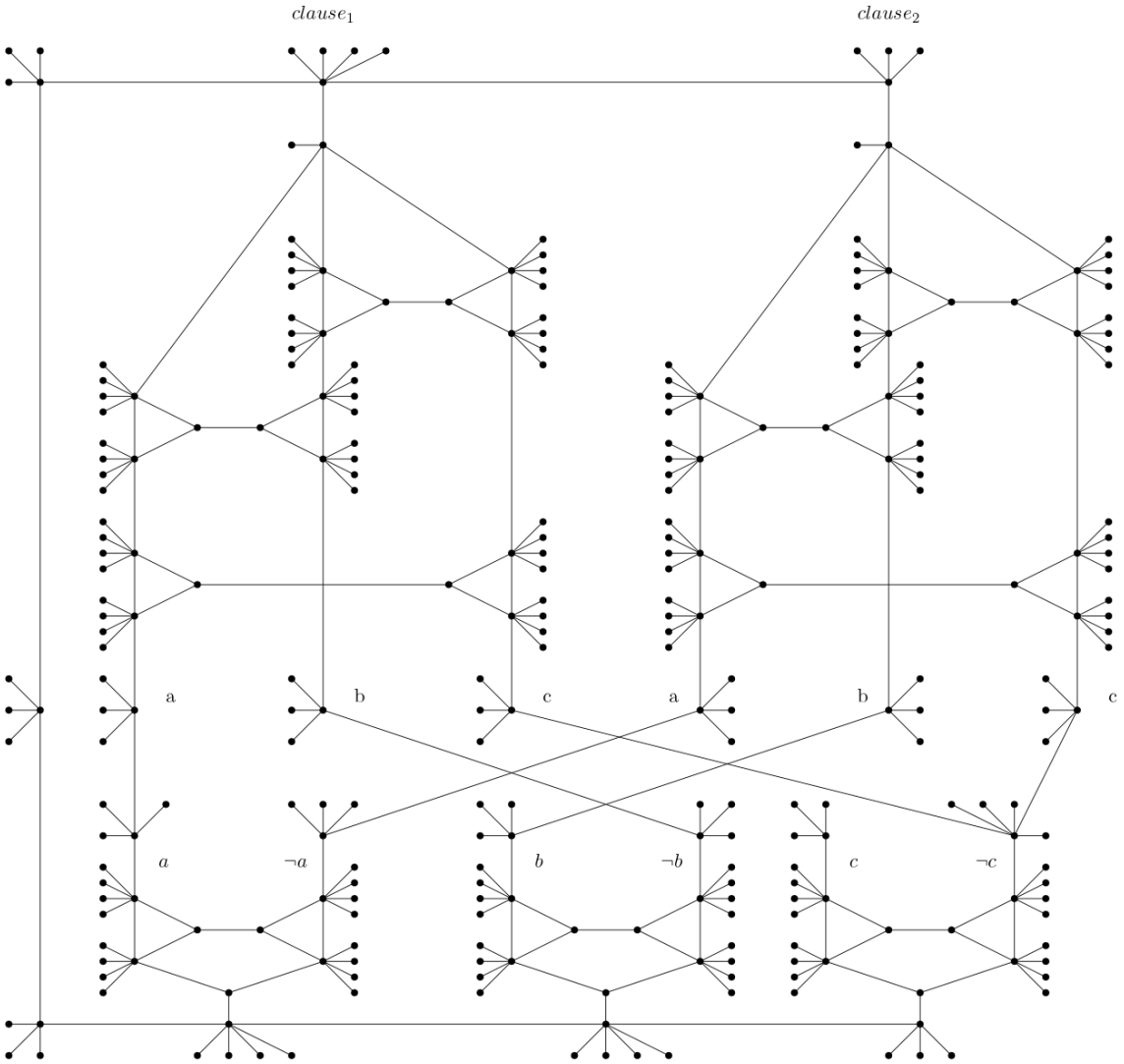


Figure 11: A full example of the X3SAT to GMDA reduction, where there are two clauses and three variables. A valid solutions is $\{a, b\}$

Definition 51. *Table-Merge Given k tables, where each row in the table consists of a vertex count, and the number of vertices that vertex count needs to be protected,t he goal is to form a new table merging the k tables, where the lowest number of required vertices is stored for each vertex count.*

Each table being considered has rows for every value of the vertex count from between 0 to its total number of vertices, with no gaps.

Theorem 52. *Table-Merge can be solved in $O(n^k k^2)$ time, where k is the number of tables.*

Proof. To construct the merged table, for each value $l \in \{1, ..., n\}$ every combination of values used in operand tables that sums up to l is tried, recording the smallest total number of external vertices needed. Some care is needed to correctly compute the number of external vertices needed for a combination, due to the count choice for each operand influencing how protected its neighbours are. To do this correctly, each vertex in the template graph takes the current minimum threshold for the vertex count it was given, and subtracts its neighbours counts from it. This is repeated for every vertex in the template graph, with the maximum value being the total number of external vertices needed.

Testing a combination can be done in $O(k^2)$ operations, assuming constant time lookup for table rows. Merging a single row can take n^k combinations of values, so this needs to be repeated n times to complete construction of a new table. □

Theorem 53. Defensive Alliance *when parameterized by Modular Width it is in $\mathcal{XP}$.*

Proof. For every leaf of the modular decomposition tree, a single vertex is defined with each getting an initial table. These tables initially consist of a row consisting of two columns, one for a vertex count and the number of neighbours outside this component needed for it to be protected. In the cases of the single vertex, the row will have the first column being one, and the second being the threshold of the vertex it contains.

As the decomposition is traversed upwards from the leaves, these tables are merged using Table-Merge to determine the minimum number of external vertices needed to use each possible value of the vertices from the component of the decomposition. If these tables always contain the minimum number of external vertices for each component, it can be seen that if the size of an alliance is less than or equal to the maximum solution size, then there will be a column of that size that requires zero external vertices. So given a suitable merging operation, these tables can be used to find if a solution of the required sizes exist.

This gives a total time complexity of $O(m^2kn^{\mathcal{O}(m)})$ given the complexity of Theorem 52. □

Theorem 54. *Table-Merge can be solved in $O(l^k k^2)$ time, where k is the number of tables and l is the solution size.*

Proof. The tables contain how many vertices a particular count of vertices needs to be protected. As this is searching for a specific solution size l, it will never need more than l entries in the table, stopping the tables from growing with n.

This gives the ultimate running time, as only l^k solutions need to be considered in this case. □

Theorem 55. Defensive Alliance *when parameterized by Modular Width and Solution Size is in $\mathcal{FPT}$.*

Proof. Given Theorem 54, it is known that Table-Merge is in $\mathcal{FPT}$ if it is parameterized by both parameters. Then it follows the same algorithm used in Theorem 53 can be directly applied with this new function used instead of the original $\mathcal{XP}$ algorithm. □

4.3.3 Upper-bounding the Complexity

Now the next few results will provide an upper bound that this algorithm is in $W[1]$ by reducing Table-Merge to k-Exact Cover, without the secondary constraint of a bounded solution size.

k-Exact-Cover The upper-bound on complexity will be shown through a problem called k-Exact-Cover, which involves covering a set exactly by selecting k subsets. The complexity of k-Exact-Cover will be shown to be $W[1] - complete$ via a reduction to another problem called k-Perfect Code

Definition 56. *Given universe of symbols, a set of sets containing symbols from the universe, the goal of k-Exact-Cover is to find exactly k sets that cover the universe with no overlap between the sets.*

To give the complexity of k-Exact-Cover, Exact Unique Hitting Set is used.

Definition 57 (k-Exact Unique Hitting Set [34, p .589])**.** *Given a universe U, a set $\mathcal{A}$ of subsets of U, and an integer k. Is there a set $X \subseteq U$ of size exactly k such that $|A \cap X| = 1$ for every $A \in \mathcal{A}$.*

Theorem 58 ([34, p .453–456])**.** *k-Exact Unique Hitting Set is $W[1]$-Complete.*

To show that k-Exact-Cover is in W[1], a reduction to k-Exact Unique Hitting Set is performed. These problem map to each other in the same way Set Cover and Hitting Set do.

Theorem 59. *k-Exact-Cover is W[1]-complete.*

Proof. k-Exact Cover can be reduced to Exact Unique Hitting Set via making the covers in k-Exact Cover the universe in Exact Unique Hitting Set. Then each member of the universe in the original problem becomes members of $\mathcal{A}$ where their sets becomes the set of covers they intersect with.

k directly maps to finding the same covers between the two problems, and each $A \in \mathcal{A}$ will only be included in one of the covers. □

A useful result to consider is how the difficulty of k-Exact-Cover varies depending on the number of overlapping sets.

Theorem 60. *k-Exact-Cover is in $\mathcal{FPT}$ when parameterized by the overlapping pairs of sets O.*

Proof. In k-Exact-Cover, any set that covers a value that no other set covers must be in the solution. So the values included in these sets can be removed, and any overlapping sets can be removed from further consideration. If no overlapping sets remain and everything is covered, a valid solution has been found.

Otherwise, a simple algorithm can be created by branching on if each member of O is included or not. If any of these branches results in a valid solution, this is a yes-instance. □

Simpler Variant of Table-Merge To make the next result easier to understand, a simpler version of Table-Merge will be considered to introduce the concepts used in the core problem. This variant is similar to Subset Sum, but with k sets, of which one value is chosen from each set such that these chosen values sum up to a target value. One constraint is that this uses a unary encoding in the reduction, which affects the number of sets in the problem.

k-Set Sum
Input: k sets of l values each, where each value is in $\mathbb{Z}^+$ and $t \in \mathbb{Z}^+$ the target sum.
Question: Can one value be taken from each of the k sets that sums up to t?

To demonstrate the complexity, Subset Sum will be reduced to k-Set Sum.

Definition 61. *Given a set S of integers and a target value T, the goal of Subset Sum is decide if a subset of S sums up to T.*

Theorem 62. *k-Set Sum is $\mathcal{NP}$-hard.*

Proof. The difficulty can be shown via a reduction from Subset Sum, which is a known $\mathcal{NP}$-hard problem.

Given a Subset Sum instance of T the target value and S of the integers, the reduction takes every value $s \in S$ and creates a new set $\{0, s\}$. So the choice becomes either using s or not. The new instance of k-Set Sum consists of these $|S|$ sets and the Target value. The correctness is based on deciding for each s whether or not it is included, which is modelled by the new sets that consist of s and 0. □

Theorem 63. *k-Set Sum can be converted to an instance of k-Exact-Cover in polynomial time.*

Proof. Consider a bit string with $t + k$ slots, with the objective of finding a covering of these $t + k$ slots. The first t slots are used by the coverings as the core of the problem, trying to find values that cover only those exact slots. The final k slots are assigned to each of the k variables, to enforce the constraint that one and only one number is taken from each set.

To cover the slots, coverings are generated for each of the numbers in the sets, which will get placed sequentially over the slots. So the covering chosen from the first set is placed directly before the second set in the grid.

The coverings for the slots are generated by finding the range of starting slot locations. This is found by looking at the minimum and maximum starting locations for the previous sets. Next, a covering is generated for each of the l values at each of the possible starting position for this value. These coverings cover the number of slots that correspond to the original variables value and one of the final k slots that represents the index of the set being considered.

If starting slot location forces any of the coverings to go beyond t, they will not be included as part of coverings considered as these can not form part of a valid solution.

Given t possible starting points for each variable, there will be t^k possible positions as an upper-bound. □

Table-Merge This is the final step to demonstrate that Defensive Alliance when parameterized by Modular-Width is in $W[1]$, where a complex conversion is done to convert into an set of instances of k-Exact-Cover. This reduction is for solving one row of the table at a time, where it is solved for every vertex count, and from the bottom up for the required number of vertices for protection.

Theorem 64. *Finding if there is set of values for a specific vertex count and protection in Table-Merge can be done by solving an instance of k-Exact-Cover.*

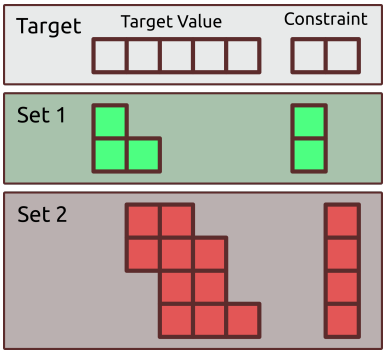


Figure 12: Visualization of the k-Set Sum reduction to k-Exact Cover, where there are two sets trying to reach a target value of 5. Each row is a new covering.

Proof. This reduction works by reducing the problem to k-Exact-Cover, where the problem is represented by a grid and a set of covers. This grid consists of two primary sections, the target component and the constraint component, each with two sub components for enforcing inclusion and sections summing to the correct value. It relies on the maximum target-value of Table-Merge being dependent on n in the original problem to bound the number of covers generated and the size of the grid.

An example of the grid being covered is shown in Figure 15, which represents the graph shown in Figure 13 with the tables shown in Figure 14. With the target sum of 2 and a maximum needed protection of 2, this problem has three solutions which are $\{a = 1, b = 0, c = 1\}$, $\{a = 2, b = 0, c = 0\}$ and $\{a = 0, b = 0, c = 2\}$.

The general structure of the reduction is that for each variable a target cover is chosen. This target cover restricts the covers that can be used in the constraint rows, which are used to ensure the vertex gets the required amount of protection.

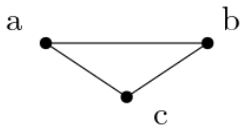


Figure 13: The fully connected graph used in this example.

0	0
1	1
2	2

(a)

0	0
1	4
2	3

(b)

0	0
1	3
2	2

(c)

Figure 14: Respective tables for each of the vertices for the graph.

Constraint rows consist of several components. Each variable that is adjacent in the graph to the variable that the constraint row was created for gets a selection section. These selection sections get filled up partly by the target covers which are used to stop mutually exclusive covers from being used at the same time. For example, three values for the same variable being used in a constraint row is not possible as otherwise they would overlap with either each other or the target cover. If no target cover was chosen for this variable, its inclusion cell would not be covered, so it would not otherwise be a solution.

There are two types of covers for the constraint rows, both with the same structure. Covers are generated for each constraint row, with no overlap between the different rows. The first type is used to sum up the neighbours, working exactly the same as the target section, just covering the selection value for the given count. Figure 17 demonstrates this for the first constraint row, with the covers for every value b and c can take.

The second type is referred to as filler covers, which are used to ensure the neighbours provide enough protection. These covers exists to fill up the remaining space on a constraint row, by filling it up from the right of the sum component of the row. So given a specific value for the variable this constraint row represents, it covers that cover between 0 to n minus the needed protection. This way, if the neighbours provide enough protection for the value, this fills up the remaining cells allowing it to be a valid solution.

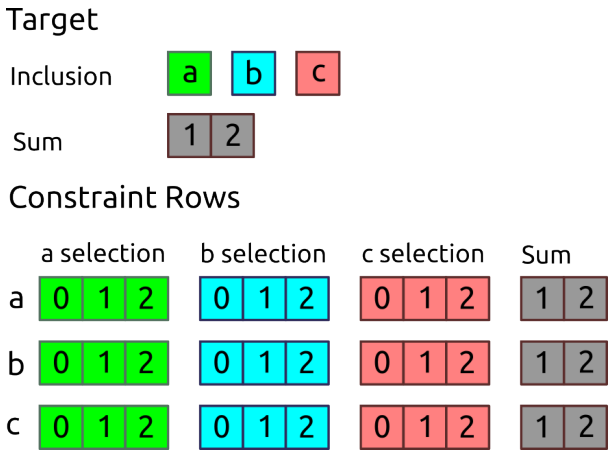
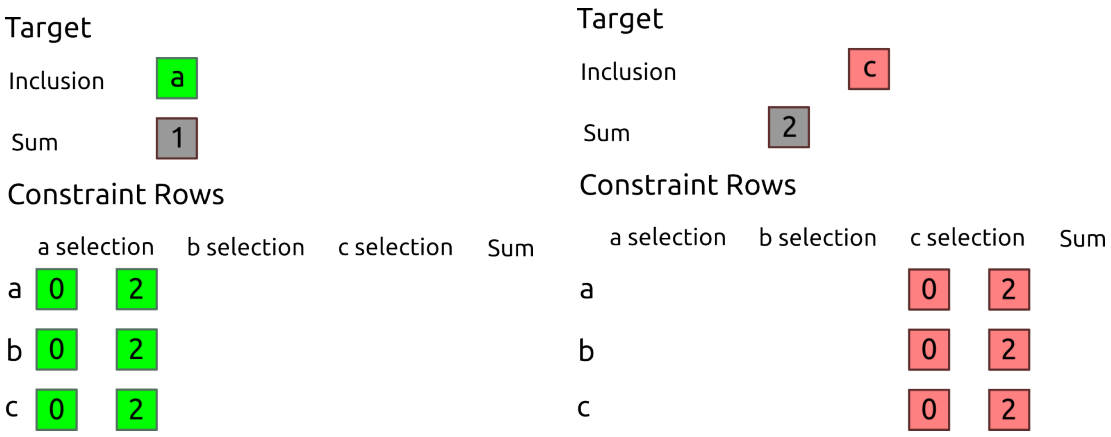


Figure 15: Example Grid, with green cells representing cells that can only be covered by covers related to a, blue for b, red for c and grey for cells that can be covered by all.



(a) vertex a, with a chosen value of 1

(b) vertex c, with a chosen value of 1

Figure 16: Example target covers for vertices a and b, that can be used together.

All the filler covers for vertex a on its constraint row are shown in Figure 18. If vertices are taken from a the filler can take any value as it has enough protection, as any value of a will need at most 2 extra vertices to be protected, which is the defined maximum for this problem instance.

In Figure 19, a full solution is shown for $\{a = 1, b = 0, c = 1\}$ with darker colors representing the target covers, and the lighter colors used for the constraint covers.

This results in a bounded number of cells, covers generated and covers selected based on n, the maximum value a variable can take (the number of vertices in the original problem), and k the number of variables.

Each constraint row (of which there are k), up to the section components represent up to $k(n+1)$ cells along with n for the sum component. For the target section, up to k inclusion cells are needed and n cells for the sum component. Resulting in at most $k^2(n+1) + kn + n + n$ cells being used.

Next, the number of covers is calculated. For filler covers, each variable can take up to n values each of which can generate up to n filler covers, which gives a total of kn^2 filler covers. For the covers generated for the target section, a given value for a variable can start at any offset in the sum. Given this, each of the n possible values can be duplicated n times, given kn^2 covers generated here. The same applies for each of the non-filler covers generated for each of the constraint rows, results in k^2n^2 covers. So in total $(2k + k^2)n^2$ covers are generated at most.

The total number of covers to be selected is dependent on the graph, and on how many other variables each variable is adjacent to in the graph, which effects the number needed to cover each constraint row. This is fixed per problem, with each constraint row needing total the number of neighbours plus one covers to be selected. So in total k for the Target section, and at most k^2 to cover each of the constraint rows.

□

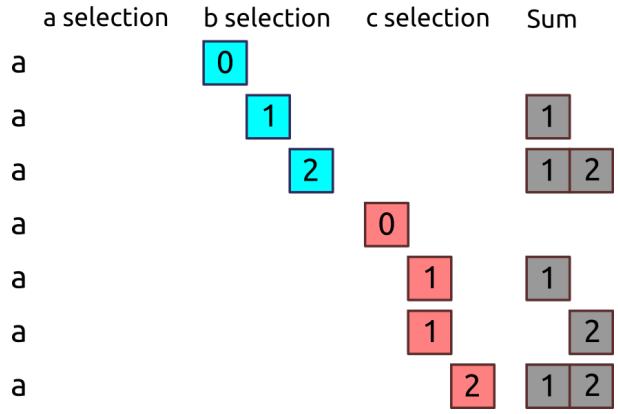


Figure 17: Constraint row covers for the protection constraint for vertex a, covering b and c.

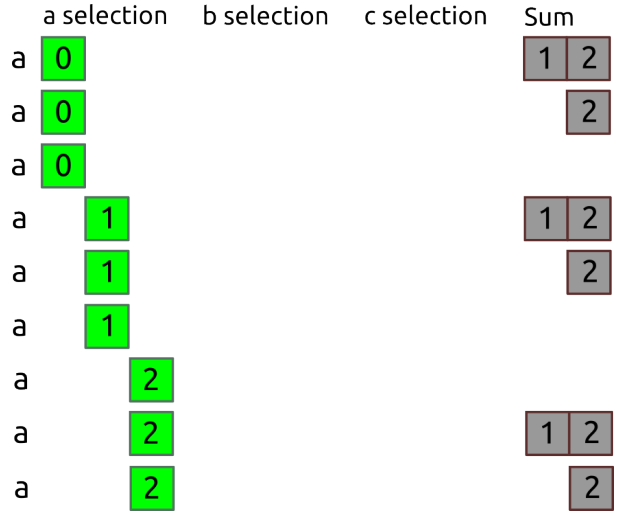


Figure 18: All the filler covers that can be applied to the vertex a's constraint row.

Theorem 65. *Solving an instance of Table-Merge can be done via $O(n^2)$ evaluations of* k-Exact-Cover.

Proof. Given Lemma 64, it is known that it can be verified if there is a selection of values that can be taken from the tables that results in a specific level of protection for a given vertex count in $W[2]$. So by trying all $O(n^2)$ combination of values in the table, each of the possible values for the vertex count and the level of protection, it can be tested to see which are valid combinations. Each instance of k-Exact-Cover has a maximum target value of n, the number of vertices in the graph, bounding the number of coverings generated. For each vertex count, each level of protection is checked from 0 to n, stopping on the first yes-instance as this is the lowest level of vertex protection available for this count.

So by solving up to $O(n^2)$ instances of k-Exact-Cover a minimal table can be found. □

Theorem 66. Defensive Alliance *when Parameterized by Modular Width in is* $W[1]$.

Proof. Constructing a table for a O4 operation was shown to involve up to $O(n^2)$ evaluations of k-Exact-Cover, a problem that is in $W[1]$, in Theorem 65, to obtain a better bounds on the complexity for Theorem 53, as Table-Merge is the primary source of complexity in the algorithm. □

4.4 Minor Results

4.4.1 Number of GMDAs

This section covers an algorithm that uses two parameters based on the number of Globally Minimal Defensive Alliances in a graph below a certain size to determine if one above the given size exists. For Globally Minimal Defensive Alliance, it is possible to solve the problem in $\mathcal{FPT}$ time when

Figure 19: A complete cover is shown for the solution $\{a = 1, b = 0, c = 1\}$. Covers for the target section are shown in the darker colors, while the lighter colors are the constraint covers.

parameterized by two parameters, k the maximum size of a minimal alliance being considered, and d the minimal defensive alliances up to size k. d can be computed in $\mathcal{FPT}$ time as discussed in Section 4.2.1, but depends on the number of vertices in a graph so it can not be used to obtain a solution size parameterized algorithm for Globally Minimal Defensive Alliance (which is impossible due to Theorem 48).

This relies on the algorithm described in Proposition 39 to check if a subset is a globally minimal defensive alliance, and removing all the smaller GMDAs from the graph. If after removing all smaller globally minimal defensive alliances, a defensive alliance is still found though this algorithm, then there must be a globally minimal defensive alliance that is larger than k.

Theorem 67. Globally Minimal Defensive Alliance *is in* $\mathcal{FPT}$ *when parameterized by the number* d *of alliances of size* k.

Proof. Given d Minimal Defensive Alliances, each up to size k, it can be seen that there are at most dk unique vertices. Denoted by $D \subseteq V$ is the set of at most dk vertices across the minimal alliances, so $|V| \geq dk \geq |D|$. Finally, $S \subseteq V$ is used to refer to the set of vertices that form an alliance in the graph, which is initially $S = V$.

The problem becomes finding a set of vertices $L \subseteq D$ that removes all of the known alliances from S but still leaves a defensive alliance in the graph. If this is found, it means all the known minimal alliances in the graph are not in S so it must contain a minimal alliance larger than k if all smaller than k are known.

The algorithm to solve this works by branching on each vertex $v \in L$, deciding whether it is explicitly removed from S or not. After reaching a leaf, all unprotected vertices are removed from S until either a protected subset remains or an empty set is left. If a protected subset remains, S is then checked to verify if it contains any of the known alliances. If not, this is a yes instance.

By removing (at least) one vertex from every minimal alliance, it is ensured that any alliances discovered can not include them. Further, if only unprotected vertices are removed, that does not change the solution as unprotected vertices can not be in a defensive alliance.

Verifying if a potential alliance contains any of the d alliances in polynomial time can be done by iterating through each of the known alliances and checking if all the vertices are included or not. This can be checked in $O(dk)$ time. Removing unprotected vertices can take up to $O(n^2)$ time, where a single vertex is removed during each loop.

This results in a branching algorithm with a $O(2^{dk}n^2)$ running time, which is in $\mathcal{FPT}$ with the two given parameters.

□

An alternative algorithm with an $O(k^d n^2)$ runtime also exists, where it branches on which vertex to remove from each alliance instead of branching on the vertices in general.

4.4.2 Split Parameters

Several Parameters like Neighbourhood Diversity can be split into two sub parameters to better model the problems complexity. For example, in Neighbourhood Diversity there are neighbourhoods that only contain one vertex while others can have many vertices inside them. Being explicit that these are different cases enables a more accurate analysis of an algorithm's complexity for certain cases. Further, problems can be parameterized with these split parameters alongside another one such as the solution size to help tackle interesting problems.

Split Neighbourhoods An alternative approach to neighbourhood diversity is to split the neighbourhoods into *small* and *large*, where small neighbourhoods only include a single vertex and large neighbourhoods include more. The known algorithms handle three cases:

1. The neighbourhood is not included.
2. One vertex from the neighbourhood is included.
3. One or more vertices are included from the neighbourhoods.

By explicitly handling the third case separately through the large parameter, this harder case can be dealt with and allow the remainder of the branching to proceed as before.

First, being able to compute these efficiently is a useful property to be aware of.

Lemma 68. *Given a list of neighbourhoods and the number of vertices inside each one, the split neighbourhood parameters s and l can be computed in linear time. s is the set of neighbourhoods that only contain one vertex, and l is the set of neighbourhoods containing more than one vertex.*

Proof. This can be done via iterating through each of the neighbourhoods, checking the number of vertices in each one and updating the respective counts for s and l in the process. □

Theorem 69. Defensive Alliance *when parameterized by the split neighbourhood parameters s and l is in $\mathcal{FPT}$.*

Proof. Using the algorithm given by Gaikwad et al. [23], the algorithm can be bounded further by only requiring l variables in the ILP compares to the $k = s + l$ originally. □

This gives a slightly tighter analysis of the algorithm by using the split parameter.

4.4.3 Solution Size with other Graph Parameters

Solution Size and graph parameters like Neighbourhood Diversity can have a synergistic effect, allowing some difficult problems to be handled. This was briefly mentioned when discussing the split algorithms for Neighbourhood Diversity, where it was noted that they could be combined with parameters like Solution size for more effective algorithms. By using two parameters, a costly ILP can be avoided for many problems.

Vertex Weights Given an optimization problem where the goal is to find an alliance of size k such that either the vertex weights are maximized or minimized it is possible to use the solution size as the only parameter. However, given some circumstances, for example the graph might have a small neighbourhood diversity or vertex cover, it would be beneficial to combine these to produce a new algorithm which takes this into account.

Lemma 70. *Given a problem involving either minimising or maximising vertex weights, where vertices with the same neighbourhood have different weights, there is a defined order of preference for using vertices in the same neighbourhood.*

Proof. In a maximisation problem, the end goal is to maximise the weight of the selected vertices. This gives the order in which each vertex in the neighbourhood should be taken, in the order of largest weight to least. In a minimization problem, the order is inverted. □

The next steps consider only the minimization case, but can be trivially modified to support maximization.

Lemma 71. *Given pre-assigned counts for each neighbourhood in a graph, it is possible verify if these counts result in a valid solution for* Vertex-Budgeted Defensive Alliance *in Polynomial time.*

Proof. Checking if these are above k, the maximum solution size, can be done by summing the pre-assigned counts up and checking if they exceed k. This can be done with n addition operations, so this check can be done in linear time.

Checking if these counts exceed l can be done by iterating through each neighbourhood and summing the smallest weights. If these are presorted, which is possible as there is a defined order due to Lemma 70, only n operations will need to be done to produce the minimum weighted subset of the vertices.

Finally, this subset needs to be checked to see if it is an alliance. In the worst case this involves checking each neighbour of a vertex in the subset to see if they are also in the subset and then checking against the required threshold, giving a worst case of $O(n^2)$ operations. As each vertex in a neighbourhood is interchangeable in this step, the vertices chosen in the previous step do not affect it. □

Theorem 72. Vertex-Budgeted Defensive Alliance *is in $\mathcal{FPT}$ when parameterized by Solution Size k and Neighbourhood Diversity d.*

Proof. The algorithm works by branching on which neighbourhoods get given a vertex. So, given d neighbourhoods and a maximum solution size of k only d^k combinations need to be considered. These generated counts are then evaluated and verified using the Polynomial time algorithm given by Lemma 71 As evaluating d^k combinations clearly places the problem in $\mathcal{FPT}$ with a $O(d^k n^2)$ running time. □

The same algorithm being discussed can also be applied to plain Defensive Alliance, as the vertex weights will be one.

5 Experiments

To study how Defensive Alliances can be found on real graphs, several of the algorithms were implemented so the performance and their ability to find solutions in a set of experimental cases could be evaluated. The focus was purely on -1-Defensive Alliance and not the other variants being discussed to simplify the process.

The experiments were done by creating a set of graphs, finding optimal solutions using an ILP formulation, and then comparing these optimal solutions against the output of a variety of other implementations.

5.1 Algorithms and Implementations

5.1.1 Implementation Overview

The experimental framework is a library and command line tool written using Python3. This library provides classes and functions for each of the algorithms, and the core data-structures used. A key part of the library, a decision to help keep the algorithms correct, was a requirement that the class that encapsulates alliances of various types verifies each alliance has the required properties. This includes properties such as having each vertex contained in the alliance is being protected, or being globally minimal.

Several libraries were used, which include:

- NetworkX [35] for the graph representation.
- PuLP [36] as an abstraction layer over several ILP solvers.
- matplotlib [37] and Plotly [38] for plotting results.
- DEAP [39] for implementing a Genetic Algorithm solver.
- The Z3 theorem prover [40] for implementing an SMT formulation.
- pandas [41] [42] for processing results.

5.1.2 Exact Algorithms

Four exact algorithms were considered in the experiments. The first was the ILP solver, which was used to find optimal solutions to the problems. The second was the Z3 based SMT solver to consider a secondary approach to finding alliances. Then the final two of the parameterized algorithms discussed. These are the Vertex Cover algorithm and the Solution Size algorithm.

ILP Solver The basic ILP formulation consists of the same formulation as given in Section 2.1.1, where the thresholds are computed by iterating over each vertex in the graph and finding its degree. The implementation supports multiple solvers, as it was implemented using PuLP [36], but Gurobi [43] was the primary solver used. During tests, Gurobi was found to perform significantly better than the default solver included with PuLP (CBC), so it was used.

SMT Solver A Satisfiability modulo theories (SMT) implementation was written using Z3 [40]. This represents each vertex as a boolean variable, with each one having a constraint added in. These constraints are in the form of an if-statements that checks if the vertex is included, and if so sums the boolean variables representing the neighbours and check if it above the threshold. To force z3 to find the optimal solution, the size of the solution found using the ILP was added as a constraint, as Z3 is less suitable for optimization problems.

This solver had a strict 4GB of RAM limit imposed as it was found that z3 would otherwise exhaust the systems memory for some graphs. To support parallel solving, Z3 was set to use the *QF FD* logic.

Vertex Cover The vertex cover ILP formulation is the same as the improved algorithm given for Theorem 27, where the components of the Vertex Cover are branched on to check if the neighbouring vertices outside the cover can protect them.

When evaluating, if an optimal solution used more than 15 members of the vertex cover the graph was skipped to save on computation time. This is because it was unrealistic for the branching get 15 members deep in the allocated time, alongside solving ILPs for a significant number of them. There is some limitations with this in the cases where there is an alliance of the same optimal size but using less members of the vertex cover.

Solution Size The solution size algorithm was the same as described by Fernau et al [20], which branches from a specific starting vertex and searches for an alliance of size k by branching on which of the adjacent vertices are included in the alliance. The implementation used in the experiments runs on multiple CPU cores, with thread being given a different starting vertex. The starting vertices were chosen from the known optimal solution from the ILP.

A second implementation that has a worker queue for evaluating each candidate alliance was written, but not used due to its poor CPU utilization. This algorithm was only run when the known optimal solution used less than 8 vertices, as due to its complexity it was unlikely to find a feasible solution in the allotted time otherwise.

If a graph had more than 30 vertices in its optimal solution, it was skipped as it would otherwise not be feasible for this algorithm to find them. This does result in some of the easier cases being skipped, with one of the generators being considered.

To give the solution size solver a chance, it was started from vertices that were known to be part of the optimal alliance solved by the ILP. As with the vertex cover solver, there could potentially be cases where an alternative (but easier for this algorithm) alliance exists of the same size.

After the experiments were run, there was a bug noticed in this solver invalidating its results, so while it was run it will not be discussed further. The issue was tied to how the recursive implementation decremented the k, which needed 1 added to it for it to work correctly.

5.1.3 Heuristics

Two heuristic based algorithms were implemented to get an overview of how well they could preform. They both used the same cost function, with some variance in how they optimized for smaller solution sizes. Each of the algorithms had three hyper-parameters which were optimized using a Genetic Algorithm.

Hyper-Parameters The two algorithms being discussed have various hyper-parameters which effect their performance. To optimize these, a genetic algorithm structured in a similar way as the one used to find defensive alliances was used. To score, each combination was run over a random set of 10 graphs, which were selected from subset of the total set of graphs.

It was run with a population of 48, a mutation probability of 0.2, and a cross over probability of 0.5 and 10 generations.

Cost Function The cost function was defined by how far a solution was away from being protected. This was computed by taking the threshold of each vertex and seeing how many neighbours the vertex needs to be protected. This was summed up over every vertex to give a final cost.

Genetic Algorithm The genetic algorithm was a standard implementation written using DEAP [39]. Each alliance was represented as a set of bits deciding, one for each vertex deciding if it was part of the alliance. The cost was slightly extended to consider the size of the alliance.

This algorithm had three hyper-parameters, which were the population size, cross-over probability and mutation probability. The hyper-parameter search gave a population of 507, a cross over probability of 0.84 and a mutation probability of 0.42 as the target values.

Cost Reduction Cost-Reduction is a custom heuristic developed for this problem. It can be thought of a modified random walk through the graph, in which vertices are added to the considered set. There is a probability that a new vertex is added or remove, giving a set of candidates, and a probability deciding when a random candidate is used or the one with the best score is used.

The cost-reduction heuristic run over a population as well, and had two different parameters. These were the probability of adding or removing a vertex from the solution, and the probability that this is a random vertex instead of the one that would lower the cost the most. The hyper-parameters chosen were a population of 711, a probability of addition at 0.43 and a probability of a random vertex being added at 0.67.

5.2 Dataset

To construct the dataset used for the experiments, several generators were used. This include four existing ones avaliable as part of networkx and two custom generators. The parameters for each of the generators were decided by solving them with the ILP solver to get a feel for how the parameters effected the overall solution size. When not noted, the range of parameters were explored linearly in the final dataset, but many of the plots used to show the exploration were done over a logarithmic scale.

Slightly under 1150 graphs were generated through all the different generators. The three main generators were given 300 graphs each, with less for the remaining generators due to their being limited reasons to explore more of their parameter ranges. Some combinations of generator values did not produce connected graphs consistently, which is why slightly below 1150 were generated.

Parameters were explored two at a time, to give a 2D grid of possible values, which made the parameters easier to visualize. For the pre-experiments, these range of values are shown using a 3D mesh plot to give a relative view of how they effect the solution size and time to find. The pre-experiment graphs were computed using only a single sample per combination of parameters, and ranges were stepped over evenly over a log scale as the primary goal was only to get a feel for how the graphs parameters develop and not be final results, using the same 15 minute time limit for computation as the final experiments.

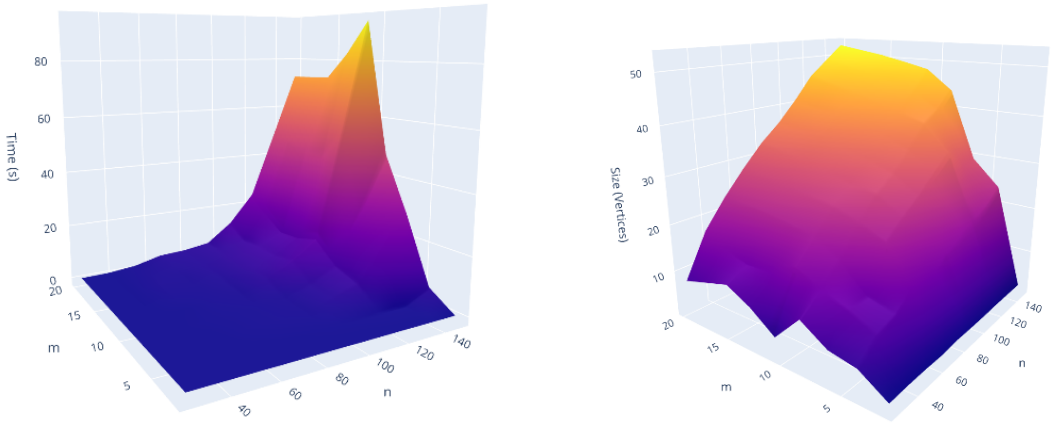
It should be noted that the time to compute a Defensive Alliance tended to grow exponentially as the parameters increased, which was a key aspect used in limiting which ares were searched.

5.2.1 Existing Generators

Four existing graph generators were used, which were the Barabasi-Albert model, Waxman model, GNP model and the Regular graph model.

Barabasi-Albert The Barabasi-Albert model is a scale-free model for generating graphs [44]. The graphs generated vary the number of vertices and the number of edges attached to a new node. The chosen ranges was between 50 and 200 for the number of vertices and between 2 and 25 for the number of edges to attach. In total, 300 graphs were generated, with 10 samples taken across each axis, and each set of parameters having 3 graphs generated.

The pre-experiment results is shown in Figure 20, which gives a feel for how the parameters change the alliance size and time to compute.



(a) Time to Compute

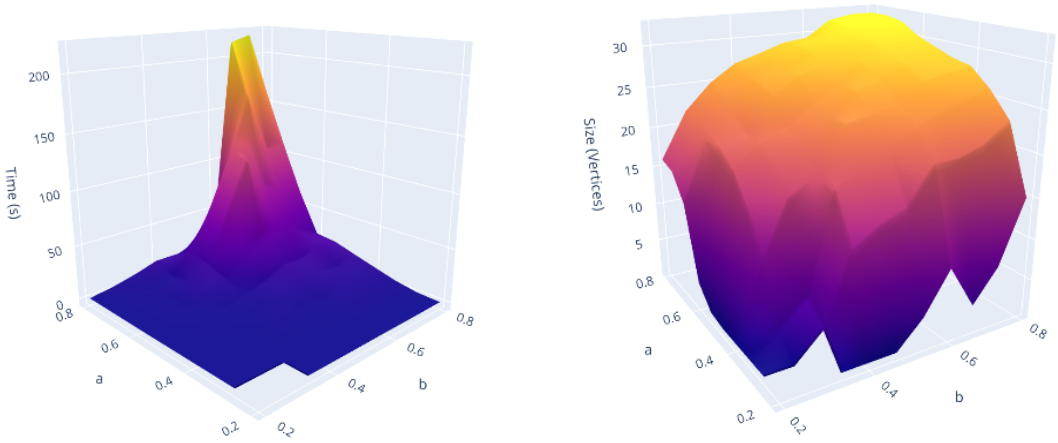
(b) Size of Alliances

Figure 20: Pre-Experimental results showing the time to compute (a) and size (b) of Defensive Alliances for a variety of parameters for values of n and m in a Barabasi-Albert Model

Waxman Waxman, or Random Geometric Graphs, are generated via placing points on a plane and connecting each point to each other with a probability based on their distance from each other [45].

For the experiments, they were generated with a fixed number of vertices, with the beta and alpha values varying. Beta was set to be between 0.2 and 0.8, with alpha varying between 0.15 and 0.3. These values were chosen based off the work by Çetinkaya et al [46], where values in these ranges were used to model telecommunication networks. 300 graphs were generated across these parameters.

Figure 21 shows the graphs slowly creating a plateau of the size of the alliance as alpha and beta get closer to 1.0.



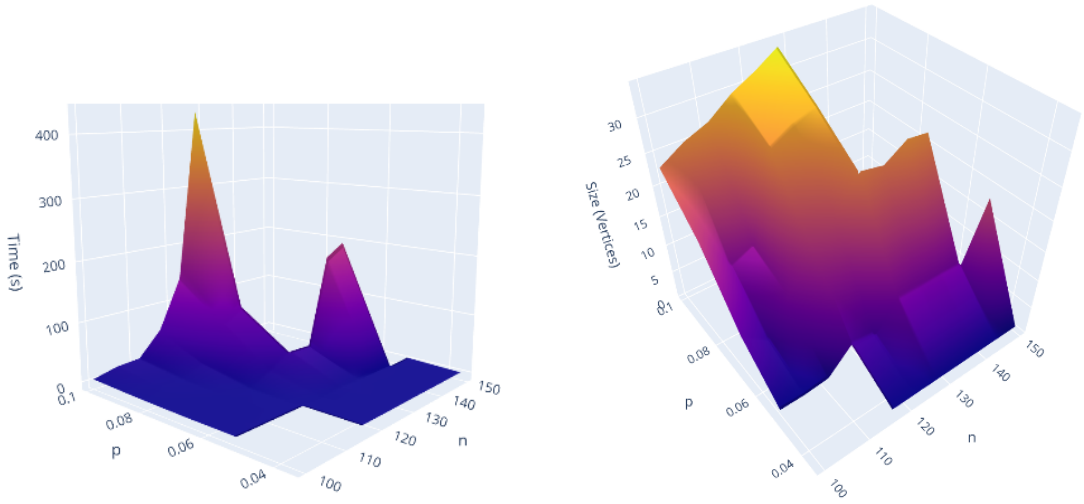
(a) Time to Compute

(b) Size of Alliances

Figure 21: Pre-Experimental results showing the time to compute (a) and size (b) of Defensive Alliances for a variety of parameters for values of beta and alpha in a Waxman Model, for 75 vertices.

GNP A GNP graph has a number of vertices, with an edge existing between any pair of the vertices existing with a given probability [47]. The number of vertices was between 100 and 150, and the probability of an edge being between 0.05 and 0.15. In total, 300 graphs were generated,of which 3 were created for each parameter combination.

Figure 22 shows a subset of the considered range, along with the time to find solutions.



(a) Time to Compute

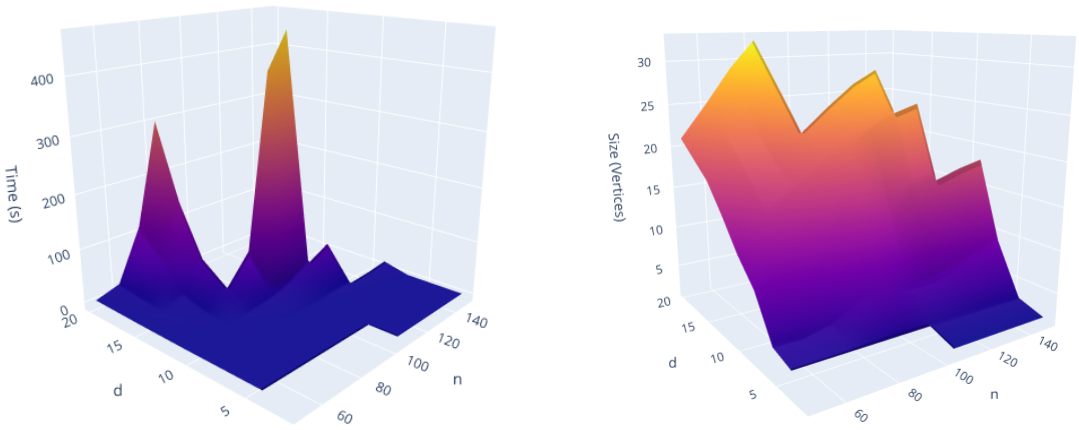
(b) Size of Alliances

Figure 22: Pre-Experimental results showing the time to compute (a) and size (b) of Defensive Alliances for a variety of parameters for numbers of vertices and edge probabilities in a GNP graph.

Regular A regular graph is one where every vertex has the same degree, and is a hard case for finding alliances. The algorithm used to generate them was the one included in networkx, which uses the work by Steger et al. [48]. The parameters being considered were between 50 and 150 vertices and degrees between 3 and 15. There were some limitations on the graphs that could be generated, as the generator requires the total number of vertices multiplied by the degree to be even.

Regular graphs were of less interest so only 25 combinations of these parameters were considered, with 3 graphs generated for each combination. This gave 75 graphs in total.

The pre-experiments shown in Figure 23 show that as n and d get larger, less graphs become possible to solve in 15 minutes.



(a) Time to Compute

(b) Size of Alliances

Figure 23: Pre-Experimental results showing the time to compute (a) and size (b) of Defensive Alliances for a variety of values of n and d for a Regular Graph.

5.2.2 Fixed GMDA

A custom generator was designed to produce a graph with a minimal alliance of a fixed known size. This provides a good case for the algorithm that considers the solution size, and a bad case for the vertex cover algorithm. The solution size algorithm will have a smallest alliance of a known size, controlled as a parameter with very edges to vertices that can not form part of the alliance. The vertex cover algorithm could struggle in this case as the optimal vertex cover would be large.

Construction The construction starts initially as a Complete Graph which has at least $2k+1$ vertices (with more are needed for larger numbers of GMDAs), which will be the core. Then, to add a GMDA of size k, a loop of size k is added. Every vertex in the loop has two connections to vertices in the core, with each vertex in the core only used once. An example is shown in Figure 24, where the box contains the "core" (with the internal connections not shown), with the loop below it.

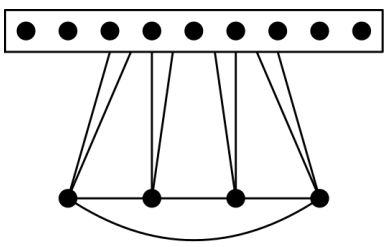


Figure 24: Sketch of the fixed GMDA construction, where the bottom path (the minimal alliance) is connected to the top component.

This is done as:

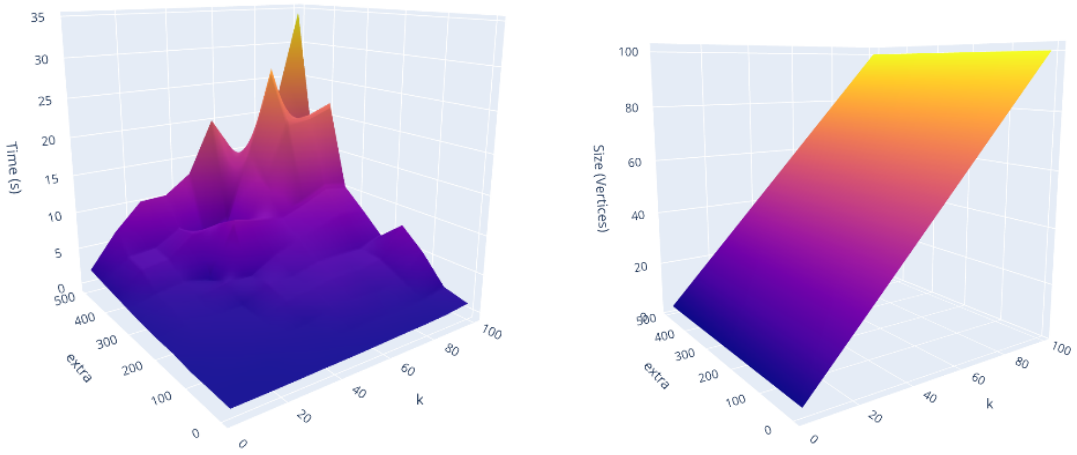
- By having connections to the core, a vertex can not be included in an alliance unless at least two of its neighbours are.

- As each vertex in the core has many internal connections, they need more than k vertices to be in the alliance for them to be included by themselves.
- Partially including part of the loop is not possible, as some vertices will be unprotected if the whole loop is not included.
- So the smallest alliances will involve the loop and will always consists of the whole loop.

There is one exception, which is the case where k is one. This requires the connections to the core to be lower, as otherwise it would not otherwise form an alliance.

An alliance made entirely of vertices in the clique would need to include at least $k+1$ vertices from it, depending on the number of extra vertices included, forcing it to be larger than the minimum size alliance outside the clique.

Parameters This was given an extra parameter that decided how many extra vertices were added to the core, which was set to be between 0 and 500. k was in the range 3 to 100. As there is no randomness in this generator, only one graph was generated for each sample point, giving 100 of these in total.



(a) Time to Compute

(b) Size of Alliances

Figure 25: Pre-Experimental results showing the time to compute (a) and size (b) of Defensive Alliances for a variety of values of k and the extra vertices in the fixed GMDA graph

5.2.3 Planted Vertex Cover

Construction Graphs with an a known upper bound on their minimum vertex cover on their vertex cover can be constructed via:

- Create a core of k vertices, add an edge between vertices in the core with a probability of p_i.
- Create the rest of the graph, by connecting each vertex in this section to the core with a probability of p_x

This approach is similar to the GNP model, just split into two components.

As vertices outside the core only directly to connect ones inside the core. This provides an upper bound on the size of the minimum vertex cover as including the whole of the the core would a vertex cover on its own. Smaller Vertex Covers are possible however.

Parameters Inside the core, the number of vertices varied from 3 to 30 and the number of vertices outside the core was fixed to 50. p_i was fixed to 0.8, and p_x was set to vary between 0.5 and 0.85 as the pre-experiments found these to be good choices. 25 combinations of these were taken, with 3 graphs generated for each combination.

5.3 Process

5.3.1 Pre-Experiments

Before performing the main experiments, several tests were performed with the ILP solver to get an idea of suitable parameters for each generator, along with how many graphs should be generated. Computational time and the ultimate size of alliances was a key concern due to the complexity of some of the algorithms being discussed. Smaller alliances, but ones that take more time to compute were the primary focus. A geometric progression was used to search the graph space to get a feel for the how the time and size complexity grows with the parameters.

These pre-experiments were performed on a desktop computer with a Ryzen 2700x and 48GB of RAM, which is a lower spec machine than the one used in the final experiments. The machine was being used for other work, while these were tests were running, so they are not representative of the final results. Each instance of the ILP was given 4 cores, and two were run at once, with a maximum allowed time of 900 seconds.

5.3.2 Experiments

Machines Several machines were used to perform the experiments. One was hosted on Google Cloud which was used for the exact algorithms, and one was hosted on DigitalOcean which was used for the heuristics. Cloud machines were used to get results in a more reasonable timeframe and to avoid issues around a desktop computer being used for other tasks.

The virtual machine hosted on Google Cloud was a *c2d-highcpu-56* instance hosted in London as part of the *europe-west2* region. The instance had an AMD EPYC Milan server with 56 vCPUs and 110GB of RAM. Each graph to be processed was writen as part of a script, which used GNU parallel [49] to run 14 jobs at once. Each graph was given 4vCPUs, with a limit of 4GB of RAM. The relativity low RAM to CPU ratio was chosen as it was noticed in the local pre-experiments that RAM was not a major concern given each Gurobi, so it was an viable limitation as a cost-control measure. Each experiment was repeated three times, recording the time and size differences between each attempt to catch any unexpected outliers.

For the heuristics experiments, a CPU optimized instance with 48 vCPUs and 96GB of RAM hosted on DigitalOcean was used. This instance was located in the AMS3 region. A similar setup was used for this machine, with each task being listed as a script and queued with GNU parallel.

Both the machines used Ubuntu 22.04 as their operating system. Figure 26 shows some of the experiments as they were running in a tmux session with htop open.

Overall Structure For the exact algorithms, each graph was solved first by the ILP solver to find an optimal. Then, a vertex cover was computed for each graph.

For all the exact solvers, processing the graph had a time limit of 15 minutes to process, and each graph was processed three times. If a graph failed to find a solution during one of the times it was being processed, it would be skipped. 15 minutes was chosen as a reasonable time bound to ensure many different graphs could be considered within the time budget allocated for the experiments, and so not otherwise become too costly to process. Each of the graphs were solved three times to help eliminate any unexpected spikes in computation time, and the number of times each one is solved was set to three as the time variance was expected to be low. Ideally more time should have been given to solve each case with the ILP solver, to give a better comparison but this was not done due to time limitations.

For the heuristics, 5 minutes on a single CPU core was allotted to solve each graph, and each one was solved three times. The best known solution at 5 minutes was then returned for each time the graph was solved.

Graph Generation The experiments were performed through a sequence of steps, first of which was generating the graphs used for the experiments. They were each generated using fixed seeds, but to avoid any potential issues they were generated once and written to disk for use in the later experiments.

Exact Algorithms First, each graph was solved (if possible) with the ILP solver to give an exact solution for each graph. Afterwards, they were then given 10 minutes to be solved with a ILP formulation of vertex cover as one of the parameterized algorithms required known vertex covers. With these known, the Vertex Cover solver was ran. Finally, both the Z3 based solver and the solution size solver were run to finish off all the exact algorithms.

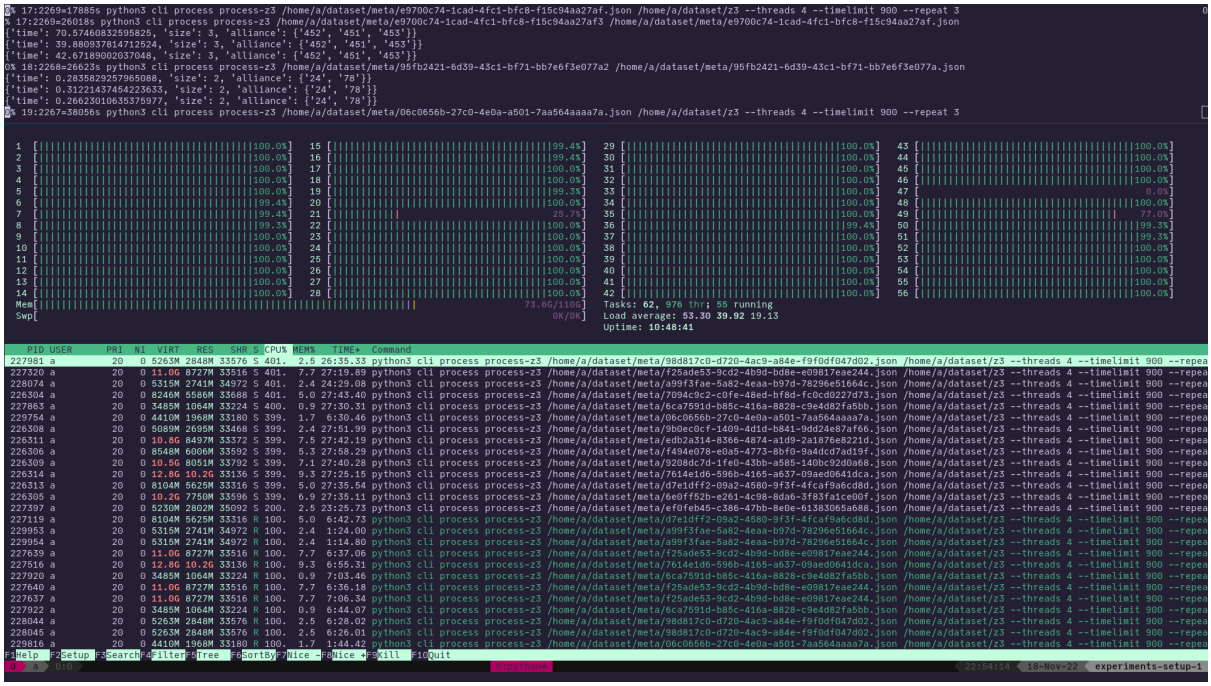


Figure 26: Running the experiments on c2d-highcpu-56 instance, in this case, some of the Z3 experiments. This was taken before Z3s memory usage was limited.

Heuristics The next set of experiments concerns the two heuristic algorithms being considered. These algorithms have several hyper-parameters that needed to be tuned, which was done through a genetic algorithm. To tune these, the data-set was split into two parts, with 10% going towards optimizing these and remainder for evaluation. This gave approximately 90 graphs for hyper-parameter optimization, of which 10 were used to evaluate each set of hyper-parameters.

After tuning, the algorithms were run with a time limit of 5 minutes and allowed to use one core for each problem instance. They were then compared on the quality of their solutions and the time to get to their best found solution.

Further Analysis Given the results, they were looked at to see the impact of various properties such as the solution size and graph parameters. The goal was to develop an intuition for the difficulty of the problem subject to these.

5.4 Results

This section will present the results plainly, before being discussed in more detail in the next section, where further analysis will be done.

5.4.1 General

Figure 27 gives an overview of the alliance size for each of the graphs generated. There is a wide variety, with some of the graph types which had less graphs generated having large gaps in the alliances sizes. Many of the graph generators produced a large number of very small alliances, as shown by the spikes for alliances of size less than three for many of them.

5.4.2 Exact Algorithms

In Table 1 the number of graphs each algorithm solved is listed.

Figure 28 shows each of the algorithms performance by graph generator. Each of the solvers results by graph parameter will be shown using heatmaps, with each cell in the heatmap showing the mean time it took to compute that a graph with those parameters. The graphs for the planted vertex cover and regular graphs will not be shown as the ILP did not manage to solve a large range of parameters for those. Alongside this, the results for the number of graphs that were solved for each viable parameter

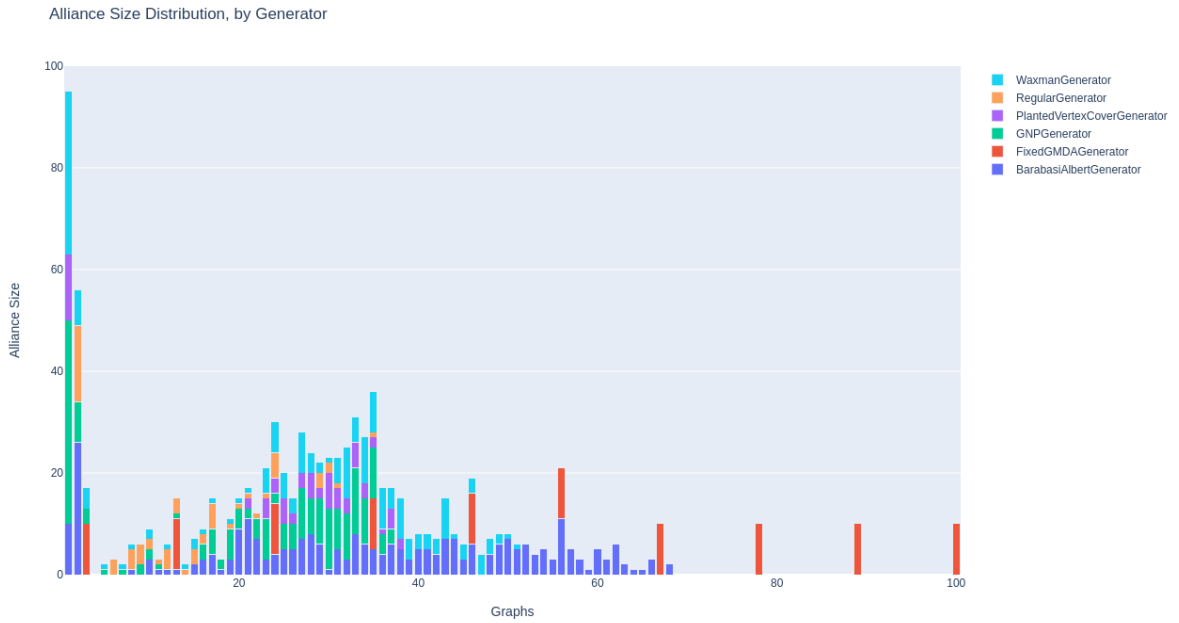


Figure 27: Solution Size Distribution by Graph Type, stacked.

Algorithm	Graphs Solved
ILP	907
SMT	373
Vertex Cover	208

Table 1: Table to show the number of graphs each algorithm solved.

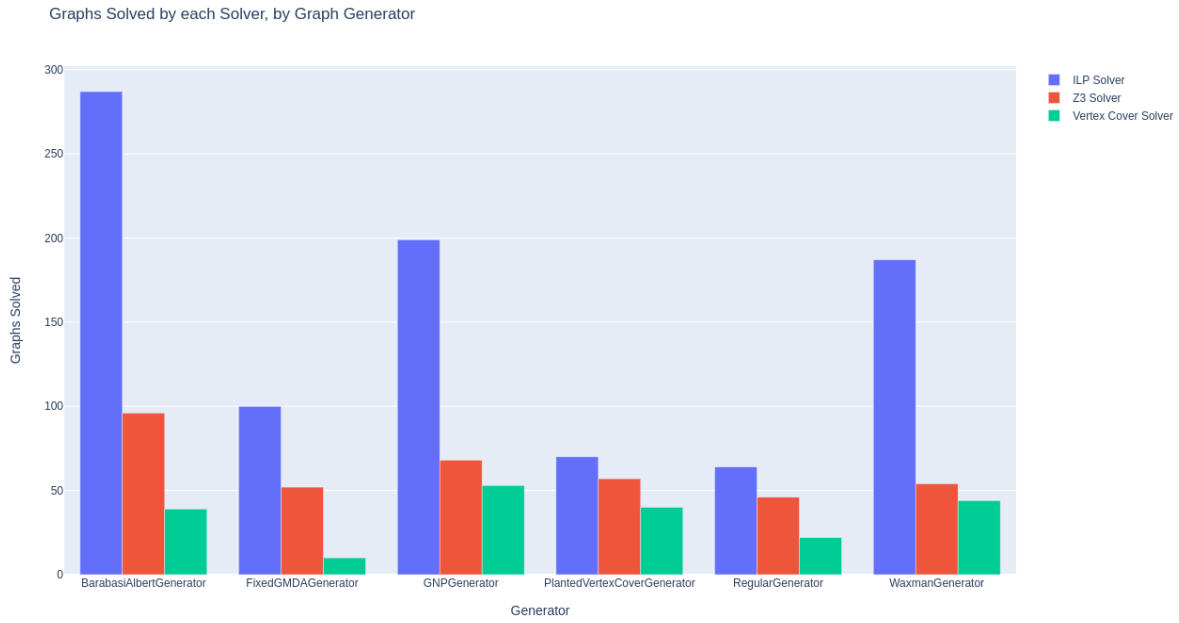


Figure 28: Results by Graph Generator for each of the Exact Algorithms

Graph Generator	Graphs Skipped	Known Solutions
BarabasiAlbert	210	300
FixedGMDA	70	100
Planted Vertex Cover	23	70
Regular	17	75
Waxman	129	298
GNP	115	300

Table 2: Graphs skipped over with the Vertex Cover Solver by Generator Type

pair will be shown for the Waxman, Barabasi-Albert and GNP generators. The Fixed GMDA generator only had one graph generated for each parameter pair due to the lack of randomness, which is why it is not shown alongside these.

ILP Solver The results by graph generator for the ILP Solver are shown in Figure 29. Figure 30 shows the number of graphs that were solved per parameter pair.

Z3 Solver The results by graph generator for the Z3 Solver are shown in Figure 31. Figure 32 shows the number of graphs that were solved per parameter pair.

Vertex Cover Solver The results by graph generator for the Vertex Cover Solver are shown in Figure 33. When processing this, 564 of the graphs were skipped as the optimal solutions used more than 15 members of the vertex cover. This resulted in roughly 60% of the graphs attempted having solutions. Figure 34 shows the number of graphs that were solved per parameter pair.

Solution Size Solver As mentioned in the algorithm section, a bug was identified that invalidated the results. The solver did manage to solve 164 graphs even with the bug.

5.4.3 Heuristics

For this, 817 graphs were considered, as 10% were used to adjust the hyper-parameters. These experiments were ran on a single core, for five minutes on a CPU-optimized instance on DigitalOcean. This led to 48 instances being evaluated at once, due to the number of CPU cores.

These results are presented as the ratio between the optimal solution size and what the heuristic returned. This was sorted by size to get a feel for the general ratio. These two graphs do not share the same scale.

Genetic Algorithm Only 27 out of the 817 graphs did not return a solution within the five minutes. In Figure 36, the ratio between the optimal alliance and the found alliance is shown for all the graphs the Genetic Algorithm managed to solve.

Cost Reduction 703 of the 817 graphs did not return a solution within the five minutes. An overview of the graph generators that Cost-Reduction returned solutions for is shown in Figure 35. It did not return solutions for any graphs generated by Waxman or GNP models.

In Figure 37, the ratio between the optimal alliance and the found alliance is shown for all the graphs solved.

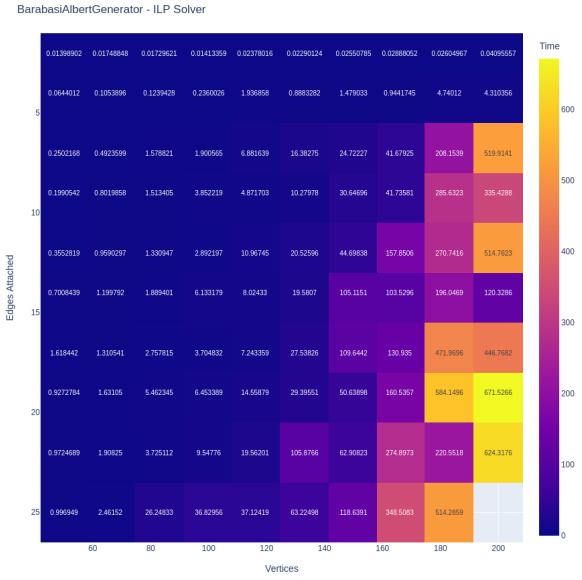
5.5 Analysis

Given the results, an analysis of what was found will be presented.

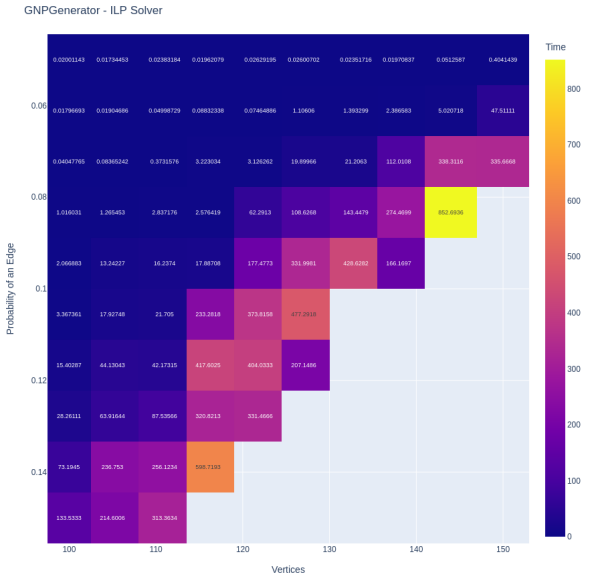
5.5.1 Dataset

The dataset had a significant amount of graphs will alliances of size one, primarily from the smaller and easier to solve generator parameters. Notably, there were not any graphs with an alliances of size four in the dataset, and there are only two graphs with an alliance of size five, so there is a sudden unexpected

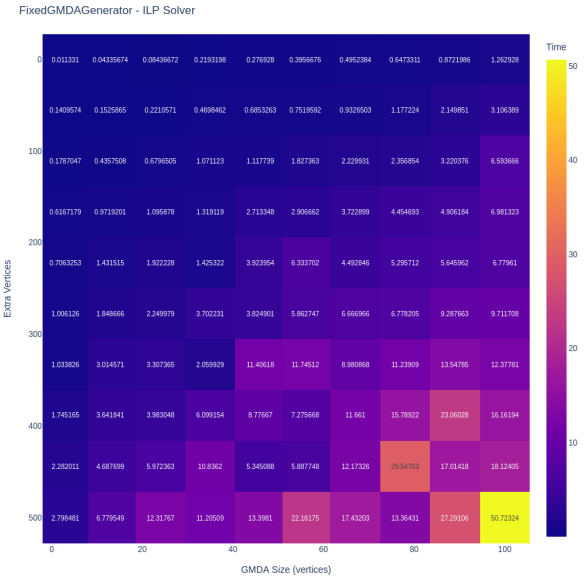
(a) Waxman



(b) Barabasi-Albert

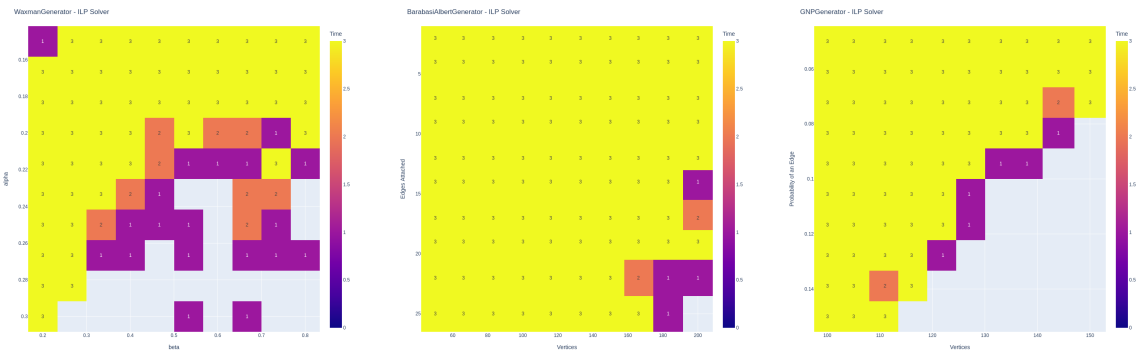


(c) GNP



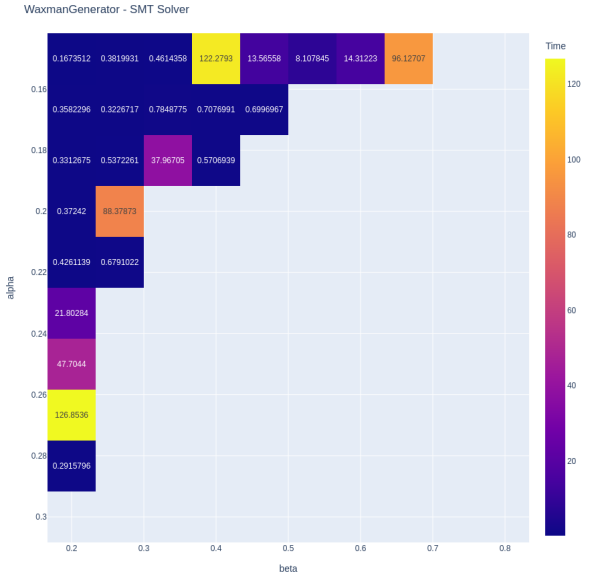
(d) Fixed GMDA

Figure 29: ILP Solver results by graph generator parameter.

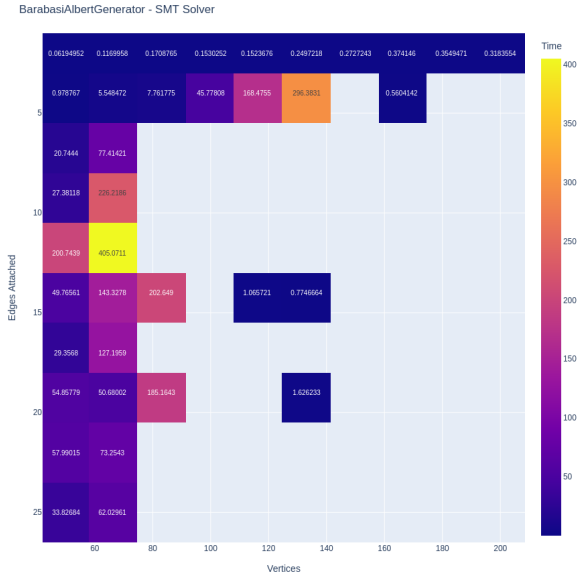


(a) Waxman (b) Barabasi-Albert (c) GNP

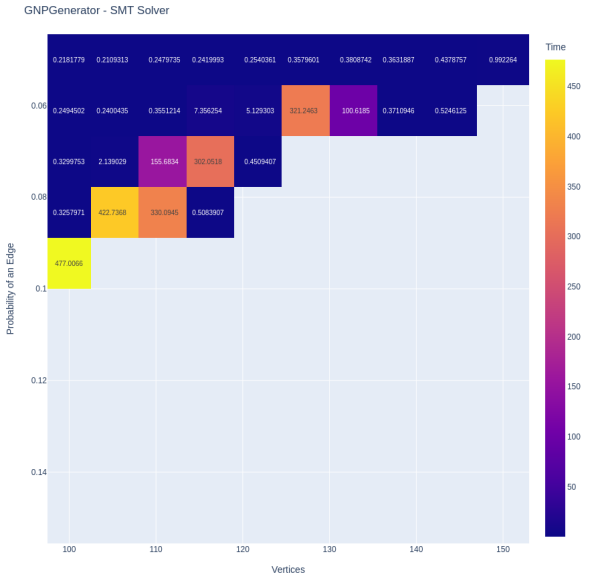
Figure 30: Number of graphs (out of three) that the ILP Solver Solved for each parameter pair



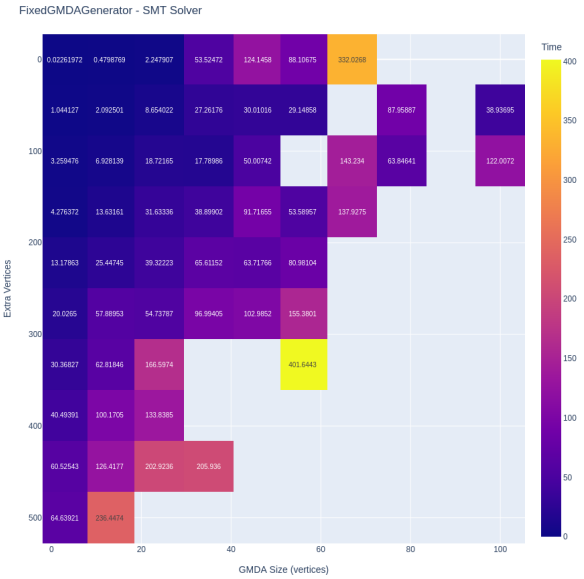
(a) Waxman



(b) Barabasi-Albert

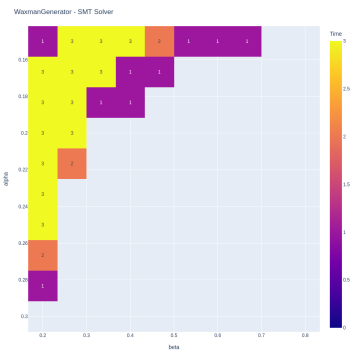


(c) GNP

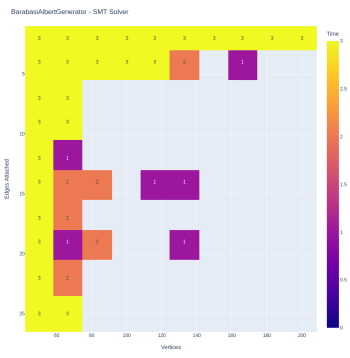


(d) Fixed GMDA

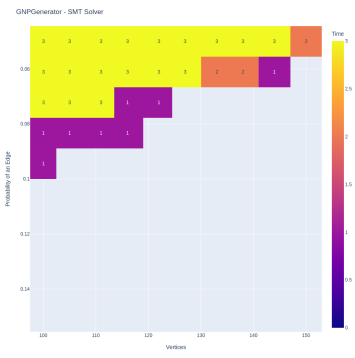
Figure 31: Z3 Solver results by graph generator parameter.



(a) Waxman

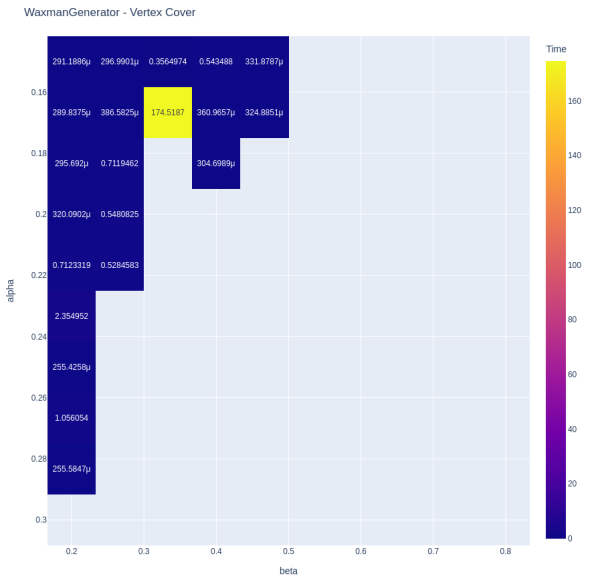


(b) Barabasi-Albert

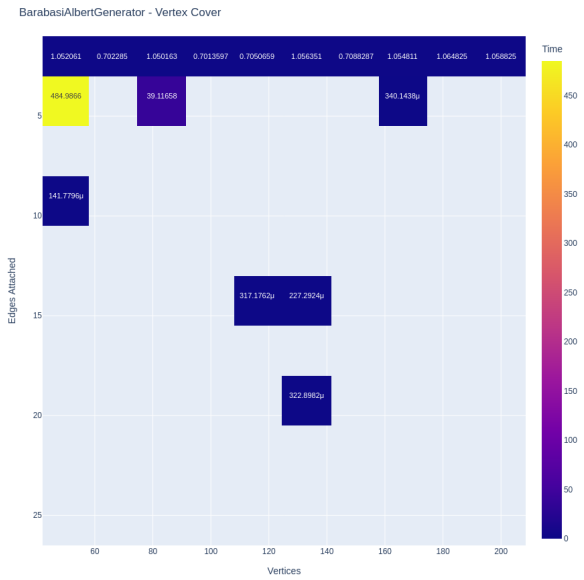


(c) GNP

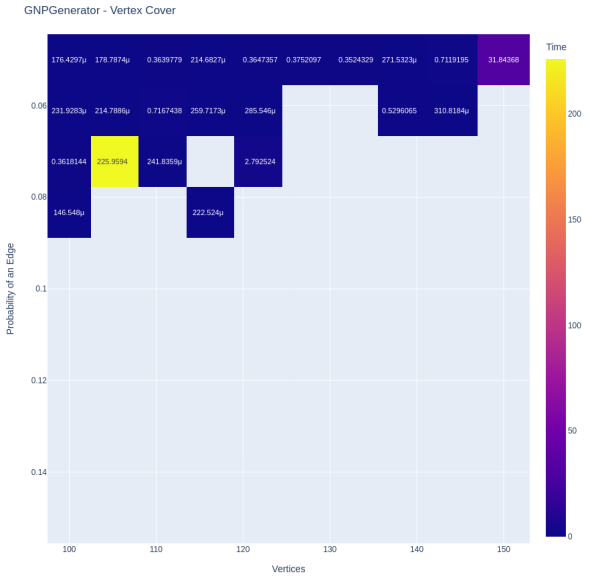
Figure 32: Number of graphs (out of three) that the Z3 Solver Solved for each parameter pair



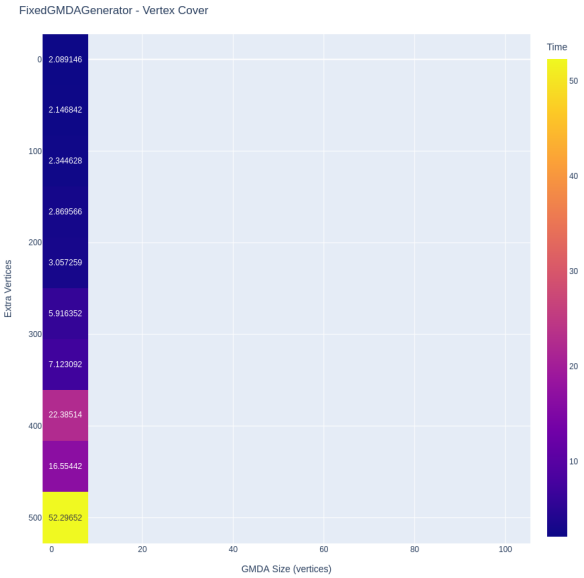
(a) Waxman



(b) Barabasi-Albert

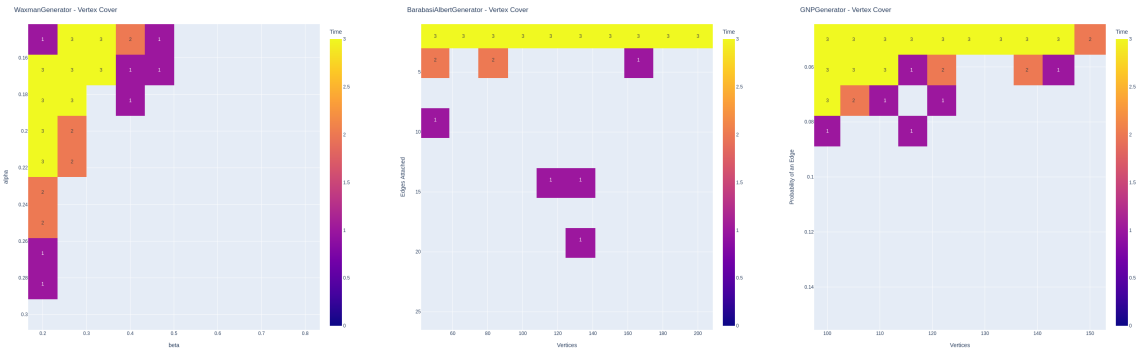


(c) GNP



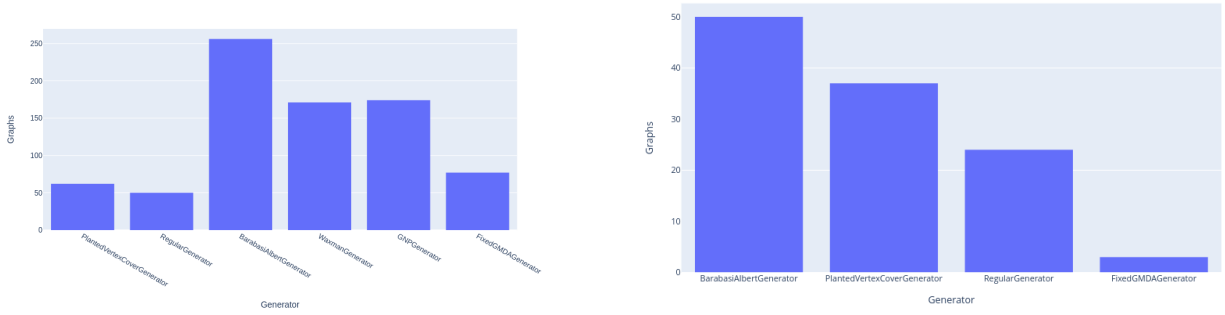
(d) Fixed GMDA

Figure 33: Vertex Cover Solver results by graph generator parameter.



(a) Waxman (b) Barabasi-Albert (c) GNP

Figure 34: Number of graphs (out of three) that the Vertex Cover Solver solved for each parameter pair



(a) Genetic Algorithm (b) Cost-Reduction

Figure 35: Results by Graph Generator for each of the Heuristic Algorithms

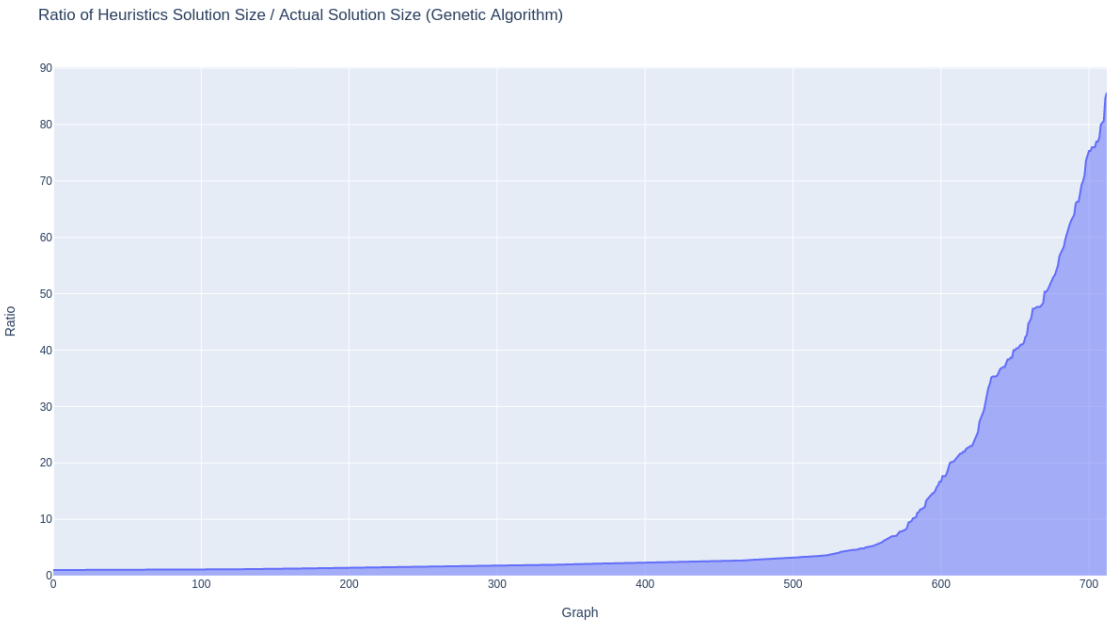


Figure 36: Genetic Algorithm Ratio

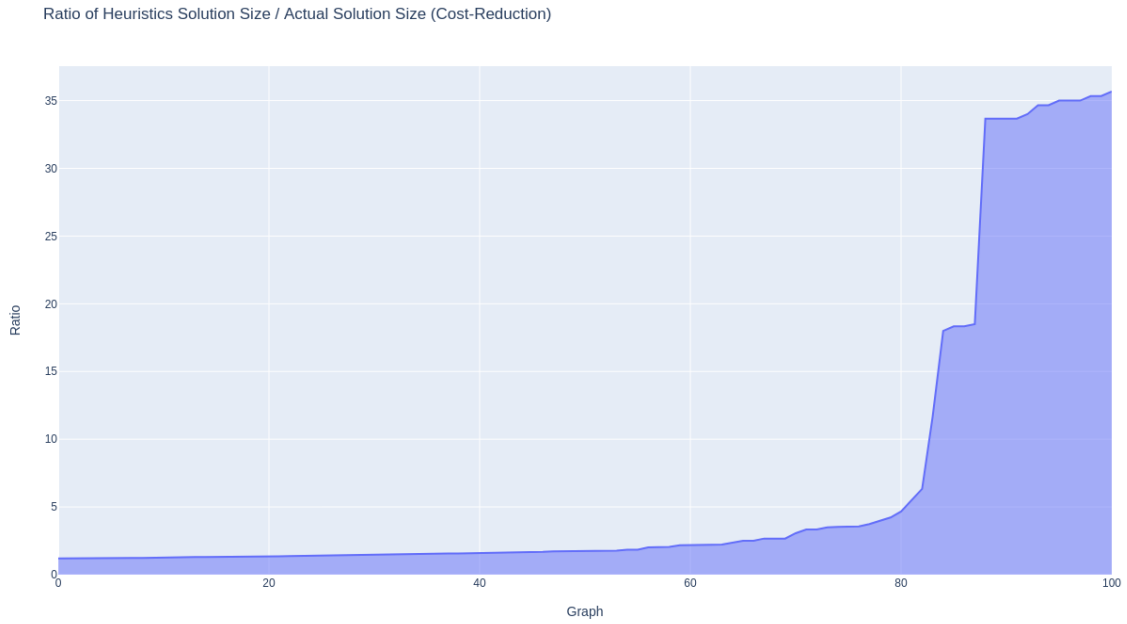


Figure 37: Cost-Reduction Ratio

drop off after three. The gap here is likely due to the generator parameters chosen, with smaller graphs more likely to lead to alliances of this size.

After this, the size of alliances generally increases to peak around sizes of 30-35 vertices, which applies for the majority of the generators. One that did not fall into this pattern was the fixed GMDA generator, which resulted in spikes around various alliance sizes. These spikes flow directly from how it was generated, as 10 points were taken throughout the parameter space, which gave alliances of specific sizes as that is what the generator was designed to do.

5.5.2 Exact Algorithms

The exact algorithms will now be considered in detail, looking at how they compare to each other and highlighting detail of note.

Comparisons This covers various graphs plotting the number of graphs solver by each generator, given an alliance size produced by each generator. This plots can be informed by looking at the pre-experiments, as the alliance sizes per each parameter of the generators are shown.

The graphs discussed in this section need careful attention, as the axis scales varies between each of the graphs.

Several of the figures shows the same general trend of the ability for each solver to find an alliance decreasing as the size of the alliance increases. Explicitly, consider Figure 38, Figure 39, Figure 40 and Figure 41 for this. Figure 43 also shows a similar trend, but due to the spacing between alliance size it is less clear. Figure 42 does show a similar trend, but the generator produced alliances with quite a large minimum size making the comparison harder.

This does show the ILP solver performing the best, as it seems in many cases to be able to find larger alliances which many of the other solvers never came close to tackling. Obviously, this is partly tied to how it was used to find optimal solutions, as graphs that the other solvers may be better at solving optimally were not considered.

Time Variance During the experiments it was noticed that depending on the properties, some of the solvers have quite a high variance in time to solve a graph with the same parameters, or even the same graph. This was noted with the Z3 solver, where the same graph could take widely different amounts of time. It is difficult to plot and compare these directly however.

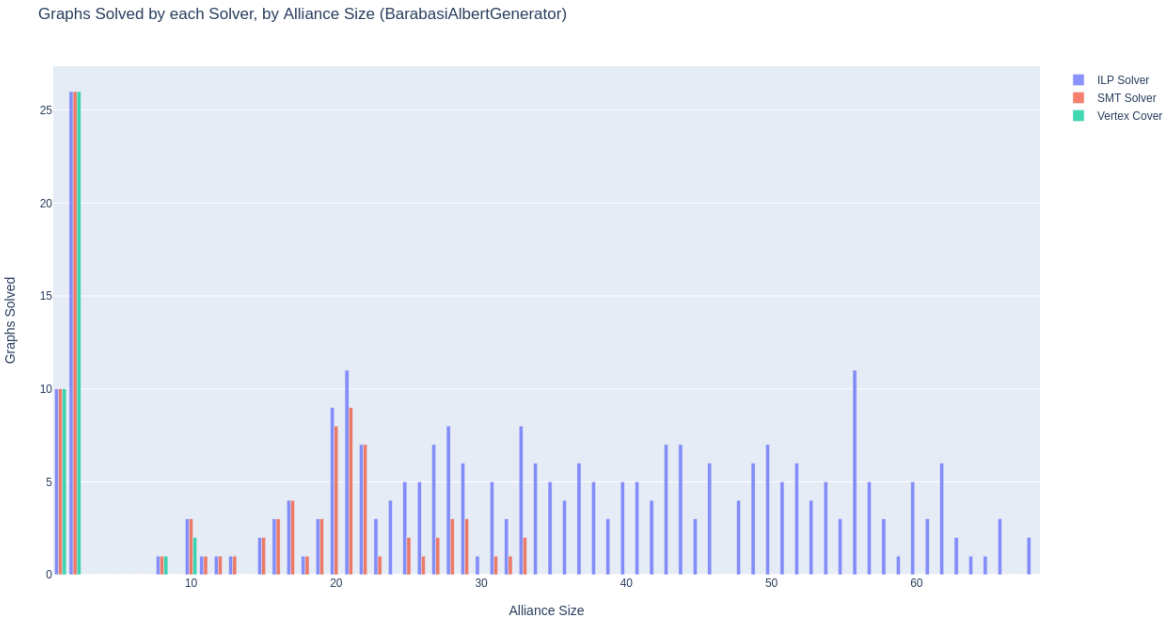


Figure 38: Barabasi-Albert Results by alliance size, comparing each solver.

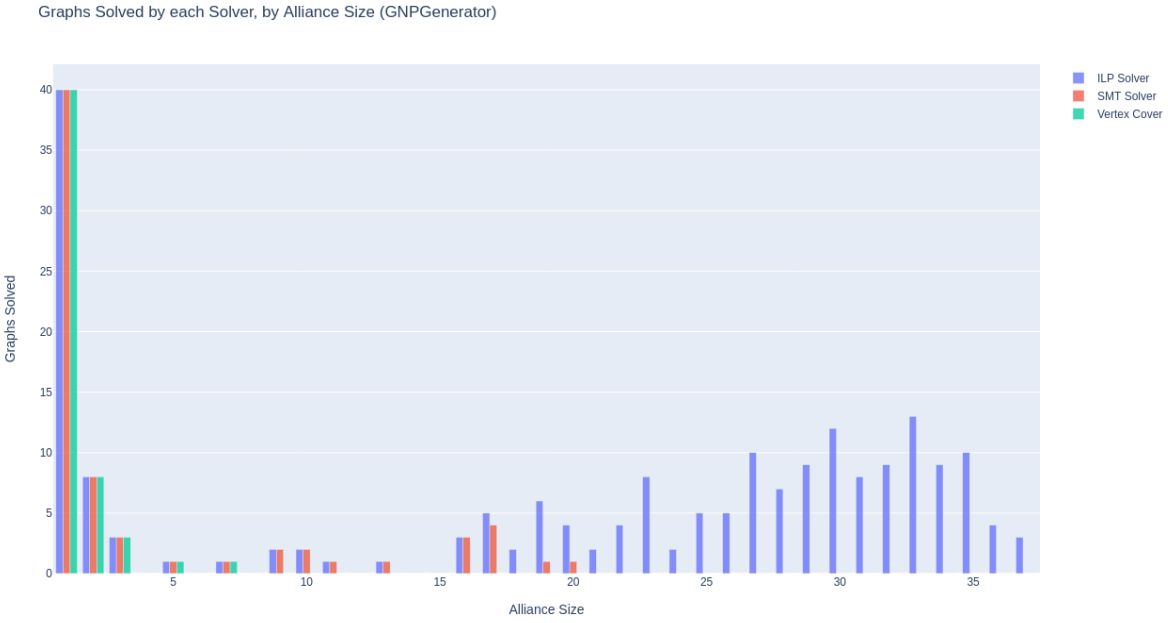


Figure 39: GNP Results by alliance size, comparing each solver.

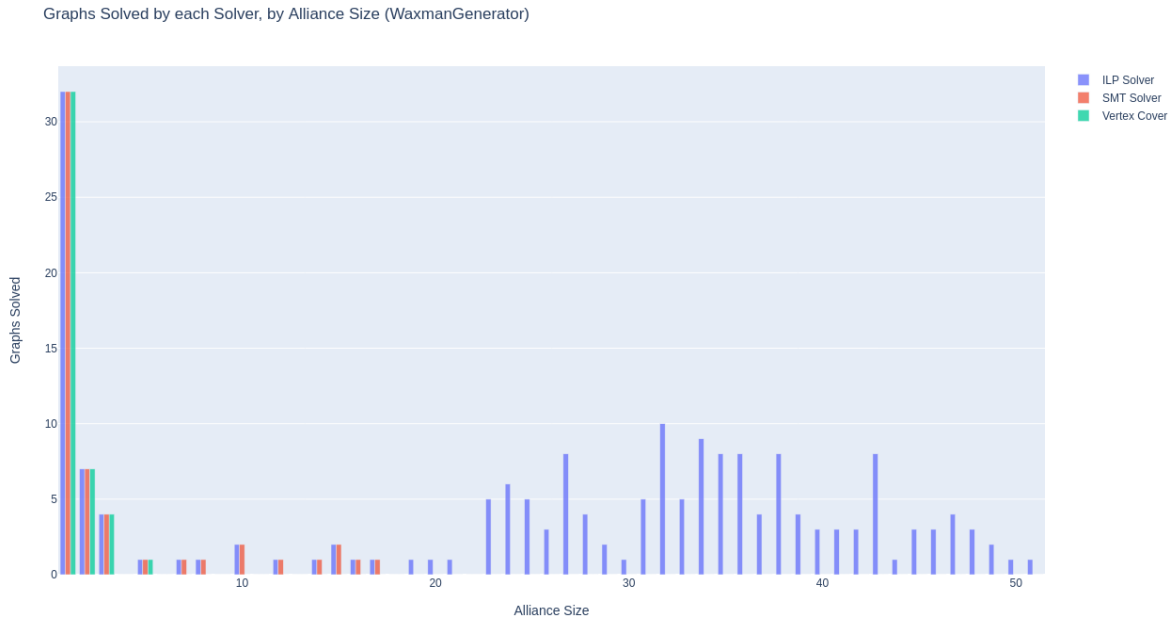


Figure 40: Waxman Results by alliance size, comparing each solver.

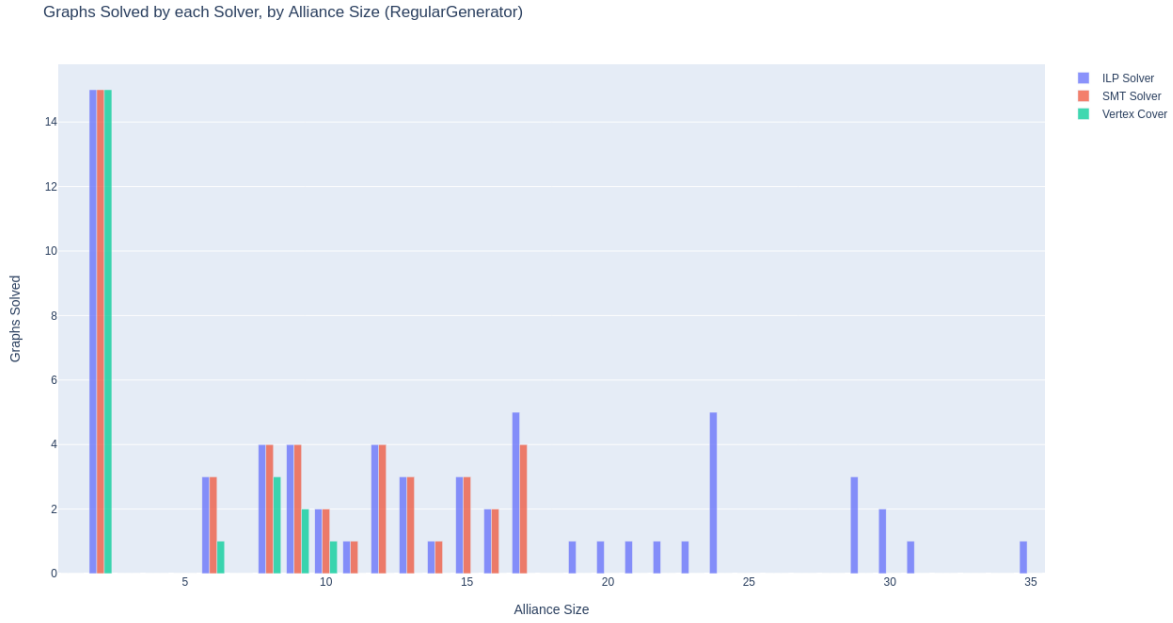


Figure 41: Regular Graphs Results by alliance size, comparing each solver.

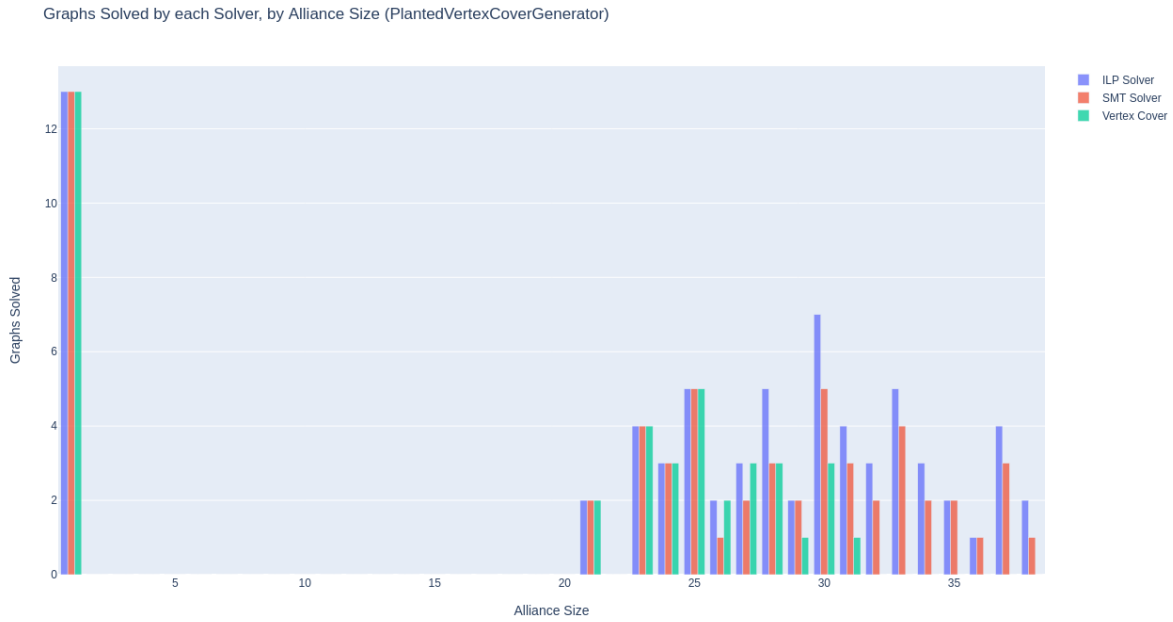


Figure 42: Planted Vertex Cover generator results by alliance size, comparing each solver.

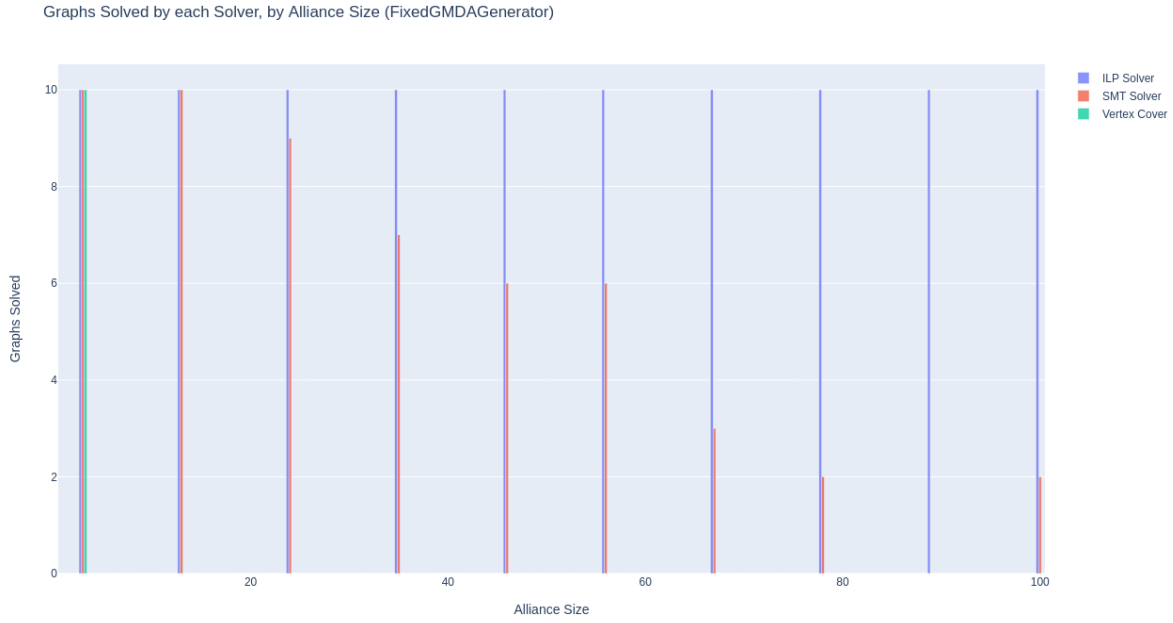
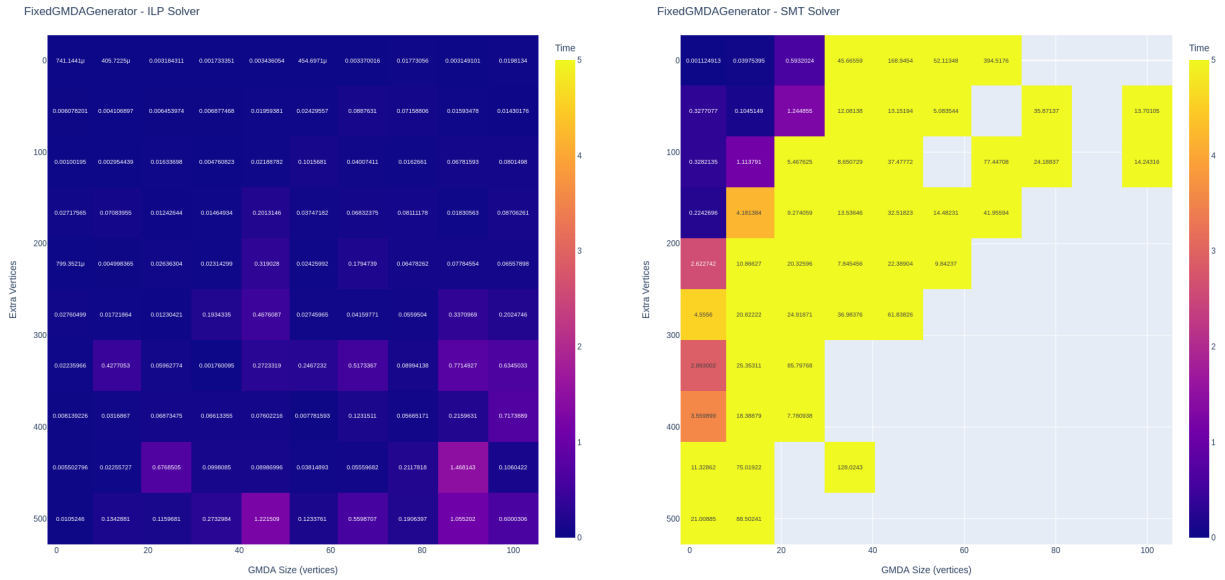


Figure 43: Fixed GMDA generator results by alliance size, comparing each solver.



(a) ILP

(b) SMT

Figure 44: Standard Deviation for the time to solve for each parameter pair in the fixed GDMA generator.

To demonstrate the variance over a single graph, one solution for each parameter for the fixed GMDA generator was plotted for the ILP and Z3 Solvers, shown in Figure 44. Only one graph was used for each value, so there is no variance from different parameter option and each solver solved the graph three times.

Issues How the solution size and vertex cover solvers avoided overly large parameters was not entirely fair, as there are cases in which they could have potentially solved larger instances. For example, if there were multiple solutions of the same optimal size in a graph, with one using less members of the given vertex cover these could potentially have been solved. Further, slightly more of the planted GMDA graphs could have been solved with the solution size algorithm if the results were valid, as it was not run if the minimal solution was larger than 30. More graphs that were similar to this, where there is a dense core and less dense component forming an larger alliance, could have also been solved.

5.5.3 Heuristics

Next the heuristic results will be considered.

Direct Comparison With 703 of the graphs not returning solutions, Cost-Reduction did not perform very well compared to the genetic algorithm. During development, it found solutions fairly quickly, but this may have been due to the graphs that were being used Potentially, some adjustments could be made to improve the design. These include adjusting how each candidate is selected, such that it is weighted based on the score instead of randomly.

Interestingly, Cost-Reduction was able to solve 9 graphs that the Genetic Algorithm was not able to find solutions for. But as the Genetic Algorithm solved 685 that Cost-Reduction was not able to, so this is not the most relevant comparison.

Out of the graphs that they both managed to get solutions for, Cost-Reduction to beat the genetic algorithms average score four times.

Solution Size Ratios Both of the algorithms achieved relativity good approximations, staying below 10 times for the vast majority of the graphs.

The best solution each of the heuristics found is shown in Figure 45, where it can be seen that the genetic algorithm beat out the Cost-Reduction for all the graphs considered. Only for a very few graphs did Cost-Reduction achieve a better ratio. It is worth noting that they were generally close in ratio for many of the graphs they both managed to solve.

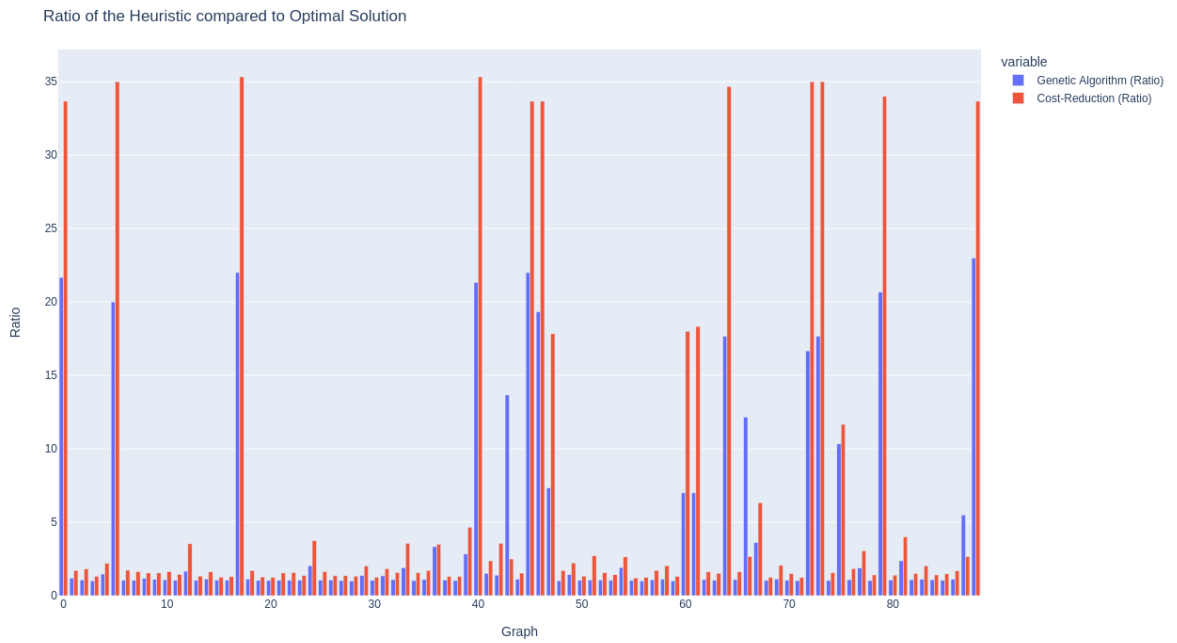


Figure 45: Comparing the Ratio for the Graphs both Cost-Reduction and the Genetic Algorithm Solved

5.5.4 Limitations

The issue invalidating the solution size solver is a significant limitation, as it caused many results to be lost. There was not enough time to re-run these experiments with correct code.

There are some significant limitations with how some parts of the experiments were performed, such as the maximum size for the solution size solver and vertex cover solver. It would be preferable to re-run the experiments without these restrictions to understand if they were needed. Further, extending the time allocated to solving each graph considered would also be useful, especially if more time was given to the ILP solver to obtain results for more of the graphs.

Aspects of the data-set construction could be improved, to give a better distribution of alliance sizes, time to solve, etc. Developing a process to do this is quite complex and hard to do while also trying to vary different parameters of the graph generators. There is also some interest in trying the algorithms on real world graphs, but defining a good test-cases is difficult. These aspects are likely to be a better fit for a more focused study.

It is unclear if there is a better way to compare the heuristics against the baseline better than using a ratio. Ratios are not the ideal solution as the size of defensive alliances is not continuous, so the heuristic could in fact be finding the next largest alliance above the optimal solution and still achieve a high ratio. But to verify this, checking if an alliance of each size exists would need to be done.

5.6 Conclusion

In conclusion, various exact and heuristic algorithms were experimented with. The exact results showed the benefits of the ILP solver compared to the other solvers tried, followed by the Z3 solver and then vertex cover solver. The heuristics experiments clearly showed that the genetic algorithm performed significantly better than Cost-Reduction.

6 Conclusion

6.1 Overview

This work set out to understand the existing literature, study variants of the Defensive Alliance, consider new parameters for the problem, make improvements to the existing work and also explore the experimental aspects of finding Defensive Alliances. The existing literature was reviewed, going over each of the of known graph parameters and their general complexity. This was used as the base on in which later aspects of the work was developed on to find areas to improve upon.

Variants Variants were then considered, with results given for a variety of problems such as Threshold Defensive Alliance and each of the problems were considered using three graph parameters. The existing graph parameters considered were solution size, neighbourhood diversity and vertex cover. Solution size generally worked for the variants, followed by vertex cover supporting more of the variants than neighbourhood diversity, which was fairly limited as to be what problems it could be applied to, at least with the techniques applied.

New Parameters Several new parameters were considered. The primary results were for distance to clique, modular width, and answering the open question about the complexity of Globally Minimal Defensive Alliance when parameterized by solution size. Some secondary results were also discussed, primarily involving multiple parameters.

Distance to clique was tackled by using it to directly bound the neighbourhood diversity, and then allowing the existing neighbourhood diversity algorithm to be directly applied.

The modular width result upper-bounds the complexity of Defensive Alliance when parameterized by modular width to $W[1]$. It could still potentially be in $\mathcal{FPT}$, but the upperbound helps give a better understanding of its complexity.

Globally Minimal Defensive Alliance was found to be in para-$\mathcal{NP}$ when parameterized by solution size, answering an open question by Gaikwad et al [14]. This was shown via a reduction from X3SAT, a variant of 3-SAT.

Improvements to Existing Work A limited set of improvements to the existing algorithms were made, as this was found to be a difficult task in most cases. A notable exception was the slightly lowered bounded for vertex cover and the use of secondary parameters.

Experiment Work The experimental work covered several exact algorithms alongside two heuristics. The ILP solver was used as the baseline, so it can not be directly said to be the best performing as everything else was only run on graphs it managed to solve, but it returning the best results was expected due to general ability of ILP solvers to tackle problems due to the many years of development they have had. Z3 performing the second best was expected as Z3 has also been extensively developed, but the limitations of SMT formulations limited the number of graphs it was able to solve.

In terms of the heuristics, the Genetic Algorithm performed as well as expected but the Cost-Reduction algorithms performance was significantly below expectations. Solving less than 15% of the graphs is a disappointing result as it worked relatively well during development.

6.2 Future Work

There are several possible future research directions. More variants and restrictions, and a further study of how these properties effect alliance problems more formally would be useful for a better understanding of the problems.

6.2.1 Alternative Graph Parameters

There remains several graph parameters that would be worthwhile to study in further detail, but the majority of them are unlikely to be in $\mathcal{FPT}$.

Shrub-Depth Shrub-Depth is a parameter based around the depth of a tree-model, which can roughly be described as a tree representation of a graph where each leaf has a color assigned to it it and each level joins parts of the tree according to a pattern of colors. This parameter was considered during research, but due to its nature it was hard to apply to an alliance problem without additional constraints to the tree-model.

Twin-Cover Twin-Cover was looked at, but its true complexity is not known. The parameter extends vertex cover by allowing edges to be either a normal edge where one member is part of the twin-cover or a twin-edge where each vertex shares the same neighbourhood. The complexity arises from a member of the twin-cover being connected to multiple cliques, each sharing twin-edges internally. These cliques become hard to treat as interchangeable.

Algorithms for problems where this parameter has led to results, such as graph motif [50], rely on not needing direct edges between these cliques. However, this is not a property Defensive Alliance can rely on, which is motivation to believe Defensive Alliance is not in $\mathcal{FPT}$ with twin-cover. Further, an $\mathcal{XP}$ algorithm is not known so it may be in para-NP or an alternative complexity class.

Vertex Neighbourhood Cover When looking at Twin-Cover, a novel parameter was considered where members of the vertex cover can be duplicated, in a manner similar to neighbourhood diversity. This preserves the property that both edges outside the cover are interchangeable, while also allowing members of the vertex cover to become interchangeable with their duplicates. The problem then becomes finding a way of balancing the protection between each of these, which does seem to make this a possible parameterization of Defensive Alliance.

Domino Tree-Width Domino Tree-width is one of the known parameters, but was not considered in this work for the variants, so extending it to the variants remains open but is likely to be possible.

6.2.2 Algorithmic Complexity

As mentioned in the overview, it was found that improving the existing algorithms complexity was quite difficult. So this remains an area where more work would be useful as this would allow more of the parameterized algorithms to be used in real world applications. Several approaches were considered, but did not ultimately lead to much progress, such as trying alternative subproblems. This was a key motivation behind the work on J-Subset Threshold Defensive Alliance, as this was a generalized case, but other problems were considered.

Secondary parameters was found to be a viable technique, so potentially more work could be done there. One aspect that was not looked at in detail was a focus on more specific graph types, so this remains open.

6.2.3 Experimental Improvements

There are a few areas of the experimental work that could be improved. Further analysis of the Solution Size algorithm to verify if in fact there was an issue with it giving the unexpected results for the GMDA generator. One aspect would be to consider if the Cost-Reduction algorithm could be improved to give better results, as its performance was lackluster.

More graphs, of more graph types, with a greater exploration of parameters would be a major improvement to the experimental work.

A longer hyper-parameter search would be more ideal, with some improvements to the Genetic Algorithm used to select them. It might be more beneficial to swap this part out for a more generic grid search, to see if this is to blame for the lower performance of the Cost-Reduction algorithm.

Exact Algorithms More exact algorithms being implemented would be a good area of work, as the algorithm for neighbourhood diversity given by Gaikwad et al [23] could potentially perform in a similar fashion to the vertex cover cover algorithm in dense graphs.

Heuristics It is rather simple to implement more population based meta-heuristics using a library like MEALPy [51]. A tool that implements this was written, but there are issues involved in deciding which of the meta-heuristics to use. This is an area that is more suited to a focused study of only heuristics approaches.

One potential idea for a heuristic algorithm would be to apply something similar to the diffusion models used in Machine Learning for image generation, by extending these towards graphs. Essentially, start with random weights for each vertex in a graph, and then progressively attempt to denoise them through a learned function that guides this towards an Alliance, which is determined by deciding what is in the alliance by a threshold for the weights and finally removing unprotected vertices. This approach maps to an intuition of what seems to make a good heuristic for the problem. Beyond this approach, it might also be worth considering using a Graph Neural Network to guide the branching behaviour in some of the other algorithms, as this has been successfully applied to many other problems.

References

[1] P. Kristiansen, S. Hedetniemi, and S. Hedetniemi, "Alliances in graphs," *Journal of Combinatorial Mathematics and Combinatorial Computing*, vol. 48, pp. 157–177, 2004.

[2] K. Ouazine, H. Slimani, and A. Tari, "Alliances in graphs: Parameters, properties and applications - A survey," *AKCE Int. J. Graphs Comb.*, vol. 15, no. 2, pp. 115–154, 2018. DOI: 10.1016/j.akcej.2017.05.002. [Online]. Available: https://doi.org/10.1016/j.akcej.2017.05.002.

[3] W. Carballosa, J. M. Rodríguez, J. M. Sigarreta, and Y. Torres-Nuñez, "Alliance polynomial of regular graphs," 2015. DOI: 10.48550/ARXIV.1506.06041. [Online]. Available: https://arxiv.org/abs/1506.06041.

[4] I. G. Yero and J. A. Rodríguez-Velázquez, *Defensive alliances in graphs: A survey*, 2013. DOI: 10.48550/ARXIV.1308.2096. [Online]. Available: https://arxiv.org/abs/1308.2096.

[5] K. H. Shafique and R. D. Dutton, "Maximum alliance-free and minimum alliance-cover sets," *Congressus Numerantium*, vol. 162, pp. 139–146, 2003.

[6] H. W. L. Jr., "Integer programming with a fixed number of variables," *Math. Oper. Res.*, vol. 8, no. 4, pp. 538–548, 1983. DOI: 10.1287/moor.8.4.538. [Online]. Available: https://doi.org/10.1287/moor.8.4.538.

[7] M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh, "Graph layout problems parameterized by vertex cover," in *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, S. Hong, H. Nagamochi, and T. Fukunaga, Eds., ser. Lecture Notes in Computer Science, vol. 5369, Springer, 2008, pp. 294–305. DOI: 10.1007/978-3-540-92182-0_28. [Online]. Available: https://doi.org/10.1007/978-3-540-92182-0%5C_28.

[8] C. Bazgan, H. Fernau, and Z. Tuza, "Aspects of upper defensive alliances," *Discret. Appl. Math.*, vol. 266, pp. 111–120, 2019. DOI: 10.1016/j.dam.2018.05.061. [Online]. Available: https://doi.org/10.1016/j.dam.2018.05.061.

[9] L. Jamieson and B. Dean, "Weighted alliances in graphs," *Congressus Numerantium*, vol. 187, pp. 76–82, 2007.

[10] R. C. Brigham, R. D. Dutton, and S. T. Hedetniemi, "Security in graphs," *Discrete Applied Mathematics*, vol. 155, no. 13, pp. 1708–1714, 2007, ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2007.03.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X07000558.

[11] A. Gaikwad and S. Maity, "Defensive alliances in graphs," *CoRR*, vol. abs/2111.05545, 2021. arXiv: 2111.05545. [Online]. Available: https://arxiv.org/abs/2111.05545.

[12] R. Enciso, "Alliances in graphs: Parameterized algorithms and on partitioning series-parallel graphs," Ph.D. dissertation, University of Central Florida, USA, 2009.

[13] R. Carvajal, M. Matamala, I. Rapaport, and N. Schabanel, "Small alliances in graphs," in *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, L. Kucera and A. Kucera, Eds., ser. Lecture Notes in Computer Science, vol. 4708, Springer, 2007, pp. 218–227. DOI: 10.1007/978-3-540-74456-6_21. [Online]. Available: https://doi.org/10.1007/978-3-540-74456-6%5C_21.

[14] A. Gaikwad and S. Maity, "Globally minimal defensive alliances: A parameterized perspective," *CoRR*, vol. abs/2202.02010, 2022. arXiv: 2202.02010. [Online]. Available: https://arxiv.org/abs/2202.02010.

[15] C. Chang, M. Chia, C. Hsu, D. Kuo, L. Lai, and F. Wang, "Global defensive alliances of trees and cartesian product of paths and cycles," *Discrete Applied Mathematics*, vol. 160, no. 4-5, pp. 479–487, 2012. DOI: 10.1016/j.dam.2011.11.004. [Online]. Available: https://doi.org/10.1016/j.dam.2011.11.004.

[16] A. Gaikwad, S. Maity, and S. K. Tripathi, "Parameterized intractability of defensive alliance problem," in *Algorithms and Discrete Applied Mathematics - 8th International Conference, CALDAM 2022, Puducherry, India, February 10-12, 2022, Proceedings*, N. Balachandran and R. Inkulu, Eds., ser. Lecture Notes in Computer Science, vol. 13179, Springer, 2022, pp. 279–291. DOI: 10.1007/978-3-030-95018-7_22. [Online]. Available: https://doi.org/10.1007/978-3-030-95018-7%5C_22.

[17] L. H. Jamieson, "Algorithms and complexity for alliances and weighted alliances of various types," Ph.D. dissertation, Clemson University, USA, 2007.

[18] A. Gaikwad, S. Maity, and S. K. Tripathi, "Parameterized complexity of locally minimal defensive alliances," *CoRR*, vol. abs/2105.10742, 2021. arXiv: 2105.10742. [Online]. Available: https://arxiv.org/abs/2105.10742.

[19] M. Cygan, F. V. Fomin, L. Kowalik, *et al.*, *Parameterized Algorithms*, 1st. Springer Publishing Company, Incorporated, 2015, pp. 12–14, ISBN: 3319212745.

[20] H. Fernau and D. Raible, "Alliances in graphs: A complexity-theoretic study," in *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings Volume II*, J. van Leeuwen, G. F. Italiano, W. van der Hoek, *et al.*, Eds., Institute of Computer Science AS CR, Prague, 2007, pp. 61–70.

[21] A. Gaikwad, S. Maity, and S. Saurabh, *Parameterized algorithms for locally minimal defensive alliance*, 2022. DOI: 10.48550/ARXIV.2208.03491. [Online]. Available: https://arxiv.org/abs/2208.03491.

[22] M. Kiyomi and Y. Otachi, "Alliances in graphs of bounded clique-width," *Discret. Appl. Math.*, vol. 223, pp. 91–97, 2017. DOI: 10.1016/j.dam.2017.02.004. [Online]. Available: https://doi.org/10.1016/j.dam.2017.02.004.

[23] A. Gaikwad, S. Maity, and S. K. Tripathi, "Parameterized complexity of defensive and offensive alliances in graphs," in *Distributed Computing and Internet Technology - 17th International Conference, ICDCIT 2021, Bhubaneswar, India, January 7-10, 2021, Proceedings*, D. Goswami and T. A. Hoang, Eds., ser. Lecture Notes in Computer Science, vol. 12582, Springer, 2021, pp. 175–187. DOI: 10.1007/978-3-030-65621-8_11. [Online]. Available: https://doi.org/10.1007/978-3-030-65621-8%5C_11.

[24] H. L. Bodlaender and J. Engelfriet, "Domino treewidth," *J. Algorithms*, vol. 24, no. 1, pp. 94–123, 1997. DOI: 10.1006/jagm.1996.0854. [Online]. Available: https://doi.org/10.1006/jagm.1996.0854.

[25] H. L. Bodlaender, "A note on domino treewidth," *Discret. Math. Theor. Comput. Sci.*, vol. 3, no. 4, pp. 141–150, 1999. [Online]. Available: http://dmtcs.episciences.org/256.

[26] A. Gaikwad and S. Maity, "On structural parameterizations of the offensive alliance problem," *CoRR*, vol. abs/2110.15757, 2021. arXiv: 2110.15757. [Online]. Available: https://arxiv.org/abs/2110.15757.

[27] B. Bliem and S. Woltran, "Defensive alliances in graphs of bounded treewidth," *CoRR*, vol. abs/1707.04251, 2017. arXiv: 1707.04251. [Online]. Available: http://arxiv.org/abs/1707.04251.

[28] M. Lampis, "Algorithmic meta-theorems for restrictions of treewidth," *Algorithmica*, vol. 64, no. 1, pp. 19–37, 2012. DOI: 10.1007/s00453-011-9554-x. [Online]. Available: https://doi.org/10.1007/s00453-011-9554-x.

[29] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier, "Fixed-parameter algorithms for cluster vertex deletion," *Theory of Computing Systems*, vol. 47, no. 1, pp. 196–217, Jul. 2010, ISSN: 1433-0490. DOI: 10.1007/s00224-008-9150-x. [Online]. Available: https://doi.org/10.1007/s00224-008-9150-x.

[30] H. N. de Ridder *et al.*, *Information System on Graph Classes and their Inclusions (ISGCI)*, https://www.graphclasses.org, 2022.

[31] T. J. Schaefer, "The complexity of satisfiability problems," in *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, ser. STOC '78, San Diego, California, USA: Association for Computing Machinery, 1978, pp. 216–226, ISBN: 9781450374378. DOI: 10.1145/800133.804350. [Online]. Available: https://doi.org/10.1145/800133.804350.

[32] J. Gajarský, M. Lampis, and S. Ordyniak, "Parameterized algorithms for modular-width," *CoRR*, vol. abs/1308.2858, 2013. arXiv: 1308.2858. [Online]. Available: http://arxiv.org/abs/1308.2858.

[33] M. Tedder, D. G. Corneil, M. Habib, and C. Paul, "Simpler linear-time modular decomposition via recursive factorizing permutations," in *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., ser. Lecture Notes in Computer Science, vol. 5125, Springer, 2008, pp. 634–645. DOI: 10.1007/978-3-540-70575-8_52. [Online]. Available: https://doi.org/10.1007/978-3-540-70575-8%5C_52.

[34] M. Cygan, F. V. Fomin, L. Kowalik, *et al.*, *Parameterized Algorithms*, 1st. Springer Publishing Company, Incorporated, 2015, ISBN: 3319212745.

[35] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[36] COIN-OR, *Pulp*, version 2.6.0, Dec. 4, 2021. [Online]. Available: https://coin-or.github.io/pulp/.

[37] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[38] P. T. Inc. "Collaborative data science." (2015), [Online]. Available: https://plot.ly.

[39] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, Jul. 2012.

[40] L. M. de Moura and N. S. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, C. R. Ramakrishnan and J. Rehof, Eds., ser. Lecture Notes in Computer Science, vol. 4963, Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24. [Online]. Available: https://doi.org/10.1007/978-3-540-78800-3%5C_24.

[41] T. pandas development team, *Pandas-dev/pandas: Pandas*, version v1.5.1, If you use this software, please cite it as below., Oct. 2022. DOI: 10.5281/zenodo.7223478. [Online]. Available: https://doi.org/10.5281/zenodo.7223478.

[42] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[43] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2022. [Online]. Available: https://www.gurobi.com.

[44] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999. DOI: 10.1126/science.286.5439.509. eprint: https://www.science.org/doi/pdf/10.1126/science.286.5439.509. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.286.5439.509.

[45] B. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, 1988. DOI: 10.1109/49.12889.

[46] E. K. Çetinkaya, M. J. F. Alenazi, Y. Cheng, A. M. Peck, and J. P. G. Sterbenz, "A comparative analysis of geometric graph models for modelling backbone networks," *Opt. Switch. Netw.*, vol. 14, pp. 95–106, 2014. DOI: 10.1016/j.osn.2014.05.001. [Online]. Available: https://doi.org/10.1016/j.osn.2014.05.001.

[47] P. Erdös and A. Rényi, "On random graphs i," *Publicationes Mathematicae Debrecen*, vol. 6, p. 290, 1959.

[48] A. Steger and N. C. Wormald, "Generating random regular graphs quickly," *Comb. Probab. Comput.*, vol. 8, no. 4, pp. 377–396, 1999. [Online]. Available: http://journals.cambridge.org/action/displayAbstract?aid=46711.

[49] O. Tange, "Gnu parallel - the command-line power tool," *;login: The USENIX Magazine*, vol. 36, no. 1, pp. 42–47, Feb. 2011. [Online]. Available: http://www.gnu.org/s/parallel.

[50] R. Ganian, "Improving vertex cover as a graph parameter," *Discret. Math. Theor. Comput. Sci.*, vol. 17, no. 2, pp. 77–100, 2015. [Online]. Available: http://dmtcs.episciences.org/2136.

[51] N. V. Thieu and S. Mirjalili, *MEALPY: a Framework of The State-of-The-Art Meta-Heuristic Algorithms in Python*, version v2.5.0, Jun. 2022. DOI: 10.5281/zenodo.6684223. [Online]. Available: https://doi.org/10.5281/zenodo.6684223.