Universiteit
Leiden
The Netherlands

Solving Breakthrough Using Binary Decision Diagrams and Retrograde Analysis

Elze de Vink

Supervisor:
Dr. Alfons Laarman

Bachelor Thesis

Opleiding Informatica

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

19/08/2021

Abstract

The field of Artificial Intelligence has long been involved with two-player games. BDDs are used to solve some of these combinatorial problems. In this thesis, we aim to strongly solve the two-player game *Breakthrough*, symbolically, using BDDs. We provide an encoding of the game board and the transition relation in order to traverse the state space. Using retrograde analysis, we are able to solve Breakthrough for board sizes up to 5x5 and 7x3.

Contents

Chapter 1

Introduction

Artificial intelligence (AI) has been a hot topic for a couple of decades already. The aim is to create an AI that can be applied in real world scenarios and research in board games and combinatorial problems are helping the field get there. Huge strides have already been made, from a neural network Backgammon player to beating champions of Go [1][2]. More recently, researchers have gotten more involved with using reinforcement learning to create video game agents capable of super human levels op play [3].

Although these researchers aimed to create very proficient players, other AI challenges include solving a game. Solving sequential combinatorial games involves state space search and storing and evaluating large amounts of board states. As an example, we focus on a board game with an easy rule-set called *Breakthrough*. This game is played on an $m \times n$ board where the goal is to reach your opponent's home row or capture all your opponent's pawns. Pawns can move one square straight forward or diagonally, only being able to take an opponent's piece with a diagonal move.

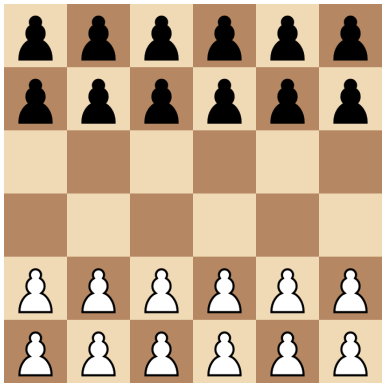


Figure 1.1: Initial board of 6x6 Breakthrough

We aim to solve Breakthrough using retrograde analysis working back from winning positions to label every board state as winning for one player. Because pawns can only move forward a draw or loop in the state space search is not possible. A game can be ultra-weakly solved if you provide the game theoretic ending, meaning won/lost/drawn for each player, for initial board states. A game is weakly solved if the game theoretic ending is given for initial board states with a

strategy on how to reach this ending. And lastly, a game can be strongly solved if the game theoretic ending is given for every legal board position, which is what we will be doing for Breakthrough.

Using this method has as a consequence that all possible board states need to be considered which will come to a very large amount of states to be stored. In order to handle this, we use Binary Decision Diagrams (BDDs) to represent board states. Although BDDs or equivalent structures that could represent Boolean functions had already been around, it was not until Bryant in 1986 that this data structure had been expanded by a fixed variable ordering and making rules to reduce the BDDs further in size[4]. These Reduced Ordered Binary Decision Diagrams (ROBDDs) were a great improvement over regular BDDs and in this thesis we also make use of this structure.

Other research on games using BDDs include an article on connect-four analyzing its state space search and its variable ordering proving that for a certain domain the BDDs can be polynomial in size[5]. Another paper uses BDD-based implementations for four parity games and compares their performance [6].

This thesis strongly solves Breakthrough boards up to the sizes 5x5, 6x4 and 7x3, for which both the second player/Black is winning from the start. A board size where the first player wins is 6x4. We also compare two methods of defining the baseline of winning board states. One where the winning move has been made and the other where a board is winning if a pawn can reach the opponent's home row. For board size 5x5 we find that we have 21 times less board states to consider.

Chapter 2 provides background knowledge for this thesis. The game of Breakthrough is discussed here and BDDs are explained as well as retrograde analysis. Chapter 3 describes the methods on how BDDs are used to represent board states for the game Breakthrough. This section continues into the use of retrograde analysis. Chapter 4 gives the experimental evaluation of the workings described in Chapter 3 and lastly Chapter 5 gives the conclusions.

Chapter 2

Related Work and Background

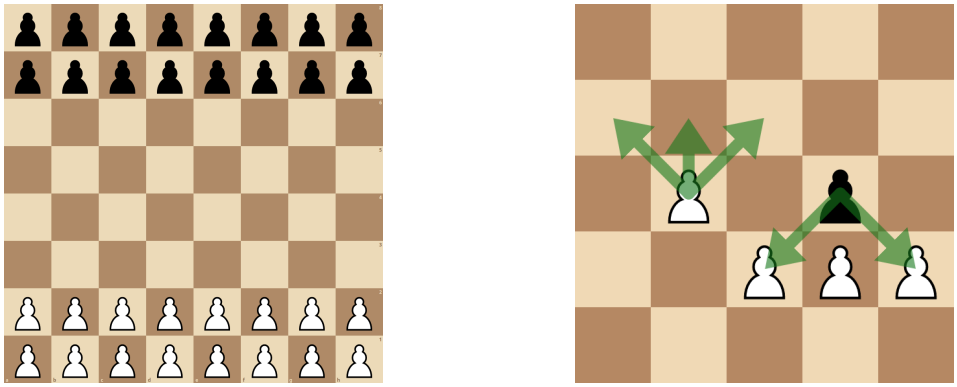
2.1 Breakthrough

Breakthrough was invented by Dan Troyka in 2000 and was originally designed to be played on a 7x7 board. He changed the board size to 8x8 to enter the 2001 8x8 Game Design Competition organised by About Board Games, Strategy Gaming Society and Abstract Games magazine. Breakthrough won the competition and it earned publications on the About Board Games website and small articles in the Strategy Gaming Society newsletter and Abstract Games magazine.

In this two-player game, the goal is to place a piece on the opponents "home" row or to take every piece of the opponent. Players White and Black take turns moving pieces where a pawn can be moved to 3 possible squares: the square directly in front of it and the two diagonal squares in front of the pawn. A pawn can never take the position that is owned by a pawn of its own colour. It can, however, take the pawn of an opposite colour if the pawn makes a diagonal move, the opponent's pawn is then removed from the game. Breakthrough can be played on a variety of board sizes, at the start of the game, each player's first two rows are filled with pawns. This starting position can be seen on an 8x8 board in Figure 2.1a along with Figure 2.1b showcasing the possible moves. Pieces can only go forward which means that no board position can be reached multiple times in a single game. Furthermore, it means that ties are not possible.

Although the game is relatively simple, it features surprising amounts of depth [7]. Defensive positions have to be maintained for as long as possible as a lone defence piece can not defend against an opponent's piece as that piece can always move directly in front of the defender making it unable to be captured. Skilled players can spot and set up forced wins comprised of several forced moves. Other researchers have created a bot using Monte Carlo Tree Search (MCTS) for playing on an 8x8 board [8]. Note that MCTS is unable to prove the game theoretic value and is designed to chose the most promising move based on many playouts of random moves. When the paper was written in 2013, this bot was ranked twelfth on Little Golem, a website for two-player combinatorial games with a Breakthrough playing community. After more improvements after the publication, the bot reached a rating of 2358, whereas the current highest rated player has a rating of 2275.

Although multiple papers have been written on the use of MCTS, other research on Breakthrough includes a solution for 6x5 boards based on race patterns and an extension of Job-Level Proof



(a) Starting Position (b) Display of possible moves

Figure 2.1: 8x8 board of Breakthrough.

Number search [9]. There have also been researchers generating end game tablebases for 6x6 boards [10].

Definition 2.1.1 will give the description of a state in Breakthrough.

Definition 2.1.1. A state A in Breakthrough is a tuple $(V^{m\times n}, p)$, with $V^{m\times n}$ being an $m \times n$ matrix where entries $V_{i,j}$ give the contents of square (i, j) on an $m \times n$ Breakthrough board. $V_{i,j} \in \{W, B, E\}$, where W and B represent an white and black pawn respectively and E an empty square. p represents which player is to move, $p \in \{0, 1\}$.

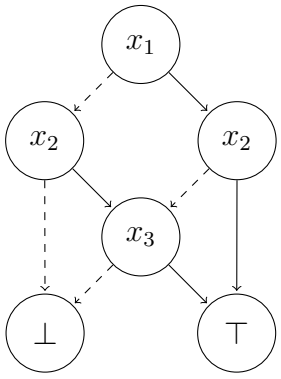
2.2 Binary Decision Diagrams

A Binary Decision Diagrams (BDD) is a data structure that is used to represent and manipulate Boolean functions. Figure 2.2 shows the BDD for the Boolean function $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$. BDDs are rooted, acyclic, directed graphs where every node represents a variable x_i and has two successors shown by the lines going down. From node x_i the dotted line, called LO, is taken when $x_i = 0$. Otherwise, when $x_i = 1$, the regular line, called HI, is taken. Following a path leads to one of the two sink nodes, either TRUE or FALSE. We use the symbol $\boxed{\top}$ to denote the TRUE sink node and $\boxed{\bot}$ to denote the FALSE sink node.

In order to keep BDDs small and simplify manipulation of BDDs, two restrictions can be enforced on BDDs. These applied constraints on a BDD make it *reduced* and *ordered*, definitions on these constraints are given by Definitions 2.2.1 and 2.2.2 respectively. These two constraints also ensure that the BDD is in a canonical form, meaning that there is a unique representative BDD for a certain function [4]. Figure 2.3 shows how a Reduced BDD (RBDD) is constructed from a non-reduced BDD. From this point, we will use the term BDD to refer to Reduced Ordered Binary Decision Diagrams (ROBDD).

Definition 2.2.1. A BDD is *reduced* if no node has an identical HI and LO. Also, multiple nodes representing the same variable may not have identical outgoing branches as these nodes can be joined.

Definition 2.2.2. A BDD is *ordered* if no node is considered multiple times when on a path from root to sink node. This is done by ensuring that whenever a LO or HI path goes from x_i to x_j, we have that $i < j$.



x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 2.2: The BDD for the Boolean function $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$ and its truth table.

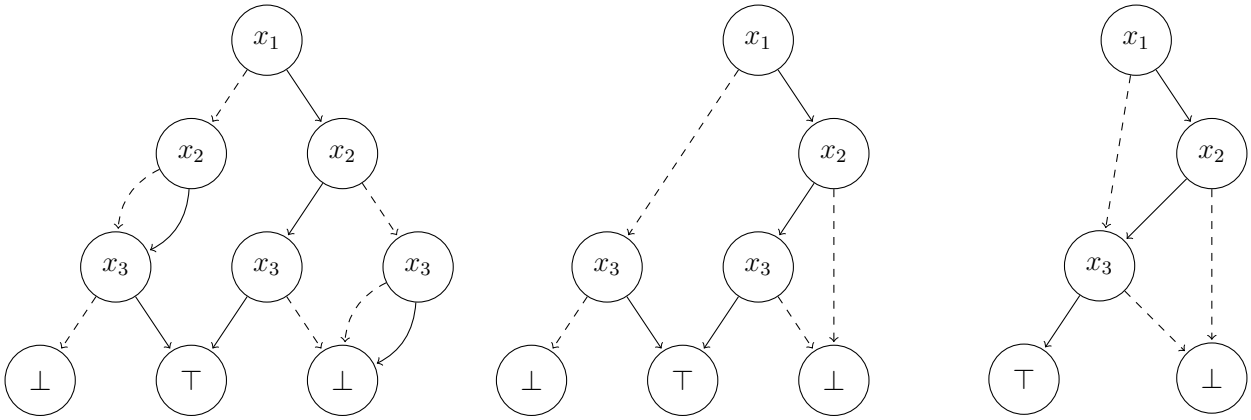


Figure 2.3: A non-reduced BDD alongside the reduced form. First the two nodes that have outgoing edges to the same node are removed. Then the two nodes that represent the same variable with identical outgoing edges are merged.

2.2.1 BDD Operations

One of the major advantages of the BDD representation, is the ability to perform operations on the structures, identical to the way you can perform operations on a Boolean function. Say we have two functions $f(x, y)$ and $g(x, y)$, we can combine these functions into $h(x, y) := f \cup g$. In the same way we can combine the BDDs representing $f(x, y)$ and $g(x, y)$, F and G, into a BDD that represents h by combining these BDDs. BDD packages allow the creation of this BDD H by calling the function *OR(F, G)*.

Along with the more common operations `OR`, `AND`, etc, BDD packages can also include the functions *Image* and *Preimage*. *Image(S,T)* is used to produce the image of a set S under a transition relation

$T(\vec{x}, \vec{x}')$, with $\vec{x}$, $\vec{x}'$ being sets of variables. It is defined as $Image(S,T) = \{x' \mid \exists x \in S, (x,x') \in T\}$. In our case, we use this operation to find successive states after a move in Breakthrough. *Preimage* is defined as $Preimage(S,T) = \{x' \mid \exists x \in S, (x',x) \in T\}$ and is used for backward induction.

2.2.2 BDD Packages

BDD packages are tools for handling and manipulating BDDs. While CUDD (Colorado University Decision Diagram) [11] is one of the most widely used package, there exist many other packages described in the comparative paper by Van Dijk et al [12]. For this thesis, we make use of Sylvan [13], a BDD package that uses work-stealing and scalable parallel data structures to provide parallelisation for the algorithms on BDDs. Sylvan has also implemented the image and preimage functions, in Sylvan named "Rel-next" and "Rel-prev". It also includes "Rel-prev for all" which use will be described in Section 3.2.3.

2.3 Retrograde Analysis

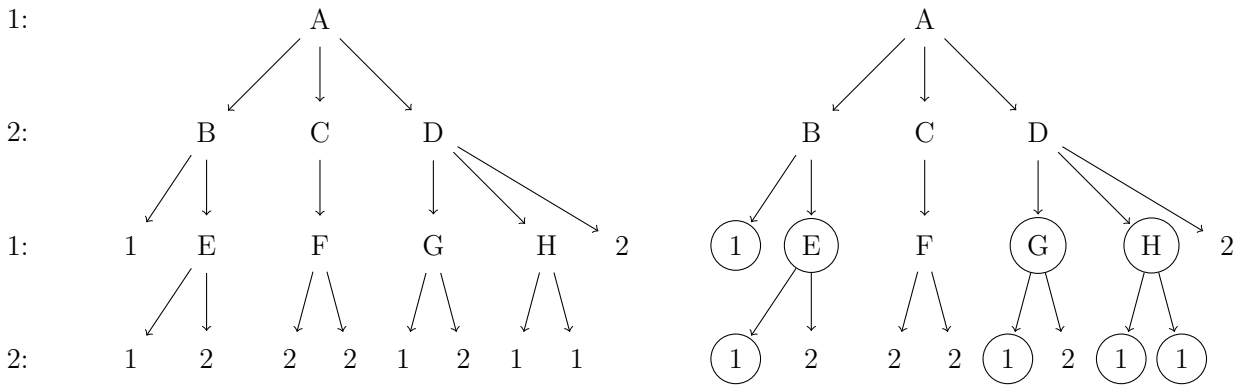
Retrograde analysis is a technique used to find the set of winning boards by working our way back from a winning position and adding those boards that can or must lead into winning positions later on in the game. This technique has everything to do with the state space of the game and the traversal between those states.

This technique is best showcased by an example of a state space of a two-player game. We name the two players Player 1 and Player 2 with a state space shown in Figure 2.4 where each state is a capitalized letter. This tree has root node A which is the initial state. Nodes that have numbers are end-states where the number is the player that has won in that position. Finding the complete set of winning states for each player is the goal and this must be done for each player individually. For this example we first consider Player 1. We start by creating the set of won states, the states have are represented by the number 1. We then check if this set can be expanded, this can be done in two ways:

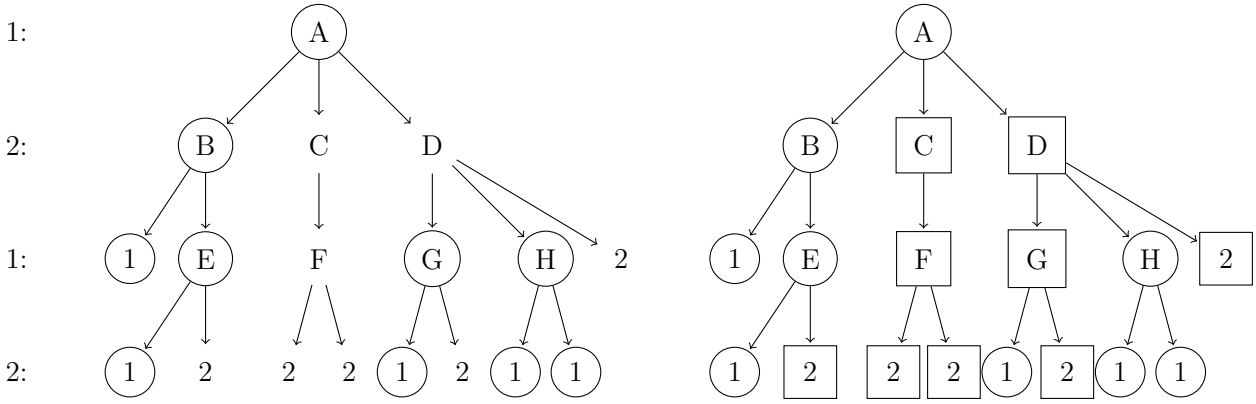
- A board is added if Player 1 can make a move towards a winning state.
- A board is added if Player 2 *has* to make a move towards a winning state (for Player 1).

The first way is shown in Figure 2.4a on the left where state E, G and H are encircled (meaning added to the set of winning states). In the next step, state B is added as Player 2 can only move from this state to a winning state. These two ways of adding states is then looped until no state can be added, this finishes here by adding state A, the initial state, in Figure 2.4b on the left. The tree on the right of Figure 2.4b also shows the entire state space with the states that are winning for Player 2 squared. It is of course also possible for a state to be tied between the players, however this is not the case for Breakthrough.

In Definition 2.1.1 we defined a game state in Breakthrough, in Definition 2.3.1 a general two-player game will be defined and in Definition 2.3.2 the game Breakthrough will be formalized in this same



(a) Left: Starting state space. Right: Winning states and immediately winning states encircled



(b) Left: Boards that can only move into winning states for player 1 are added along with the root node A that now has a winning board as an option. Right: Fully filled-in state space divided into 2 sets.

Figure 2.4: Example of retrograde analysis. Winning Boards for Player 1 are encircled and winning boards for Player 2 are squared. The number on the left represents the player to move from the states in that row.

form for an 8×8 board.

Definition 2.3.1. A two-player game is a tuple $(Q_0, Q_1, S, E_0, E_1, M)$ with Q_0/Q_1 the set of reachable game states where P_0/P_1 is to move, S the set of starting states, E_0/E_1 the sets of end states for P_0/P_1 and M the move relation. The tuple also has the following properties:

- $Q \triangleq Q_0 \uplus Q_1$ (Q is the union of the two disjoint game state sets)
- $S \subseteq Q_0$ (P_0 moves first)
- $E_0, E_1 \subseteq Q$ with $E_0 \cap E_1 = \emptyset$ (Disjoint sets of end states for P_0/P_1)
- $M \triangleq M_0 \uplus M_1$ with $M_0 \subseteq Q_0 \times Q_1$ and $M_1 \subseteq Q_1 \times Q_0$ (Move relation for P_0/P_1)

Definition 2.3.2. Breakthrough is a two-player game $B = (Q_0, Q_1, S, E_0, E_1, M)$ played on an $m \times n$ board. Player 0 plays with the white pawns and Player 1 with the black pawns, both players

start with two rows of pawns opposite each other. Pawns can move one square straight forward or diagonally forward if the square is free. A pawn can take an opponents pawn if a diagonal move is made. A player has won if one of its pawns has reached the furthest row or after having taken all pieces of the opponent. Note that a player has won if the winning move has been made, meaning if P_0 has won, the won board state is in Q_1.

- Q_i is the set of states with a state $A = (V^{m \times n}, i)$ with $V_{i,j} \in \{W, B, E\}$ for white pawn, black pawn and empty square.
- $S \triangleq \{A\}$ with $A_{i,j} = \begin{cases} W & \text{if } 1 \leq i \leq 2 \\ E & \text{if } 3 \leq i \leq 6 \\ B & \text{if } 7 \leq i \leq 8 \end{cases}$
  Starting positions with rows of white and black pawns with empty rows in between.
- $E_i \triangleq F_i$ (stone on furthest row) $\cup\, T_i$ (all pieces taken) with:
  $F_i \triangleq \{A = (V^{8\times 8}, p) \mid p \in \{0,1\},\ p \neq i,\ A \in Q,\ A \notin Q_i, \begin{cases} \exists x : A_{8,x} = W & \text{if } i = 0 \\ \exists x : A_{1,x} = B & \text{if } i = 1 \end{cases} \}$
  T_i is the subset of $Q_{!i}$ such that $\texttt{image}(M, T_i) = \emptyset$

Calling a game solved describes the knowledge of the games outcome. There are still distinctions to be made in what way a game can be solved. Three definitions have been stated for two-player games with perfect information [14]. Allis describes the distinctions between a game that is *ultra-weakly solved*, *weakly solved* and *strongly solved.*

- **Ultra-weak**: From the start of the game the game theoretic ending (either a draw or win for a certain player) assuming that both players play perfectly. Such a "perfect" game need not be given.
- **Weak**: From the start of the game the game theoretic ending is determined and a strategy is given how to reach this ending.
- **Strong**: The game theoretic ending is determined for every legal position.

Chapter 3

Execution of Breakthrough in BDDs

3.1 Encoding a Breakthrough State in BDDs

To solve Breakthrough we will need to traverse the state space. To be able to do this we will have to convert a board state to the form of a BDD. Every square on the board will have to be considered and represented. Each square can be occupied in three ways, it can either be empty or occupied by a white or black pawn. Because there are three possibilities we will need two Boolean variables to represent a square, in Table 4.1 the chosen variable encoding can be seen. Note that when x_1 and x_2 for square x are both 1 we reach a square state named "impossible". This is because it is not possible for a square to reach this state. At the start of a game each square has one of the three possible contents and it never leaves one of these three states by any possible move, this can be seen in Figure 3.1.

x_1	x_2	Square State
0	0	Empty
0	1	White Pawn
1	0	Black Pawn
1	1	Impossible

Table 3.1: Chosen square encoding for Breakthrough.

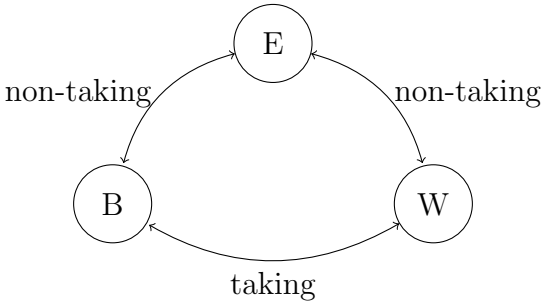


Figure 3.1: Square state transitions with taking and non-taking moves.

In addition to the encoding of the $(m \times n)$ squares for an $m \times n$ board we need one additional variable. This variable P is 1 when White is to move and 0 when Black is to move. So, to encode a Breakthrough state $2(m \times n) + 1$ variables are required.

3.2 Implementing Retrograde Analysis

Using retrograde analysis to solve a game by backward induction and BDDs requires creating the transition relation between a set of states and its successor after every possible move. With this transition relation we can not only reach every possible board state by calling the image function but also move backwards using the preimage function. Using the image function we move from the initial board state to end states while for retrograde analysis we move from a player's complete set of winning board states to board states earlier in the game. This has to be done for both players separately as will be discussed further below. We can expand each player's set of winning boards until all board states have been categorized to be in either player's set of winning boards. When we find in which set the initial board state lies we know which player can force a win from the start of the game.

We applied two methods of creating the initial sets of winning board states. The first more straightforward method takes those boards where the state has already been won while the second method takes those boards where a winning move can be made. This change already decreases the amount of states visited greatly.

3.2.1 Move Translation

To make use of *Image(S,T)* we must not only give the *source* or *unprimed* variables $\{x_0, ..., x_n\}$ of the current set of states S. These variables are also paired with the *target* or *primed* variables $\{x'_0, ..., x'_n\}$. The used BDD package Sylvan requires interleaving variable ordering meaning that the variables are combined as $\{x_0, x'_0, ..., x_n, x'_n\}$. After the computation the target variables have their new values and a target set of states S' has the set of states after one move coming from S. S then takes the value of S' to be used again.

To create the translation relation to be used for *Image(S,T)* and *Preimage(S,T)* we operate on the accumulation of subsets of variables. In a Breakthrough move, not all state variables need to be changed. The partial relations are made up of all the possible pairs of squares that are connected with one move. Using algebraic board notation as used in chess, we say that a1 is paired with b1 and b2 and that b2 is paired with c1, c2, c3, a1, a2 and a3. There are then three different possibilities of moves to be made for a combination of connected squares we will call sq_0 and sq_1. These three possibilities are described for the White player:

1. Straight across: A move can be made if sq_0 is White and sq_1 is empty (non-taking move).
2. right or left across: A move can be made if sq_0 is White and sq_1 is empty (non-taking move).
3. right or left across: A move can be made if sq_0 is White and sq_1 is Black (taking move).

After a move, not only are the variables of sq_0 and sq_1 altered, but the Player variable is also swapped.

To calculate the amount of reachable board states in Breakthrough we move through the state space using breadth-first search starting from the single initial board state. This is done by repeatedly calling *Image(S,T)* and adding unseen board states to a set of BDDs. Board states can be reached by a different order of moves but because all pawns only move forward it is not possible to revisit a board state or create a loop in this complete run through of all board states. When a state is reached that is a winning or end state, it is first added to the reached states and then removed from the set of states to be further worked on. This is to avoid unreachable states, for instance those states where both players reach their opponents home row.

3.2.2 Winning Boards

Before the retrograde analysis can begin and the set of winning boards can be expanded we need to create the sets of board states that are won for each player. In our first method this is when a pawn has *reached* the opponent's home row or when all opposing pawns have been taken. For White's set of won boards, this is when White has *made* the winning move and the resulting board state has the player variable reading Black to move.

The second method is different in that a board state is declared as winning for White if all Black's pawns have been taken, meaning the player variable has Black to move. This is still the same as for the first method, but where that method takes boards where White has reached Black's home row, here we take those boards where White is to move when a pawn is on the row before Black's home row. These board states have an immediately winning move for White meaning they can already be seen as winning board states themselves.

3.2.3 Algorithm

Applying retrograde analysis is done by expanding the two disjoint sets of winning board states for the two players. We must first look at how a board state can be added to such a set. There are two distinct ways a board state can be added and they are based on which player is to move. Let us think of increasing the White player's winning boards, a board is added when White is to move and a move can be made to go to a board state that is in the set of winning board states. This can be formulated like so with board state x, y and set of winning board states for White W_w: $\{x \mid \exists Image(x, T) \in W_w\}$.

Board states where the opponent is to move are also added when all possible moves lead to states that are in the winning set of states. For this you need the preimage function, the function that gives $Preimage(S, T) = \{\forall x' \mid \exists x \in S, (x', x) \in T\}$. For White, these boards can be defined by $\forall\exists Preimage(W_w, T) = \{x' \mid \forall x' \in Image(x, T) \in W_w,\ x \in Preimage(W_w, T)\}$. $\forall\exists Preimage(S, T)$ will give all board states one move before S that will definitely end up in S as all its possible moves lead to S. These two possible ways of expanding the set of winning board states follow each other in a loop until states are no longer added.

We separate these two sets by who is to move as they can only grow from each other. It is not possible to move from a state where White is to move to a winning state where White is to move, only a winning state where Black is to move because after a single move the player variable only

flips once. The loop is given in Algorithm 1 for White where BaselineWhiteWin are those states where White has already made the winning move, and so Black is to move. The operation '|' is the symbol for `OR` meaning the expansion of a set here.

Algorithm 1 Retrograde Analysis Loop for White

```
WhiteWinFinal = BaselineWhiteWin
WhiteWinBlackMove = BaselineWhiteWin
WhiteWinWhiteMove = Bdd 0
temp, Prev = Bdd 0
while WhiteWinFinal != temp do
    if WhiteWinBlackMove == Bdd 0 then
        break

    temp = WhiteWinFinal
    Prev = Preimage(WhiteWinBlackMove, T)
    WhiteWinWhiteMove |= Prev
    WhiteWinFinal |= Prev

    Prev = ∀∃Preimage(WhiteWinWhiteMove,T)
    WhiteWinBlackMove = Prev
    WhiteWinFinal |= Prev
return WhiteWinFinal
```

When this loop has been completed for both players, every legal board position has been added to either W_w or W_b. Simple checks can be done to see whether there is no overlap between the two disjoint sets of winning states and whether the sets add up to the total amount of reachable states. Now, you can also inspect in what set the initial position has ended up in to see which player is winning from the starting position.

Chapter 4

Experimental Evaluation

To get the results of this chapter code has been written in c++ that uses the BDD package Sylvan as mentioned in Subsection 2.2.2. The code for this project can be found on https://github.com/elzedevink/Breakthrough-BDD.

In Table 4.1 the results for varying board sizes are given. In the table the winning player is given alongside the number of reachable states where most board states have the second player as their winner while on a 6x4 board the first player wins. The amount of White/Black winning boards per board size do not differ that much. For instance, for method 1, on a 5x5 board, White has $2.57 \cdot 10^{10}$ winning board states while Black has $2.52 \cdot 10^{10}$. This difference is similarly small for each board size/method.

	5x4	5x5	6x4	7x3
Winning	2nd Player	2nd Player	1st Player	2nd Player
M1 #reachable states	$3.7 \cdot 10^8$	$5.1 \cdot 10^{10}$	$1.5 \cdot 10^{10}$	$3.2 \cdot 10^8$
M2 #reachable states	$3.5 \cdot 10^7$	$2.4 \cdot 10^9$	$2.0 \cdot 10^9$	$1.0 \cdot 10^8$

Table 4.1: Winning player and number of reached states for several board sizes.

Table 4.2 shows the amount of nodes of the BDD that represents the reached board states. In Figure 4.1 you can see that when you calculate this number of nodes it reaches a peak before the BDD has fully reached all possible states.

Board Size	Peak	End
5x4	170075	15506
5x5	2693209	2494183
6x4	350532	292921
7x3	57865	51133

Table 4.2: Number of nodes representing the BDD of visited states.

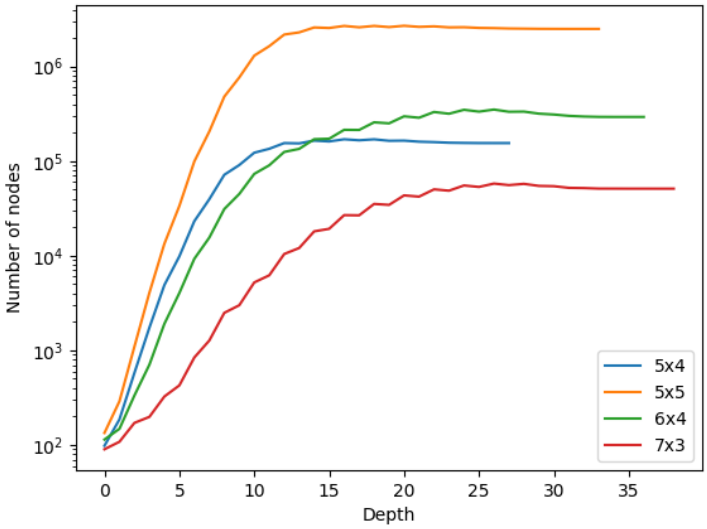


Figure 4.1: Number of nodes of the BDD that represents the reached states logarithmically scaled for different board sizes.

Chapter 5

Conclusions and Further Research

In this thesis, the two-player combinatorial game Breakthrough was strongly solved for board states up to 5x5, 6x4 and 7x3 being able to state the game theoretic ending for all reachable states. We showed how Breakthrough can be encoded as a BDD and how we can traverse the state space using the image function to reach all possible states. We showed how to build the transition relation needed for this function and how it can also be used to apply retrograde analysis to ultimately expand sets of winning board states.

Even though we used BDDs to be able to handle large amounts of board states, we were not able to solve board sizes 6x5 which has $4.2 \cdot 10^{11}$ possible board states and 5x6 which has $1.6 \cdot 10^{11}$.

Breakthrough was not too complex to make work with BDDs and its move translation as only two nearby squares need to be considered for a move and the fact that a square can only be in one of three states. It is possible that further research can be done on more complex problems. Also, as was done for connect-four[5], research can be done for Breakthrough to look into variable ordering as this has been demonstrated to change the size of the resulting BDD[15].

Bibliography

[1] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.

[2] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[3] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," *CoRR*, vol. abs/1912.06680, 2019.

[4] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, 1986.

[5] S. Edelkamp and P. Kissmann, "On the complexity of bdds for state space search: A case study in connect four," in *AAAI*, 2011.

[6] L. Sanchez, W. Wesselink, and T. A. Willemse, "A comparison of bdd-based parity game solvers," *Electronic Proceedings in Theoretical Computer Science*, vol. 277, p. 103–117, Sep 2018.

[7] K. Handsomb, "8x8 game design competition: The winning game: Breakthrough ...and two other favorites," *Abstract Games Megazine*, vol. 7, pp. 8–9, 2001.

[8] R. Lorentz and T. Horey, "Programming breakthrough," pp. 49–59, 2013.

[9] A. Saffidine, N. Jouandeau, and T. Cazenave, "Solving breakthough with race patterns and job-level proof number search," vol. 7168, November 2011.

[10] A. Isaac, "Generating an end game tablebase for the game of breakthrough using quasi-retrograde analysis," 2016.

[11] F. Somenzi, "Cudd: Cu decision diagram package release 2.4.1," 2005.

[12] T. van Dijk, E. M. Hahn, D. N. Jansen, Y. Li, T. Neele, M. Stoelinga, A. Turrini, and L. Zhang, "A comparative study of bdd packages for probabilistic symbolic model checking," in *Dependable Software Engineering: Theories, Tools, and Applications* (X. Li, Z. Liu, and W. Yi, eds.), (Cham), pp. 35–51, Springer International Publishing, 2015.

[13] T. van Dijk and J. van de Pol, "Sylvan: multi-core framework for decision diagrams," *International Journal on Software Tools for Technology Transfer*, vol. 19, pp. 675–696, Nov 2017.

[14] L. Allis, "Searching for solutions in games and artificial intelligence," 1994.

[15] R. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, p. 293–318, Sept. 1992.