Heuristic Algorithms in Maker-Breaker Secure Domination Games

A thesis submitted for the degree of
Masters of Science (Mathematics)

Patrick Dunn-Lawless

College of Science & Engineering

Flinders University

June 2023

Contents

List of Figures

Summary

This thesis considers the Maker-Breaker domination game, a special case of Maker-Breaker games introduced in 2020 in which the winning sets are the dominating sets of the underlying graph. Since these games always have a theoretical winner, the majority of analysis in the literature has been towards identifying graphs where a specific player has a guaranteed winning strategy. However, little consideration has been given to how these games play out when a winning strategy is not known to either player.

The Maker-Breaker domination game can be naturally extended by considering variants of dominating sets, and this thesis does so by considering secure dominating sets. This extension has not previously been considered in the literature. A simulation environment is designed whereby both players (Maker and Breaker) are assigned a play strategy, and the game unfolds until one player wins. There is a stochastic element to the simulation environment, and so the simulation can be run many times to observe different possible outcomes. The results of these simulations are analysed to see which strategies are more successful. The strategies considered are based on various heuristics and results in the literature. Most notably, these include a heuristic based on the Erdős-Selfridge criterion, a seminal result in positional games literature, and also a recently proposed heuristic based on relaxations of a mixed integer linear programming formulation for secure domination. The analysis reveals that the proposed strategies can be categorised into one

of three tiers, based on the degree of computational intensity and observed level of success.

Declaration

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Patrick Dunn-Lawless

Acknowledgements

I would like to thank my supervisors, Michael Haythorpe and Alex Newcombe, for their steady guidance and support throughout my studies. I would also like to thank my family and my partner for their love and encouragement.

Chapter 1

Introduction

1.1 Graphs

Graphs are commonly defined as a tuple (V, E), where V denotes a vertex set, and $E \subseteq V \times V$ an edge set, representing connections between vertices. In general, two vertices may have multiple edges between them, or an edge may connect a vertex to itself. Edges may also be assigned weights to represent some cost of traversal or some other variable of interest. However, in this thesis we restrict our focus to *simple graphs*, which have no multiple edges between vertices, no looping edges from a vertex to itself, and no weights associated with their edges.

Graphs are extensively used in a wide variety of applications [43, 46, 52, 53], and there are many questions regarding graphs themselves that can be asked. Some common questions are whether certain structures exist within a graph, or whether the graph exhibits certain properties. The computation-based difficulty of these questions can range from easy to intractable. In this thesis we will be concerned with a certain kind of graph structure called a *dominating set*, which will be discussed in Section 1.2.

Certain games can be played on graphs, where the latter essentially constitutes the game board. Depending on the rules of the game, the player(s) may traverse or claim vertices or edges in pursuit of some goal, which itself is often a particular graph structure. In this thesis we will consider a particular kind of game called a *Maker-Breaker game* which may be played on graphs. We will discuss Maker-Breaker games in Section 1.3

1.1.1 Notation

We now introduce notation which will be useful throughout the thesis.

If two vertices v, u are *adjacent* in a graph G, we write $v \sim u$ or equivalently $u \sim v$. *Nonadjacency* is written as $u \not\sim v$.

For a vertex v in the vertex set V of a simple graph G, the *open neighbourhood* of v is $N(v) := \{u \in V | v \sim u\}$. The *closed neighbourhood* of v is $N[v] = N(v) \cup \{v\}$.

A subset S of a vertex set V is said to be an enclave of a vertex v if $N[v] \subseteq S$.

The notion of neighbourhoods can be extended to sets. For some set of vertices S, we define

$$N(S) = \bigcup_{s \in S} N(s), \text{ and } N[S] = \bigcup_{s \in S} N[s].$$

A subset S of a graph's vertices is said to be an *independent set* if
$\forall u, v \in S,\ \ u \not\sim v$.

A vertex set exhibiting some property $\mathcal{P}$ is said to be a *maximal $\mathcal{P}$-set* if the addition of any other available vertices would mean the resulting set no

longer exhibits property $\mathcal{P}$. Similarly, a *minimal $\mathcal{P}$-set* is a set which no longer exhibits property $\mathcal{P}$ on removal of any vertex.

A vertex x in a subset of vertices X is said to be *redundant* if
$N[x] \subset N[X - \{x\}]$

A *matching* in a graph is a subset of edges such that every vertex is an endpoint of at most one edge . A matching M is a *perfect matching* if every vertex is an endpoint of exactly one edge in M.

1.2 Dominating Sets of Graphs

Given a graph $G = (V, E)$, a collection of vertices $S \subseteq V$ is said to be a *dominating set* of G if $\forall v \in V$, either $v \in S$ or $\exists w \in S$ such that $v \sim w$. Given a set S, a vertex $v \in S$ is said to *cover* all vertices in $N[v]$. A dominating set is thus a set that covers all vertices in V.

The concepts of domination and dominating sets are often attributed to Ore, who first used the term in Chapter 13 of [45], and Berge, who had previously defined the concept under another name [6]. However, the concept (and several of its variants) appeared with other names as early as the 19th century [35], notably in the context of considering positions in chess games [5].

The concept of a *minimal dominating set* was of interest to Ore. These are dominating sets that cease to be dominating on the removal of any member vertex. A related concept is that of a *minimum dominating set*. This is a dominating set of the smallest possible cardinality in a given graph. Note that all minimum dominating sets are minimal, but a minimal dominating

set is not necessarily a minimum dominating set, an example of which is given in Figure 1.1.

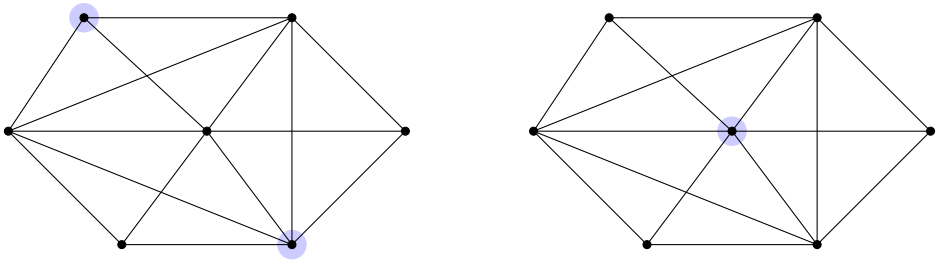

Figure 1.1: Comparison of a minimal dominating set (left) and a minimum dominating set (right) of the same graph. Set members are highlighted in blue

The cardinality of a minimum dominating set for a graph is referred to as that graph's *domination number*. In their 1977 survey, Cockayne and Hedetniemi [19] denoted this number as $\gamma(G)$, which has since become the standard notation. Computing $\gamma(G)$ is known as the *domination problem* and was shown to be an NP-hard problem in general [42]. Dominating sets and their variations are often applied to issues of network coverage or facility location [34, 4]. Following the survey by Cockayne and Hedetniemi, there was a marked increase in research about dominating sets, and parameters such as $\gamma(G)$. By requiring further properties beyond domination, several variants of domination that can be defined. As such, to avoid confusion we will at times refer to the original form of domination as *plain domination*. The variant in the following subsection is a major focus of this thesis.

1.2.1 Secure Domination

The subsequent years after Cockayne and Hedetniemi's paper saw a considerable increase in publications concerning domination. This is, in part, credited to the wide variety of domination parameters that can be defined, and their natural interpretations in real world applications [38]. We will presently discuss a select few of these parameters.

The *vertex independence numbers*, $i(G)$ and $\alpha(G)$ denote the respective minimum and maximum cardinality of a maximal independent set. Berge showed that every maximal independent set in a graph G is a minimal dominating set of G [6], and so $i(G)$ is also called the *independent domination number* of G, and $\alpha(G)$ the *independence number* of G. A *private dominating set* S is a dominating set such that for every $u \in S, \exists v \notin S$ such that $N[v] \cap S = \{u\}$. The *private domination number* is the minimum cardinality of such a set, and is written as $\gamma_{pvt}(G)$. A well known result of Bollobas and Cockayne [9] established that, for graphs with no isolated vertices, $\gamma(G) = \gamma_{pvt}(G)$.

A *secure dominating set* S is a dominating set which satisfies the following property: $\forall t \notin S,\ \exists s \in N(t) \cap S$ such that $\{S \setminus \{s\}\} \cup \{t\}$ is a dominating set. A common analogy used in the discussion of secure dominating sets is that those members $s \in S$ are guards supervising the locations $t \notin S$. S is then secure dominating if every location t has some adjacent guard which can "relocate" to t, with the resulting configuration remaining a dominating set. As seen in Figure 1.2, the same underlying graph can produce different values for γ and γ_s (having values 1 and 2 respectively). By definition, all secure dominating sets are dominating sets, and so $\gamma \leq \gamma_s$ for any graph.

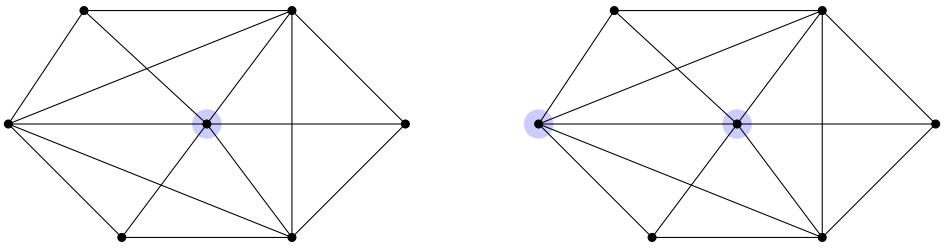

Figure 1.2: An comparison of a minimum plain dominating set (left) and a minimum secure dominating set (right)

This variant of domination was introduced and explored in a 2005 paper by Cockayne [20]. The *secure domination number* $\gamma_s(G)$ is the number of vertices in the smallest secure dominating set of G. It is clear that calculating γ_s

is NP-hard in general, and Merouane and Chellali [44] proved that calculating $\gamma_s(G)$ is NP-hard even for bipartite and split graphs. Despite this, there have been various efforts to calculate the secure domination number exactly. These approaches can be broadly distinguished as either Branch-and-Bound / Branch-and-Reduce algorithms [16, 51] or linear programming approaches [17]. In particular, Burdett and Haythorpe [14] developed a binary programming formulation of the secure domination problem, correctly calculating γ_s for a collection of test graphs of manageable size, as well as identifying a mistake in the results of [17].

1.3 Maker-Breaker games

Maker-Breaker games were first described by Erdős and Selfridge [24]. These games are commonly represented as a hypergraph with vertex set V and a finite set $\mathcal{F} \subseteq 2^V$ of hyperedges. In this context, V is often referred to as the board of the game. The game is typically played by two players (referred to as Maker and Breaker) who take turns claiming vertices from V. Maker's objective is to claim all of the vertices in some hyperedge $e \in \mathcal{F}$, winning the game if successful. The hyperedges in $\mathcal{F}$ are said to make up the *winning sets* of the game. Breaker's objective is to prevent Maker's victory, that is, claim at least one vertex in every winning set. If Breaker achieves this, they are said to win the game. Note that Maker and Breaker's goals are complementary, meaning a draw is not possible. Maker-Breaker games belong to the field of study known as *positional games*, also called *combinatorial game theory*. Combinatorial game theory is a vast and mathematically complex branch of study, and a thorough discussion of it is beyond the scope of this thesis; however, we refer the interested reader to the seminal publication by Hefetz et al.

A common assumption made in this area of study is that both players have

infinite computational capabilities [39], and a common research question is for which of the players a *winning strategy* exists. Such a strategy means that for any sequence of moves their opponent makes, there is always some response the player can make to ensure their victory. A logical result known as Zermelo's Theorem ensures that for any Maker-Breaker game, exactly one player must have a winning strategy [7].

Maker-Breaker games are perfect-information games, meaning that each player has complete knowledge of previous moves and their opponent's objective. Hypothetically, to check whether a player P has a winning strategy one could examine every possible sequence of moves that a game permits. With this goal, the initial position of the game can be thought of as the root of a tree. The branches from this root correspond to the different possible first moves a player can make (choices of the board's vertices), with subsequent moves branching from them in turn. This structure is called the *game tree*, and the playing of the game can be thought of as the construction of a path from the root of the game tree to a leaf, with each vertex in the path being a child of the previous vertex. The leaves of this tree are the end states of the game, when a winner has been determined.

Suppose P is playing against another player called Q. We can mark these leaves according to their respective winners. Decrementing in depth from the leaves, we can consider the decision making process of the players at each turn. Recalling that Maker-Breaker games are perfect information games, if a player has the opportunity to navigate to one of their winning end states, it is sensible that they would do so. Furthermore, if all children of a given node have the same marking, that node should also have the same marking. Applying these rules for all nodes is referred to as traversing the game tree. We'll now consider a simple example of this process with the game tree

displayed in Figure 1.3.

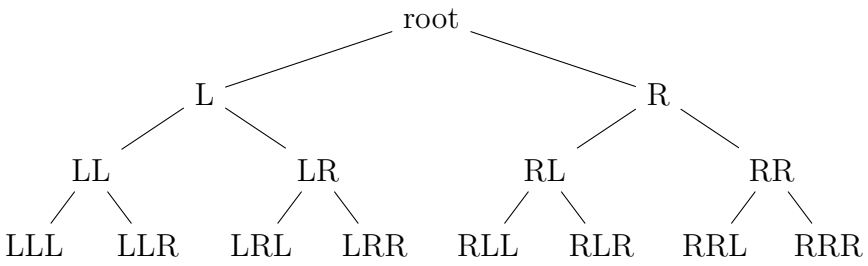

Figure 1.3: A simple game tree corresponding to a three turn game, each turn corresponding to a binary choice between "left" or "right". The game history is labeled at each node

Suppose that we designate the Maker's winning sets to be $\mathcal{F}$= {LLL, LLR, LRL, RLL}. We may then label these end states accordingly as in Figure 1.4, and work backwards to determine the game outcome.

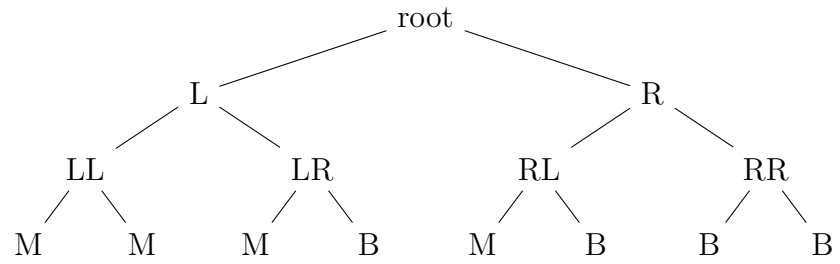

Figure 1.4: The first step of labeling the game tree, beginning at leaf vertices

Our marking of non-leaf vertices is dependent on which player moves first. The tree displayed in Figure 1.5 corresponds to Maker moving first. Hence, the odd levels correspond to Maker's turn, while the even levels correspond to Breaker's turn. For each vertex, the corresponding player makes the choice leading to their victory if possible, and hence the vertex is labelled as a win for that player if and only if its level corresponds with their turn, and at least one of its child vertices is labelled as a win for them. Repeating this process from the bottom level up, the entire tree can be labelled, and the player with a winning strategy is revealed.

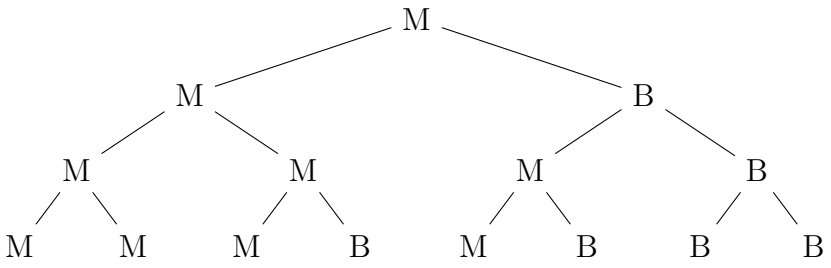

Figure 1.5: The fully enumerated game tree for Maker moving first

By inspecting the root of this tree, we observe that when Maker moves first, they have a winning strategy. This is no longer the case, however, if Breaker moves first. In this case, the resulting marked tree is displayed in Figure 1.6.

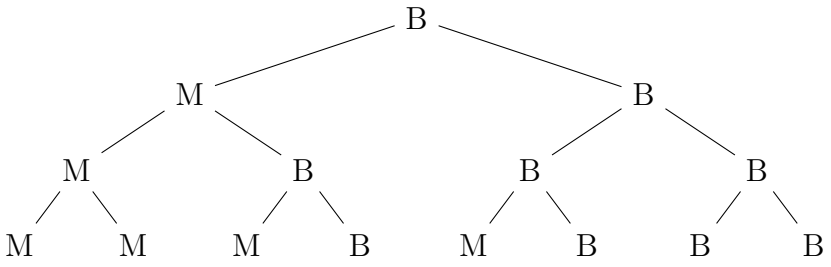

Figure 1.6: Enumerated game tree for Breaker moving first

Here we see when Breaker moves first, they have a winning strategy. This example is deliberately simple to illustrate the backtracking technique. In it, Maker's objective can be described as "steering left" twice in the game tree. The actual results we obtain regarding winning strategies in this instance are somewhat obvious, as when Maker moves second they only have one turn. The advantage of moving first, however, does extend to non-trivial games. In particular, Hefetz et al. proved a valuable result for general Maker-Breaker games which we restate here, as we will use it several times in this thesis.

Proposition 1.1. *(restating of 2.1.6 in [39]) Let $(V, \mathcal{F})$ be a hypergraph. In the Maker-Breaker game over $\mathcal{F}$, a player has a winning strategy moving first if they have a winning strategy moving second.*

The size of the game tree is exponential in the number of possible moves at each turn, and so "brute-force" traversal quickly becomes intractible. Consequently, a lot of results on winning strategies make use of probabilistic arguments to specify conditions which ensure the existence of a winning strategy, rather than attempting to explicitly construct the strategy itself [7, 1].

A seminal result in this field which describes both a sufficient condition for a winning strategy, and provides the strategy itself, is known as the Erdős-Selfridge criterion [24]:

In the Maker-Breaker game played over $\mathcal{F}$, for some hypergraph $(X, \mathcal{F})$,

$$\sum_{A \in \mathcal{F}} 2^{-|A|} < \frac{1}{2} \implies \text{Breaker has a winning strategy.} \tag{1.1}$$

$$\text{Also, } \sum_{A \in \mathcal{F}} 2^{-|A|} < 1 \implies \text{Breaker has a winning strategy moving first.} \tag{1.2}$$

A loose interpretation of this criterion is that if there are sufficiently few winning sets, or if the winning sets are sufficiently large enough, then Breaker always has the ability to claim vertices blocking them all before Maker can guarantee a win. The proof of this result also provides a method for constructing such a winning strategy for Breaker, however in practice the method is often intractable. We will discuss this method further in Section 3.1.

If a subset of the game board can be partitioned into pairs such that each winning set contains at least one of the pairs, Breaker can win the game by

following what is called a *pairing strategy* [39]. According to this strategy, whenever Maker claims a vertex, Breaker responds by claiming the other vertex of the pair containing that Maker's previous choice. This way, Breaker will successfully occupy every winning set, winning the game.

Another common concept in the field of combinatorial games is that of *strategy stealing*. Privately invented by John Nash [7] and first published by Hales and Jewett [33], the idea is that a player with knowledge of their opponent's strategy will pre-emptively "steal" the vertex their opponent would have wanted to claim next. Strategy stealing has been used to show the existence of winning strategies for various games, without ever specifying what the strategy is [36, 33].

There has been recent interest in games on graphs for which the winning sets are exactly the dominating sets of the graph, with numerous different (but related) games being referred to as *the domination game*. The domination game described by Brešar et al. has come to be referred to as the standard version [12, 23]. In this game, one player is referred to as Dominator, and has the objective of constructing a dominating set in as few moves as possible. Their opponent is referred to as Staller, and has the objective of maximising the amount of moves before a dominating set is constructed by Dominator. An additional requirement of the game is that each player may only claim vertices which increase the total number of covered vertices (that is, those vertices adjacent to either Dominator or Staller's claimed vertices). When both players adopt an optimal strategy, the number of vertices chosen is called *game domination number*, and is written $\gamma_g(G)$. Another useful concept introduced by Brešar et al. is that of the *imagination strategy*. To employ this strategy, one player "imagines" another game being played, and plays an optimal strategy according to this imagined game.

Recently, a Maker-Breaker version of the domination game was described by Duchêne et al. [23]. Adopting the naming convention of Brešar et al., this version assigns Dominator's objective as to simply construct any dominating set, and Staller's to prevent any such construction. Players must only choose free vertices (those not previously chosen) but beyond that there are no restrictions on which vertex a player may choose. Several results from Duchêne et al. are relevant to this thesis, and will be addressed in Chapter 3. It is easy to extend this concept to other variants of domination. In this thesis we will consider secure domination in this context, which has previously not been explored in the literature.

The remainder of this thesis is laid out as follows. In Chapter 2, we review the existing algorithmic approaches to solving the domination problem and the secure domination problem. In Chapter 3, we look more closely at the Maker-Breaker version of the domination game, and in Chapter 4, we describe a simulation approach to analysing the game. In Chapter 5 we present experimental results arising from the simulation approach, and discuss their implications. Finally, in Chapter 6 we give conclusions and discuss future research work arising from this thesis.

Chapter 2

Algorithmic Approaches to Finding Dominating Sets

2.1 Determining the (Secure) Domination Number Algorithmically

As mentioned in the previous chapter, finding the domination number and the secure domination number are both NP-hard problems. Furthermore, it is NP-complete to check whether a (secure) dominating set exists of cardinality at most k, for $k \in \mathbb{Z}^+$. Any graph $G = (V, E)$ contains a dominating set. If nothing else, we can always take the whole set of vertices V and it will satisfy the definition of a dominating set, and indeed a secure dominating set. Checking whether a given subset of vertices constitutes a dominating set is also efficient. We need only iterate over each of the subset's members and record each of their neighbouring vertices. After this, if the total set of neighbouring vertices is equal to the entire vertex set V, then the subset is dominating. This procedure has a runtime $\mathcal{O}(|V|\Delta)$, where Δ is the maximum degree in the graph. Checking secure domination requires additional constraints to be checked, but this is still achievable in polynomial time. For

checking if a set $X \subset V$ is secure dominating, Burger et al. [17] outlined an algorithm composed of three tests (arranged in order of computational complexity) on a vertex set partitioned into five components. Notably, the first of these test is to determine whether X is a plain dominating set.

In an attempt to solve the dominating set problem or one of its variants, an unsophisticated approach might be to construct such a dominating set and then show that no smaller set can satisfy the corresponding variant of domination. However, developing an algorithm to show no such set exists remains a computationally difficult task. There are various algorithmic approaches to solving the (secure) domination problem. Broadly speaking, these approaches can be categorised as either exact algorithms, approximation algorithms, or heuristic algorithms. Exact algorithms are guaranteed to solve the problem to optimality, and can be expected to produce an example of a minimum (secure) dominating set, but are inefficient to run. The brute force method to find the domination number $\gamma(G)$ has exponential complexity and is described below. Algorithms that improve on the brute-force solution for calculating $\gamma(G)$ exactly have been developed, but nonetheless still take exponential time in the worst case [27]. Exact algorithms with the purpose of enumerating every possible (secure) dominating set are similarly intractible. Approximation algorithms are not guaranteed to solve the problem to optimality, but instead guarantee that they will find a solution which lies within a bound (usually a multiplicative factor) of the optimal solution in a polynomial amount of time. Finally, heuristic algorithms are designed to be efficient to run, but give no guarantees at all about solution quality. As such, it is common for heuristics to be run many times, with the best solution so produced taken as the final output.

The most straightforward exact algorithm is to simply consider all possible

subsets of the vertices for sizes $k = 1, 2, ...$, until a valid (secure) dominating set is found. This approach could also be adapted to enumerate all minimal dominating sets. However, the runtime of this procedure is $\mathcal{O}(|V|\Delta 2^{|V|})$, and so this very quickly becomes intractable and is not useful for anything beyond very small graphs.

The first improvement on the above naive approach was made by Grandoni [31], and various others have since been developed [25, 48, 50]. Currently, the best-known exact algorithm for plain domination is by van Rooij and Bodlaender [51] which solves the dominating set problem in $\mathcal{O}(1.4969^n)$ time and polynomial space. For the plain domination problem, a greedy approximation algorithm exists with an $ln(n + 1)$ margin of error [41].

In the following sections, we will highlight two algorithmic approaches in particular which will be relevant to the later chapters of this thesis. First, both the dominating set problem and the secure dominating set problem can be formulated as mixed-integer linear programs (MILPs), allowing them to be passed to a highly optimised solver. While still NP-hard to solve, MILP problems have been the subject of intensive study for several decades [8] and have the benefit of advanced tooling such as IBM's CPLEX being available to address them. Secondly, we will cover a recently described heuristic approach, which is based on iteratively solving the relaxation of the MILP formulations, and using the outputs to select vertices for inclusion in a (secure) dominating set.

2.2 Mixed Integer Linear Programming (MILP) Approaches

The dominating set problem admits a binary programming formulation as follows. First, assign binary variables for each of the n vertices in the graph, denoted x_i, for $i = 1, 2, \ldots, n$. For these variables, a value of 1 represents membership in the dominating set, and a value of 0 represents non-membership. Denote by $\mathbf{x}$ the vector containing all such x_i. The dominating set problem is then equivalent to solving:

$$\min_{\mathbf{x}} \sum_{i=1}^{n} x_i$$

subject to

$$\sum_{j \in N[i]} x_j \quad \geq \quad 1, \qquad \forall i = 1, 2, \ldots, n, \tag{2.1}$$

$$x_i \quad \in \quad \{0, 1\}, \qquad \forall i = 1, 2, \ldots, n. \tag{2.2}$$

The intuition behind constraint (2.2) is that each vertex in the graph should have at least one member of the dominating set in its closed neighbourhood.

Although the above formulation includes binary variables, CPLEX is nonetheless able to successfully solve the problem for graphs with up to several hundreds of vertices. Burger et al. [17] extended upon this and developed a binary programming formulation for the secure domination problem. In addition to the x_i variables from the previous formulation, the authors introduced new binary variables z_{kl}, for $k, l = 1, ..., n$ and $k \neq l$. These variables are assigned a value of 1 to indicate that a guard at vertex l is assigned to move to a currently unoccupied vertex k in the current configuration of the graph, and are assigned a value of 0 otherwise.

The authors then proposed the following formulation for secure domination:

$$\min_{\mathbf{x}} \sum_{i=1}^{n} x_i$$

subject to

$$\sum_{j=1}^{n} a_{ij}x_j \geq 1, \qquad \forall i = 1, ..., n, \tag{2.3}$$

$$\sum_{\substack{l=1 \\ l\neq k}}^{n} z_{kl} \geq 1 - x_k, \qquad \forall k = 1, ..., n, \tag{2.4}$$

$$a_{kl}(x_l - x_k + 1) \geq 2z_{kl}, \qquad \forall k, l = 1, ..., n, \ \ l \neq k, \tag{2.5}$$

$$\sum_{\substack{j=1 \\ j\neq k \\ j\neq l}} a_{ij}x_j + a_{ik} \geq z_{kl}, \qquad \forall i, k, l = 1, ..., n, \ \ l \neq k. \tag{2.6}$$

This formulation involves n^2+n binary variables and n^3+n^2+2n constraints.

Interestingly, Burger et al. [17] noted that simply submitting this formulation to CPLEX provided superior performance to the previously best known exact algorithms for the secure dominating set problem, and suggested that an improved formulation may produce even greater performance. Subsequently a superior formulation was provided by Burdett and Haythorpe in 2020 [14], which we outline now.

Like that of Burger et al., Burdett and Haythorpe's formulation also used binary x_i variables to denote set membership. However, instead of the binary variables z_{kl}, Burdett and Haythorpe instead used continuous variables $y_{jk} \geq 0$ to represent whether a guard at vertex j should be assigned to move to vertex k. If so, then $y_{jk} > 0$, otherwise $y_{jk} = 0$.

Burdett and Haythorpe then proposed the following formulation for the se-

cure domination problem.

$$\min_{\mathbf{x}} \sum_{i=1}^{n} x_i$$

subject to

$$\sum_{j \in N[i]} x_j \geq 1, \quad \forall i = 1, ..., n, \tag{2.7}$$

$$y_{ij} - x_i \leq 0, \quad \forall i = 1, ..., n, \ \ \forall j \in N(i), \tag{2.8}$$

$$\sum_{k \in N[i]} x_k - \sum_{k \in N(i) \cap N(j)} y_{kj} \geq 1, \quad \forall i, j \text{ such that } dist(i, j) = 2, \tag{2.9}$$

$$\sum_{i \in N(j)} y_{ij} + x_j = 1, \quad \forall j = 1, ..., n, \tag{2.10}$$

$$y_{ij} \geq 0, \quad \forall i = 1, ..., n, \ \ \forall j \in N(i), \tag{2.11}$$

$$x_i \in \{0, 1\}, \quad \forall i = 1, ..., n. \tag{2.12}$$

The authors implemented the above formulation in CPLEX. On standard hardware, this allowed exact answers to be produced for a variety of graph families with up to around 100 vertices. Burdett and Haythorpe's formulation comes with some advantages over its predecessor. Firstly, since the additional variables are continuous, the formulation has n^2 fewer binary variables than that of Burger et al. and is accordingly much faster to solve. Secondly, it involves a significant reduction in the number of constraints needed, now $\mathcal{O}(m + n^2)$, where m is the number of edges in the graph. As such, it constitutes the best known exact algorithm for secure domination in the literature.

2.3 Heuristic Algorithms

There are many heuristic algorithms for the different variants of the domination problem [3, 15, 37]. A very general procedure for constructing a

minimal (secure) dominating set is described below in Algorithm 2.1. In the following, we will discuss an adaptation of this procedure which allows many different strategies to be defined for a Maker-Breaker domination game context. This procedure consists of two phases. In the first phase, vertices are iteratively added to the set S, and we call this the Bottom-Up phase. In the second phase, vertices are considered for removal from S, and this is called the Top-Down phase.

Algorithm 2.1 General procedure for constructing minimal (secure) dominating sets

Inputs: An empty graph
Outputs: A minimal (secure) dominating set

$S = \{\}$
while S is not (secure) dominating **do**:
    Use a selection function to pick a vertex $v \notin S$
    Add v to S
for v in S **do**
    if $S \setminus \{v\}$ is (secure) dominating **then**
        Remove v from S

return S

The majority of heuristics for domination fit into the framework of Algorithm 2.1. Then, the major point of distinction between them is the selection functions used to pick vertices for addition or removal. Selection functions can range from trivial, random choices, to more sophisticated calculations. For example, one such sophisticated calculation for the Bottom-Up phase selects the vertex with the most previously uncovered neighbours. This is repeated until the Bottom-Up phase is completed, and the number of vertices in the dominating set can be inspected. Using this particular selection function is actually known to be an approximation algorithm, and is close to the best possible approximation algorithm for the case of general graphs [47], although it is outperformed by approximation algorithms tailored for specific families of graphs.

In the upcoming adaptations, our focus will be on modifying the selection function in the Bottom-Up phase. This is due to the nature of Maker-Breaker games, where both players start with no claimed vertices and constructively add to their set of claimed vertices throughout the game's duration. Since there is no functionality in Maker-Breaker games for vertices to be rescinded by a player after claiming them, the Top-Down phase is not relevant here. In any event, the primary purpose of the Top-Down phase is to ensure the constructed (secure) dominating set is minimal, which is not something we need to consider in the Maker-Breaker version of the domination game.

In [13], it was proposed that the MILP formulation of Burdett and Haythorpe described in the previous section can be adapted to fit the framework of Algorithm 2.1, to produce a heuristic for secure domination. Burdett proposed that the most significant effort should go towards the Bottom-Up phase, with the rationale being that if that phase is done well, then the set S may already be close to optimal before the Top-Down phase begins. At each iteration of the Bottom-Up phase, the MILP formulation is constructed, along with additional constraints $x_v = 1$ for any vertices v which are already contained in S. Then, the binary constraints are relaxed to produce a linear program. The linear program is solved (for instance, using CPLEX), with one of two outcomes occurring. If the solution has all binary values for the x variables, then the unit values correspond to the best possible secure dominating set subject to the choices already made, and so this secure dominating set is returned. Alternatively, if the solution has some x values strictly between 0 and 1, then the solution values themselves aid in selecting the next vertex. Those variables with a value closer to 1 can be thought of as being more desirable selections to form a dominating set, and those close to 0 as less desirable. By removing the values corresponding to vertices already selected, and normalising the remaining values, a probability mass function (PMF)

is produced. An observation from the probability mass function can then inform the next vertex to add to S, with the vertices corresponding to higher solution values being selected with higher probability. Burdett also proposed transforming the PMF, for instance raising the solution values to a constant power (selected in advance), manipulating the relative weight given to the higher or lower values. Once the new vertex is selected and added to S, the formulation is updated and the process begins again, continuing until a secure dominating set is found.

Burdett demonstrated that this heuristic, although an order of magnitude slower than some of the best previous approaches, was able to find significantly higher quality solutions. There was also some work done exploring different options for the Top-Down phase which resulted in some minor additional improvements. In Chapter 3, we will discuss the recently introduced concept of the Maker-Breaker domination game, and introduce the secure version of this game. Subsequently, in Chapter 4, we will detail how an analogous approach to that proposed by Burdett can be used to aid the Maker (or the Breaker) in their decision-making.

Finally, we conclude this chapter by noting that the approach advocated by Burdett is applicable beyond just secure domination. In fact, it can be applied to any variant of domination for which there exists a mixed-integer linear programming formulation, as long as the following property is true: any valid solution for the relaxation in which the x_i variables are binary corresponds to a dominating set of the desired variant. We note that this excludes the secure domination formulation by Burger et al. since it is possible to obtain a valid solution for the relaxation where all the x_i variables are binary but the y_{ij} variables are not, and this may not correspond to a secure dominating set. However, the standard formulation for the dominating set

problem given at the start of Section 2.2 meets this condition, and hence a heuristic for plain domination could be constructed following this approach.

Chapter 3

Maker-Breaker domination games

3.1 Definition and Key Results

In Section 1.3, we discussed Maker-Breaker games in which two players, Maker and Breaker, take turns claiming elements from the game board. Maker is successful if they are able to claim every element from one of the *winning sets*, while Breaker is successful if they prevent Maker from doing so. It is common for Maker-Breaker games to be played on graphs, in which case the elements being claimed could be edges or vertices (or both). Then, the winning sets may correspond to various structures or objects within the underlying graph. For example, in [18], Chvátal proposes the *Hamiltonicity game* in which the elements being claimed are edges, and the winning sets correspond to the Hamiltonian cycles in the underlying graph.

Along the same lines, in 2020 Duchêne et al. [23] proposed the *Maker-Breaker domination game*. In this context, the elements being claimed are vertices in a graph, and the winning sets $\mathcal{F}$ correspond precisely to the dominating sets of the graph. As such, Maker is attempting to collect vertices constituting

a dominating set while Breaker is trying to prevent it. Indeed, Duchêne et al. refer to Maker as *Dominator*, and Breaker as *Staller* in this context. An analagous game could be proposed for variants of domination, and indeed, some publications have already appeared considering total domination in this context [29, 11]. In this thesis, we consider secure domination in this context, which thus far appears not to have been discussed in the literature. To avoid confusion, we will often refer to the Maker-Breaker domination game as the *Maker-Breaker plain domination game*, in order to contrast it with the *Maker-Breaker secure domination game*. In their paper, Duchêne et al. also proved several results regarding the Maker-Breaker plain domination game, which are outlined below.

If the Erdős-Selfridge criterion is met, Breaker has a winning strategy. This strategy is given by Hefetz et al. in their proof [39] of the Erdős-Selfridge criterion, which we now paraphrase. To employ their winning strategy, Breaker examines every winning set that is still *active*, that is, those winning sets for which no vertices have yet been claimed by Breaker, and so are still a feasible way for Maker to win the game. Those winning sets that do contain a vertex which has been claimed by Breaker are herein referred to as *neutralised*, or *deactivated*. Breaker considers the scenario in which Maker plays at random, and calculates a quantitative measure of the "danger" of a winning set being fully claimed by Maker:

$$\texttt{danger}(\mathcal{H}) = \sum_{A\in\mathcal{H}, A\cap B=\emptyset} 2^{-|A\setminus M|} \tag{3.1}$$

Here, $\mathcal{H}$ is the hypergraph representation of the game, with M and B representing the claimed vertex sets of Maker and Breaker respectively. At each turn, Breaker should choose a vertex that gives the biggest reduction in value for this `danger` function. In the case of multiple such candidate vertices, Breaker may choose to claim any of these arbitrarily.

If the Erdős-Selfridge criterion is met, then the above greedy algorithm is guaranteed to be a winning strategy for Breaker, with execution time being polynomial in the number of winning sets (not including the time taken to enumerate the winning sets in the first place). Its usefulness is furthered in the context of Maker-Breaker plain domination games, due to the fact that these games have equivalent formulations for which either player can be assigned the role of Breaker.

In particular, Duchêne et al. applied the Erdős-Selfridge criterion to the context of Maker-Breaker domination games. Their discussion assumed the following result, stated without proof in [23]. For the sake of completeness, we provide a proof here.

Proposition 3.1. *Breaker wins the plain domination Maker-Breaker game* $\iff$ *Breaker fully claims the closed neighbourhood of a vertex* v*.*

Proof. $\implies$: Assume Breaker has made their final move and won the game. This means that there is some vertex which is adjacent to none of Maker's vertices. Call this vertex v. As Breaker has won, the neighbourhood of v must be fully occupied by Breaker, otherwise Maker could still claim one of v's unoccupied neighbours.

$\impliedby$: Breaker claiming the entire closed neighbourhood of v means Maker has claimed no adjacent vertex to v and is no longer able to, so cannot form a dominating set. □

With this result in hand, Duchêne et al. were able to reformulate the Maker-Breaker domination game M as an equivalent game M', with the player in the role of Maker in M acting as Breaker in M', and vice versa. The winning sets of M' are those subsets of V containing an *enclave*, that is, those sets

of vertices containing at least one vertex's entire closed neighbourhood. The authors were then able to apply the Erdős-Selfridge criterion to M' to obtain a winning condition for Maker in the original game M.

Proposition 3.2. *[23] Let G be a starting position of the Maker-Breaker domination game M and let δ be the minimum degree of G. If $|V| < 2^\delta$ then Maker has a winning strategy for the Maker-Breaker domination game on G, even if they are the second player to move.*

As the Erdős-Selfridge criterion provides an explicit winning strategy when its conditions are met, this result provides an explicit winning strategy for Maker on M. Even in situations for which the sufficient condition of the Erdős-Selfridge criterion are not met, the strategy provided may still prove to be a winning one. In Section 4.3, we will describe an adaptation of this strategy that makes use of the above reformulation.

Duchêne et al. also considered the concept of a *pairing dominating set*. A graph $G = (V, E)$ admits a pairing dominating set if there exists a collection of vertex pairs $\{\{u_1, v_1\}, \ldots, \{u_k, v_k\}\}$ where all the vertices are distinct, and such that the union of the intersection of each pair's neighbourhoods is V, i.e. $V = \cup_{i=1}^{k} N[u_i] \cap N[v_i]$. If so, then there exists a winning strategy for Maker. Specifically, Maker can win as long as it obtains at least one vertex from each of the pairs. Since the vertices in the pairs are all distinct, Maker can just observe what move Breaker makes at each iteration. If Breaker selects a vertex from one of the pairs, Maker can immediately select the other vertex if they haven't already done so; otherwise, Maker can select any vertex. Unfortunately, the authors also proved that it is NP-complete to determine whether G admits a pairing dominating set.

Finally, Duchêne et al. also proved that deciding the outcome of a Maker-Breaker domination game position is PSPACE-complete on bipartite and

chordal graphs.

3.1.1 Maker-Breaker Secure Domination Game

We now formally define the Maker-Breaker secure domination game as follows.

Definition 3.3. For a given graph G with vertex set V, the Maker-Breaker secure domination game is a Maker-Breaker game in which the underlying hypergraph has vertex set V, and the set of hyperedges F contain all secure dominating sets of G.

We now briefly consider the results of Duchêne et al. in the context of the Maker-Breaker secure domination game. Unlike for plain domination, Proposition 3.1 does not hold for secure domination. To be precise, the result holds in only one direction; that is, if Breaker can fully claim an enclave around a vertex v, then they win the Maker-Breaker secure domination game. However, it is possible for Breaker to win without obtaining an enclave. Specifically, Proposition 3.4 indicates the properties which allow Breaker to win the Maker-Breaker secure domination game.

Proposition 3.4. *For the Maker-Breaker secure domination game played on graph G, if M is the set of vertices claimed by Maker, then Maker has not yet won the game if either of the following are true.*

(i) M is not a dominating set on G, or

(ii) There exists $u, v, w \in V$ with $dist(u, v) = 2$, and $\{N[u] \cup N[v]\} \cap M = w$.

Since, by definition, any secure dominating set is also a dominating set, it is no harder for Breaker to win an instance of the Maker-Breaker secure domination game [7] than it is for them to win the equivalent instance of the Maker-Breaker plain domination game. The strategy of having the Maker

obtain a vertex from each vertex pair in a pairing dominating set is not sufficient to ensure a Maker win in the Maker-Breaker secure domination game. Indeed, the example given in Duchêne et al. is a good counter-example. They considered the graph displayed in Figure 3.1 and highlighted $(u_1, v_1), (u_2, v_2), (u_3, v_3)$ as a pairing dominating set. Any combination of single vertices from the three pairs is indeed a dominating set; however, none of them is a secure dominating set.

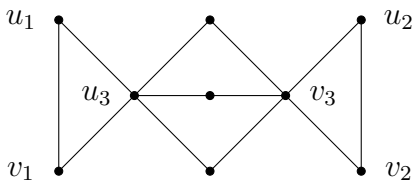

Figure 3.1: An example of a graph which admits a pairing dominating set, as given by Duchêne et al. [23]

However, that is not to say that this kind of approach is without merit. In many cases a pairing dominating set may indeed provide a winning strategy for Maker. In the below result, we consider perfect matchings, which are themselves special cases of pairing dominating sets.

Lemma 3.5. *If the Maker-Breaker secure domination game is being played on a graph G, and G contains a perfect matching, then there is a winning strategy for the Maker regardless of who plays first.*

Proof. Consider any perfect matching in G, and suppose that the Breaker plays first. At each iteration, Maker observes which vertex is selected by Breaker, and they then select the vertex which is paired with it in the perfect matching. Since G contains a perfect matching, it has an even number of vertices, and hence the final move is made by Maker. Hence, at the end of this process, Maker certainly possesses vertices which constitute a dominating set, as per the discussion on pairing dominating sets in Duchêne et al. Denote by S the set of vertices possessed by Maker. Then all that remains is to ensure that S is also secure.

Consider any vertex v not claimed by Maker, and denote by w its pair in the perfect matching. It is clear that $(S \cup w) \setminus v$ is also a dominating set, by the same argument as in the previous paragraph. Hence, S is also a secure dominating set, and hence this is a winning strategy for Maker.

Finally, if Maker plays first, the result from Proposition 1.1 implies there is an analagous winning strategy for Maker. □

Corollary 3.6. *If the Maker-Breaker secure dominating game is being played on a graph $G = (V, E)$, and there is a vertex $v \in V$ such that $G \setminus v$ contains a perfect matching, then there is a winning strategy for Maker if Maker plays first.*

Proof. Maker begins by claiming vertex v, after which the remaining situation is equivalent to that in Lemma 3.5. □

In the following section, we consider a very simple instance of a Maker-Breaker plain domination game. Specifically, we consider the 3×3 grid, and prove that a specific strategy guarantees a win for Maker. The 3×3 grid is a graph which satisfies Corollary 3.6, and so can be won by Maker if they play first, even in the case of the Maker-Breaker secure domination game. However, there is no immediate conclusion when Maker plays second. As will be seen, the proof is surprisingly complex and involves the consideration of many cases, even though the underlying graph is so small and we are only considering plain domination. This highlights the complexity of analysing these games directly, and motivates our upcoming approach of using simulation techniques to explore more complicated instances.

3.2 3×3 Grid Example

In the upcoming Chapter 4, we describe various strategies that Maker or Breaker could use to try to win an instance of Maker-Breaker domination game. These strategies are broadly sensible and defensible approaches, but establishing whether or not they can guarantee a win for a given instance can be challenging. We will consider one of our strategies, which we call `neighbourhoodWatch` (See Section 4.3.4), and show that it constitutes a winning strategy for Maker in the case of the 3×3 grid. To aid the upcoming proof, we establish some notation in advance. Throughout the proof it will sometimes be necessary to refer to specific vertices, and so we will use the labelling of the grid given in Figure 3.2:

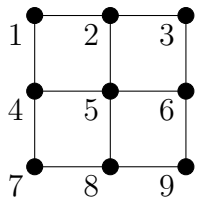

Figure 3.2: The labelling of the 3×3 grid used in this section

Although the theorem will be posed for Maker, it will convenient for us to think of winning sets from the perspective of Breaker. As mentioned in Proposition 3.1, a win for Breaker on a graph G is equivalent to Breaker having fully claimed the closed neighbourhood $N[v]$ for at least one vertex $v \in V(G)$. Hence, there are nine such minimal winning sets for Breaker in the 3×3 grid[1]. We will refer to these as A^i for the vertices $i = 1, \ldots, 9$ respectively. We will refer to A^i as having *Form* k if $|N[i]| = k$; hence, A^i has Form 3 for $i = 1, 3, 7, 9$, Form 4 for $i = 2, 4, 6, 8$, or Form 5 for $i = 5$.

For a particular game state and a given (Breaker) winning set A we will

[1]We note that there are more than nine winning sets for Breaker, since any superset of a closed neighbourhood is also a winning set. However, the nine closed neighbourhoods are the minimal winning sets for Breaker.

use $m(A)$ and $b(A)$ to refer to the number of vertices of A which have been claimed by Maker and Breaker respectively. Note that the game state itself is abstracted from the notation here; it will always be clear from context what the game state is. Finally, we define $c(A)$ to be a measure of how close Breaker is to fully claiming A, noting that if Maker claims even a single vertex of A then it is impossible for Breaker to claim fully A. Hence:

$$c(A) := \begin{cases} |A| - b(A), & \text{if } m(A) = 0, \\ \infty, & \text{otherwise.} \end{cases}$$

If $c(A) = \infty$ we will refer to A as having been *neutralised* by Maker. If all $c(A) = \infty$ then Maker has won the game, whereas if $c(A) = 0$ for any A then Breaker has won the game. If there is no winner yet, then there must be some finite, positive value $c_{min} = \min_A c(A)$, equal to the minimum number of vertices Breaker still needs to claim to win the game. We will refer collectively to those winning sets A such that $c(A) = c_{min}$ as being the *closest winning sets* (to being fully claimed by Breaker) at that game state.

`neighbourhoodWatch`, which will be introduced in more detail in Section 4.3.4, can be summarised as follows. Choose the available vertex v which is contained in the greatest number of closest winning sets. If there is a tie, break it by choosing the vertex which neutralises the most currently active (Breaker) winning sets. If there is still a tie, choose from the remaining choices uniformly by random.

We begin by establishing a minor result which will be useful in the proof of the upcoming theorem.

Lemma 3.7. *Suppose Breaker plays first in the 3×3 grid, and Maker follows the `neighbourhoodWatch` strategy. Then Maker will not claim a corner vertex or the middle vertex on their first move.*

Proof. Suppose that Breaker has made their first move, and now it is Maker's turn. Suppose that Maker chooses a corner vertex. Note that in the first turn, choosing a corner vertex neutralises only three (Breaker) winning sets (the minimum possible amount). Hence, this choice would only be made if corner vertices were the only ones contained in the most closest winning sets. However, this is impossible; it can easily be checked that each winning set has at least two non-corner vertices, and since Breaker has claimed only one vertex, at least one more must remain. Hence, Maker must not claim a corner vertex. Then, suppose Maker claims the middle vertex; clearly, Breaker must not have claimed the middle vertex in their first turn. However, in this case, there must be at least one winning set A for which $c(A) = 2$ and A does not contain the middle vertex. Hence, the middle vertex is not contained in the most number of winning sets, so this situation is also impossible, completing the proof. □

We are now ready to establish the result.

Theorem 3.8. *`neighbourhoodWatch` constitutes a winning strategy for Maker in the Maker-Breaker plain domination game on the* 3×3 *grid.*

Proof. From Proposition 1.1, if Maker has a winning strategy as the second player, they can extend this to a winning strategy as the first player. Hence, we will assume that Breaker plays first.

Now, suppose that the game has played out and Breaker has won, with Maker utilising the `neighbourhoodWatch` strategy; we will assume that the game immediately stops once Breaker has claimed an enclave, say A. That is, after Breaker's final move we have $c(A) = 0$, but prior to that move $c(A) > 0$. Hence, immediately before Maker's final move we must have had $c(A) = 1$. Furthermore, before Maker's final move, there must be some other winning set $B \neq A$ with $c(B) = 1$, or else Maker's strategy would have instructed it

to neutralise A. Note that since there are only nine vertices in the 3×3 grid, Breaker can only have claimed at most four vertices prior to Maker's final move. From here, there are two possibilities; either $A \cap B = \emptyset$, or $A \cap B \neq \emptyset$.

We first examine the case of $A \cap B = \emptyset$. By analysing the three different forms of winning sets, it is clear that neither A nor B can have Form 5, or else they will intersect with the other. Likewise, they cannot both have Form 4. Hence, there are only two possibilities; either both have Form 3, or one has Form 3 and the other has Form 4. However, in the latter case, it implies that Breaker must have made at least five moves already before Maker's last move, which as mentioned above is impossible. Hence, both A and B must have Form 3. Up to symmetry, this is only possible in the configuration displayed in Figure 3.3. Hence we have $c(A) = c(B) = 2$, and prior to Maker's final move they have chosen three other vertices. From Figure 3.3 it is clear that these must be the three vertices outside of $A \cup B$. However, from Lemma 3.7 this is impossible. Hence, it must not be the case that $A \cap B = \emptyset$.

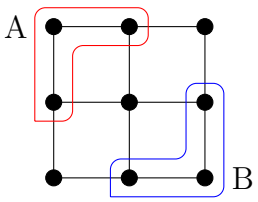

Figure 3.3: The only possible configuration for which A and B have Form 3 (up to symmetry)

Therefore, A and B must overlap. It can be easily checked that for all pairs of winning sets, their union occupies between five and seven vertices. Suppose that $|A \cup B| = 7$. Then Breaker must have chosen at least five vertices prior to Maker's final move, which as above, is impossible. Then, suppose that $|A \cup B| = 6$. There are only two possibilities; either the winning sets have Form 3 and Form 5, or both winning sets have Form 4. These possibilities

are displayed in Figure 3.4.

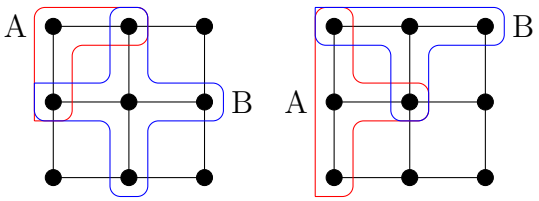

Figure 3.4: The two possible configurations for $|A \cup B| = 7$ (up to symmetry)

If the winning sets have Form 3 and Form 5, then it is clear that Breaker must have made four moves before Maker's final move. Hence, Maker must have chosen the three vertices not contained in $A \cup B$. However, since these are all corner vertices, Lemma 3.7 implies this situation is impossible. Hence, the winning sets must both have Form 4. Note that in this case there is precisely one corner vertex not contained in $A \cup B$, refer to it as vertex v, and the other two vertices as x and y. From Lemma 3.7 we know that v cannot be claimed in Maker's first move; without loss of generality, suppose that x is claimed instead. For the second move, if v is contained in a closest winning set then y is as well, and choosing y will neutralise two more winning sets (as opposed to one for v). Hence, Maker must claim y in the second move, and hence claim v in the third move. However, at the time of Maker's third move, v is not contained in any closest winning sets and so could not be chosen by `neighbourhoodWatch`. Hence, this situation is also impossible.

The only remaining situation is that $|A \cup B| = 5$. Hence, either the winning sets have Form 3 and Form 4, or both winning sets have Form 3. These possibilities are displayed in Figure 3.5.

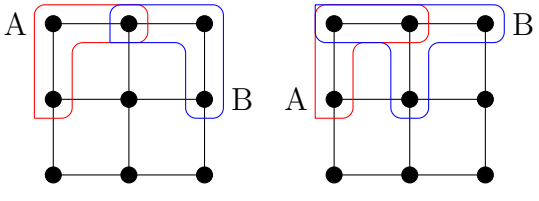

Figure 3.5: The two possible configurations for $|A \cup B| = 5$ (up to symmetry)

Consider first the situation where both winning sets have Form 3. Due to symmetry, we can assume without loss of generality that it is equivalent to the situation displayed in the left configuration of Figure 3.5. After Breaker makes their first move, Maker must claim a vertex not contained in $A \cup B$, and from Lemma 3.7 it cannot be a corner vertex or the middle vertex. Hence, Maker must claim vertex 8 for their first move. However, this choice would only be made if vertex 8 is contained in one of the closest winning sets, and hence Breaker's first move must have been to claim either vertex 4 or 6. Again, due to symmetry both choices are equivalent, so we will assume Breaker claimed vertex 4 initially. Then, Breaker needs to claim either vertex 1, 2, 3 or 6 for their second turn. However, if Breaker claims vertices 1, 3 or 6, then `neighbourhoodWatch` instructs Maker to select vertex 2. If Breaker claims vertex 2, then `neighbourhoodWatch` instructs Maker to select vertex 1. Hence, in all cases, Maker selects a vertex from $A \cup B$ for their second turn, which is impossible.

Finally, consider the situation where the winning sets have Form 3 and Form 4. Again, due to symmetry, we can assume without loss of generality that it is equivalent to the situation displayed in the right configuration of Figure 3.5. At this stage, we now need to consider all possible ways Breaker could play, and show that in each case `neighbourhoodWatch` would instruct Maker to choose a vertex in $A \cup B$ before their final move takes place. However, we first make two observations which help to reduce the number cases.

Firstly, one complicating factor with this case is that Breaker could theoretically win in four moves, and hence it is conceivable that Breaker might choose to claim a vertex outside of $A \cup B$ in their first three moves and still end up winning. However, we can eliminate this from consideration with the following simple argument. It is clear that choosing a vertex from outside

$A \cup B$ does not assist Breaker; it does not contribute towards claiming all of A, and none of those vertices are advantageous for Maker, so preventing Maker from claiming them does not help Breaker. However, if Breaker claims a vertex from outside $A \cup B$, it prolongs the game and enables the Maker to claim an extra vertex. Since these vertices must be outside $A \cup B$, what results is a stronger situation for the Maker than would have been possible if Breaker had selected only from within $A \cup B$. Hence, this situation is never beneficial for Breaker, and we do not need to consider it, if we can show that the weaker situation for Maker nonetheless results in a Maker victory.

Secondly, recall that before Maker's final move, we have $c(A) = c(B) = 1$. It cannot be the case that it is the same vertex which remains from both winning sets, or else `neighbourhoodWatch` will instruct Maker to claim it, hence neutralising both A and B simultaneously. Hence, Breaker must have claimed both overlapping vertices prior to Maker's final move.

With the above observations in mind, it is clear that Breaker's first three moves must be to either claim (in some order) vertices 1, 2 and 3, or vertices 2, 3, and 5. We now consider this reduced set of cases.

If Breaker claims vertex 1 in their first turn, `neighbourhoodWatch` instructs Maker to claim either vertex 2 or 4. Since vertex 2 is in $A \cup B$, Maker must claim vertex 4. Then, Breaker must claim either vertex 2 or 3 next. In either case, `neighbourhoodWatch` instructs Maker to claim a vertex from within $A \cup B$ (vertex 3 or vertex 2 respectively). Hence, this is impossible.

If Breaker claims vertex 2 in their first turn, `neighbourhoodWatch` instructs Maker to claim either vertex 4 or 6. Since vertex 6 is in $A \cup B$, Maker must claim vertex 4. Then, Breaker must claim vertex 1, 3, or 5 next. In each

case, `neighbourhoodWatch` instructs Maker to claim a vertex from within $A \cup B$ (vertices 3, 6 and 3 respectively). Hence, this is impossible.

If Breaker claims vertex 3 in their first turn, `neighbourhoodWatch` instructs Maker to claim either vertex 4 or 6. Both of these are in $A \cup B$, so this is impossible.

If Breaker claims vertex 5 in their first turn, `neighbourhoodWatch` instructs Maker to claim one of vertices 2, 4, 6 or 8. Since vertices 2 and 6 are in $A \cup B$, Maker claims either vertex 4 or vertex 8. Suppose Maker claims vertex 4 as its first move. Breaker must then claim either vertex 2 or 3 next. In either case, `neighbourhoodWatch` instructs Maker to claim a vertex from within $A \cup B$ (vertex 3 or vertex 6 respectively). Hence this is impossible, and so Maker must claim vertex 8 as its first move. Breaker must then claim either vertex 2 or 3 next. If Breaker claims vertex 2 as its second move, `neighbourhoodWatch` instructs Maker to claim either vertex 1 or 3, both of which are in $A \cup B$. If Breaker claims vertex 3 as its second move, `neighbourhoodWatch` instructs Maker to claim vertex 2, which is also in $A \cup B$. Hence, this is impossible as well.

We have now considered all possible cases and found that none of them are possible. Hence, the initial assumption that Breaker has won the game must be incorrect, completing the proof. □

As we have seen, there is a considerable amount of case analysis required for even this small graph example. To iterate over larger and larger graphs in this manner is clearly infeasible. Interestingly, the `neighbourhoodWatch` strategy which we employ effectively here is a (polynomial-time) adaptation of the strategy implied by the Erdős-Selfridge criterion, and yet this graph does not actually meet the conditions for Erdős Selfridge criterion.

As outlined in Section 1.3, the Erdős Selfridge criterion states that if

$$\sum_{A \in \mathcal{F}} 2^{-|A|} < \frac{1}{2}, \quad \text{then Breaker has a winning strategy in } \mathcal{F} \tag{3.2}$$

In the plain domination game, these $A \in \mathcal{F}$ are exactly the dominating sets of the game's underlying graph. As mentioned in Section 1.2, however, it is generally intractible to enumerate all dominating sets for all but the smallest of graphs. However, the observation of Duchêne et al. in Proposition 3.1 states that each Maker-Breaker domination game $\mathcal{F}$ can be equivalently represented as a game $\mathcal{F}'$ for which the Maker player of M is the Breaker of M' and vice versa. Considering the "Maker-as-Breaker" formulation M' in the context of the Erdős Selfridge criterion, the winning sets are all subsets of the vertex set that contain an enclave. In this case, we have

$$\sum_{A \in \mathcal{F}'} 2^{-|A|} \geq \sum_{v \in V} 2^{-|N[v]|}, \tag{3.3}$$

and in the 3×3 grid,

$$\sum_{v \in V} 2^{-|N[v]|} = 2^{-5} + 4 \times 2^{-4} + 4 \times 2^{-3} = 0.78125 > \frac{1}{2}. \tag{3.4}$$

So, the condition for the criterion is not satisfied and yet the Breaker of M' still has a winning strategy. We will attempt to investigate other such strategies through a simulation-driven approach, outlined in the following section.

Chapter 4

Strategies and Simulation

4.1 Core Simulation Environment

As seen in the previous chapter, exhaustively analysing an instance of the Maker-Breaker domination game on anything but small graphs is likely to be intractable. As a result, in order to examine a strategy's performance with any reasonable computation time, it must be done by restricting consideration to some subet of all possible move sequences. One convenient means of doing this is through simulated play, and this is the approach used in this thesis. In the simulation model, we define operating procedures to represent the decisions of Maker and Breaker, as well as procedures for representing the game state. As both Maker and Breaker's strategic approaches are of interest, both players are fully simulated, and can play according to any valid strategy one may define in the codebase. This allows a greater variety of comparison between strategies compared to simply hardcoding a single approach for one player. In doing so, it is important to ensure that the design of such a simulation environment is fair to both players.

The simulation can be run for a given graph, and a choice of Maker strategy and Breaker strategy. Then, each simulated player will take turns claiming a

free vertex from the board according to their strategy until one of the players has won the game and/or no free vertices remain. We have implemented this simulation model in Python in order to examine the various strategies in terms of their observed success rate against one another. We will also consider the computational requirements of each strategy against their observed successes. It is worth noting that fairness in the above respect is distinct from the "fairness" of the underlying graph on which the game is played. Certain graphs may be structurally favourable to either the Maker or the Breaker, in which case one player may be able to win even with a much weaker strategy than their opponent. In Algorithm 4.1, we describe the top level of the simulation. It accepts as input the graph, and the choice of strategies for the Maker and Breaker. It then iteratively performs the process of simulating turns until there is a winner of the game. Note that in Algorithm 4.1 the Maker is the first to act; however, if the Breaker is to act first, the algorithm can be modified accordingly. We will now discuss how the simulation model

Algorithm 4.1 `(playGame)`
The top-level routine for simulating gameplay between two players P and Q

Input: Graph, Maker Strategy, Breaker Strategy
Output: Winner of the simulation

while no player has won **do**:
    Maker selects a vertex according to their strategy and the game state
    check if Maker has now won the game
    Breaker selects a vertex according to their strategy and the game state
    check if Breaker has now won the game
return winner of the simulated game

determines whether a player has won the game or not. We first consider plain dominating sets. As discussed in Section 2.1, we can evaluate if a vertex set S is dominating by checking if all vertices in the graph have a neighbour in S. If so, Maker has won the game. On the other hand, if there is any vertex whose neighbours have all been selected by Breaker, then Breaker has won the game.

For secure dominating sets, determining the winner is a more involved question. In Algorithm 4.2 we outline a procedure for determining if a dominating set is a secure dominating set. If so, the Maker has won the game. However, it is more challenging to identify that the Breaker has won midway through the simulation. To avoid this computational difficulty, we opt not to check at each iteration to see if the Breaker has won the game. Instead, unless otherwise stated, we only judge the Breaker to be the winner if there are no more free vertices, and Maker still has not won the game. It is worth

Algorithm 4.2 (`isSecure`)
Checking if a dominating set is also secure dominating

```
Input: Game state
Output: Boolean value

SD = True
for each vertex v not claimed by Maker do:
    find guard g of v that can move to v and preserve domination
    if no such g exists then:
        SD = False
        break
return SD
```

noting that the manner in which Algorithm 4.2 (and others of its nature) is implemented can have a significant impact in how quickly the simulation runs. For example, rather than checking each vertex at each iteration, one can store certain data and then cleverly update it each iteration, reducing the overall computation time by up to an order of magnitude. However, such implementation details are beyond the scope of this thesis, and so we omit them here.

Clearly, on a given graph the choice of strategies for Maker and Breaker is the major determining factor of how the simulation unfolds. As such, the remainder of this chapter will be devoted to detailing various strategies that can be employed in Algorithm 4.1. In Section 4.2, we will address the

differences between what we refer to as strategies in this thesis, and the conventional definition of strategy in positional games literature. Then, in Sections 4.3 and 4.4 we will outline the strategies we have developed for the simulation.

4.2 What is a Strategy?

As mentioned in Section 1.3, a primary concern in the study of positional games is determining for which of the players a winning strategy exists. A strategy for the first (second) player can be thought of as a mapping from the set of all even (odd) length sequences of a graph's vertices (corresponding to the game's play history) to a single vertex corresponding to the next move. For example, after the k^{th} turn, a partially played game has the move history given by the sequence $\{v_1, v_2, \ldots, v_k\}$, and a strategy maps this sequence to the next move corresponding to the vertex v_{k+1}. This definition paraphrases that in Appendix C of [7], which gives the total number of distinct strategies for a given graph on N vertices to be equal to

$$\prod_{i=0}^{\lfloor N/2 \rfloor} (N-1-2i)^{\Pi_{j=0}^{i}(N-2j)} = e^{e^{NlogN/2+O(N)}} \tag{4.1}$$

When used in the sense of positional games, the term strategy corresponds to a function which gives a specific, deterministic output for any possible game state. However, the present work will loosen this definition to include functions with nondeterministic output, and which indeed may not be optimal (or at least, not provably optimal). Techniques with the aim of comparing such strategies with one another will also be employed. These comparisons are of interest particularly when there is no realistic hope of identifying optimal strategies. Instead of the purely deterministic definition of strategy, our approach will be to consider many possible deterministic strategies at each turn, and choose from them probabilistically. This approach is novel

in that it is positioned between deterministic strategies, which are the focus in postitional game literature, and simulation-based techniques often used in artificial intelligence approaches. Considering strategies in this probabilistic way is similar to mixed strategies in traditional game theory, but applied to this more combinatorial setting.

4.3 Strategies for the Maker-Breaker Plain Domination Game

In this section we discuss some strategies for Maker or Breaker to employ in the simulation. In addition to the upcoming strategies, either Maker or Breaker (but not both simultaneously) can choose to use *strategy stealing*. In such a case, the player will determine what move their opponent would have chosen to make next if it were their turn, and make that move. This can be employed regardless of which strategy their opponent is using, and in fact, the player can choose to act as if their opponent is employing a different strategy than they are in practice, and play as if they were stealing that strategy.

Although most of the strategies discussed will be motivated from the perspective of either Maker or Breaker, in practice they may be employed by either player. Of course, since the objectives of Maker and Breaker are different, in some cases it is not sensible to do so; however, in those cases, stealing the strategy may be effective. The distinction between employing a strategy and stealing it is somewhat subtle, so we elaborate here. Suppose that employing a strategy calls for the player to consider their previous moves in order to make their next choice. Stealing the strategy would instead have that player consider their *opponent's* previous moves. For the purposes of this chapter, we will simply introduce each strategy, and will then comment

in Chapter 5 on the specific strategies where it is sensible for a particular player to steal the strategy rather than employing it. In each case, we provide a justification for why the strategy might be expected to perform well for at least one of the players. In some cases, there are established results that motivate the strategies, while for others a common-sense argument can be made. The majority of strategies outlined in this section can be employed very efficiently. However, we will also consider two more intelligent strategies which are likely to be slower to run. In Chapter 5 we will compare the performance of these strategies against one another in terms of both win rate, and computational time.

4.3.1 Pairing Strategy

Algorithm 4.3 (`tryNaiveMatching`)
Claiming random neighbours of Q's previous choices

Input: Game state
Output: Choice of free vertex

if Q has moved and Q's most recent move has a free vertex as a neighbour **then**
    choose a free neighbour of Q's most recent move
else
choose a free vertex with maximum degree uniformly by random
return choice of vertex

A pairing strategy is a reactive strategy defined by some collection of disjoint pairs of vertices $\{a_i, b_i\}$. When the opposing player claims a vertex in a pair, the strategy is to choose the other vertex in the pair. As mentioned in Section 3.1 if the graph admits a pairing dominating set S, Maker wins the plain domination Maker-Breaker game by following a pairing strategy with respect to S. The problem of identifying a pairing dominating set is NP-complete [23], and so a player might naively attempt to achieve a matching by claiming some neighbour of their opponent's previous choice.

If the player using such a strategy moves first there are no previous moves to consider and so the player may choose arbitrarily, for example claiming the vertex with the maximum degree in the graph. We refer to this strategy as `tryNaiveMatching`.

4.3.2 Greedily Seeking New Neighbours

Algorithm 4.4 (`greedyForNewNeighbours`)
Strategy for greedily increasing number of covered neighbours, used by player P against opponent Q

Inputs: Game state
Outputs: Choice of free vertex

Choose an available vertex which would add the most new neighbours to P's claimed vertex set
return choice of vertex

As discussed in Section 2.1, there is a greedy approximate algorithm for the dominating set problem which involves iteratively choosing the vertex which dominates the most new vertices. This can be readily adapted to a strategy in the context of a Maker-Breaker domination game, claiming a vertex which will maximally increase the number of vertices dominated at each turn. If multiple such vertices exist, then one is chosen uniformly at random. This strategy is referred to as `greedyForNewNeighbours` in this text. It should be noted that in this new context, this algorithm is not guaranteed to produce a dominating set. If Maker uses the strategy and there is some vertex in the graph such that at each turn, no member of its closed neighbourhood offers the most new vertices dominated, Maker will not claim any vertex in this closed neighbourhood. Breaker can subsequently claim each of these vertices, prevent a dominating set and win the game. Nonetheless, this strategy is reasonably efficient, and seems likely to work well in general.

4.3.3 Greedily Seeking Enclaves

Algorithm 4.5 (`decMin`)
Strategy for claiming vertex from active neighbourhood closest to realisation, used by player P against opponent Q

Inputs: Game state
Outputs: Choice of free vertex

Randomly select an available vertex from the closed neighbourhood closest to being fully claimed by P
return choice of vertex

Algorithm 4.6 (`decAll`)
Strategy for claiming vertex which will reduce total proximity to being fully claimed, used by player P against opponent Q

Inputs: Game state
Outputs: Choice of free vertex

Select an available vertex which will minimize the number of free vertices in all neighbourhoods untouched by Q
return choice of vertex

If Breaker can fully claim an enclave, then no dominating set of any kind can be constructed. A strategy that achieves this would then be a winning Breaker strategy for both plain domination and secure domination Maker-Breaker games. A simple approach for Breaker is to greedily choose the vertex which will reduce the minimum number of free vertices in a closed neighbourhood not yet occupied by Maker. This is a reasonably efficient strategy and, as outlined above, gives Breaker an opportunity to quickly claim an enclave if Maker overlooks their choice of vertices. This strategy will be referred to as `decMin`.

A similar greedy strategy for Breaker is to choose the vertex which will minimise the sum of free vertices in all active neighbourhoods. The intention of this approach is to quickly increment towards claiming an enclave, as with `decMin`. However, by considering the total sum of free vertices Breaker

may be able to employ a "hedged" approach that is more resilient against which seek to block enclaves from being claimed. By reducing the overall sum of free vertices, Breaker is working towards multiple enclaves at once, challenging Maker to have to block them all before Breaker can claim one.

4.3.4 Drawing on the Erdős-Selfridge Criterion

Algorithm 4.7 (`neighbourhoodWatch`)
Strategy for playing according to a danger function, giving preference to those closed neighbourhoods closest to being fully claimed by opponent, used by player P against opponent Q

Inputs: Game state
Outputs: Choice of free vertex

Identify those vertices that intersect with the highest number of active neighbourhoods closest to realisation by Q
if there are multiple such vertices **then**
    identify which of those vertices intersect with the greatest number of active neighbourhoods (in addition to those closest to realisation)
    if there are multiple of these vertices as well **then**
        choose one uniformly at random
return choice of vertex

In the strategies described thus far, we have essentially attempted to provide a "score" indicating how valuable each vertex is to the player, and the player then chooses one of the most valuable vertices to claim. However, these scores have been fairly blunt. Ideally, at any given game state, the player would like to analyse all remaining winning sets. Recall that if the Erdős-Selfridge criterion is met, there is a winning strategy for the Breaker, which can be identified by considering all remaining winning sets and selecting a vertex which eliminates large numbers of these at each step. When the criterion is met, the proof concludes that by the end of the game, all winning sets will be eliminated. However, attempting to employ such an approach in the simulation is intractable, since it requires identifying all remaining winning sets at each turn. In particular, it is known that graphs may contain an

exponential number of minimal dominating sets [26].

Nonetheless, motivated by this approach, we propose a compromise strategy along the same lines, but from the perspective of Maker. At each turn, the player is to select a vertex that occupies the greatest number of those closed neighbourhoods which are the closest to being fully claimed by Breaker. If there are multiple such vertices, preference will be given to the one which intersects with the most other closed neighbourhoods still untouched by Maker. If there are still multiple such vertices, one of them is chosen uniformly by random.

Although considerably slower to run than the other strategies listed so far in this section, this approach can still be carried out in polynomial time, and gives a much more nuanced measurement of which vertices are important to be claimed at each step. In Chapter 5, we will explore whether this strategy results in a significant enough improvement in results to justify the extra time taken compared to the simpler strategies.

4.4 Algorithms for the Maker-Breaker Secure Domination Game

4.4.1 Preliminaries

Of course, the plain domination strategies outlined in Section 4.3 may also be employed in the Maker-Breaker secure domination game. Maker requires a dominating set to achieve a secure dominating set, and so preventing a dominating set implies a win for Breaker. A strategy for the Maker-Breaker secure domination game may be divided into two separate stages. The first of which aims to achieve a dominating set, and the second aims to extend

this to a secure dominating set. Existing strategies for the Maker-Breaker plain domination game could be paired with a "second stage" strategy to determine how Maker proceeds upon constructing a plain dominating set, and we discuss this at the end of this chapter. However, first we need to develop strategies with the particular features of secure domination in mind. We note that secure domination strategies may also be used in the plain domination game, and we will explore their effectiveness in Chapter 5.

The following definitions are relevant to the strategies in this section.

Two vertices u, v in a graph G are said to be *2-neighbours* if $dist(u, v) = 2$, i.e. the shortest path between them consists of exactly two edges.

In the context of examining whether a set S is a secure dominating set of G, we say that a vertex $v \notin S$ is *1-guarded* if $|N[v] \cap S| = 1$. In other words, there is only one guard vertex supervising v. The term *2-guarded* is defined analogously.

4.4.2 Random Selection of 2 Neighbours

Algorithm 4.8 (`random2Neighbour`)

Inputs: Game state
Outputs: Choice of free vertex

Claim a free 2-neighbour of a previous choice of P. If no such option exists, choose free vertex at random
return choice of vertex

In the secure domination context there is the added requirement relating to guard movement. At the site of such a move, a guard vertex is essentially traded with one of its neighbours (the site of the guard's new location) to construct a new subset S. This means the guard's new neighbours may have

previously been 2-neighbours, some 1-neighbours may now be 2-neighbours, and some previous 2-neighbours may have become 1-neighbours. To preserve domination under a guard relocation, all of these vertices must remain dominated. It follows that strategies for secure domination Maker-Breaker games may benefit from considering the 2-neighbourhoods of the board's vertices.

4.4.3 Considering 2-Guarded Vertices

Algorithm 4.9 (`greedyFor2Guarding`)

Inputs: Game state
Outputs: Choice of free vertex

Greedily choose a vertex that will produce the biggest increase in 2-guarded vertices
if multiple such vertices exist **then**
    choose one uniformly at random
return choice of vertex

If every non-Maker vertex is 2-guarded, then Maker has achieved a secure dominating set. Even when this is not possible, vertices which are 2-guarded are valuable as they do not prohibit adjacent guards from moving elsewhere to satisfy the secure domination condition.

4.4.4 Considering Minimum Guarded Vertices

Algorithm 4.10 (`minGuardedNeighbour`)

Inputs: Game state
Outputs: Choice of free vertex

Claim the free neighbour of a previous choice of Q with the fewest current guards.
if multiple such vertices exist **then**
    choose one uniformly at random
return choice of vertex

Algorithm 4.11 (`minGuardedFreeVert`)

Inputs: Game state
Outputs: Choice of free vertex

Claim the free vertex with the fewest current guards.
if multiple such vertices exist **then**
    choose one uniformly at random
return choice of vertex

Unlike in the plain domination case, a vertex having an adjacent guard can still represent a threat to Maker in the Maker-Breaker secure domination game, as that guard may need to move. As such, an adaptation of the greedy enclave seeking strategy for the secure domination case may be to greedily choose a vertex (whether a free neighbour of a previous choice or simply a free vertex) with the minimum number of guards in its neighbourhood. In practice, this means the choice of vertex will have zero or one adjacent guard, as all non-guard vertices having two adjacent guards implies a Maker victory.

4.4.5 Using a Linear Programming Formulation

Algorithm 4.12 (`bh19LP`)
Strategy based on the LP formulation for player P, with opponent Q

Inputs: Game state
Outputs: Choice of free vertex

Build relaxed formulation from Burdett and Haythorpe [14]
Add constraints for vertex play history, $x_Q = 0$, $x_P = 1$
Solve linear program with CPLEX
Process and normalise x_i values of solution
Select a free vertex probabilistically according to the x_i values
return choice of vertex

As discussed in the context of plain domination in Section 4.3, it may be beneficial to consider a more nuanced approach to scoring the vertices. Unfortunately, the additional requirements of secure domination mean that an equivalent strategy to Algorithm 4.7 would likely be at least an order of magnitude slower again to run. Instead, we now propose an alternative approach

to obtaining a score for the vertices. As with Algorithm 4.7, this approach is significantly slower than the simple strategies described earlier in this section, but seems likely to provide a much more nuanced measurement of which vertices are important to be claimed at each step. We will investigate the effectiveness of this strategy in Chapter 5.

Recall that a mixed-integer linear programming formulation for secure domination was given in [14]. The formulation could be modified to accommodate a partially-completed Maker-Breaker secure domination game, by fixing variables corresponding to Maker's choices to be equal to one, and fixing variables corresponding to Breaker's choices equal to zero. The result of solving this formulation would be to find an optimal secure dominating set which includes the choices of Maker, and none of the choices of Breaker, if such a set exists. Although the optimality is not directly relevant in the context of Maker-Breaker games, it seems reasonable to expect that a secure dominating set which requires fewer vertices to be claimed is easier for Maker to achieve. Meanwhile, if the formulation has no solution then Breaker has won the game. However, solving the formulation requires an integer programming solver, and hence becomes intractable for larger instances.

Instead, after fixing variables according to the choices so far of Maker and Breaker, we can then relax the integer constraints in the formulation, instead constraining those variables to lie in the continuous interval [0,1]. We will use the name `bh19LP` to refer to the strategy involving the relaxed version of the formulation. Since it just involves solving a linear program, it can be run in polynomial time using standard software such as CPLEX. What results is not a secure dominating set (unless the solution happens to be binary), but nonetheless has some meaning. If, in the solution, one variable has a significantly larger value than another, then it seems reasonable to expect the vertex

corresponding to the former is more "valuable" than the vertex corresponding to the latter when seeking a secure dominating set. Although nothing has been definitively shown in the literature, values resulting from solving `bh19LP` have been effectively employed to provide a heuristic approach for finding minimal or near-minimal secure dominating sets [13]. We will use these values to provide scores for the vertices. As discussed in Section 2.2, these values can be processed, for example raised to some power to accentuate or obfuscate differences in values. In our implementation, we chose to raise the values to the power of 2. The resulting collection of values is then normalised and used as a probability mass function informing the player's next choice.

Using the Secure Domination LP Strategy for Plain Domination

It is worth noting that the strategy described in Algorithm 4.12 can be adapted to the plain domination case. Note that the constraints in Algorithm 4.12 include those from [14], as well as the constraints relating to the play history. Instead, the constraints (2.1) and (2.2) of Burdett and Haythorpe's formulation (numbered as they appear in Section 2.2) can be combined with the constraints relating to play history to give a version of `bh19LP` which is designed specifically for plain domination. We will refer to this strategy as `bh19LP_d`.

4.4.6 Hybrid Strategies

As mentioned in Subsection 4.4.1, strategies for the plain domination game can be employed in the secure domination game. However, strategies designed with Maker's plain domination objective in mind should not be expected to perform as well in a secure domination context. Instead, if a dominating set is achieved, Maker could then switch to a second strategy

intended to "secure" their existing dominating set. It will be shown that hybrid strategies of this type can achieve superior performance over either of the individual strategies they are comprised of.

By distinguishing between a "dominating" stage and a "securing" stage, various combinations of strategies are available. Combining plain domination strategies such as `greedyForNewNeighbours`, `neighbourhoodWatch` with a second stage is of interest. We will discuss these choices further in Section 5.1.

Chapter 5

Results

In this chapter, we will look at results produced by the simulation model described in Chapter 4. All results are produced by performing 1000 simulated games per instance (i.e. for each combination of graph, Maker strategy, and Breaker strategy). We adopt the convention that Maker plays first in all of our experiments.

The experiments will be conducted on grid graphs of various sizes. Grid graphs have been used extensively in the literature around domination and its variants, and in that context they possess surprisingly intricate and complex properties. For example, the complete characterisation of domination numbers of $m \times n$ grid graphs was derived over a 27-year period [40, 32, 21, 49, 2, 30], and consists of 23 special cases before settling into a standard formula for $m, n \geq 16$. For secure domination, the domination numbers are only known for $m, n \leq 10$. These instances provide an interesting and complex, but also consistent test bed for our experiments. We will consider the strategies outlined in Sections 4.3 and 4.4. However, although those strategies can always be played for either player, in practice some of them are only sensible for a certain player. For example, the `neighbourhoodWatch` strategy is designed for Maker, to try to prevent Breaker from obtaining an

enclave. If the Breaker were to employ the `neighbourhoodWatch` strategy, it would try to prevent Maker from obtaining an enclave, but Maker is not attempting to do so, and it is not beneficial to Maker even if it occurs by chance. As such, this strategy is nonsensical for Breaker to employ. However, it could be sensible for Breaker to *steal* the `neighbourhoodWatch` strategy; that is, to consider what choice Maker would make, and then claim that choice for themselves. With this in mind, in the experiments that follow, rather than considering nonsensical strategy choices for each player, when sensible we will instead have the players employ the strategy stealing versions of those strategies.

In particular, the `decAll`, `decMin` and `minguardedFreeVert` strategies are designed for Breaker. Hence, in the upcoming experiments, Maker will only use the strategy stealing version of those strategies. Likewise, the `neighbourhoodWatch`, `tryNaiveMatching`, `bh19LP` and `bh19LP_d` strategies are designed for Maker, so Breaker will only use strategy stealing versions of those strategies.

5.1 Hybrid Approach

As discussed in Section 4.4, it is possible to use strategies designed for the plain domination game in the secure domination game, and vice versa. We will investigate how effective this approach is in this chapter. In particular, in Section 5.2 it will be seen that the strategies designed for secure domination perform no worse in the Maker-Breaker plain domination game than those strategies designed with plain domination in mind. Strategies designed for plain domination, however, do not perform as strongly in the secure domination game. It follows that for the secure domination game, these plain domination strategies might benefit from being extended as hybrid strategies.

To examine the effectiveness of the hybrid approach, we now conduct the following experiment. We run simulations of the Maker-Breaker secure domination game on square grids of increasing size, where Breaker's strategy is fixed to be random choice. For Maker, we consider various different plain domination strategies, secure domination strategies, and also three hybrid strategies. In particular, the plain domination strategies considered are `greedyForNewNeighbours`, `neighbourhoodWatch`, and `bh19LP_d`. The secure domination strategies considered are `greedyFor2Guarding`, `bh19LP` and `tryNaiveMatching`. The three hybrids are given a suffix of `_h`, and in each case they utilise one of the plain domination strategies until a dominating set is found, at which point they switch to `greedyFor2Guarding`. In addition to the above, we also include `randomChoice` as a Maker strategy to give a baseline comparison. The results of the simulation, displayed in Figure 5.1, are quite striking. We note first that the standalone secure domination strategy of `greedyFor2Guarding` was actually outperformed by `randomChoice`. Clearly, this is not a useful strategy to employ by itself. Despite this, however, it appears to offer some value when used as part of a hybrid strategy. The hybrid versions of both `neighbourhoodWatch` and `bh19LP_d` significantly outperformed their respective standalone versions, while the hybrid version of `greedyForNewNeighbours` gave almost identical performance to the standalone version. However, by far the most effective strategies here were the standalone secure domination strategies, `tryNaiveMatching` and `bh19LP`. It seems that for secure domination, it is sensible to restrict our consideration to standalone secure domination strategies and hybrid strategies. In the interests of providing a detailed analysis, we will retain most of the standalone plain domination strategies in the upcoming section; however, given the extremely weak performance of `greedyFor2Guarding` as a standalone strategy, we will remove it from consideration. As such, it will con-

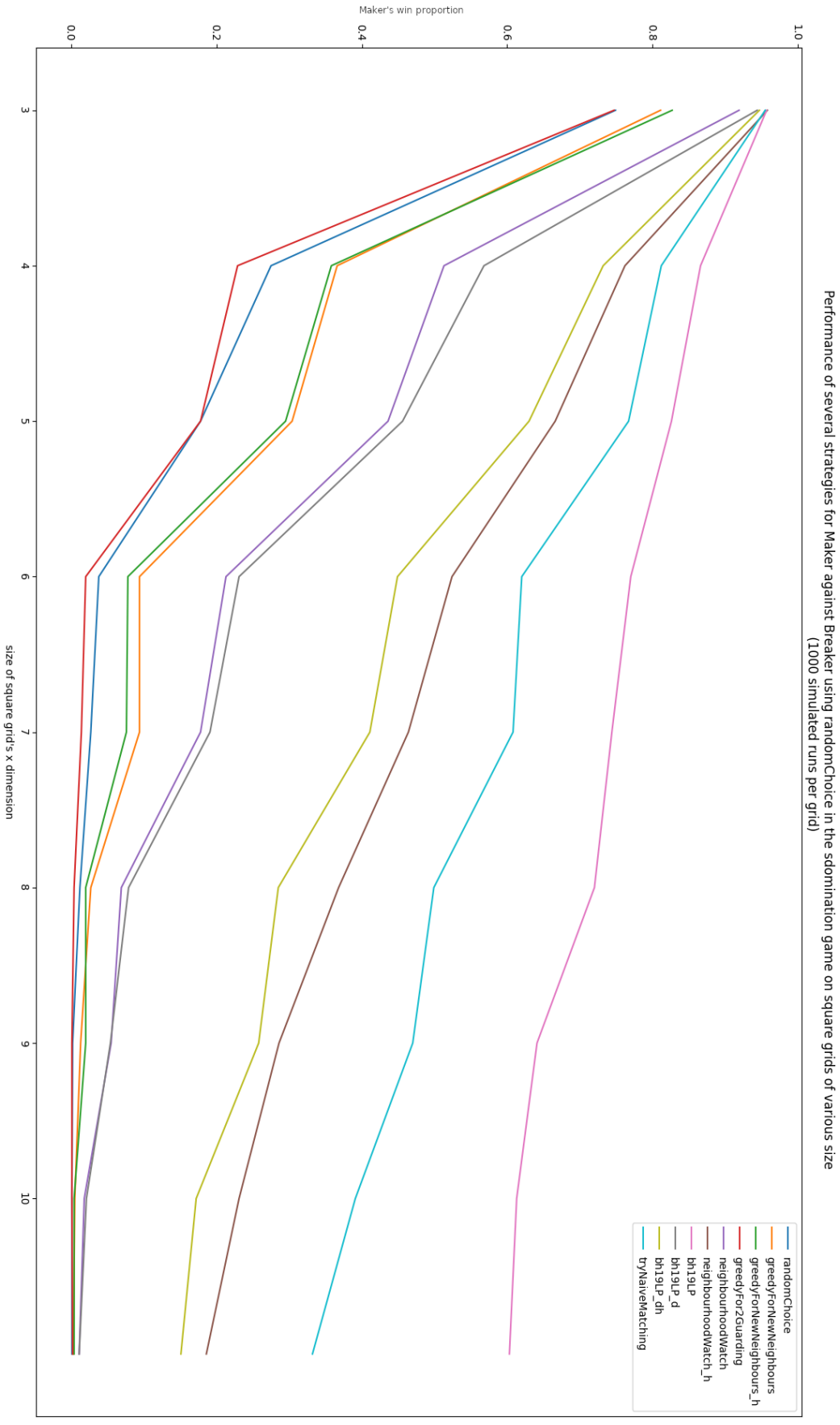

Figure 5.1: Examination of various hybrid strategies, their component functions, and several other strategies for the secure domination game on square grids of various dimension

tinue to exist in our simulations only in the sense that it is employed in the second stage of the hybrid strategies.

5.2 Comparing Strategies Against Each Other

As a means of examining the quality of various heuristic algorithms, their experimental win-rate against one another was calculated. Not only might this provide insight into which strategies are generally strong (i.e. win more often than not against other strategies) for a particular player, interesting relationships may exist between specific strategies, with a generally strong strategy perhaps being defeated by a generally weaker strategy. In certain cases, results such as these may potentially highlight properties of the strategies themselves, the rules of the game, or properties of the underlying board. Both the secure and plain domination versions of the game were considered in this way on the 7×7 grid as the game board. This graph does not admit a perfect matching, and so the theoretical winner is not obvious. The results of the simulations are displayed in Figures 5.2 and 5.3 for the plain and secure domination game respectively. We note that in Figure 5.2, results are included for some hybrid strategies. However, since Figure 5.2 corresponds to the plain domination game, the hybrid strategies are equivalent to the standalone plain domination strategies; we include them here only for easy comparison with the upcoming Figure 5.3, where the secure domination game will be considered. The two figures are quite revealing, and we now discuss various insights gained from analysing them.

First of all, we note that the simulated results are just for a particular game board, the 7×7 grid. It is perhaps interesting to consider whether this board favours one player or another, but it is not immediately obvious how to do this. Of course, in a theoretical sense any given game board has exactly one

winning player; indeed, since the 7×7 grid graph G contains a vertex v such that $G \setminus v$ has a perfect matching, and Maker plays first in our experiments, Corollary 3.6 implies that there is a winning strategy for Maker. But, if the winning strategy is not known, this may not reflect the reality of playing the game. It is conceivable that a game board may have a theoretical winner, say Maker, but that employing almost any strategy other than the winning strategy leads to a Breaker win. In such a case, it would perhaps be accurate to say that the game board favours Breaker, even though Maker is the theoretical winner.

A simple measure of favourability could be to consider the results when both Maker and Breaker play randomly, which according to Figure 5.2 would suggest that the 7×7 grid is heavily favoured towards Breaker. However, it can be seen in the same figure that Maker has significant success with some other strategies, regardless of which strategies Breaker employs. On the other hand, when Maker does not employ these "good" strategies, it seems to lose most of the time. Clearly, the question of which player is favoured by the game board is complicated to answer. Perhaps the most accurate statement here is that Maker requires a good strategy to be successful; if both players employ poor strategies, Breaker seems to be successful. We will explore the question of board favourability further in Section 5.4

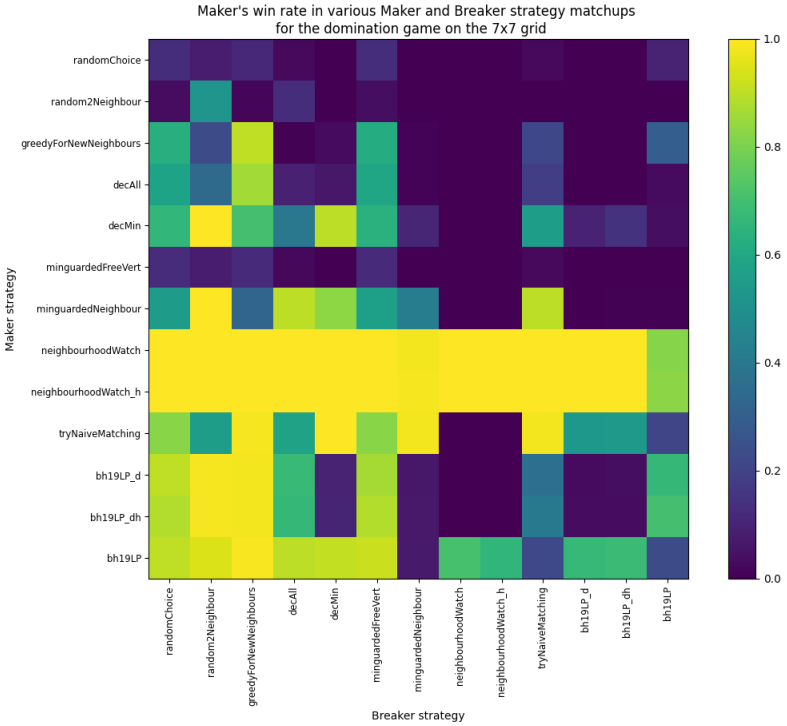

Figure 5.2: Examining various strategies for Maker and Breaker in the plain domination game on the 7×7 grid

It is clear that `neighbourhoodWatch` is an almost undefeated Maker strategy in the observed simulation. In fact, the only Breaker strategies that appears to have won any significant number of games are `minguardedNeighbour` and the strategy stealing version of `bh19LP`. This suggests that while no longer a guaranteed winning strategy as in the 3×3 grid (see Theorem 3.8), `neighbourhoodWatch` is still a strong strategy for the plain domination game on this larger grid graph. The second strongest Maker strategy appears to be `bh19LP`, a strategy designed with the secure domination game in mind. This is interesting as the strategy only considers the secure domination LP formulation, yet nonetheless performs well here in the plain domination setting. The third strongest Maker strategy is `tryNaiveMatching`, which performs well ($\geq$ 60% observed win rate) against all strategies ex-

cept for Breaker's strategy stealing versions of `neighbourhoodWatch` and the variations of `bh19LP`. This suggests that on the 7×7 grid graph, pairing strategies are promising approaches for Maker.

Regarding strategies for Breaker, the stealing versions of the most successful Maker strategies appear to be successful, except for `tryNaiveMatching`. It is perhaps unsurprising that `tryNaiveMatching` is relatively unsuccessful here, as the strategy involves claiming adjacent vertices to your opponent's previous choice. The standard (non-stealing) version of this is not particularly useful for Breaker, as it will claim a vertex which has already been dominated; for this reason, we identified this strategy as one where the Breaker should employ the strategy-stealing version. However, that version will cause Breaker to select vertices next to its previous choices, with no regard for what Maker has done, including if Maker has already claimed a vertex in the surrounding area. If Maker is foolish, Breaker will quickly claim an enclave and win the game, but if Maker plays intelligently then Breaker is unlikely to be successful. This is reflected in Figure 5.3, where we see the strategy is roundly defeated by Maker strategies which react to Breaker's moves, but is otherwise successful for Breaker.

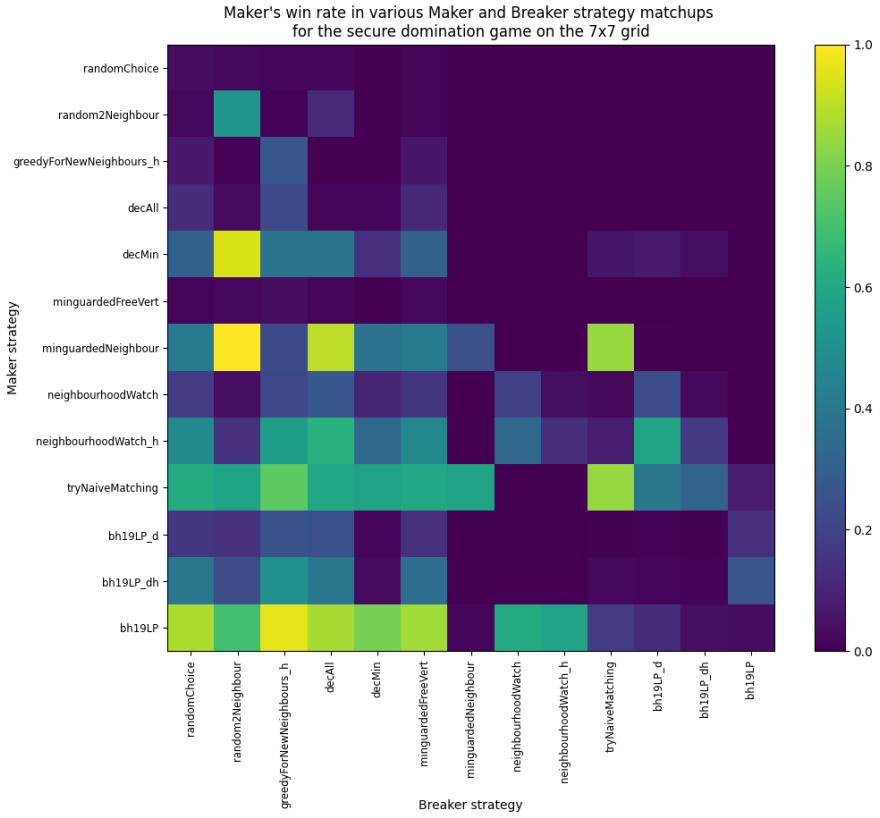

Figure 5.3: Examining various strategies for Maker and Breaker in the secure domination game on the 7×7 grid

Turning our attention now to Figure 5.3, in general Breaker is observed to be successful much more frequently in the Maker-Breaker secure domination game than in the Maker-Breaker plain domination game. This is not surprising, of course; to successfully win the secure domination game, Maker has to win the plain domination game in addition to satisfying other criteria.

The hybrid `greedyForNewNeighbours_h` has little success for Maker even against random Breaker play. The stealing version of `decAll` is similarly unsuccessful. Comparatively speaking, the stealing of `decMin` appears to better retain its success rate in the Maker-Breaker secure domination game, as does `minguardedNeighbour`. The latter also largely retains its appar-

ent quality as a strategy for both Maker and Breaker. Perhaps the most drastic difference from the plain domination game is the reduced success of `neighbourhoodWatch`, now losing much more often than winning. As seen previously, though, its hybrid version performs markedly better. The `tryNaiveMatching` and `bh19LP` strategies are by far the strongest strategies for Maker, and exhibit some interesting comparative results. While `bh19LP` is almost always defeated by Breaker employing `minGuardedNeighbour` or the strategy stealing version of `tryNaiveMatching`, it appears `tryNaiveMatching` performs quite well against each of these Breaker strategies. However, `bh19LP` is the only Maker strategy to exhibit strong performance against Breaker stealing the `neighbourhoodWatch` strategy (or its hybrid).

There are several effective strategies for Breaker in this instance, but the only one which does not seem to have a weakness is the strategy stealing version of `bh19LP`. The only Maker strategy which was even remotely competitive was the hybrid version of `bh19LP_d`. It is perhaps fascinating to note that the strategy stealing versions of good Maker strategies appear to provide better options for Breaker than their own dedicated strategies, which we designed with Breaker's objectives in mind. Again, one possible interpretation of this is Maker requires a good strategy to be successful on this instance, which leaves them susceptible to having their strategy stolen.

There are some interesting results for specific strategy pairs. The choice of `minguardedNeighbour` is only modestly successful for Maker, while the strategy stealing version of `tryNaiveMatching` is very successful for Breaker. Yet, the combination of the two leads to an unexpectedly strong result for Maker. A possible explanation for this might be that the selection functions for both strategies give preference to neighbours of vertices which have been previously chosen by Breaker, which increases the likelihood that both play-

ers choose adjacent vertices. As discussed earlier, this is typically beneficial for Maker as it means Breaker is more likely to claim vertices which have already been dominated.

Another interesting case is when Maker chooses the strategy stealing version of `decMin` and Breaker chooses `random2Neighbour`. The former is a poor performing strategy for Maker, and the latter performs moderately well for Breaker, but their combination leads to around a 90% win rate for Maker. Analysing these two strategies, we see that Breaker will want to choose vertices which are a distance of two away from one of their previous choices. Meanwhile, Maker is stealing the strategy where Breaker tries to claim an enclave if possible. As such, Maker gets to follow along with Breaker, neutralising these potential enclaves as they arise, resulting in likely success for Maker. Unsurprisingly, this was also a very effective combination of strategies for Maker in the plain domination setting.

5.3 Time Analysis

Every strategy given in Chapter 4 of this thesis can be run in polynomial time. However, since some strategies are more sophisticated than others, it should be expected that there will be significant differences in how long they take to perform their calculations. We now seek to analyse how long the strategies take to run. However, it is not immediately obvious how such an analysis should be undertaken. One might think that we could simply employ two timers (one for Maker, one for Breaker) and toggle them on and off whenever one player is executing their strategy. Then, the individual times for each player can be reported. However, there is a significant issue with this approach; the strategy employed by the opponent can have an impact on how many moves are required to finish the game. For example, if

Breaker makes poor choices, Maker might be able to obtain a dominating set much quicker. This would mean that for two different Breaker strategies, a given Maker strategy would report two very different times. Alternatively, we could take an average of the time taken per move, but it might be the case that a good strategy enables the player to finish earlier. For example, when the `bh19LP` strategy is employed (either by Maker, or by Breaker using the strategy stealing version), if Breaker has been successful then the LP often returns an infeasible result, which may allow the game to be terminated early. This is a unique benefit of using `bh19LP` (the strategy stealing version is the only Breaker strategy that allows us to terminate early) that would be disguised by reporting average time per turn.

Given it is unclear how best to report the time taken for a strategy, we have conducted the following experiment on the secure domination game. We fix the Breaker strategy to be `randomChoice`, and consider a collection of Maker strategies over grid graphs of increasing size. Then, we simply report the average time taken for the entire simulation to conclude. Although this time includes both players and other overheads from the simulation, it is at least consistently counted across all Maker strategies, allowing for direct comparisons. Since Breaker always plays randomly, no advantage or disadvantage is imparted to a particular Maker strategy. Also, since `randomChoice` is a very fast strategy, the bulk of the differences in runtime should be attributable to the Maker strategy. The results are displayed in Figure 5.4, where the vertical axis (corresponding to time taken) is on a logarithmic scale.

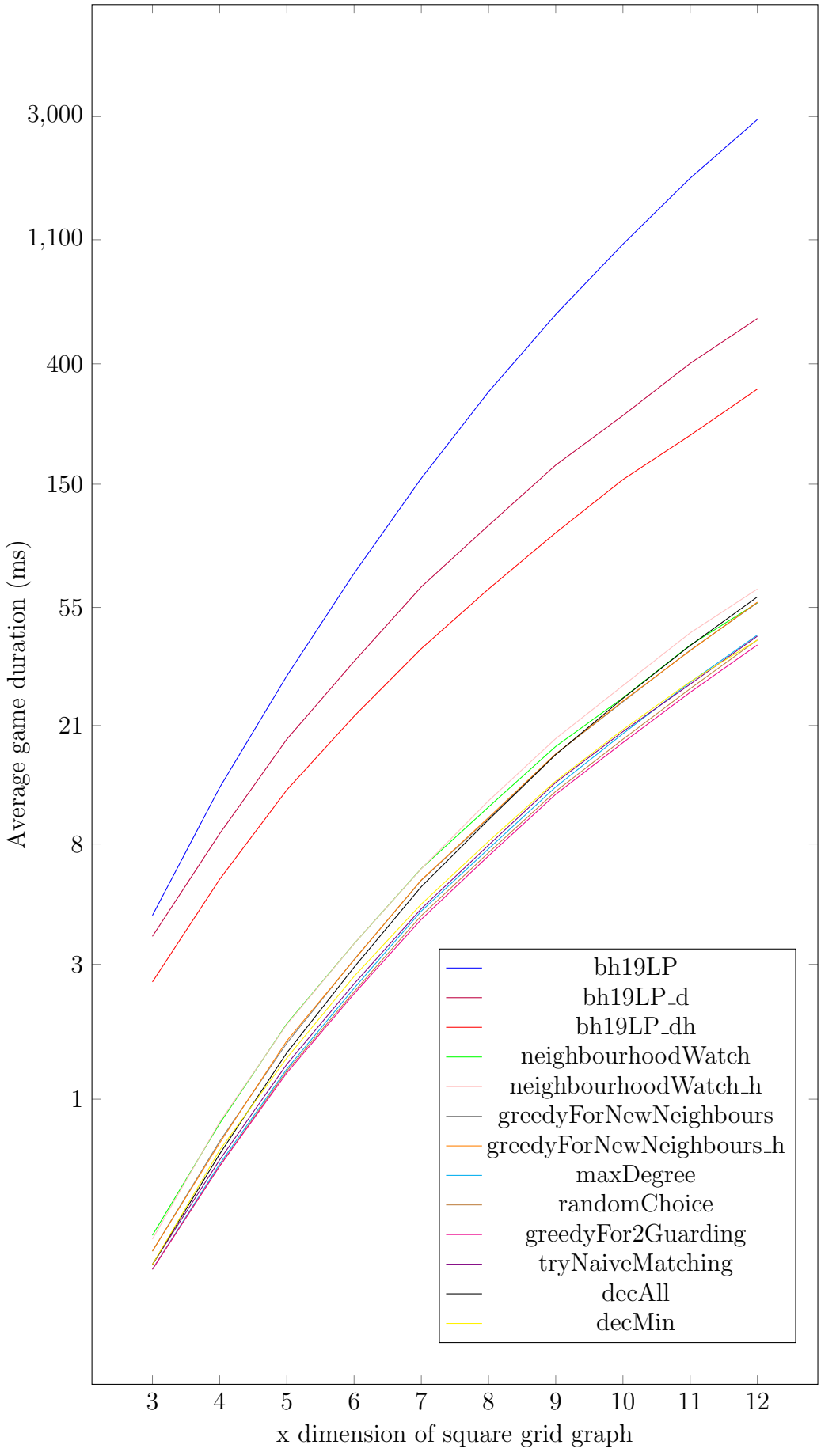

Figure 5.4: A logarithmic plot of the average game duration for various Maker strategies against Breaker playing randomly in the Maker-Breaker secure domination game on square grids of various dimensions

In Figure 5.4 there appears to be three "tiers" of strategies, in terms of how quickly they run. Clearly, `bh19LP` and its variants are by far the slowest strategies considered in this thesis. This is unsurprising considering that, at each iteration, these strategies involve first constructing a linear program with a considerable number of constraints, and then sending it to be solved by an external program. We have seen that `bh19LP` has displayed strong performance in various simulations, but this should be balanced against its computational overhead if time is an important consideration. The `neighbourhoodWatch` strategy (and its hybrid version) are the next-slowest, followed by `tryNaiveMatching`. These two strategies comprise the second tier of strategies in terms of time taken. Both of these strategies were seen to be effective in the previous section, although further analysis is required to see if `tryNaiveMatching` remains as effective in general graphs without perfect matchings. The following scenario may also result in decreased success of `tryNaiveMatching` as a Maker strategy. If the game is played on a graph in which a low-degree vertex v has neighbours of high degree, a clever Breaker player might select each of these high degree neighbours, and at each turn Maker's response using `tryNaiveMatching` would have relatively low probability of selecting v. Having selected all of these high degree neighbours, Breaker can then claim v and achieve an enclave, winning the game. Finally, the remaining strategies make up the third tier; however, while these strategies are quick to run, they were for the most part unsuccessful in the previous section.

Perhaps unsurprisingly, there appears to be an inverse relationship between the success rate of strategies, and their efficiency. Of course, the exact time taken for these strategies does depend on how efficiently they are implemented, but such discussions are beyond the scope of this thesis. For Maker, it is arguable whether `bh19LP` is worthwhile, given it performs similarly to

`tryNaiveMatching` but takes at least an order of magnitude longer. For Breaker, `bh19LP` is so successful that it is perhaps worth the extra time, at least in the secure domination version of the game.

5.4 Strategy Stealing

As discussed in Section 5.1, an interesting property to consider is to what extent the graph on which the game is played favours a particular player. In the combinatorial sense, a Maker-Breaker game on a given board admits a winning strategy for exactly one player, and so (if it is known for which player this holds) a certain binary classification can be made accordingly. However, this form of favouritism may not be reflected in the stochastic sense. For example, the game of Hex can be thought of as a Maker-Breaker game, and has been shown to be a first-player win on the 11×11 board. However, the exact winning strategy is not known. Hex continues to be an active competitive board game for both human and computer players, suggesting that this theoretical favouritism has not yet surfaced in real-world play to the point of rendering the game obsolete. One study has reported a first player win percentage of 62.8% [10]. As such, it certainly seems that the first player has some advantage in a stochastic sense, but not a massive advantage.

It is not immediately clear how to characterise which player has the advantage for a given game board. As discussed earlier, simply having both players make random selections and analysing the outcome does not necessarily give meaningful information for how the game will unfold when the players play intelligently. Instead, we propose the following experiment. We will analyse the outcome of having one player employ a strategy, while their opponent employs the strategy stealing version of the same strategy. This seems like a relatively "fair" experiment, since both players will be trying to

do the same thing. In addition, it will allow us to analyse how susceptible certain strategies are to being stolen. We will choose specific strategies which are sensible for one player to employ, and the other to steal.

In particular, we consider first consider `neighbourhoodWatch` (and its hybrid version), `tryNaiveMatching`, and `bh19LP` as Maker strategies, while Breaker employs the corresponding strategy stealing variants. These were run on grid graphs of increasing size, with the results displayed in Figure 5.5. Likewise, we consider Breaker choosing the `minguardedNeighbour`, `decMin` and `decAll` strategies, while Maker employs the corresponding strategy stealing variants. These were run on grid graphs of increasing size, with the results displayed in Figure 5.6.

We observe that as the size of the graph increases, the success rate for Maker decreases. There is a sensible interpretation for this; as the graph grows larger, Maker has more work to do since the entire graph needs to be dominated, whereas Breaker still only needs to gain an enclave for one vertex. For most of the strategies considered, the win rate for Maker appears to be heading towards zero; however, interestingly, it seems that `tryNaiveMatching` is much less susceptible to strategy stealing than the other strategies. As discussed back in Section 5.2, when both players employ `tryNaiveMatching`, an odd situation occurs where the Maker and Breaker end up picking vertices next to each other for the entire game, leading to a situation where Breaker is less likely to claim an enclave.

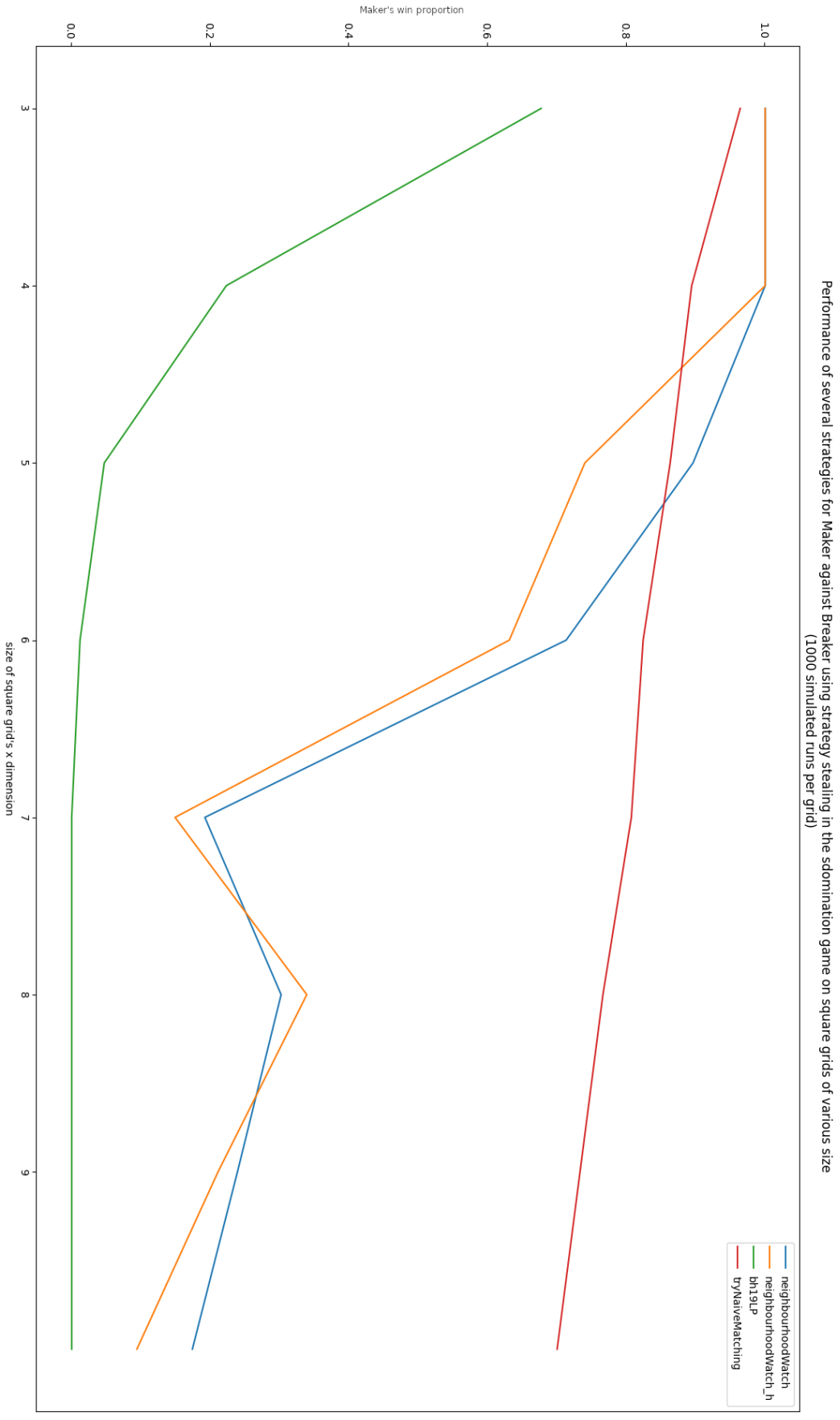

Figure 5.5: Various Maker strategies' performance against strategy stealing on square grids of differing dimension

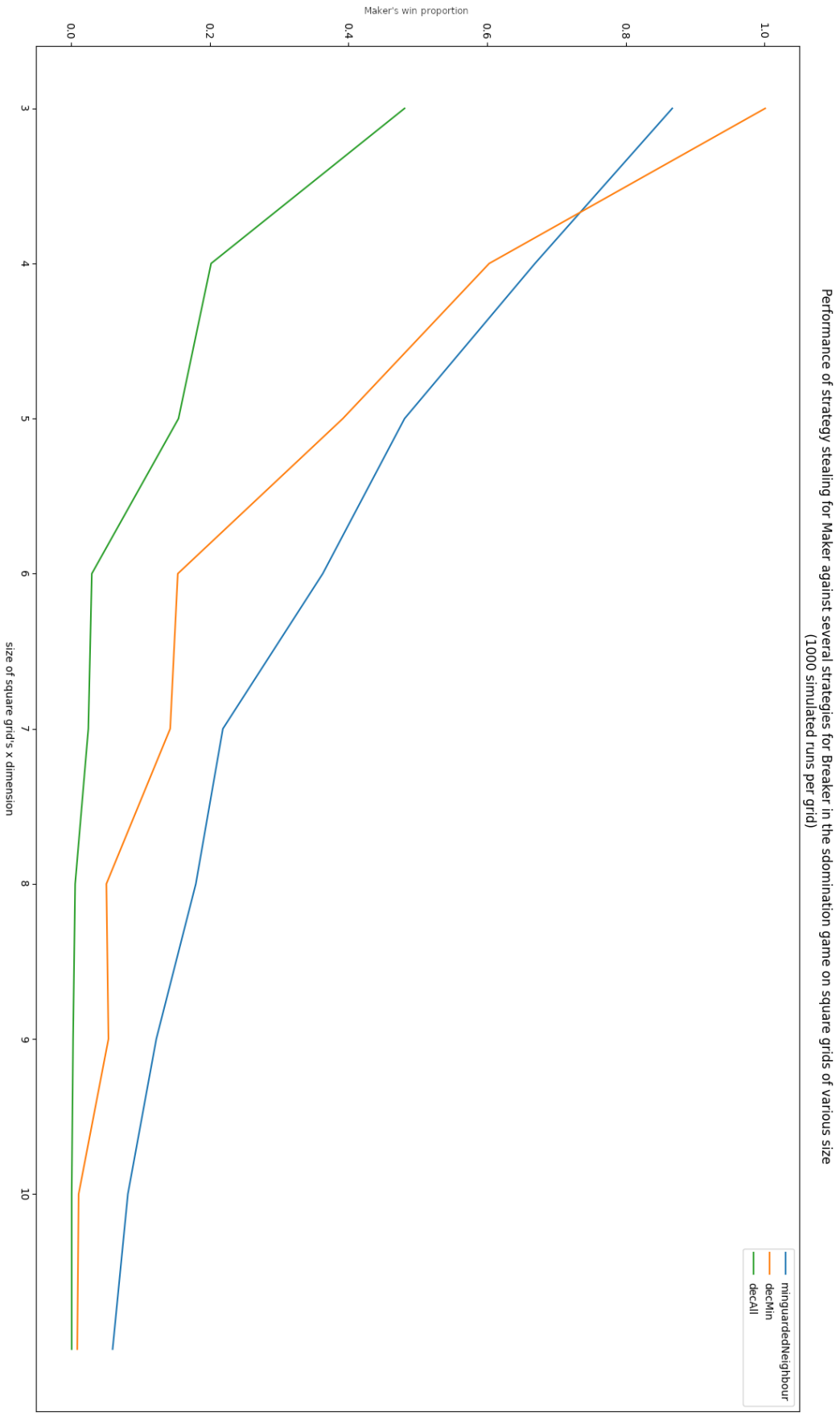

Figure 5.6: Various Breaker strategies' performance against strategy stealing on square grids of differing dimension

Chapter 6

Conclusions and Directions for Future Research

In this thesis, several heuristic strategies for both the Maker-Breaker plain domination and secure domination games were developed and examined in an experimental context. Various measures for quality were defined, and these strategies were compared to one another according to these. Square grids of various sizes were exclusively used in the simulations, and so naturally an avenue of future research would be to perform these evaluations on various other graph families of interest to see what impact this has on results. A few examples might be random graphs, graphs that do not admit a perfect matching, or graphs with a higher variance in vertex degree. It is of interest to expand the scope of consideration as the "generality" of heuristic strategies is another measure that could be used to indicate quality. The `tryNaiveMatching` strategy proved to be a successful, efficient Maker strategy for the secure domination game on various square grids, however a scenario was outlined in which the graph's structure allows the strategy to be exploited by a clever Breaker player. Whether the more computationally expensive `bh19LP` or `neighbourhoodWatch_h` are more resistant to such exploitation on a variety of graphs would be an interesting future experiment.

Hybrid strategies that adapt heuristics for Maker in the plain domination game to the secure domination (or other variant of domination) game are another area of interest. In this thesis these strategies were broken into two stages, the second initiating after Maker has achieved a dominating set. This thesis only examined three first stage strategies and one second stage strategy. Different combinations of both first and second stage strategies (or the introduction of additional stages) may be an interesting avenue of future research. In addition to the choices of the strategies that make up the hybrid, the point at which the transition between stages occurs might also be subject to experimentation, allowing the second stage to begin early once it becomes apparent the first stage will be successful.

Several interesting relationships between various heuristics were observed in Chapter 5. As these are experimental observations, a natural next step would be to work towards establishing proofs regarding these relationships. `tryNaiveMatching` as a Maker strategy seemed particularly resistant to strategy stealing. The intuition behind this might be extended to include other Maker strategies whose selection functions gave preference to the neighbours of Breaker's previous choices. On graphs that possess certain structures, one heuristic strategy may be provably superior to another.

Since the strategies we have employed are stochastic in nature, there is also scope to take some kind of combination of multiple strategies, hopefully leveraging the benefits of each of them. It seems natural to assume that strategies which make decisions which are generally good at any stage of the game would would be amenable to this kind of approach. Conversely, strategies which require a particular plan to be followed for a long period would be unsuitable for this.

The domination game is noteworthy due to the "interchangeablity" of the players under equivalent formulations. Similar formulations for secure domination or other variants are desirable as they allow the application of general Maker-Breaker game results for one player to be applied to both players in these games. Characterising a Maker-Breaker game beyond the player who possesses a guaranteed winning strategy is still a difficult endeavour, however the observed performance of heuristic strategies may provide additional insight into the complex properties these games possess.

Bibliography

[1] Alon, N. and Spencer, J. H., 2016. *The probabilistic method*. John Wiley & Sons.

[2] Alanko, S., Vercals, S., Isopoussu, A., Östergård, P.R., Pettersson, V., 2011. Computing the domination number of grid graphs. *Electronic Journal of Combinatorics*, p.141.

[3] Alzoubi, K.M., Wan, P.J. and Frieder, O., 2002. Distributed heuristics for connected dominating sets in wireless ad hoc networks. *Journal of Communications and Networks*, 4(1), pp.22-29.

[4] Balasundaram, B. and Butenko, S., 2006. Graph domination, coloring and cliques in telecommunications. *Handbook of optimization in telecommunications*, pp.865-890.

[5] Ball, W.R., 1893. Mathematical Recreations and Essays. *Bulletin des sciences mathématiques, 17*, pp.105-107.

[6] Berge, C., 1958. *La theorie des graphes et ses applications*. Dunod.

[7] Beck, J., 2008. *Combinatorial games: tic-tac-toe theory* (Vol. 114). Cambridge: Cambridge University Press.

[8] Bixby, R.E., 2012. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica, 2012*, pp.107-121.

[9] Bollobás, B. and Cockayne, E.J., 1979. Graph-theoretic parameters concerning domination, independence, and irredundance. *Journal of Graph Theory, 3*(3), pp.241-249.

[10] Brausen, H., Hayward, R.B., Müller, M., Qadir, A. and Spies, D., 2011. Blunder Cost in Go and Hex. *Advances in Computer Games, 13*, pp. 220-229.

[11] Brešar, B., Henning, M.A., Klavžar, S. and Rall, D.F., 2021. *Domination games played on graphs*. Berlin: Springer.

[12] Brešar, B., Klavžar, S. and Rall, D.F., 2010. Domination game and an imagination strategy. *SIAM Journal on Discrete Mathematics, 24*(3), pp.979-991.

[13] Burdett, R., 2020. *Exact and heuristic algorithms for secure domination in graphs*. (Honours thesis, Flinders University, South Australia, Australia).

[14] Burdett, R. and Haythorpe, M., 2020. An improved binary programming formulation for the secure domination problem. *Annals of Operations Research, 295*(2), pp.561-573.

[15] Bouamama, S. and Blum, C., 2021. An improved greedy heuristic for the minimum positive influence dominating set problem in social networks. *Algorithms, 14*(3), p.79.

[16] Burger, A. P., De Villiers, A. P. and Van Vuuren, J.H., 2013. Two Algorithms for Secure Graph Domination. *Journal of Combinatorial Mathematics and Combinatorial Computing, 85*, pp.321-339.

[17] Burger, A.P., De Villiers, A.P. and Van Vuuren, J.H., 2013. A binary programming approach towards achieving effective graph protection. *Proceedings of the 2013 ORSSA annual conference*, pp.19-30.

[18] Chvátal, V. and Erdös, P., 1978. Biased positional games. *Annals of Discrete Mathematics, 2,* pp.221-229.

[19] Cockayne, E.J. and Hedetniemi, S.T., 1977. Towards a theory of domination in graphs. *Networks, 7*(3), pp.247-261.

[20] Cockayne, E.J., Grobler, P.J.P., Grundlingh, W.R., Munganga, J. and Van Vuuren, J.H., 2005. Protection of a graph. *Utilitas Mathematica, 67.*

[21] Chang, T.Y. and Clark, W.E., 1993. The domination numbers of the 5×n and 6×n grid graphs. *Journal of graph theory, 17*(1), pp.81-107.

[22] Davis, M. and Putnam, H., 1960. A computing procedure for quantification theory. *Journal of the ACM (JACM), 7*(3), pp.201-215.

[23] Duchêne, E., Gledel, V., Parreau, A. and Renault, G., 2018. Maker-Breaker domination game. *arXiv preprint arXiv:1807.09479.*

[24] Erdős, P. and Selfridge, J.L., 1973. On a combinatorial game. *Journal of Combinatorial Theory, Series A, 14*(3), pp.298-301.

[25] Fomin, F.V., Grandoni, F. and Kratsch, D., 2005. Some new techniques in design and analysis of exact (exponential) algorithms. *Bull. EATCS, 87*, pp.47-77.

[26] Fomin, F.V., Grandoni, F., Pyatkin, A.V. and Stepanov, A.A., 2005. Bounding the number of minimal dominating sets: a measure and conquer approach. In *Algorithms and Computation: 16th International Symposium, ISAAC 2005, Sanya, Hainan, China, December 19-21, 2005. Proceedings 16* (pp. 573-582). Springer Berlin Heidelberg.

[27] Fomin, F.V., Grandoni, F., Pyatkin, A.V. and Stepanov, A.A., 2008. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms (TALG), 5*(1), pp.1-17.

[28] Fomin, F.V., Grandoni, F. and Kratsch, D., 2009. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM (JACM), 56*(5), pp.1-32.

[29] Gledel, V., Henning, M.A., Iršič, V. and Klavžar, S., 2020. Maker–Breaker total domination game. *Discrete Applied Mathematics, 282*, pp.96-107.

[30] Gonçalves, D., Pinlou, A., Rao, M., Thomassé, S. (2011). The domination number of grids. *SIAM Journal on Discrete Mathematics, 25*(3), pp.1443–1453.

[31] Grandoni, F., 2006. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms, 4*(2), pp.209-214.

[32] Hare, E.O.M., 1989. *Algorithms for grids and grid-like graphs*(Ph.D. thesis, Clemson University, South Carolina, United States of America).

[33] Hales, A.W. and Jewett, R.I., 1963. Regularity and positional games. *Transactions of the American Mathematical Society, 106*(2), pp.222-229.

[34] Haynes, T.W., Hedetniemi, S. and Slater, P., 1998. *Fundamentals of domination in graphs*. CRC press.

[35] Haynes, T.W., Hedetniemi, S.T. and Henning, M.A. eds., 2020. *Topics in domination in graphs* (Vol. 64). Cham: Springer.

[36] Hayward, R.B. and Toft, B., 2019. *Hex: The full story*. CRC Press.

[37] Hedar, A.R. and Ismail, R., 2012. Simulated annealing with stochastic local search for minimum dominating set problem. *International Journal of Machine Learning and Cybernetics, 3*, pp.97-109.

[38] Hedetniemi, S.T. and Laskar, R.C., 1991. Bibliography on domination in graphs and some basic definitions of domination parameters. *Annals of discrete mathematics, 48*, pp.257-277.

[39] Hefetz, D., Krivelevich, M., Stojaković, M. and Szabó, T., 2014. *Positional games* (Vol. 44). Basel: Birkhäuser.

[40] Jacobson, M.S. and Kinch, L.F., 1983. On the domination number of products of graphs: I. *Ars Combinatoria, 18*, pp.33-44.

[41] Johnson, D.S., 1973, April. Approximation algorithms for combinatorial problems. *Proceedings of the fifth annual ACM symposium on Theory of computing* (pp. 38-49).

[42] Kann, V., 1992. *On the approximability of NP-complete optimization problems* (Ph.D. thesis, Royal Institute of Technology, Stockholm, Sweden).

[43] Majeed, A. and Rauf, I., 2020. Graph theory: A comprehensive survey about graph theory applications in computer science and social networks. *Inventions, 5*(1), p.10.

[44] Merouane, H.B. and Chellali, M., 2015. On secure domination in graphs. *Information Processing Letters, 115*(10), pp.786-790.

[45] O. Ore, *Theory of Graphs* (Vol 38), American Mathematical Society Colloquium Publications.

[46] Phillips, J.D., Schwanghart, W. and Heckmann, T., 2015. Graph theory in the geosciences. *Earth-Science Reviews, 143*, pp.147-160.

[47] Siebertz, S., 2019. Greedy domination on biclique-free graphs. *Information Processing Letters, 145*, pp.64-67.

[48] Schiermeyer, I., 2008. Efficiency in exponential time for domination-type problems. *Discrete Applied Mathematics, 156*(17), pp.3291-3297.

[49] Spalding, A., 1998. *Min-Plus Algebra and Graph Domination.* (Ph.D. thesis, University of Colorado, Colorado, United States of America)

[50] van Rooij, J.M. and Bodlaender, H.L., 2008. Design by measure and conquer, a faster exact algorithm for dominating set. *arXiv preprint arXiv:0802.2827*.

[51] van Rooij, J.M., 2011. *Exact exponential-time algorithms for domination problems in graphs*. BOXpress.

[52] Vecchio, F., Miraglia, F. and Rossini, P.M., 2017. Connectome: Graph theory application in functional brain network architecture. *Clinical neurophysiology practice, 2*, pp.206-213

[53] Xue, H.L., Liu, G. and Yang, X.H., 2016. A review of graph theory application research in gears. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 230*(10), pp.1697-1714.