Universiteit
Leiden
The Netherlands

Solving Breakthrough Using Binary Decision Diagrams and Retrograde Analysis

Elze de Vink

Supervisors:
Dr. Alfons Laarman
Dr. Walter Kosters

Bachelor Thesis

Opleiding Informatica

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

04/01/2022

Abstract

The field of Artificial Intelligence has long been involved with two-player games. Binary Decision Diagrams (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 solve Breakthrough for board sizes 5×4, 5×5, 6×4 and 7×3.

Contents

1 Introduction **1**

2 Background **3**
2.1 Breakthrough . . . 3
2.1.1 Rules . . . 3
2.1.2 On Breakthrough . . . 4
2.2 Binary Decision Diagrams . . . 4
2.2.1 BDD Operations . . . 5
2.3 Retrograde Analysis . . . 6

3 Related Work **9**
3.1 Breakthrough research . . . 9
3.2 Connect Four . . . 9
3.3 BDD Packages . . . 9

4 Execution of Breakthrough in BDDs **11**
4.1 Encoding a Breakthrough State in BDDs . . . 11
4.2 Implementing Retrograde Analysis . . . 12
4.2.1 Encoding of the Move Relation . . . 12
4.2.2 Winning Boards . . . 13
4.2.3 Algorithm . . . 13

5 Experimental Evaluation **15**

6 Conclusions and Further Research **17**

References **19**

Chapter 1

Introduction

Artificial intelligence (AI) has been a hot topic for a couple of decades already. The aim is to create agents that can be applied in real world scenarios. By conducting research in areas like combinatorial problems and board games the field of AI has been able to apply techniques to more complex problems. Huge strides have 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 of play [3].

Although these researchers aimed to create very proficient players, other AI challenges include solving a game. Solving sequential combinatorial games often involves state space search and storing and evaluating large amounts of board states. In this paper, 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 the opponent's home row or capture all the opponent's pawns. Pawns can move one square forward or diagonally, only being able to capture an opponent's pawn with a diagonal move. See Figure 1.1 for the starting position for a 6×6 board.

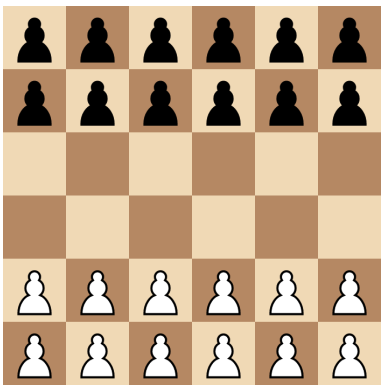


Figure 1.1: Initial board of 6×6 Breakthrough.

We aim to solve Breakthrough but this can mean multiple things. There are still distinctions to be made in what way a game can be solved. Three definitions have been proposed for two-player games with perfect information [4]. Allis describes the distinctions between a game that is *ultra-weakly solved*, *weakly solved* and *strongly solved*.

A game is *ultra-weakly solved* if one provides the game theoretic value, meaning won/lost/drawn for each player assuming perfect play, for initial board states [4]. A game is *weakly solved* if the game theoretic value is given for initial board states with a strategy on how to reach this value. Lastly, a game is *strongly solved* if the game theoretic value is known for every legal board position, which is what we will be doing for Breakthrough. This also includes an optimal strategy for every game position.

Using this method has as a consequence that all possible board states need to be considered, which will lead 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 providing rules to reduce the BDDs further in size [5]. 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.

This thesis strongly solves Breakthrough boards for several sizes. For board sizes 5×4, 5×5 and 7×3 the second player is winning from the start. For the board size 6×4 the first player is winning. 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 state is winning if a pawn can reach the opponent's home row. For a 5×5 board, we find that there are 21 times fewer board states to be considered if we use the method of defining winning board states where the winning move has not yet been made.

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 provides related works on the subjects discussed to give a good view on how some of these techniques are used. Chapter 4 describes the methods on how BDDs are used to represent board states and how retrograde analysis has been implemented. Chapter 5 gives the experimental evaluation of the workings described in Chapter 4 and lastly Chapter 6 gives the conclusions and discusses further research. This bachelor thesis was created under the supervision of Alfons Laarman and Walter Kosters from the Leiden Institute of Advanced Computer Science (LIACS).

Chapter 2

Background

This chapter provides background information to understand this thesis.

When referring to the game pieces of Breakthrough we will call them 'pawns' as they move similarly to pawns in chess. The two players will be called 'White' and 'Black' with White being the starting player. White will also be referred to as Player 0 or P_0 and Black as Player 1 or P_1.

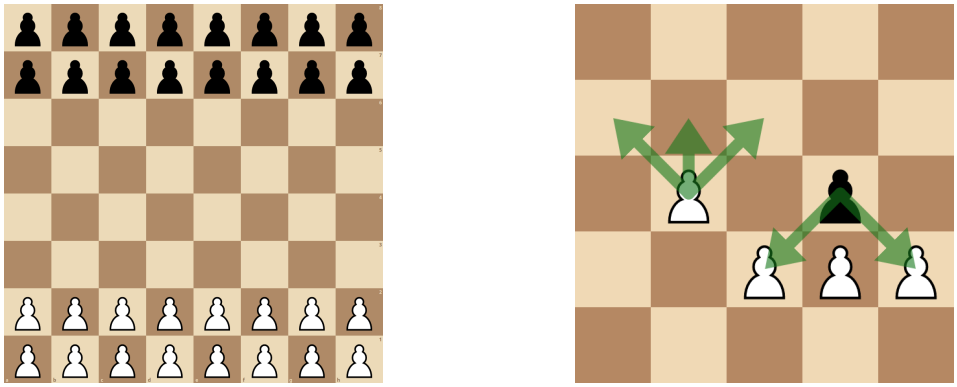
2.1 Breakthrough

Breakthrough was introduced by Dan Troyka in 2000 and was originally designed to be played on a 7×7 board. He changed the board size to 8×8 to enter the 2001 8×8 Game Design Competition organised by About Board Games, Strategy Gaming Society and Abstract Games magazine. Breakthrough won the competition and it earned a publication on the About Board Games website and a article in the Abstract Games magazine [6].

2.1.1 Rules

In this two-player game, players White and Black take turns moving pawns on a $m \times n$ board. A pawn can be moved to three 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, capture (or take) a pawn of an opposite colour if the pawn makes a diagonal move, the opponent's pawn is then permanently removed from the game. The goal is to place a pawn on the opponent's "home" row or to capture every pawn of the opponent.

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 8×8 board in Figure 2.1a along with Figure 2.1b showcasing the possible moves. Because pawns can only move forward a draw or loop in the state space search is not possible.



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

Figure 2.1: The initial position of an 8×8 board of Breakthrough. A pawn can move a square forward if that square is vacant, it can also move one square diagonally forward thereby taking the opponent's pawn.

2.1.2 On Breakthrough

Although the game is relatively simple, it features surprising amounts of depth [6]. Defensive positions must be maintained for as long as possible as a lone defence piece can not defend against an opponent's pawn as that pawn 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.

Definition 2.1.1 gives 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. Here $V_{i,j} \in \{W, B, E\}$, where W and B represent a white and black pawn, respectively, and E an empty square. Furthermore, p represents which player is to move, $p \in \{0, 1\}$. If p is 0 it means that White is to move, otherwise Black is to move.

2.2 Binary Decision Diagrams

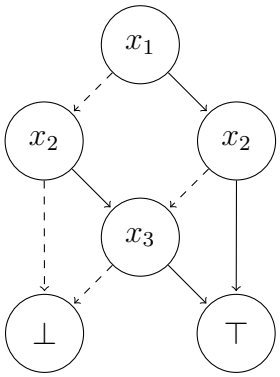
A Binary Decision Diagrams (BDD) is a data structure that is used to represent and manipulate Boolean functions. Figure 2.2 shows the BDD for the Boolean function $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$. BDDs are rooted, acyclic, directed graphs where every node 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 solid 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 [7].

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.

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 on any 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 $i < j$.

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 [5]. 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).



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: 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.

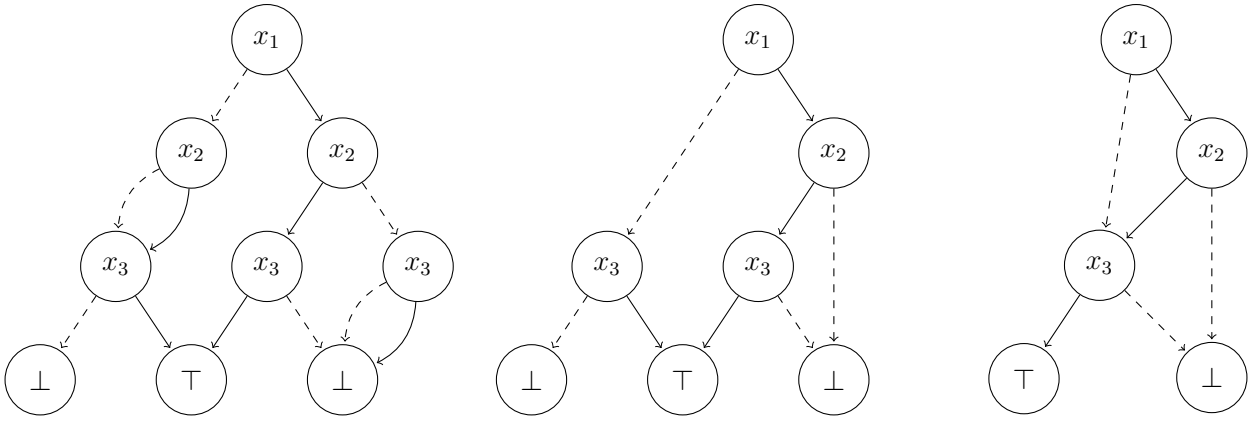


Figure 2.3: A non-reduced BDD alongside the reduced form. First the two nodes that have outgoing edges to the same node are removed. Then the two nodes that represent the same variable with identical outgoing edges are merged. Note that the BDD is ordered.

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 one can perform operations on a Boolean function. Say we have

two functions $f(x, y)$ and $g(x, y)$, we could then combine these functions into $h(x, y) := f \vee g$. 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 more common operations like `OR` and `AND` 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 $T(\overrightarrow{x}, \overrightarrow{x}')$, with $\overrightarrow{x}$, $\overrightarrow{x}'$ 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.

2.3 Retrograde Analysis

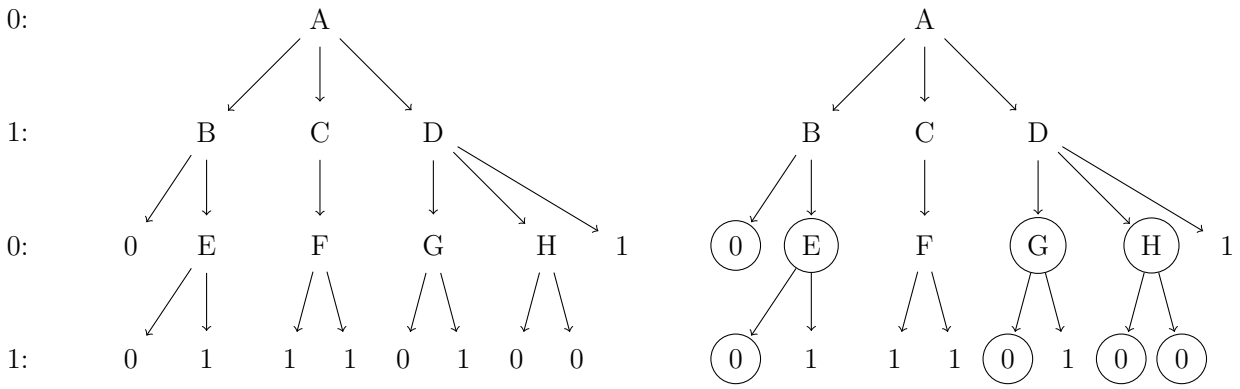
Retrograde analysis is a technique used to find the set of winning boards by working our way back from a winning position and adding those boards that can or must lead into winning positions later on in the game. This technique has everything to do with the state space of the game and the traversal between 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 0 and Player 1 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 terminal 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 is done for each player individually. For this example we first consider Player 0. We start by creating the set of won states, the states represented by the number 0. We then check if this set can be expanded, which can be done in two ways:

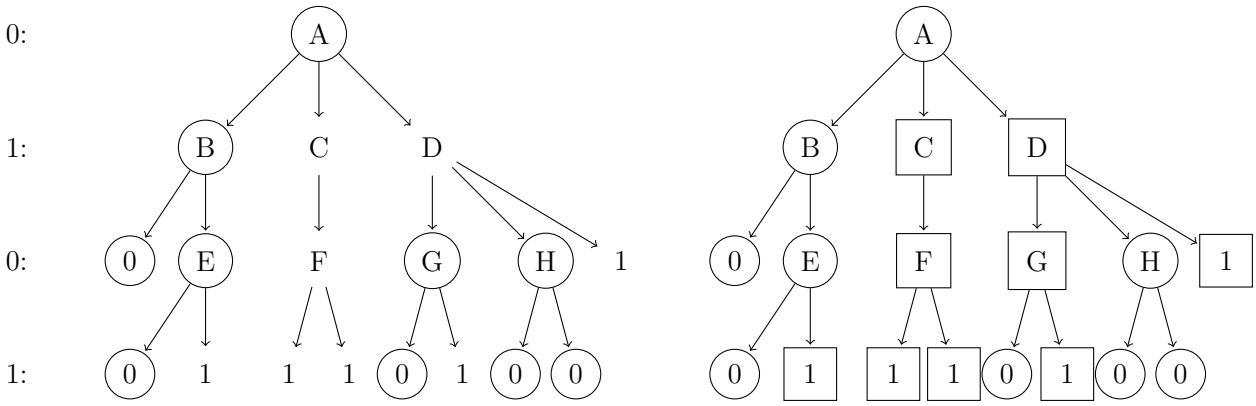
- A board is added if Player 0 can make a move towards a winning state.
- A board is added if Player 1 must make a move towards a winning state (for Player 0).

The first way is shown in Figure 2.4a on the left where state E, G and H are encircled (meaning that they are added to the set of winning states). In the next step, state B is added as Player 1 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. This game has now been weakly solved by providing the winning player and the winning strategy. The tree on the right of Figure 2.4b also shows the entire state space with the states that are winning for Player 1 squared. Now that all possible states have been labeled the game is strongly solved. In general it is also possible for a state to be tied between the players, however this is not possible 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 form for an 8×8 board. This follows the definition of a two-player game proposed in [8].



(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 0 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 two sets.

Figure 2.4: Example of retrograde analysis. Winning boards for Player 0 are encircled and winning boards for Player 1 are squared. The number on the left represents the player to move from the states in that row.

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 sets of 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 that are won 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 Q_0 and Q_1)
- $S \subseteq Q_0$ (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)
- $E_0 \subseteq Q_1$ and $E_1 \subseteq Q_0$ (All games won by P_i are such that P_i has made the winning move thus placing the resulting board in Q_{1-i})
- $M \triangleq M_0 \uplus M_1$ 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 starting from each player's home row. Pawns can move one square forward or diagonally forward if the square is free. A pawn can capture an opponent's pawn if a diagonal move is made. A player has won if one of its pawns has reached the furthest row or after having capture all pawns 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_p is the set of states with a state $A = (V^{m\times n}, p)$ with $V_{i,j} \in \{W, B, E\}$ for white pawn, black pawn and empty square respectively. Also, p represents which player is to move, $p \in \{0, 1\}$. If p is 0 it means that White is to move, otherwise Black is to move.

- If $m > 4$: $S \triangleq \{A\}$ with $A_{i,j} = \begin{cases} W & \text{if } 1 \leq i \leq 2 \\ E & \text{if } 3 \leq i \leq (m-2) \\ B & \text{if } (m-1) \leq i \leq m \end{cases}$
  Starting position with two rows of white and black pawns with empty rows in between. If there are fewer than five rows, there is only one row of pawns per player:
  $S \triangleq \{A\}$ with $A_{i,j} = \begin{cases} W & \text{if } i = 1 \\ E & \text{if } 2 \leq i \leq (m-1) \\ B & \text{if } i = m \end{cases}$

- $E_p \triangleq F_p$ (stone on furthest row) $\cup\, T_p$ (all pawns captured) with:
  $F_p \triangleq \{A = (V^{m\times n}, 1-p) \mid p \in \{0,1\},\ A \in Q,\ A \notin Q_p, \begin{cases} \exists x : A_{m,x} = W & \text{if } p = 0 \\ \exists x : A_{1,x} = B & \text{if } p = 1 \end{cases} \}$
  T_p is the subset of Q_{1-p} such that $Image(M, T_p) = \emptyset$

- $(A, A') \in M_0$ with $A_p = 0$, $A'_p = 1$ and (A, A') as one of these three moves:
  - Left diagonal move: $V_{k,\ell} = W,\ V_{k+1,\ell-1} \in \{E, B\},\ k < m,\ \ell > 1$,
    $V'_{i,j} = \begin{cases} V_{i,j} & \text{if } (i,j) \notin \{(k,\ell),\ (k+1,\ell-1)\} \\ E & \text{if } i = k,\ j = 1 \\ W & \text{if } i = k+1,\ j = \ell - 1 \end{cases}$
  - Forward move: $V_{k,\ell} = W,\ V_{k+1,\ell} = E,\ k < m$,
    $V'_{i,j} = \begin{cases} V_{i,j} & \text{if } (i,j) \notin \{(k,\ell),\ (k+1,\ell)\} \\ E & \text{if } i = k,\ j = 1 \\ W & \text{if } i = k+1,\ j = \ell \end{cases}$
  - Right diagonal move: $V_{k,\ell} = W,\ V_{k+1,\ell+1} \in \{E, B\},\ k < m,\ \ell < n$,
    $V'_{i,j} = \begin{cases} V_{i,j} & \text{if } (i,j) \notin \{(k,\ell),\ (k+1,\ell+1)\} \\ E & \text{if } i = k,\ j = 1 \\ W & \text{if } i = k+1,\ j = \ell + 1 \end{cases}$

  Moves in M_1 are similarly defined, going in the opposite direction vertically and changing the position of black pawns instead of white pawns.

Chapter 3

Related Work

This chapter provides related works on the topics discussed.

3.1 Breakthrough research

Lorentz and Horey [9] have created a bot using Monte Carlo Tree Search (MCTS) for playing on an 8×8 board. Note that MCTS is unable to prove the game theoretic value and is designed to choose 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 6×5 boards based on race patterns and an extension of Job-Level Proof Number search [10]. There have also been researchers generating end game tablebases for 6×6 boards [11].

3.2 Connect Four

Edelkamp and Kissmann [12] have determined bounds for BDD sizes for representing the reachable states in Connect Four. They used several different ordering methods while using a cell-wise encoding of the game. They show that BDDs are great for representing the reachable states if search is continued after finding a terminal state. When the termination criterion is taken into account a BDD with an exponential number of nodes is needed. When this can be disregarded, they prove that polynomial sized BDDs can represent the set of all possible states.

3.3 BDD Packages

BDD packages are tools for handling and manipulating BDDs. While CUDD (Colorado University Decision Diagram) [13] is one of the most widely used package, there exist many other packages described in the comparative paper by Van Dijk et al. [14]. They show that no single BDD package

dominates in each field. However, many packages could benefit from parallelisation of BDD operations, something the package Sylvan outperformed other packages with.

Sylvan [15] is a package develop by the Formal Methods and Tools group of the University of Twente and differs from other packages by implementing parallelized operations on BDDs and Multi-Terminal Binary Decision Diagrams (MTBDDs) thereby exploiting the power of multi-core machines.
For this thesis, we use Sylvan which is written in C. Sylvan has implemented the image and preimage functions, in Sylvan named "Rel-next" and "Rel-prev". It also includes "Rel-prev for all" which application will be described in Section 4.2.3.

Chapter 4

Execution of Breakthrough in BDDs

We encode Breakthrough as a two-player game in BDDs. To do this we first encode a single Breakthrough board in BDDs and explain the amount of variables needed for this. We explain the encoding of the set of winning states and the transition relation and we then implement retrograde analysis using the *Preimage* and the $\forall\exists$*Preimage* BDD operations.

4.1 Encoding a Breakthrough State in BDDs

To encode a state in Breakthrough 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 Figure 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 not a state a square in the game can be in either in theory or in our implementation.

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

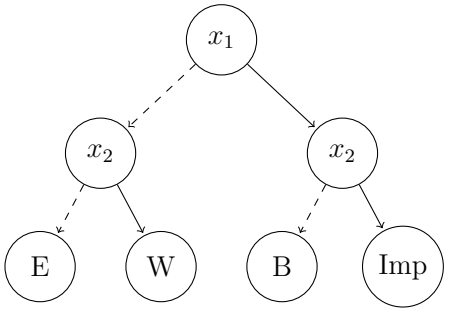


Figure 4.1: Square encoding for Breakthrough used in our implementation.

An improvement over our implementation is to remove the unnecessary impossible option altogether. This can be done by removing the impossible node and setting both outward paths from the rightmost x_2 to Black. Even better would be to remove both the rightmost x_2 and the impossible node and setting the HI path from x_1 to Black. This is shown in Figure 4.2.

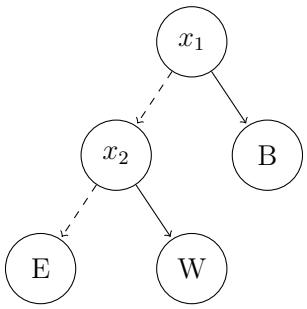


Figure 4.2: Improved encoding of a Breakthrough square requiring one less variable to encode a Black pawn.

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.

4.2 Implementing Retrograde Analysis

Using retrograde analysis to solve a game by backward induction and BDDs requires creating the move relation. In forward search, this is used to compute predecessors using the BDD image operation. Since retrograde analysis is a backward search procedure, we use the BDD preimage operation to compute predecessors. We start from the set of winning states and expand this set for each player separately. 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.

4.2.1 Encoding of the Move Relation

To make use of *Image*(S, T) we must not only give the *source* or *unprimed* variables $\{x_0, \ldots, x_k\}$ of the current set of states S. These variables are also paired with the *target* or *primed* variables $\{x'_0, \ldots, x'_k\}$. The used BDD package Sylvan requires interleaving variable ordering meaning that the variables are combined as $\{x_0, x'_0, \ldots, x_k, x'_k\}$. 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. The set 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. We see that a corner square like $A_{1,1}$ is paired with $A_{2,1}$ and $A_{2,2}$. A central square like $A_{2,2}$ is connected with $A_{3,1}$, $A_{3,2}$, $A_{3,3}$, $A_{1,1}$, $A_{1,2}$ and $A_{1,3}$. 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 as follows:

1. Forward: 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 the board states. When a state is reached that is a winning or is an 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 opponent's home row.

4.2.2 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 captured. 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 captured, 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.

4.2.3 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 and the 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 we 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 $\forall\exists Preimage(W_w, T) = \{x' \mid \forall x' \in Image(x, T) \in W_w,\ x \in Preimage(W_w, T)\}$. The function $\forall\exists Preimage(S,T)$ 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.

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 ← bddZero()                                   /* Empty BDD */
temp, Prev ← bddZero()
while WhiteWinFinal ≠ temp do
    if WhiteWinBlackMove is bddZero() then
        break
    end
    temp ← WhiteWinFinal
    Prev ← Preimage(WhiteWinBlackMove, T)
    WhiteWinWhiteMove ← WhiteWinWhiteMove | Prev
    WhiteWinFinal ← WhiteWinFinal | Prev

    Prev ← ∀∃ Preimage(WhiteWinWhiteMove, T)
    WhiteWinBlackMove ← Prev
    WhiteWinFinal ← WhiteWinFinal | Prev
end
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, one can also inspect in what set the initial position has ended up in to see which player is winning from the starting position.

Chapter 5

Experimental Evaluation

To get the results of this chapter code has been written in C++ that uses the BDD package Sylvan as mentioned in Chapter 2. The code for this project can be found on https://github.com/elzedevink/Breakthrough-BDD.

In Table 5.1 the results for varying board sizes are given. In the table, M1 (method 1) represents the method we first employed, where a board state is declared as winning if the winning move has been made. Likewise, M2 (method 2) represents the method where a board state is winning if the winning move can be made. Note that the two methods are in fact equivalent.

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 6×4 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 5×5 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.

	5×4	5×5	6×4	7×3
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 5.1: Winning player and number of reached states for several board sizes.

Table 5.2 shows the amount of nodes of the BDD that represents the reached board states. In Figure 5.1 it is shown that this number of nodes reaches its peak before all possible states have been reached.

Board Size	Peak	End
5×4	170,075	155,056
5×5	2,693,209	2,494,183
6×4	350,532	292,921
7×3	57,865	51,133

Table 5.2: Number of nodes representing the BDD of visited states.

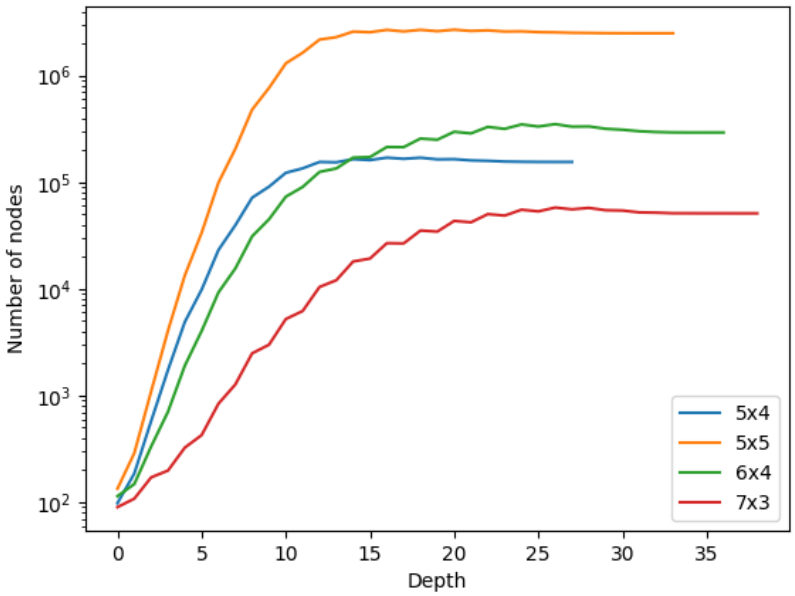


Figure 5.1: Number of nodes of the BDD that represents the reached states logarithmically scaled for different board sizes. The horizontal axis gives the depth of the game or the number of moves made.

Chapter 6

Conclusions and Further Research

In this thesis, the two-player combinatorial game Breakthrough was strongly solved for board sizes up to 5×5, 6×4 and 7×3. For this, the game theoretic ending for all reachable states was given. 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.

We also compared two methods for defining our start point of winning boards: one method where the winning move has already been made and the other method where a board is said to be winning if such a move can be made. For a board size of 5×5 the former method reaches $5.1 \cdot 10^{10}$ boards whereas the latter reaches $2.4 \cdot 10^{9}$ boards.

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

Before we can use AI techniques in real life scenarios they have to be tried and tested in a safe environment. While using BDDs along retrograde analysis to solve a game is not a novel method, applying it on different games or scopes of games can still be beneficial towards knowledge on the game.

Improvements on the methods shown in this thesis could result in larger board sizes being solved. As was done for Connect-Four [12], research can be done for Breakthrough to look into variable ordering as this has been demonstrated to change the size of the resulting BDD [16]. As mentioned in Chapter 4 simple improvements on board encryption could already lead to slightly better results.

References

[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, 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] V. Allis, "Searching for solutions in games and artificial intelligence," PhD thesis, Maastricht University, 1994.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, p. 677–691, 1986.

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

[7] D. E. Knuth, *The Art of Computer Programming*, vol. 4A. Addison-Wesley Professional, 2011.

[8] R. Greenlaw, H. J. Hoover, and W. Ruzzo, "A compendium of problems complete for P," 1992.

[9] R. Lorentz and T. Horey, "Programming Breakthrough," in *International Conference on Computers and Games*, pp. 49–59, Springer, 2013.

[10] A. Saffidine, N. Jouandeau, and T. Cazenave, "Solving Breakthrough with race patterns and job-level proof number search," in *Advances in Computer Games*, pp. 196–207, Springer, 2012.

[11] A. W. Isaac, "Generating an end game tablebase for the game of Breakthrough using quasi-retrograde analysis," Master's thesis, California State University, Northridge, 2016.

[12] S. Edelkamp and P. Kissmann, "On the complexity of BDDs for state space search: A case study in Connect Four," *Inproceedings of the AAAI Conference on Artificial Intelligence*, pp. 18–23, 2011.

[13] F. Somenzi, "CUDD: CU Decision Diagram package release 2.4.1." URL: http://web.mit.edu/sage/export/tmp/y/usr/share/doc/polybori/cudd/cuddIntro.html, 2005.

[14] 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*, pp. 35–51, Springer, 2015.

[15] 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, no. 6, pp. 675–696, 2017.

[16] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, p. 293–318, 1992.