SOLVING FANORONA

Maarten Schadd

Master Thesis MICC-IKAT 06-08

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Faculty of Humanities and Sciences
of the Universiteit Maastricht

Thesis committee:

Prof. dr. H.J. van den Herik
Prof. dr. E.O. Postma
Dr. ir. J.W.H.M. Uiterwijk
Dr. H.H.L.M. Donkers
Dr. P.H.M. Spronck

Universiteit Maastricht
Maastricht ICT Competence Centre
Institute for Knowledge and Agent Technology
Maastricht, The Netherlands
August 2006

Preface

In my Master Thesis I describe research performed at IKAT (Institute for Knowledge and Agent Technology) of the Universeit Maastricht. The subject of this study is solving the game of FanoRona.

To me, the study of board games is challenging since I believe (1) that many situations in the real world are translatable to a board-game position, and (2) that even if the rules are rather easy, playing well can be hard. These two beliefs are the main reasons why I chose to make the game FanoRona the subject of my Master Thesis.

Subsequently, I wrote the program RALOMBO in Java 1.5.0 which uses standard .txt files for storage. The computations made for this thesis are performed on a Pentium IV 1900 with 512MB Ram and 2GB Ram virtual memory. The endgame databases are computed by an Athlon 2000+ with 512MB Ram.

I would like to acknowledge Professor Jaap van den Herik for inspiration and corrections and my daily advisor Jos Uiterwijk for the good advice and the useful guidelines. Furthermore I am grateful to Pieter Spronck and Mark Winands for reading my Thesis and comments. I would also like to recall the course of "Intelligent Search Techniques" given as part of the Master of Artificial Intelligence which inspired me for this research. Finally, I would like to thank my family and friends who supported me during this research.

Maarten Schadd
Maastricht, August 2006

Summary

A few thousands years ago humans created board games. From then on humans competed against each other to see who has the strongest mind. With the upcoming of computers, scientists in the field of Computer Science and Artificial Intelligence tried to design computer programs which were capable of winning against a human player. This has been successfully achieved for an incomplete list of games. It is also possible to take it one step further and to solve a game. This means computing what the result of a game would be when both players play optimally and to know the best move to play in any position (strongly solving the game).

FanoRona is a Madagascars game for two persons with complete information. It is comparable to Checkers. In this thesis, all the rules of the game are explained and also smaller versions of FanoRona are constructed. These smaller versions are meant to show how the complexity of this game grows by increasing the board size.

The problem statement of this thesis reads: Can a computer program be written so that it weakly solves the board game FanoRona? The game-tree complexity and the state-space complexity are important measures. According to those complexities adequate methods have been selected and tested.

Proof-number search which has proven to be successful in many games is also introduced in the world of FanoRona and applied to the different versions of FanoRona. We show that proof-number search is able to solve the small versions of the game. Endgame databases are created with the help of retrograde analysis for the bigger versions of FanoRona and it is shown that this method is applicable with success in some of these versions. Also transposition tables and PN^2 have been tested on their usability for FanoRona.

Due to memory limitations the standard FanoRona game is not solvable at this point of time. A variety of extensions have to be exploited in order to restrict the use of memory as much as possible. However most smaller versions have been solved during this research and we may conclude that FanoRona is a game which soon will be solved.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This Chapter starts describing the history of computers playing games (in Section 1.1) and the related work in the field of Artificial Intelligence and the research of games (in Section 1.2). Then we will state our problem statement and research questions in Section 1.3. The outline of the thesis is described in Section 1.4.

1.1 Computers playing games

Competing against each other in the form of a game is nothing new. Egyptians and Chinese have archived games which date back to far before the year zero. Games can be categorized according to different dimensions. Three examples are (1) the number of players, (2) whether chance is involved, and (3) how many information a player has. With the upcoming of computers human beings were tempted to let the computer play those games. The reason why scientists are interested in research on board games is that the rules of games are mostly exact and well defined which makes it easy to translate them to a program that is suitable for a computer to run (Van den Herik, 1983).

The research in board games obtained a huge impulse in 1944 when Von Neumann republished his article about the minimax algorithm (Von Neumann, 1928) together with Morgenstern in the book "Theory of Games and Economic Behavior" (Von Neumann and Morgenstern, 1944). These ideas were picked up by Shannon (1950) and Turing (1953) who tried to let a computer play Chess as intelligently as possible. Since then much research is performed on new methods, on a variety of games (Murray, 1952) and on other problems to make the computer a worthy opponent for the human player (Schaeffer and Van den Herik, 2002).

One field in this area of research are the board games which have full information and are played by two persons. Chess is the classical example of this kind of a game and a great deal of effort has been devoted in the past to the construction of a good chess player. Chess will not be solvable in a foreseeable future according to Levy and Newborn (1991). Currently we may cast some doubt on these statements (Van den Herik, 2005). The most pregnant success so far in this area was the result when Deep Blue achieved to win against world chess champion Garry Kasparov (Newborn, 1996).

1.2 Related work

In this section we will shortly describe other research in the field of Artificial Intelligence performed on board games. An increasing number of games can be played by a computer on a high-performance level since the research on games increases steadily. Also the list of solved games is increasing with games such as Nine Men's Morris (Gasser, 1996). Recently Awari has been solved (Romein and Bal, 2003). The key to the success for solving Awari is the creation of endgame databases with the help of retrograde analysis.

A large amount of effort is performed trying to solve the game of Checkers (Schaeffer and Lake, 1996) which is not achieved yet. Recently the white-doctor opening in Checkers was solved and current expectations are that Checkers can be solved within a few years (Schaeffer *et al.*, 2005).

Allis (1994) showed that the game of Connect-Four was a first-player win. Knowledge rules were implemented so that large subtrees can be pruned without the need to investigate these subtrees. This resulted in a solution tree which was small enough to solve the game. The game-theoretical value of Connect-Four is a win for White (Allis, 1988). The size of the solution tree for FanoRona would also be reduced if knowledge rules can be found.

The strength of a computer program playing a game can be measured in different categories. Figure 1.1 shows an overview of a prediction made in 1990 for the Computer Olympiad in 2000 (Allis, Van den Herik, and Herschberg, 1991a). Afterwards this prediction was shown to be accurate. An overview of solved games and predictions about solving more complex games can be found in Van den Herik, Uiterwijk, and Van Rijswijk (2002). It is not known yet in which category FanoRona belongs.

Solved or Cracked	Over Champion	World Champion	Grand Master	Amateur
Connect-Four	Checkers (8×8)	Chess	Go (9×9)	Go (19×19)
Qubic	Renju	Draughts (10×10)	Chinese Chess	
Nine Men's Morris	Othello		Bridge	
Go-Moku	Scrabble			
Awari	Backgammon			

Figure 1.1: Predicted program strength for the computer olympiad in the year 2000.

1.3 Problem statement and research questions

To solve a board game, the game-theoretic value of the start position has to be determined. The definition of weakly-solvable by Allis (1994) is if we can determine the game-theoretical value and a winning strategy for the start position. We designed the problem statement for this thesis from this notion. It reads as follows:

Can a computer program be written so that it weakly solves the board game FanoRona?

In order to answer the problem statement, three related research questions will be investigated first.

1. *What complexity does FanoRona have?*

In order to answer this question the game-tree complexity and the state-space complexity can be computed.

2. *What methods are suitable for solving FanoRona?*

The methods investigated are chosen based on the result received for the complexity of the game. For example the chosen methods have to be knowledge rich if the complexity of the game is high.

3. How do these methods perform when applying to solve FanoRona?

In order to answer question 3 tests have to be performed with a range of methods on different board sizes to measure the performance in terms of (1) nodes used in the search tree and (2) time needed.

Once these three questions are answered a program to solve FanoRona can be built. The goal of this research is to solve the game of FanoRona.

1.4 Outline of the thesis

The outline of this thesis is as follows.

- Chapter 1 contains a general introduction on how computers play games and a short history of research in board games. Furthermore this Chapter contains related work and the problem statement and the research questions.
- Chapter 2 gives an introduction to the board game FanoRona. The history of the game is described and all the rules are explained.
- Chapter 3 discusses the state-space complexity and the game-tree complexity of FanoRona on different board sizes. This Chapter also gives a comparison with other board games.
- Chapter 4 introduces the techniques investigated during this research. Techniques as Proof-Number search, Transposition Tables, Retrograde Analysis, and Symmetry are discussed.
- Chapter 5 describes the experiments and results which answer the third research question and show how well the techniques described in Chapter 4 work.
- Chapter 6 concludes the thesis by providing (1) answers to the research questions, (2) an evaluation of the problem statement, (3) final conclusions and (4) recommendations for future research.

Chapter 2

Rules of the FanoRona game

In this Chapter the rules of FanoRona will be explained in detail. The Chapter is based on the rules stated in Bell (1980) and Chauvicourt (1980). FanoRona is the national game of Madagascar and was derived from the game "Alquerque" which might be over 3000 years old. FanoRona has three standard versions: Fanoron-Telo, Fanoron-Dimyand, and Fanoron-Tsivy. The difference between these variants is the size of board played on. Fanoron-Telo is played on a 3×3 board and the difficulty of this game can be compared to the game of tic-tac-toe. Fanoron-Dimyand is played on a 5×5 board and Fanoron-Tsivy is played on a 9×5 board. We will call Fanoron-Tsivy from now on FanoRona since the Fanoron-Tsivy variant is the widest-known variant and the main subject of this thesis. The goal of the game is to capture all enemy stones. For indicating the size of a board we do not use the standard matrix notation. A board with size A×B has A columns and B rows.

2.1 Board

FanoRona is played on a 9×5 board and it is played along the lines and on its intersections just as Go. A line represents the way along which a stone can move during the game. There are strong and weak points (intersections). On a weak point it is only possible to move a stone horizontally and vertically, while on a strong point it also is possible to move stones diagonally. A stone can only move from one point to another at a time.

Each player has 22 stones at the start position as shown in Figure 2.1. The player to start is White, as may be expected from a western game.

2.2 Capturing

Players are allowed to play a move alternatively. Even if a move consists of multiple movements of one single stone we will still denote it as a single move.

Capturing stones of the opponent can be done in two different ways, either by approach or by withdrawal. An approach is the movement of the capturing stone to an adjacent point of an opponent stone if the opponent's stone is placed on the extension of the movement line of the capturing stone. A withdrawal is the movement away from the opponent's stone if the capturing stone is placed on an adjacent point of the opponent's stone and if the opponent's stone is placed on the extension of the movement line of the capturing stone. When an opponent's stone is captured by approach or by withdrawal, all opponent stones in line behind that stone (as long as there is no interruption by an empty point or an own stone) are captured too.

Figure 2.2 shows how the capturing mechanism works. In the situation shown in Figure 2.2 White

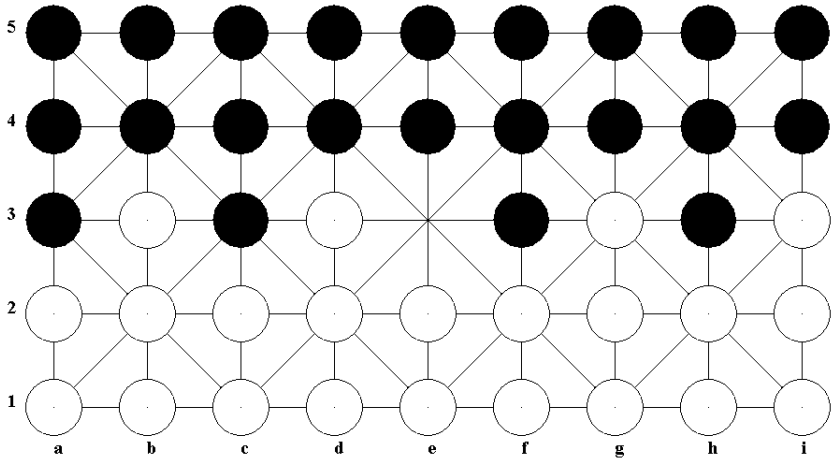


Figure 2.1: The start position for a FanoRona game.

can capture Black's stone on d2 by moving his[1] white stone from b2 to c2. With this move also Black's stone on e2 will be captured. g2 will not be captured since there is no black stone on f2. This was a capturing by approach since White moved his stone towards Black's stone on d2.

White can also capture by withdrawal if he moves his white stone from f4 to e4 since he is moving away from Black's stone on g4. i4 will not be captured since there is a white stone disrupting the line at h4.

White cannot capture c4 with f4 since for a capturing move the own stone has to be next to the one captured after this move and f4 is too far away to capture c4. In order to allow a capturing the capturing stone has to be moved to an adjacent point next to the captured stone if it is approached. If a capturing is done by withdrawal then the capturing stone has to start at an adjacent point next to the captured stone and move away from it.

White also cannot capture c4 with b2 since c4 is not on the extension of a movement line from b2. Only stones can be captured which are located in the extension of the movement line of the capturing stone. Thus capturing "around a corner" is not allowed.

For describing a move we will define the following notation: a1-b2A means that the stone on a1 will move to b2 and approaches(A) the stone on c3. For a withdrawal the letter "W" is used. If a stone is moved without capturing opponent stones then no letter "A" or "W" will be used.

Like Checkers it is allowed to keep capturing with the same stone as long as possible. Figure 2.2 shows that White can capture c4 with the move: b2-c2A-c3A. A player must choose a capturing move above a non-capturing move but a player is free to stop with capturing after any number of opponent stones are captured. In Figure 2.2 White is allowed to stop early and only play b2-c2A. A move where no stones are captured is called a *paika* move. Thus White is not allowed to play the paika move b2-b1 here since capturing is possible.

There are three more rules concerning capturing stones. The first is that it is not allowed to capture by approach and withdrawal at the same time. This is the case at the start position shown in Figure 2.1 where White could play d3-e3 as an approach or a withdrawal. In such a situation the player has to choose if the current move is an approach or a withdrawal. The second rule is that it is not allowed to make a capturing move in the same direction as the capturing move directly before. We illustrate this rule by referring to Figure 2.2. White is not allowed to play: f4-e4W-d4A since his stone would move two times in a row in the same direction. A player is allowed to play a capturing sequence with two times the same direction if a capturing with another direction is done in between them. The last movement

[1]For brevity, we use 'he' or 'his' when 'he or she' or 'his or her' would be meant.

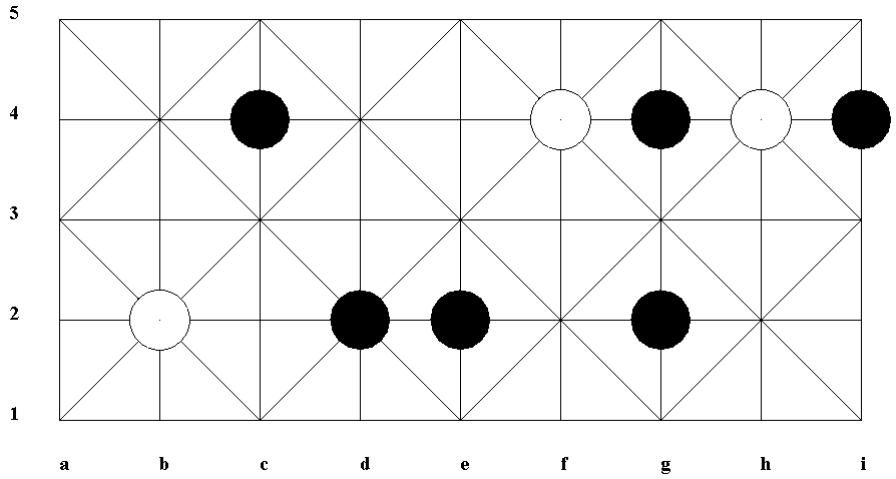


Figure 2.2: An example position during a FanoRona game.

direction of the capturing in the turn before (i.e., before the previous opponent move) does not influence allowed capturing directions in the current turn. The last rule is that the current capturing stone is not allowed to arrive at the same point twice during a capturing sequence. If in Figure 2.2 a black stone would be on g5 then White is not allowed to play f4-e4W-e3A-f4A because of this rule.

2.3 End of the game

The player who first captures all opponent stones wins the game. There exist draw situations in FanoRona and both players can call it a draw if both do not see an opportunity to win. Such a draw situation is detectable in Figure 2.3 with White to move. White cannot move his stone in the middle or he will lose the game. The tactic for the black player is to stay always on the other side of White's moving stone in order to achieve a draw.

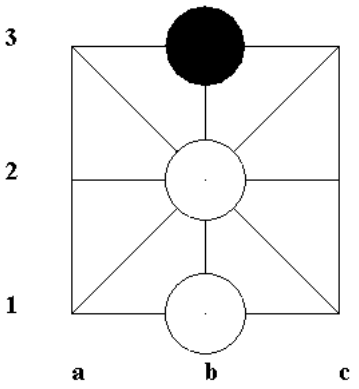


Figure 2.3: This position is a draw with White to move.

To illustrate why the position in Figure 2.3 is a draw we can analyse all possible moves for White

excluding symmetry.

1. **b2-a3** - Black answers with b3-b2A-c1W and White loses the game.
2. **b2-a2** - Black answers with b3-b2A-c2W and White loses the game.
3. **b2-a1** - Black answers with b3-b2A-c3W and White loses the game.
4. **b1-a1** - The answer of Black could be
   - **b3-a3** - White wins the game by playing a1-a2A.
   - **b3-c3** - White has four moves because of symmetry.
     - **b2-a2** - Black wins by playing c3-b2A-c2W.
     - **b2-a3** - Black wins by playing c3-b2A-c1W.
     - **b2-b3** - Black wins by playing c3-b2A-b1W.
     - **a1-b1** - Now Black has only two moves.
       * **c3-c2** - White wins by playing b2-a2W.
       * **c3-b3** - With this move the game arrived at the same position again which means that this will be repeated forever and this game is a draw.

So the draw strategy for Black is to circle around the middle and always stay on the exact opposite side of the White's outer stone.

The tradition of FanoRona allows the losing player to play a "VELO" game with different rules to regain his honor. These rules do not allow the player who won the first game to capture enemy stones until the winner only has 5 stones left in the Velo game. If he now still can win against the loser of the first game then the loser did also lose his honor. This thesis will not go deeper into this velo game. An example FanoRona game is shown in Appendix B.1.

Chapter 3

Analysis

This Chapter is meant to give insight into the complexity of FanoRona and smaller variants. Therefore we start by analyzing the FanoRona board (Section 3.1) and continue with the analysis of the the 5×5 board (Fanoron-Dimyand) in Section 3.2 as representative of the smaller board variants.

3.1 9×5 board, FanoRona

In this section we determine the complexity of FanoRona. We distinguish the game-tree complexity (Subsection 3.1.3) and the state-space complexity (Subsection 3.1.4). For the game-tree complexity we need the average for game length and branching factor, which are therefore determined first (Section 3.1.1 and Section 3.1.2).

3.1.1 Game length

A first important factor for the complexity of FanoRona is the average length of a game. In order to determine this factor a database would be needed with games played in the past. But such a database does not exist at this moment since FanoRona is not yet a well-known game. Therefore, random games are examined (cf., Donkers and Uiterwijk, 2002) in order to estimate the average game length. Of course, this includes games which come to an early end since a player made a rather bad move and also games which are unnaturally long since the players are not trying to win the game. Figure 3.1 shows a histogram of game lengths for 100,000 games randomly played. The length of the game is denoted on the x-axis and the number of games is denoted on the y-axis. Two maxima located at 22 and 24 moves can be identified in this Figure. We assume that the average game length is located between 21 and 25 moves according to these maxima.

Because it is questionable whether this result on random behavior represents the actual game length of a FanoRona game another approach is investigated, too. Instead of a random player an alpha-beta search is used with a shallow depth. The pseudo code for this alpha-beta search is shown in Appendix A.1. To prevent that always the same game is played a random factor is introduced. The chance to choose a move is related to the score given by alpha-beta search. A greedy heuristic is used here which uses the number of pieces on the board. A random choice is then made based on the results of the 4-ply deep alpha-beta search. As an example, if White has only two moves to play and a 4 ply deep search with greedy evaluation gives move A the value 1 and move B the value 10 then the chance of choosing move B is ten times as high as choosing move A. We assume that such a player resembles the behavior of a human player better than a random player. Figure 3.2 shows a histogram of the length of games played with the shallow alpha-beta search. For computational reasons, the histogram is based on 1,000 played games only.

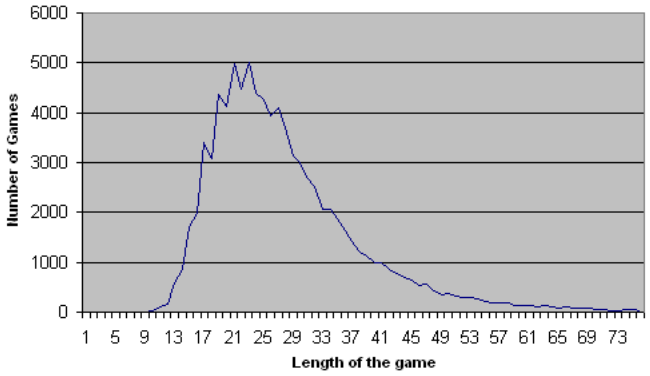


Figure 3.1: Histogram of the game length of a FanoRona game with random players.

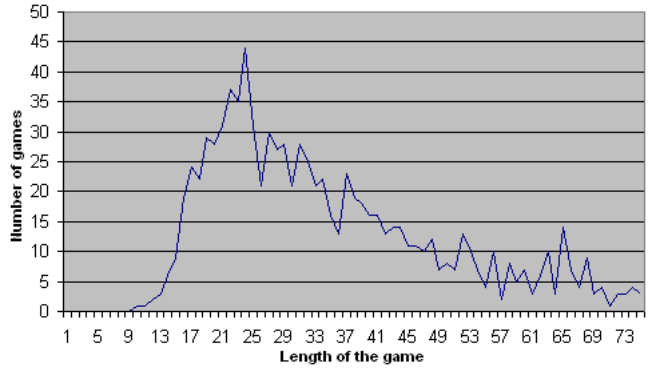


Figure 3.2: Histogram of the game length of a FanoRona game for players using a 4-ply deep alpha-beta search with a randomized decision making.

Due to an increased amount of games with a large amount of moves the average game length has increased in comparison to the experiment with random play (Figure 3.1). Apparently random moves giving away a good position occur more frequently than random moves which do not try to win the game. Also moves which postpone the loss which are played instead of moves which prevent the loss increase the average game length. From the data used for Figure 3.2 the average game length is calculated as 39.84.

3.1.2 Branching factor

A second important factor for the complexity of a game is its average branching factor. The branching factor indicates how many different moves a player can choose on average. If FanoRona has a high branching factor then the search for the best move will take longer since more moves have to be examined at each search level (ply) of the search tree.

There are positions possible in FanoRona with a higher branching factor compared to the start position in Go (361 possible moves not counting symmetry). Appendix B.2 shows a situation where 414 different capturing moves are possible. Figure 3.3 shows the average branching factor on the y-axis as a function of the number of pieces on the board on the x-axis.

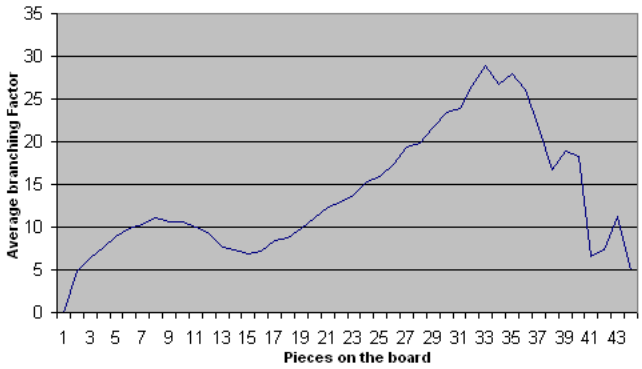


Figure 3.3: The average branching factor as a function of the number of pieces on the board.

In order to approximate an adequate branching factor players with a shallow search are used to play games. At every move the branching factor is computed and stored. The graph shows that at the opening (where there are 44 pieces on the board), the branching factor is equal to five. This histogram shows a quite interesting property of FanoRona. The histogram indicates a maximum branching factor when there are about 34 pieces left on the board. This is during a phase of the game where there are many of capturing moves available since on the one hand there is space on the board for long capturing sequences and on the other hand there are a lot of stones left to capture. The average branching factor decreases when there are fewer pieces on the board; a minimum is reached for about 15 pieces.

A second local maximum is reached with 8 pieces. This second increase of the branching factor is due to the transition between capturing moves and paika moves (i.e., moves where no capturing is possible). With 15 pieces left the player will only have a couple of capturing moves left and if those are played then the branching will increase again since the player now has the opportunity to play with every stone a paika move.

It seems that the most complex part of a game is located around 34 on the board, where a lot of moves can be played. However, one has to realise that during this stage of the game only capturing moves are played and more than one piece can be captured at a time so that the transition through this phase can be fast. It might happen that a great deal of moves will be played when only a few pieces are on the board before a piece might be captured.

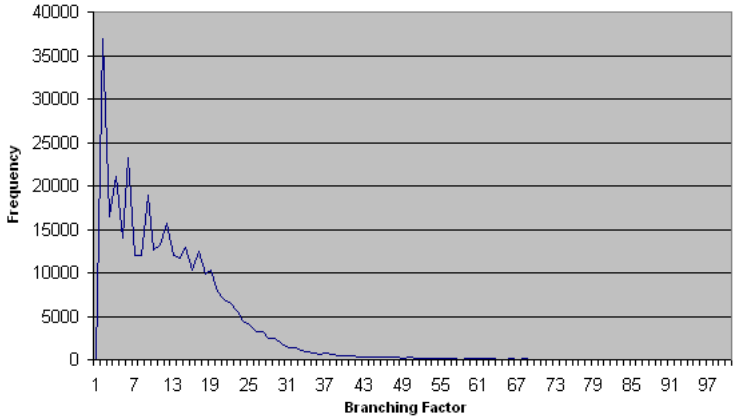


Figure 3.4: Histogram of the branching factor with 10,000 shallow played games.

Figure 3.4 shows a histogram for the branching factor. We can conclude from this figure that positions with more than 50 different moves are exceptionally rare. After 10,000 games the average branching factor is calculated as 10.3.

3.1.3 Game-tree complexity

The game-tree complexity is defined as the average branching factor to the power of the average game length. Given the calculated averages in Subsections 3.1.1 and 3.1.2 we obtained:

$$log(GameTreeComplexity) = log(10.3^{39.84}) \approx 40. \tag{3.1}$$

Table 3.1 is summary of complexities of games made by Winands (2000) and Allis (1994). As Table 3.1 shows the game-tree complexity of FanoRona is larger than the game-tree complexity of Checkers and Awari.

3.1.4 State-space complexity

The state-space complexity (Allis, 1994) is a measure for the total number of possible positions. As an example we may take the game of Go. Go is played on a 19×19 board and each point can be empty or occupied by a white or a black stone. This would mean that Go has a state-space complexity of $3^{361} \approx 1.74 \times 10^{172}$. Of course, not all of these positions are legal. In Go a surrounded stone is captured which decreases the number of legal positions. It is also possible that a position is legal but it is not reachable from the start of the game. All these positions will not play a role during solving or playing the game but are included in most calculations of the state-space complexity.

We start with the same idea as explained for Go to calculate the state-space complexity of FanoRona. This would lead to a log complexity of $log(3^{45}) = 21.47$ which gives us an initial rough estimation of the state-space complexity. But it is possible to make a much more accurate calculation for FanoRona. The earlier calculation includes positions with 45 white stones on the board which is not possible since every player starts a game with 22 stones. The number of positions with 22 white stones on 45 points is $\binom{45}{22} = 411,671,536,800$. This has to be multiplied with the possibilities to place 22 black stones on the remaining 23 points resulting in about 9 billion possible positions for positions where both White and Black have 22 stones. To create the final number all combinations of white and black stones have to be examined where both players have a maximum of 22 stones on the board.

Game	*log (Game-Tree complexity)*	*log (State-Space complexity)*
Connect-Four	21	14
Othello	28	58
Checkers	31	18
Awari	32	12
FanoRona	40	21
Nine men's morris	50	10
Lines of Action	56	24
Chess	123	50
Go	360	172

Table 3.1: State-space and game-tree complexities of games.

$$log_{10}\left(\sum_{w=0}^{22}\sum_{b=0}^{22}\binom{45}{w}\times\binom{45-w}{b}\right)\approx 21.46$$

It is interesting to see that the this number is almost the same as $log(3^{45})$. A comparison of the state-space complexity of FanoRona to other games can be seen in Table 3.1.

The conclusion can be drawn that the state-space complexity of FanoRona resembles the complexity of Checkers, but the overall complexity is more alike the complexity of Connect-Four. Connect-Four has been solved by Allis (1988). The key to the success of Allis has been the implementation of knowledge rules with the help of which it is possible to detect whether a position is won or lost in an early stage (Uiterwijk, Allis, and Van den Herik, 1989). With this method huge subtrees which would have to be created can be pruned.

Both Van den Herik *et al.* (2002) and Allis *et al.* (1991a) come to the conclusion that knowledge-based methods are more appropriate for solving games with a low decision complexity, while brute-force methods are more appropriate for solving games with a low state-space complexity. Our analysis showed that FanoRona is a game with a low decision complexity and a mediocre state-space complexity. Thus knowledge rules would be important for solving FanoRona.

Such knowledge rules do not exist yet or are not formulated since FanoRona is not a well-known game. So the only other option is to try to solve this game with brute force without knowledge rules or create own knowledge rules. In this research we have mainly pursued the first approach. We designed one knowledge-rich method for FanoRona which directs the search with the help of a greedy function. This function is then tested against the knowledge-poor methods. Knowledge-rich methods proved were able to create cut-offs at the solution tree of Connect-Four. The game was subsequently weakly-solved (Allis, 1988).

3.2 5×5 board, Fanoron-Dimyand

The growth of the complexity when the size of the board grows can also be characteristic for a board game. In order to show the growth of the complexity the analysis for the smaller board sizes are made analogously. The detailed results for Fanoron-Dimyand will be shown here and a summary of the complexities of all board sizes is given in Section 3.3.

Figures 3.5 and 3.6 show that the average game length of a Fanoron-Dimyand game is 15.258. A remarkable point to observe is that both figures do not show big differences such as the Figures 3.1 and 3.2 for the FanoRona game.

The results of the analysis of the average branching factor of Fanoron-Dimyand show the same behavior as for FanoRona. There are two maxima where one is located at a point where there is space for capturing

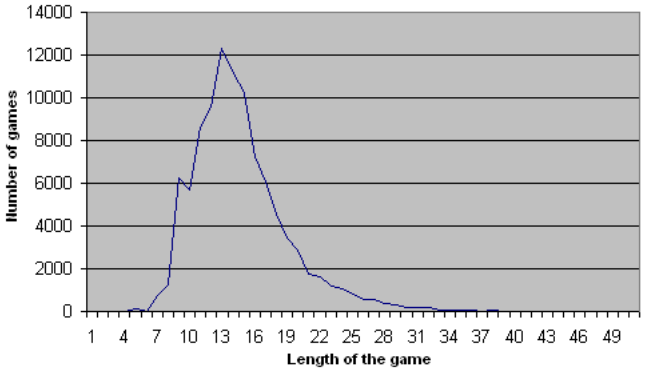


Figure 3.5: Histogram of the game length of a Fanoron-Dimyand game with random play.

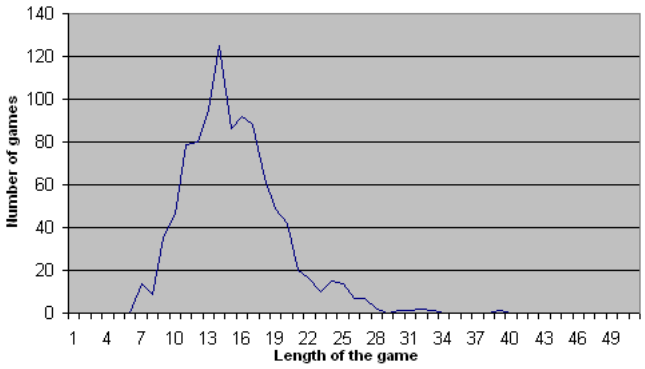


Figure 3.6: Histogram of the game length of a Fanoron-Dimyand game with a 4-ply deep search.

and the second maximum is located where there are hardly capturing moves possible and mostly paika moves are played. Figure 3.7 would indicate a branching factor around 8. Using players with a shallow alpha-beta search the average branching factor is computed as 6.7.

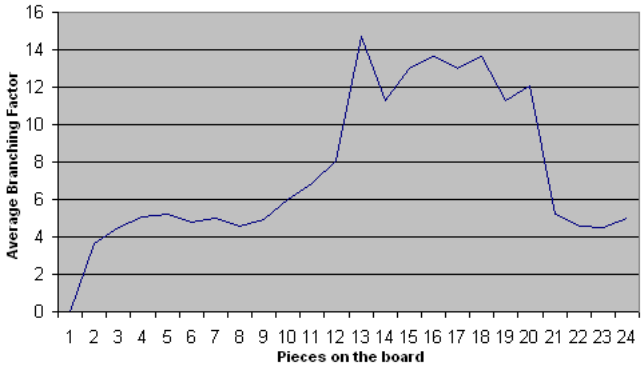


Figure 3.7: The average branching factor of Fanoron-Dimyand

With this result the log game-tree complexity can be computed as $log(6.7^{15.258}) \approx 12.6$ The state-space complexity of Fanoron-Dimyand is computed in the same way as FanoRona and results in a log complexity of 11.89.

The increase in the game-tree complexity and state-space complexity imply that changing from the 5×5 board to the 9×5 board is a change from a theoretically easy to solve game to a game which is much harder to solve.

3.3 Summary of all board sizes

The results of all board sizes are summarized in Table 3.2[1] to show how the game-tree and state-space complexities develop while changing the board size.

Game	3 × 3	5 × 3	3 × 5	5 × 5	7 × 3	3 × 7	9 × 3	3 × 9	7 × 5	5 × 7	9 × 5
log game-tree complexity	4.3	7.8	6.6	12.7	14.5	16.6	26.8	38.3	24.1	28.7	40.4
log state-space complexity	4.1	7.1	7.1	11.9	10.0	10.0	12.8	12.8	16.7	16.7	21.5

Table 3.2: State-space and game-tree complexities of different board sizes of FanoRona.

There are two remarkable points visible in Table 3.2. First it is noted that the game-tree complexity increases faster than the state-space complexity on long and narrow boards. These values may originate from a not-realistic playing behavior from computer players. We expect these numbers to decrease if they are based on data of human players. The main reason for a high game-tree complexity during the simulation is the increase in game length. Apparently a 4-ply deep search as described in Section 3.1.1 is not realistic enough on a long and narrow board. If the search tree is not deep enough then all adjacent points receive the same value, resulting in a random move by the player. We suspect that this is the reason for an increase in game length and therefore the game-tree complexity can be overestimated. The experiments support this assumption (Section 5.2.3).

Second we observe that that the game-tree complexity changes if the board is rotated 90 degrees. Please note that A×B indicates a board with A columns and B rows. Apparently the average branching

[1]These numbers were not accurate at the time of the publication of this thesis and have been corrected afterwards.

factor is smaller if the game is played on a vertical board. The explanation for this could be that the capturing moves at the start of a game are normally vertically directed. On a vertical board these moves will capture more stones and there will be less stones to capture once there is space for capturing. This could be the reason for a lower average game-tree complexity of vertical games compared to horizontal games.

If these numbers prove to be correct then this increase of complexity can be called a *complexity jump*. The reason for why such an instant increases in the game-tree complexity from $10^{16.6}$ (3×7) to $10^{38.3}$ (3×9) has yet to be investigated.

An unexpected result in Table 3.2 is the game-tree complexities of the board size 3×9. This board has almost the same complexity then the 9×5 board. One might expect that the addition of two rows would influence more then these numbers show.

The main reason for these results is the increase of game length for the long and narrow boards. Apparently the alpha-beta search players get in situations where the last pieces are far away from each other and therefore need many steps to finish the game. Also the search depth could not be deep enough so that the alpha-beta player does not have the ambition to move towards the other side of the board. Of course, this could also be true for the 9×5 board. Making the board more like a square could result in a playing strategy during the midgame which is completely different from the strategy on a smaller board. It might be that there is more room for capturing on the 9×5 board because it is not so narrow as the 9×3 board. Therefore it may be that capturing is faster on a square board.

We estimate that the 7×3, 3×7, 9×3 and 3×9 are still solvable in spite of the high game-tree complexity since the state-space complexity is not high. Games with a low state-space complexity may encounter same positions often. Transposition Tables is a technique to store encountered positions and reuse the computed information in another branch of the search tree. Programs using this techniques would benefit from it.

Chapter 4

Applied Methods

This Chapter gives a description of the methods chosen to solve FanoRona on different board sizes and explains why they are chosen. The methods are: Proof Number Search (4.1), Transposition Tables (4.2), Retrograde Analysis (4.3), and Using Symmetry (4.4).

4.1 Proof-number search

Proof-number (PN) search (Allis, Van der Meulen, and Van den Herik, 1994) is a best-first AND/OR tree-search algorithm which is inspired by conspiracy-number search described by McAllester (1988), Schaeffer (1990) and Lorenz and Rottman (1997). The strength of proof-number search is that it uses a heuristic to prefer shallow trees above wide trees. PN search has been successfully applied in games such as Chess (Breuker, Allis, and Van den Herik, 1994a), Shogi (Seo, Iida, and Uiterwijk, 2001) and Lines of Action (Winands and Uiterwijk, 2001).

We will discuss the algorithmic details in Subsection 4.1.1, the possible heuristics in Subsection 4.1.2, and the motivation in Subsection 4.1.3.

4.1.1 Algorithmic details

The complete search tree has to be stored in memory since PN search is a best-first search. Different methods are implemented to reduce the memory needed. They will be discussed later. PN search tries to prove if a player can achieve his goal in the root. In the context of this research the goal is if White can win the game with optimal play by Black. In order to prove if Black can always win a game a new PN search for Black has to be started for each of White's first moves. The game-theoretical value for the root is a draw if both White and Black can not achieve their goal.

Each node in the search tree has a proof and disproof number. The proof number represents how many nodes minimally have to be evaluated to prove the goal in this node and the disproof number represents the minimum number of nodes which have to be evaluated to disprove the goal in this node. The goal in a node is proven when the proof number becomes 0 and the disproof number becomes infinity. To disprove a node the proof number has to be infinity and the disproof number has to be 0.

The proof and disproof numbers of a node are dependent on the children. As an example we take the tree shown in Figure 4.1 which was taken from Breuker *et al.* (1994a). On the left side of a node is the proof number and on the right side the disproof number. A square represents an OR node and a circle represents an AND node. In the situation of a two-player board game the OR nodes correspond to the max player and the AND nodes to the min player.

Node C in Figure 4.1 has proof number 1 and disproof number 2. The proof number is the minimum proof number of its children because node C is an OR node and only one of its children has to be a certain win for the max player in order to be a win for the max player in the current node. In contrast,

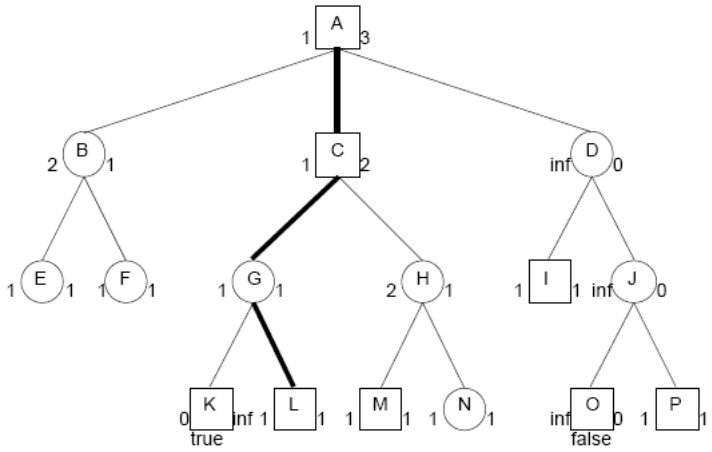


Figure 4.1: AND/OR tree with proof and disproof numbers.

the disproof number is the sum of the disproof numbers of all children. If C has to be disproven then all children of C have to be disproven. The reverse principle holds for AND nodes like B where the sum of the proof numbers and the minimum disproof number are taken.

Allis (1994) describes two variants of creating the search tree.

1. *Immediate evaluation.* Each node in the tree is immediately evaluated after it has been generated. The tree is built as follows: the root is first generated and evaluated. Then, at each step, a leaf is selected and expanded, and all its children are immediately evaluated, etc.

2. *Delayed evaluation.* Each node is only evaluated when it is selected, and not immediately after it has been generated. The tree is built as follows: the root is first generated (without evaluation). Then, at each step, a leaf is selected and evaluated. If the evaluation value is *unknown*, the node is expanded (without evaluating its children), etc.

The standard PN search generates children with proof number and disproof number 1. If a mate position is found the proof and disproof number will be assigned zero and infinity (or vice versa) and thus PN search is a form of immediate evaluation.

Proof-number search is a best-first search and the selection of the next node to evaluate is based on the proof and disproof numbers. The most-proving leaf is selected by starting at the root. A child with the same proof number is selected if the parent is an OR node or a child with the same disproof number is selected if the parent is an AND node until the algorithm arrives at a leaf. This node is the most-proving node and is expanded. After the children are generated and new proof and disproof numbers are determined the new values are propagated upwards to the root and a new most-proving node is selected.

If pn search is applied to the start position then the information is obtained if White can win the game or not. In order to investigate if the game could be a win for Black every possible move for White has to be examined with a new search and if all moves are a win for Black then the start position is a win for Black. If it turns out that it is not possible to force a win by Black for each possible move for White and White can not force a win at the start position, then the game-theoretical value is a draw. More details of this algorithm are available in Allis *et al.* (1994) and the pseudo code is given in Appendix A.2.

This method has proven quite valuable since it can search certain subtrees rather deep if they are promising while it does not search deep in areas of the tree which are not useful for the proof. This makes proof-number search successful in Chess problems (Breuker *et al.*, 1994a) and Shogi (Nagai, 2002) where certain endgames require deep searches to be solved.

4.1.2 Heuristics

Instead of doing an immediate evaluation with proof number 1 and disproof number 1 also more knowledge can be used to create a more directed search (Allis, 1994). With a directed search less nodes have to be kept in memory and a forced win can be found faster. We describe three examples of directed search.

Greedy

Instead of giving a node the proof number 1 and disproof number 1, the pieces on the board could be counted. We designed this heuristic in a way that the PN is based on the balance between white and black stones on the board. This is a form of immediate evaluation since the PN numbers are computed directly.

$$proofnumber = \frac{\#blackpieces}{\#whitepieces} \tag{4.1}$$

$$disproofnumber = \frac{1}{proofnumber} \tag{4.2}$$

This would direct PN search to those subtrees where White has more pieces than Black and it should be easier to force a win in those subtrees. This heuristic should be enriched with more game knowledge by experts. We designed this simple approach to investigate the impact of a small amount of knowledge on the size of the solution tree.

This method resembles *stones pn* described by Allis (1994). Stones pn uses the amount of opponnent stones on the board which still have to be captured as proof and disproof number. We believe that using the ratio of white and black stones is a more accurate measure for the strength of a position.

Children dependent

The children-dependent method is basically looking a step ahead. The PN numbers are determined by the number of children of this node. So it would be the same if the node would be evaluated only that the children are not kept in memory until they are needed. This methods saves memory space since the stored tree is overall 1 step shallower than the investigated tree. A disadvantage of this method is that it takes much longer if the generation of children is slow.

PN^2 Search

PN^2 search is a search method consisting of two levels of PN search (Breuker, Uiterwijk, and Van den Herik, 2001). It uses PN search for the normal tree but uses PN search a second time for determining the PN numbers for evaluated nodes. There also exist other two-level search algorithms such as PDS-PN but Winands, Uiterwijk, and Van den Herik (2004) showed that PDS-PN used more nodes than PN^2 for certain LOA positions, but is not restricted by the memory. We will call the first PN tree pn_1 and the second tree to evaluate the nodes pn_2.

Breuker *et al.* (2001) showed that a delayed evaluation has a better performance than an immediate evaluation with PN^2 search. If every node would be evaluated immediately then more PN searches have to be made and thus the immediate evaluation would take longer than the delayed evaluation.

The size of the subtree which is allowed for determining the PN numbers is an important factor for PN^2 search. With a larger tree the search would be more directed but it would take more time to compute it. Breuker *et al.* (2001) propose a variable size for the pn_2 tree which is dependent on the size of the pn_1 tree. Berkey (1988) describes a logistic growth model which can be used to determine the size of the pn_2 tree. Using this formula the size of the pn_2 will increase more with a large pn_1 tree. A more directed search is provided in that way when a large tree is stored in the memory.

$$f(x) = \frac{1}{1 + e^{(a-x)/b}}$$

Parameter a denotes the transition point of the function. As soon as pn_1 reaches a nodes pn_2 will use half as many nodes. Parameter b determines the steepness of the sigmoid function. The value for a and b have to be determined by experiments. Breuker *et al.* (2001) show that it would be wise to choose them in a way so that $\frac{b}{a} > 0.1$ in order to compute the game-theoretic value of every solvable position. For chess endgame problems (a b) should be equal to (600K 80K) and we will use these values in order to calculate the performance of this method for FanoRona.

4.1.3 Motivation

PN search is chosen because it has proven to be a successful method to solve games and endgame positions. We expect that PN search is also efficient for solving FanoRona.

4.2 Transposition tables

During the search in the game tree it may happen that an identical position is searched twice because the position can be reached by a different sequence of moves. As an example we can take an opening in the game of Chess. The position after 1. e4 c5 2. Nf3 is identical to the position after 1. Nf3 c5 2. e4. If this happens time has to be spent on a situation which already was examined earlier. These occurrences of identical positions are called *transpositions*. Implementing a transposition table is a way to store earlier examined situations and reuse the calculated information. The use of transposition tables can lead to a reduction of the search tree (Breuker, 1998). Roijakkers (2005) encountered this for the game of Octi.

4.2.1 Hashing

In the best case, all possible positions are stored in the transposition table. But since there are too many possible positions only part of them can be stored in the table. In order to achieve this a hashing function is used which makes the transposition table in fact a hash table.

In order to translate a certain game position to a number Zobrist hashing is used (Zobrist, 1970). FanoRona has two different pieces (white and black) and 45 points where these pieces can be. A random number is generated for each kind of piece on each possible point. In order to create the hash code the XOR operation is used with all random numbers corresponding to the current position. This concept was designed in order to give each hashing index the same chance of occurring. It would not be efficient if many positions are hashed on the same number and only a few are hashed on all other numbers. In addition to this hash key an extra bit is added to indicate which player is to move.

The actual size of the transposition table can be varied according to the number of bits used for indexing. During this research 64 bits are used for the hash key and 20 of those bits are used for determining the index in the table. This allows $2^{20} = 1,048,576$ positions to be stored at the same time. The other 44 bits are stored for identification since the chance of two different positions being mapped on the same table entry increases with using less bits (Zobrist, 1970).

4.2.2 Values stored

A typical transposition-table entry stores the following values.

- **hash key** - The identification key which was not used for the hash code.
- **move** - The best move to make in this situation.
- **value** - The result of a search for this node.
- **depth** - The depth reached for the computation of the value.

This research focuses on solving the game FanoRona and since proof-number search is used there is no information of a node except if it is a sure win for the white player or not. So, the transposition table implemented in this research only uses the hash key field for identification purposes and the value field as a boolean to indicate if White wins the current position.

4.2.3 Error probability

We distinguish between two kinds of errors, a *type-1 error* and a *type-2 error*. A type-1 error happens if two different positions have the same hash value. Thus they are mapped on the same entry in the table and also the hash key value is the same. This is an error which may remain undetected. In this case it can happen that if such an error occurs the solution tree could have another outcome.

Let N be the number of distinguishable positions and M be the number of different positions to be stored in the table. Breuker (1998) gives the following equation to determine the chance that a type-1 error will not happen:

$$P(NoErrors) \approx e^{-\frac{M^2}{2N}}$$

Assuming that a search would consider 10,000 positions per second and runs for 20 minutes 1.2×10^7 positions are evaluated. Breuker (1998) assumes that 30% of the examined positions are attempted to store in the transposition table. Since we will only store positions if they are completely solved we estimate this number to be 10%. In this example this would mean that 1.2×10^6 positions are attempted to be stored in the transpostion table. With a 64 bit hash value the probabilty on a type-1 error would be

$$1 - e^{-\frac{1.2\times 10^{6\,2}}{2\times 2^{64}}} \approx 3.9 \times 10^{-8}.$$

This should be small enough for solving large trees without having type-1 errors.

The hash key field is stored to detect type-2 errors or collisions (Breuker, 1998). A type-2 error happens when two different positions are hashed on the same entry in the table because they have the 20 hashing bits the same. To detect these the other 44 bits are stored in the hash key field. If this happens different actions can be done which are called *replacement schemes* (Breuker, Uiterwijk, and Van den Herik, 1994b). If information about depths would be available then the deepest one could be stored for example. This information is not stored in a node with this approach with using PN search and therefore the newest one is saved and old entries are deleted.

4.2.4 Motivation

Transposition Tables is a technique which already proved to be successful. We decided to investigate this method to be able to prune large subtrees in the endgame. We expect that transpositions can often occur in FanoRona. The reason for this is the way of capturing. A player is allowed to stop capturing. If we imagine that White has the capturing move a2-a3A-b4A-b3W, he can choose to stop capturing after a2-a3A and play a3-b4A-b3W next turn (assuming the move of the opponent does not interfere). Another transposition would be if the player decides to play a2-a3A-b4A first and plays b4-b3W next turn.

4.3 Retrograde analysis

Retrograde analysis is a method to create an endgame database for board games. These endgame databases prove vital to the strength of many current game programs (Schaeffer, 1997). The more situations are stored in the endgame database, the earlier the search process can be interrupted. Thus a deeper play is possible in the mid-game. Retrograde analysis dates back to 1970 where Ströhlein (1970) made a chess endgame database and since then many other game programs such for Awari (Allis, Van der Meulen, and Van den Herik (1991b); Van der Goot (2001)) were implemented with endgame databases.

The implementation of a large endgame database with parallel retrograde analysis did lead to the fact that Awari was solved by Romein and Bal (2003). In the game of Checkers for example endgame databases have been built for up to 10 pieces left on the board (Schaeffer *et al.*, 2003).

Section 4.3.1 will show the algorithmic details of a retrograde analysis and Section 4.3.2 will describe the motivation behind this method.

4.3.1 Algorithmic details

The idea of retrograde analysis is intuitive. The first thing needed are two functions which translate a position to a number and the other way around. These functions have to be a one-to-one mapping. Making these functions efficiently can save much space in the database (Lake, Schaeffer, and Lu, 1994). With these functions it is possible to access all possible game positions for a certain amount of pieces on the board just by iterating the index number.

An initial index function for FanoRona can be designed as follows. Every position on the board is numbered. We call the total amount of positions on the board p. The position of White's stone i is W_i, and the position of Black's stone j is called B_j. The index is calculated as follows if White has M stones and Black has N stones.

$$\sum_{i=1}^{M} \left(p^{i-1} \times W_i\right) + p^M \times \sum_{j=1}^{N} \left(p^{j-1} \times B_j\right)$$

It is possible with this function to create a unique index number for every position. This index number is dependent on the number of pieces. As long as it is known how many pieces each player has it is also possible to revert this formula and create the corresponding board according to a given index.

This function has two disadvantages. (1) It does not filter out positions where a white and a black stone are both located at position one, which is not possible in FanoRona. (2) Positions are counted multiple times since swapping two identical stones will change the index number. If we define that the stones are numbered according to the position on the board then still a large amount of index numbers is redundant. Index numbers are still reserved for positions where the stone with the highest stone number is located on a position with the lowest number on the board. This can not happen since in case that the stone is moved to the position with lowest index number the stone numbers would change as described above.

We used an index function which is capable of removing problem (2). This function computes a combined index for all stones of a player (Lake *et al.*, 1994). There exist $\binom{m}{n}$ possibilities to place n stones on m positions. The new index function works as follows:

$$\sum_{i=1}^{M} \binom{W_i}{i} + \binom{p}{M} \times \sum_{j=1}^{N} \binom{B_j}{j}$$

In order to be able to use this function for retrograde analysis it has to be possible to convert the corresponding board to an index number. First the index number is partitioned into a combined white index and a combined black index which is possible with the knowledge of how many pieces of each color are on the board. Now it is possible to retrieve the correct board by "trial and error". We start with the position of Black's stone with the highest stone number nr. The possible position pb is increased until $\binom{pb}{nr} > CombinedBlackIndex$. Black has a stone positioned on $pb-1$. We remove $\binom{pb-1}{nr}$ from the combined index and continue until the position of all black stones is resolved.

Table 4.1 compared to Table 4.2 shows the decrease in size of the database if the second function is used in stead of the first one. However the advanced function still contains the cases where a black and a white stone are located on the same position. If those can be filtered out by an even better function then the size would decrease even more.

Since during this research PN search is used it is only needed to store a 1 if the player to move can win the position and a 0 if the player can not win the position. By only changing bits in a database

file the size for 4 white stones against 3 black stones would require 252 MB disc space. It is possible to reduce the required space by replacing long sequences of for example M 0's by (M,0) (Lake *et al.*, 1994), but this was not implemented during this research.

		White stones				
		1	2	3	4	5
Black stones	1	2025	91,125	4,100,625	184,528,125	8,303,765,625
	2	91,125	4,100,625	184,528,125	8,303,765,625	
	3	4,100,625	184,528,125	8,303,765,625		
	4	184,528,125	8,303,765,625			
	5	8,303,765,625				

Table 4.1: Overview of database size for FanoRona using a simple index function.

		White stones					
		1	2	3	4	5	6
Black stones	1	2025	44,550	638,550	6,704,775	54,979,155	366,527,700
	2	44,550	980,100	14,048,100	147,505,050	1,209,541,410	
	3	638,550	14,048,100	201,356,100	2,114,239,050		
	4	6,704,775	147,505,050	2,114,239,050			
	5	54,979,155	1,209,541,410				
	6	366,527,700					

Table 4.2: Overview of database size for FanoRona using an advanced index function.

In order to compute the endgame database an iteration through all possible index numbers is started. For every index number the corresponding position is created and all possible moves in this position are examined. If there is a mate for the opponent in X moves within the possible moves, then the current position is a mate in X+1 moves, with X the fastest mate amongst the possible moves. If all moves are a win for the opponent, then the current position is a mate for the current player in Y moves, where Y is the maximum number of moves to mate. The last ensures that even if there is no possible way to escape mate, still the moves are played which maximize the game length.

Retrograde analysis iterates through all possible positions and applies the method described above until all positions are examined and not a single change is found anymore. The calculation of the database has finished when this has happened and all positions where no value has been assigned are draws.

The pseudo code for the retrograde analysis can be found in Appendix A.3.

4.3.2 Motivation

Our motivation for implementing endgame databases during this research is the second maximum in branching factor with a small amount of pieces on the board. Figure 3.3 showed the average branching factor for a certain amount of pieces left on the board and the curve has 2 maxima. An endgame database would cover a large amount of the maximum at the area with a few pieces on the board. We expect this to be helpful since in this area more than one move has to be made to capture a stone of the opponent whereas at the other maximum every move is a capturing move.

4.4 Using Symmetry

Symmetry can happen at almost every game and is quite common in endgame positions. Figure 4.2 shows a position with symmetry. White could move his stone to d5, d4, e4, f4 or f5. But d5 and f5 are symmetrical since this position can be mirrored along the y axis. This means that the game-theoretical value for both moves is the same and it does not matter which move is played. Also d4 and f4 are symmetrical.

If only one of these 2 moves for such a symmetrical pair would be investigated then the search tree is reduced. The position in Figure 4.2 would yield only 3 different moves instead of 5, with White to move. In case Black is to move, only 8 instead of 16 moves have to be considered when using symmetry. If this reduction of possible moves happens at an early stage the size of the solution tree would be significantly smaller.

The chance of symmetry is higher in endgame situations which would result in being able to determine the game-theoretical value a lot faster. If the board of FanoRona would be a square then positions could also be mirrored along the diagonals and the program would benefit even more from it.

Subsection 4.4.1 will describe which possible improvements can be achieved with using symmetry. Subsection 4.4.2 will describe why we have chosen to implement symmetry.

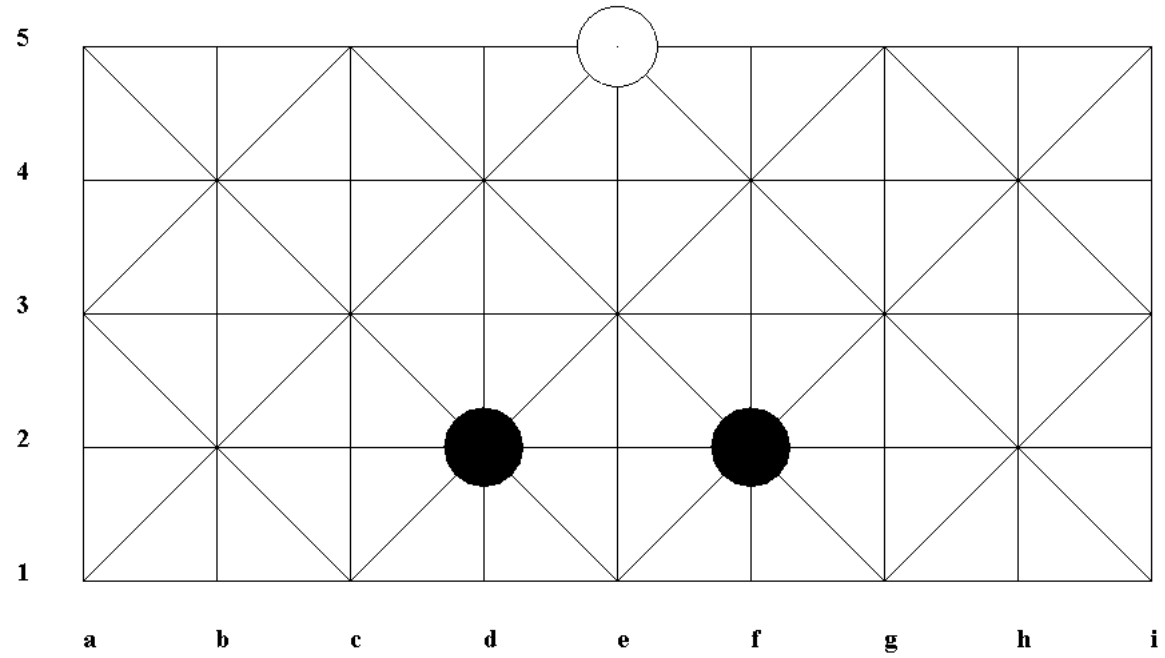


Figure 4.2: An endgame position with symmetry.

4.4.1 Possible gains

This research used the symmetry of FanoRona in three different ways.

- **Move Generation** - When all moves are generated of a position then a move is discarded if it is symmetrical in respect to the x-axis or y-axis with another move. This ensures a smaller search tree which still determines the correct values for proof-number search.

- **Draw determination** - As described in Section 4.1 a node in the search tree is a draw if it yields the same position as one of its predecessors. It also is a draw if one of its predecessors is symmetrical with this position since the game-theoretical value will be the same.
- **Transposition Tables** - If a node is not stored in the transposition table then the symmetric positions can be checked. If one of these was already examined the value of that position can be used. This makes the transposition table far more efficient because more positions can retrieve information from the table.
- **Endgame Database** - In order to fasten up the process of making an endgame database a lookup is done on the symmetric positions. Nothing has to be done if one of those positions has a lower index number than the current one. If a position has to be retrieved from the endgame database the lowest index of all symmetric positions has to be taken. This should decrease the size of the database by approximately 75% since every position has 3 other symmetric ones (fore rectangular boards). Furthermore, less hard drive access is needed and therefore a lot of time is saved.

4.4.2 Motivation

We expect that using symmetry will speed up the search and also speed up the creation of the endgame databases. Transposition tables is a method to save work if the search tree arrives at the same node and combined with symmetry the usefulness of transposition tables will increase, since more positions can be looked up. Also a smaller tree has to be examined since symmetry reduces the branching factor. The motivation for using symmetry is that it will increase the usability of some methods and reduces the size of the solution tree.

Chapter 5

Experiments

This Chapter describes the experiments which are performed for this research. Section 5.1 describes the experimental setup. Section 5.2 describes the results obtained by our experiments. Board sizes with almost the same complexity are summarized. The experiments are performed by the program Ralombo. In Section 5.3 we provide the Chapter conclusions and Section 5.4 will provide ideas for the discussion.

5.1 Experimental setup

In this section we first will describe the experimental setup for small boards (Subsection 5.1.1). After that the setup for the 5×5 board will be shown (Subsection 5.1.2). At the end of this section the results for the large board sizes are investigated (Subsection 5.1.3). All the boards with either a horizontal or vertical side equal to three is counted as a small board. All boards except 5×5 with their smaller side equal to five are considered as a large board.

As a measurement of effectiveness of the introduced methods we will focus on the size of the solution tree and not on the time needed. Preliminary experiments show that the total amount of created nodes is largely proportional to the time needed. Of course, time measurements would be the first choice. However, there are two reasons for basing conclusions on the amount of nodes. (1) The node counts are machine independent, whereas time measurement is not. During the research experiments have been done on different computers which might not need the exact same time for the same experiment. Reasons for this can be the difference in speed of the computer or background programs running during the computation. (2) The move generator for FanoRona is not fast. This means that the time needed to create nodes is the most important part of the total amount of time needed for a computation. These two reasons made us choose the total amount of created nodes as the most important performance measure during these experiments.

5.1.1 Small board sizes

In order to understand the basic tactics of FanoRona the small boards are analyzed. Experiments are done with simple PN search because the complexity described in Table 3.2 indicates that these games would be easy to solve. The 3×3 board can be solved by hand. The 3×5 and 5×3 should be no problem for a computer. The boards 3×7, 7×3, 3×9 and 9×3 will be tested whether they are solvable with simple PN search.

We will look at the optimal strategy on these board sizes after they have been solved. Tactics used here to force a win against a player can give an insight into this game and could help with the generalization of tactics to larger boards. These generalized tactics could be translated to knowledge rules which may reduce the size of the solution tree for larger board sizes.

5.1.2 5×5 board, Fanoron-Dimyand

The Fanoron-Dimyand game could be too complex for the standard PN search algorithm. With a game-tree complexity of 12.7 and a state-space complexity of 11.9 this board size will be much harder to solve than the smaller board sizes.

Since the 5×5 board could be a representative of the more complex board sizes we will test different methods on this board. Dependent on the results we will choose adequate methods for solving the larger board sizes. The methods described in Chapter 4 will be tested on this variant. The number of nodes and the time needed to solve the 5×5 board are used as performance measures.

Also tests are done to investigate different parameters of a method. For example we will change the size of the endgame database to see what its influence is.

There are five properties which are interesting in order to see the applicability of Transposition Tables. These properties are:

- **Nodes** - The number of nodes used to solve the game with PN search.
- **Stored** - The number of times where the game-theoretical value is stored in the Transposition Table. Since PN search is used this is a 0 or 1 if White can win or not.
- **Used** - The number of times where the Transposition Table has been used. Every time where the table is used a subtree is pruned.
- **Collision** - The number of type-2 errors occurring during the run.
- **Time** - The time needed to solve the tree.

These values will be determined on the 5×5 board in order to see the applicability of Transposition Tables.

5.1.3 Large board sizes

After obtaining the results for the experiments on the 5×5 board the best combination of methods is used to derive the game-theoretic value for the root node of these search trees.

5.2 Results

In this section the results of the experiments are described to answer the research question on how effective the selected methods are. We have partitioned the results according to the board sizes into small (Subsection 5.2.1), 5×5 (Subsection 5.2.2), and large board sizes (Subsection 5.2.3). All the experiments described below have symmetry implemented as described in Section 4.4.

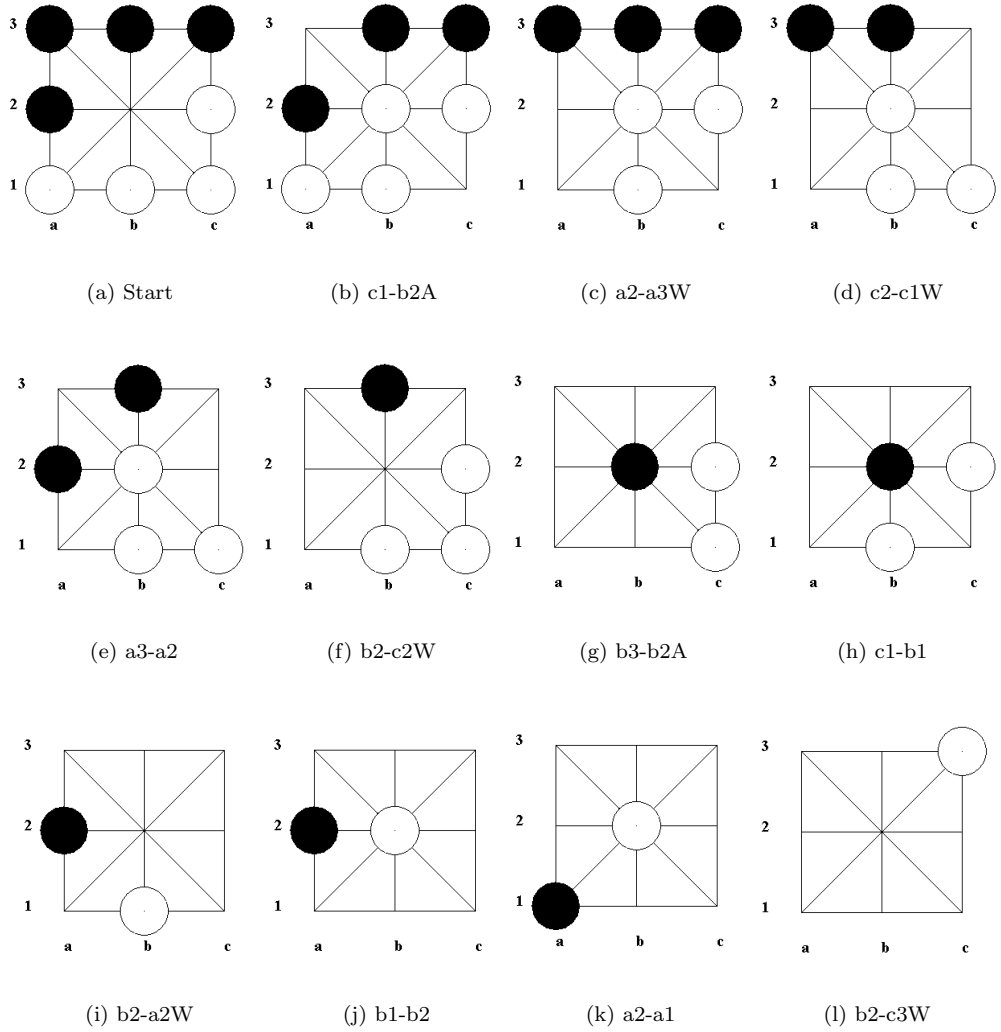
5.2.1 Small board sizes

We will first investigate the Fanoron-Telo variant and then turn to the other small board sizes.

3×3, Fanoron-Telo

Fanoron-Telo is solvable by PN search. During analysis the game-tree complexity was computed to be 4.3 and the state-space complexity to be 4.1 (Section 3.3). Knowing these numbers this was not an unexpected result, just as the analysis in Section 3.2 would expect. Only 107 nodes were used to determine the game-theoretic value of the root. This turned out to be a win for White. Figure 5.1 shows the game played when both players play optimally.

In order to understand the move sequence in Figure 5.1 we can analyse it step by step. White's first move forces Black to play the withdrawal shown in (c). Black does not have another possibility since a



(a) Start (b) c1-b2A (c) a2-a3W (d) c2-c1W

(e) a3-a2 (f) b2-c2W (g) b3-b2A (h) c1-b1

(i) b2-a2W (j) b1-b2 (k) a2-a1 (l) b2-c3W

Figure 5.1: White can force a win.

capturing move has to be played above a paika move. After White plays c2-c1W shown in Subfigure (d) Black has only two possibilities. If he would play b3-c3 he would lose the next move since White could play b2-a1W-a2A and so Black chooses a3-a2. Now White has only one choice to play leaving Black with two possible moves. Black chooses not to capture two stones but to stop capturing after one stone. The reason for this is that if he would capture also the next stone White would move to the middle leaving Black with the position shown in Subfigure (j) which is a loss no matter what Black does. So in fact by stopping early Black postpones his loss for another two moves. White plays c1-b1 leaving Black with two possible moves which actually are the same because of symmetry. After capturing White's stone White moves his last stone to the middle of the board. Now Black can move to either side but White can in both cases capture Black's last stone and win the game.

There are three things that we can learn from this result. First it is obvious how a player has to position his stone to trap an opponent on the board. Second, an important point visible here is that it can be important to sacrifice stones in order to force a player to play a capturing move. This gives the

possibility to control the next move of your opponent and bring the pieces of the opponent to a position where you want them to be. Third, finishing a capturing sequence early can be beneficial for a player.

Other small board sizes

After solving 3×3 the other board sizes can be investigated to see if they also can be solved. In order to do this standard PN search is applied to board sizes 5×3 and 3×5 (see Table 5.1). 1,343 nodes were used in the solution tree for 5×3 to prove that the game-theoretical value is a win for White. Almost the same number of nodes were used in order to prove that it is also possible for White to force a win on the 3×5 board. The games played if both players play optimally can be found in Appendix B.4 and B.5.

We will not discuss every solution in detail since the solution trees are getting more complex with each larger board size. In the optimal game played for the 5×3 size we can find an interesting property of the game. This property is called *Zugzwang* and we can see it in Subfigure h and following of Figure B.5 in Appendix B.4. White sacrifices a stone (e2-e3) in order to force Black to do a move which White wants Black to play. This is possible due to the rule that a capturing move has priority over a non-capturing move. We saw this property already in the optimal play of the 3×3 board and it seems that this way of playing the game is important in order to be able to force a win. Also on other board sizes this principle can be found in the optimally played games.

The optimal played game on the 3×7 board shows that having stones in a good position is much more important than the number of stones (Appendix B.7). In this variant White sacrifices almost all his stones for a good position. White wins the game after having three white stones against eight black stones (Subfigure (g) of Figure B.8). Due to this possibility it could be challenging to find a good evaluation function for this game. With a good evaluation function a more directed and faster search could be possible.

Table 5.1 gives an overview of how difficult it is to solve the small board sizes and what methods have been used. We see that 3×9 takes considerably longer than the other sizes. During the analysis it has been computed that 3×9 has a game-tree complexity of 38.3 as shown in Figure 3.2. Also a 4-piece endgame database was needed in order to be able to solve this board size. This confirms that this variant is indeed the hardest one.

Size	*Winner*	*TT*	*EGDB size*	*Greedy PN*	PN^2	pn_1 *tree nodes*	pn_2 *tree nodes*	*Time (sec)*
3×3	White	no	0	no	no	107	0	0
5×3	White	no	0	no	no	1,343	0	1
3×5	White	no	0	no	no	1,266	0	0
7×3	White	yes	0	yes	no	53,211	0	6
3×7	White	yes	0	yes	no	35,741	0	4
9×3	White	yes	0	yes	no	732,602	0	104
3×9	White	yes	4	yes	yes	280,923	176,649,748	13,879

Table 5.1: Overview of the results on forcing a win with optimal play on small board sizes.

5.2.2 5×5 Board, Fanoron-Dimyand

This section will describe results obtained for the 5×5 board. Depending on these results methods are chosen for solving the larger board sizes.

It is possible to assess the importance of endgame databases if a significant reduction of the search tree is achieved with a larger database. Databases for the 5×5 board were created which stored all positions up to 5 pieces on the board to make this effect visible. Normal PN search was applied to prove that it is not possible for White to force a win on the 5×5 board. The number of nodes used of PN search for

different endgame databases can be found in Table 5.2. We see that increasing the number of pieces in the endgame databases has a high influence on the size of the search tree.

Database size	$NoDB$	$2PieceDB$	$3PieceDB$	$4PieceDB$	$5PieceDB$)
Nodes used	6,128,509	2,690,780	471,286	203,793	115,353

Table 5.2: The effectiveness of endgame databases on the 5×5 board.

All the experiments showed from now on are done with different endgame databases. With more data it is easier to assess if methods are suitable for solving FanoRona or not.

Table 5.3 shows the results of the experiments with Transposition Tables. If this table is compared with Table 5.2 then a small decrease in the size of the solution tree is noticeable. The reason why the decrease is only so small is unknown. Another remarkable point here is that the usability of Transposition Tables increases when smaller endgame databases are used. If no database is used the table is used more often than the number of nodes stored. With a large database only 20% of the stored nodes are used.

Table 5.4 shows an overview of nodes in the solution tree for different extensions of the PN search algorithm. PN^2 pn_1 denotes the number of nodes which have to be stored in the memory and PN^2 *total* the total number of nodes created, i.e., including the pn_1 and pn_2 nodes.

Table 5.4 shows that using greedy PN numbers can have a positive effect on the size of the solution tree but it is also possible that the size increases. The increase in size of the solution tree happens when large endgame databases are used. This means that the solution tree benefits the most of greedy PN numbers when there are not many pieces left on the board. The reason for these larger solutions trees can also be that 5×5 is a draw and not a win for White. Therefore a greedy directed search could create a solution tree containing more nodes.

If the 5×5 board would be a win for White then maybe the greedy PN numbers are even more beneficial. Table 5.5 shows that if White can win the game, i.e., on the 7×3 board, the use of greedy PN numbers indeed decreases the size of the solution tree. Remarkable is that if no endgame database is used greedy PN numbers store fewer nodes in the solution tree than PN^2. Also the child numbers make a better impression here than on the 5×5 board.

The solution tree is smaller for large endgame databases when child numbers are used but the solution tree is much larger when the endgame databases are small. The child numbers simulate a solution tree which is overall one step deeper and this is apparently beneficial during the opening and the midgame of FanoRona but not during the endgame.

Furthermore Table 5.4 shows that PN^2 is useful in terms of stored nodes. Many more nodes need to be created in order to solve a position but the number of nodes stored in memory is decreased and that is exactly why PN^2 was invented.

5.2.3 Large board sizes

The methods which are used for the large board sizes are chosen based on the results for the 5×5 board described in Section 5.2.2. The methods chosen are Transposition Tables, Endgame Databases, Greedy

Database size	*Nodes*	*Stored*	*Used*	*Collision*	*Time(sec)*
No DB	6,124,803	70,562	112,789	34,573	716
2 piece	2,482,912	50,133	30,414	23,981	243
3 piece	460,186	10,440	2,369	2,124	44
4 piece	201,891	4,396	1,030	215	20
5 piece	114,743	2,508	439	149	13

Table 5.3: The effectiveness of transposition tables on the 5×5 board for different endgame databases.

Game	*NoDB*	*2PieceDB*	*3PieceDB*	*4PieceDB*	*5PieceDB*
Normal	6,128,509	2,797,698	474,850	205,603	117,330
Transposition Tables	6,124,803	2,403,748	464,804	203,693	116,784
Greedy PN numbers	2,528,452	1,561,654	485,221	272,478	137,897
Child Numbers	>10,000,000	5,211,099	672,301	192,456	114,117
PN^2 pn_1	233,722	117,216	94,893	68,190	50,819
PN^2 *total*	18,699,539	6,323,686	946,532	417,403	209,645

Table 5.4: The number of nodes created in the search tree for the 5×5 board using different endgame databases, different PN-search variants, and transposition tables.

Game	*NoDB*	*2PieceDB*	*3PieceDB*	*4PieceDB*
Normal	286,271	217,427	55,040	14,951
Transposition Tables	202,306	213,300	55,345	14,332
Greedy PN numbers	52,360	37,646	45,277	20,174
Child Numbers	99,117	74,725	29,401	10,484
PN^2 pn_1	55,141	53,743	28,008	9,222
PN^2 *total*	419,247	368,001	115,026	25,489

Table 5.5: The number of nodes created in the search tree for the 7×3 board for different endgame databases using different methods.

PN numbers and PN^2. We have combined the Greedy PN numbers with PN^2 so that the pn_2 tree uses the Greedy PN numbers instead of the standard numbers.

Table 5.6 shows the results of this method on the larger board sizes. 7×5 was solved and shown to be a draw. The numbers given are for a PN-search for the start position (proving that this is not a win for White) plus a PN search after White plays d2-d3 (proving that White can at least achieve a draw). *TT* is an abbreviation for Transposition Tables and EGDB is an abbreviation for endgame database.

Board Size	*Winner*	*TT*	*EGDB size*	*Greedy PN*	*PN^2*	*pn_1 nodes*	*pn_2 nodes*	*Time(sec)*
7×5	Draw	yes	6	yes	yes	135,060	44,938,999	5,150
5×7	Not solved	yes	6	yes	yes	>7,000,000	>30,000,000,000	>1 week
9×5	Not solved	yes	6	yes	yes	>10,000,000	>50,000,000,000	>1 week

Table 5.6: Overview of the results on forcing a win with optimal play on large board sizes.

Unfortunately the other large variants were not solved during the time of this research. Table 5.6 shows the progress of solving at the end of this research. These results show the increase of the solution tree when increasing the board size. Not being able to solve these board sizes in time of this research also underlines the importance of knowledge rules to create cut-offs in the solution tree.

5.2.4 Summary of the results

This section is meant to give an overview of the achieved results in order to be able to compare all board sizes. The increase in complexity is visible when the board size increases (Table 5.7). All board sizes up to 7×5 are easy to solve but then it is considerably harder to solve the 5×7 and 9×5 board.

Another remarkable result of these experiments has been that 7×5 is solvable while 5×7 is not. The state-space complexity of these variants is the same and the game-tree complexity is almost the same. In spite of this, 5150 seconds where needed to solve 7×5 while 5×7 could not be solved in a week time.

This shows that the computation of the game-tree complexity is not very accurate.

Board Size	*Winner*	*TT*	*EGDB size*	*Greedy PN*	PN^2	pn_1 *nodes*	pn_2 *nodes*	*Time(sec)*
3×3	White	no	0	no	no	107	0	0
5×3	White	no	0	no	no	1,343	0	1
3×5	White	no	0	no	no	1,266	0	0
7×3	White	yes	0	yes	no	53,211	0	6
3×7	White	yes	0	yes	no	35,741	0	4
9×3	White	yes	0	yes	no	732,602	0	104
3×9	White	yes	4	yes	yes	280,923	176,649,748	13,879
5×5	Draw	yes	5	no	yes	45,464	173,572	13
7×5	Draw	yes	6	yes	yes	135,060	44,938,999	5,150
5×7	Not solved	yes	6	yes	yes	>7,000,000	>30,000,000,000	>1 week
9×5	Not solved	yes	6	yes	yes	>10,000,000	>50,000,000,000	>1 week

Table 5.7: Overview of the results on forcing a win with optimal play.

5.3 Conclusions

We have seen that the smaller variants of the boards are solvable. With the help of the introduced methods also 5×5 and 7×5 have been solved. 5×7 and 9×5 were not solvable yet in the time of this research.

Furthermore we noticed from the experiments that using child numbers only gives a small advantage in number of nodes with large endgame databases but performs worse when no large databases are available on the 5×5 board. However, child numbers do perform better on the 7×3 board. Thus we do not advise to use child numbers for FanoRona until further research is done. The most successful methods examined were Endgame Databases, Greedy PN numbers and PN^2 which reduce the solution tree for FanoRona.

A conclusion which may be drawn from the results is that White has an advantage on the smaller boards where one of the two sides has size 3. If this is the case then White can play a strategy which results in a forced win for White. When the board is larger White apparently loses this advantage and the game-theoretical value is a draw. Because the 5×7 and the 9×5 board have not been solved during this research, this claim so far is only based on the game-theoretical values of two board sizes (i.e., 5×5 and 7×5).

5.4 Discussion

This section will discuss important findings of the experiments. Subsection 5.4.1 will discuss our findings concerning the greedy PN heuristic. Subsection 5.4.2 is a general discussion about knowledge-rich methods and Subsection 5.4.3 compares our results to Uiterwijk and Van den Herik (2000) with respect to the advantage of the initiative.

5.4.1 Greedy PN numbers

Section 5.2.2 describes a reduction in the size of the solution tree when the greedy heuristic is used during PN search. The greedy heuristic is entirely based on the number of pieces on the board. However there exist situations where White can win the game in spite of a 3 vs 8 situation. This implies that the location of a stone is an important factor for the strength of a position. Appendix B.7 shows an example of a situation where White sacrifices most of his stones for a good position. In Chess this would resemble a situation where a player sacrifices his Queen in order to mate.

Also the player's turn is a factor for the strength of a position which is not implied by the greedy heuristic. Since capturing is obliged and long capturing sequences are possible the value of the greedy function can change completely after only one turn of the opponent and an advantage can turn into a disadvantage in the evaluation function.

Having too many stones could also be a disadvantage. It is possible that capturing is not possible because the player's own stones block movement lines. We suspect that it is disadvantageous for a player to have too many pieces on the board as well as it is to have too few pieces on the board.

In spite of these scenarios the greedy function decreases the size of the solution tree. Consequently, we believe that in most positions the player with the majority of stones wins the game. However, if more research is done to design an evaluation function which incorporates positioning of pieces then we believe that such a heuristic will perform even better than a greedy function.

5.4.2 Knowledge-rich methods

In this research we showed that knowledge-rich methods should be applied to solve FanoRona. This research only investigates one knowledge-rich method (Greedy PN search). We observed that using greedy PN numbers the size of the solution tree decreases, especially when there are no databases available (Table 5.4). In order to solve FanoRona other knowledge-intensive methods should be implemented and tested.

We believe that such knowledge-rich methods can be improvements on the greedy methods, which also take positioning into account. During the endgame it can be very important to have stones on points with diagonal lines. This implies more directions in which the stone could capture opponent stones. This restricts the space for opponent stones and can lead to a win. Including such game-knowledge into the PN search algorithm could lead to a reduction of the solution tree.

5.4.3 The advantage of the initiative

Uiterwijk and Van den Herik (2000) concludes in the article "The advantage of the initiative" that in a great deal of games the first player can win the game if the board is large enough. This has been shown for the game of Domineering (Breuker, Uiterwijk, and Van den Herik, 1999) and many K-in-a-row games (Uiterwijk and Van den Herik, 2000).

In Section 5.3 we conclude that White looses his advantage when the board size increases. Resulting from this FanoRona would be an example where White has an advantage when the board is small and not when the board is large. We assume that a small board with either horizontal or vertical size of three is a win for White. On such a small board White has the possibility to play in a way that Black can not win the game. In Appendix B.7 we show that the optimal play for White is to sacrifice stones to force Black to play capturing moves.

Summarizing we can say that White has the advantage of the initiative when playing on small boards and not on large boards. So, FanoRona does not belong to the majority of games where White has an advantage on large boards.

Chapter 6

Conclusions

This Chapter contains our conclusions on the research performed. Section 6.1 will investigate to what extent the research questions are answered and gives conclusions concerning the problem statement. Section 6.2 will describe further research possibilities on the game of FanoRona.

6.1 Problem statement and research questions revisited

In Section 1.3 the following three research questions where stated:

1. What complexity does FanoRona have?

In Chapter 3 we estimated the game-tree complexity as $O(10^{40})$ and the state-space complexity as $O(10^{21})$. This means that FanoRona is harder than Checkers. We showed during this research that the game-tree complexities might be overestimated, especially for some smaller board sizes (Table 3.2). It is important to create more reliable data to estimate the game-tree complexity. Based on the state-space complexity (and in spite of the unreliable game-tree complexity) for FanoRona we expect that FanoRona will soon be solved.

2. What methods are suitable for solving Fanorona?

In Chapter 4 different methods are presented which are applied during this research. Both Van den Herik *et al.* (2002) and Allis *et al.* (1991a) come to the conclusion that knowledge-based methods are more appropriate for solving games with a low decision complexity. Thus such knowledge-based methods should be implemented. Since FanoRona is not a well-known game the knowledge for making knowledge rules is not available and consequently brute-force methods have been applied. The only knowledge-rich method described are the Greedy PN Numbers which create proof and disproof numbers according to the number of pieces on the board.

3. How do these methods perform when applying to solve FanoRona?

Chapter 5 shows the results of the methods described in Chapter 4. We saw that child numbers for nodes in the PN search did not perform well.

Furthermore using Transposition Tables did not make a big difference on the size of the solution tree, at least when endgame databases are used. This is rather an unexpected result since Transposition Tables perform well for other board games. We also saw that PN search with Endgame Databases, Greedy PN numbers and PN^2 is effective with respect to the size of the solution tree.

Especially the Greedy PN numbers did lead to a decrease in size of the solution tree since this method has implemented game knowledge.

Can a computer program be written so that it weakly solves the board game FanoRona?

We showed that it is not possible at this moment to solve the game of FanoRona with a program which uses Proof-Number search. Endgame Databases, Greedy PN Numbers and PN^2 are methods which are proven to be effective for the size of the solution tree. To make solving FanoRona possible knowledge-rich pruning methods should be implemented. With such knowledge-rich methods we expect that FanoRona is a game which will be solved within the year.

6.2 Further Research

The game-tree complexity of FanoRona can be subject of further research. During this research alpha-beta players with a shallow search depth were used to determine the average branching factor and average game length. This gives an estimate of the game-tree complexity of FanoRona. With real games played by humans or by strong computer players these values can be computed more reliable. Since FanoRona is mainly played in Madagaskar and is not a well-known game yet these data were not available during the time of this research.

Furthermore, other methods can be examined which can result in a smaller search tree. These should be knowledge-rich methods. First if it is possible to prove that a player will win the game far before the endgame database is reached then a cutoff can be made and the size of the solution tree decreases.

A second example is to be able to make use of a database even if the current situation is not stored in the endgame database. If a 5-piece database is available and there are 6 pieces left on the board with one piece far away from the others without influencing the rest of the game then the possibility of using the database has to be examined to create a cutoff. Otherwise, the optimal move in the 5-piece database can be used as a move-selection heuristic in the 6-piece situation.

During this research Greedy PN Numbers have been proven to be successful. This heuristic is only based on the number of pieces on the board. This method can be enriched with more knowledge of the game. For example standing on a point where also diagonal movement is possible can be important in the endgame. Extending the Greedy PN Numbers with knowledge of the game could improve PN search.

In order to achieve a better understanding of the game, it could be investigated if the optimal play for the larger board sizes resembles the optimal play for the smaller board sizes. This could lead to identifying tactics which improve the play on even large boards.

Finally, a better implementation can be made which is more memory efficient. Also a faster move generator can be made to increase the program speed. More research can be done regarding the creation of the endgame databases. An improved index function for the retrograde analysis with less gaps can lead to a faster database creation, using less memory, and therefore larger endgame databases.

As a last point more research can be done on the solution tree of the PN search algorithm where child numbers are used. This research showed varying results of this method and more research is to be done to examine if child numbers are an improvement for FanoRona.

References

Allis, L.V. (1988). *A knowledge-based Approach of Connect-Four. The game is solved: White wins.* M.Sc. thesis, Faculty of Methematics and Computer Science, Free University, Amsterdam, The Netherlands. [2, 13]

Allis, L.V. (1994). *Searching for Solutions in Games and Artificial Intelligence.* Ph.D. thesis, Rijksuniversiteit Limburg, Maastricht. ISBN 9090074880. [2, 12, 18, 19]

Allis, L.V., Herik, H.J. van den, and Herschberg, I.S. (1991a). Which games will survive? *Heuristic Programming in Artificial Intelligence 2: the second computer olympiad (eds. D.N.L. Levy and D.F. Beal)*, pp. 232–243. Ellis Horwood Limited, Chichester, UK. ISBN 0–13–382615–5. [2, 13, 35]

Allis, L.V., Meulen, M. van der, and Herik, H.J. van den (1991b). Databases in Awari. *Heuristic Programming in Artificial Intelligence 2: the second computer olympiad (ed. D.N.L. Levy and D.F. Beal)*, pp. 73–86. Ellis Horwood Limited, Chichester, UK. [21]

Allis, L.V., Meulen, M. van der, and Herik, H.J. van den (1994). Proof-Number Search. *Artificial Intelligence*, pp. Vol. 66, No. 1, pp. 91–124. [17, 18]

Bell, R.C. (1980). *Board and Table Games from Many Civilizations.* Dover Publications. ISBN 0486238555. [5]

Berkey, D.D. (1988). *Calculus.* Saunders College Publishing, New York, NY. [19]

Breuker, D.M. (1998). *Memory versus Search in Games.* Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands. [20, 21]

Breuker, D.M., Allis, L.V., and Herik, H.J. van den (1994a). How to Mate: Applying Proof-number Search. *Advances in Computer Chess 7 (eds. H.J. van den Herik, I.S. Herschberg, and J.W.H.M. Uiterwijk)*, pp. 251–272. University of Limburg, Maastricht, The Netherlands. ISBN 90–621–6101–4. [17, 18]

Breuker, D., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1994b). Replacement Schemes for Transposition Tables. *ICCA Journal*, Vol. 17, No. 7, pp. 183–193. [21]

Breuker, D.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1999). Solving 8×8 Domineering. *Theoretical Computer Science, 230*, pp. 195–206. Also published in Technical Report in Computer Science, CS 98–05, Department of Computer Science, Universiteit Maastricht, Maastricht, 1998. [34]

Breuker, D.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2001). The PN^2-search algorithm. *Advances in Computer Games 9 (eds. H.J. van den Herik and B. Monien)*, pp. 115–132. IKAT, Universiteit Maastricht, Maastricht, The Netherlands. [19, 20]

Chauvicourt, J & S (1980). *Le Fanorona - Jeu National Malgache.* Nouvelle Imprimerie de Arts Graphiques. [5]

Donkers, J. and Uiterwijk, J.W.H.M. (2002). Programming Bao. *Technical report*, Seventh Computer Olympiad: Computer-Games Workshop Proceedings (ed. J.W.H.M. Uiterwijk), Universiteit Maastricht, Maastricht, The Netherlands. CS 02-03. [9]

Gasser, R. (1996). Solving Nine Men's Morris. *Games of No Chance (ed. K.J. Nowakowski)*, Vol. 29, pp. 101–113.MSRI Publications, Berkeley, USA. [1]

Goot, R. van der (2001). Awari Retrograde Analysis. Computers and Games Second International Conference. *Lecture Notes in Computer Science LNCS 2063*, pp. 87–95. ISBN 3540430806. [21]

Herik, H.J. van den (1983). *Computerschaak, Schaakwereld en Kunstmatige Intelligentie*. Ph.D. thesis, Delft University of Technology. Delft, The Netherlands. In Dutch. [1]

Herik, H.J. van den (2005). *When will chess be solved? (with H.H.L.M. Donkers)*. Symposium Propagation of Heuristic Evaluation Errors in Game Graphs. University of Ljubljana, Slovenia, August 29,2005. [1]

Herik, H.J. van den, Uiterwijk, J.W.H.M., and Rijswijk, J.H. van (2002). Games Solved: Now and in the Future. *Artificial Intelligence 134*, pp. 277–311. [2, 13, 35]

Lake, R., Schaeffer, J., and Lu, P. (1994). Solving large retrograde-analysis problems using a network of workstations. *Advances in Computer Chess 7 (eds. H.J. van den Herik, I.S. Herschberg, and J.W.H.M. Uiterwijk)*, pp. 165–162. Universiteit Maastricht, Maastricht, The Netherlands. [22, 23]

Levy, D.N.L. and Newborn, M.M. (1991). *How Computers Play Chess*. Computer Science Press. ISBN 0–716–78239–1. [1]

Lorenz, U. and Rottman, V. (1997). Parallel controlled conspiracy-number search. *Advances in Computer Chess 8 (eds. H.J. van den Herik and J.W.H.M. Uiterwijk)*, pp. 129–152. Universiteit Maastricht, Maastricht, The Netherlands. [17]

McAllester, D.A. (1988). Conspiracy Numbers for Min-Max Search. *Artificial Intelligence 35*, pp. 287–310. ISSN 0004–3702. [17]

Murray, H.J.R. (1952). *A History of Board Games other than Chess*. Oxford University Press, Oxford, UK. ISBN 0878172114. [1]

Nagai, A. (2002). *Df-pn Algorithm for Searching AND/OR Trees and Its Applications*. Ph.D. thesis, Dissertation, Department of Information Science, University of Tokyo. [18]

Neumann, J. von (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen 100*, pp. 295–320. Translated by Bargmann (1959). [1]

Neumann, J. von and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press. ISBN 0691003629. [1]

Newborn, M. (1996). *Kasparov versus Deep Blue. Computer Chess Comes of Age*. Springer-Verlag, New York. ISBN 0–387–94820–1. [1]

Roijakkers, C.P.E. (2005). *Analysis and Implementation of the game of Octi*. M.Sc. thesis, Universiteit Maastricht, The Netherlands. [20]

Romein, J.W. and Bal, H.E. (2003). Solving awari with parallel retrograde analysis. *Computer*, Vol. 36, pp. 26–33. ISSN 0018–9162. [1, 22]

Schaeffer, J. (1990). Conspiracy numbers. *Artificial Intelligence, 33 (1)*, pp. 67–84. [17]

Schaeffer, J. (1997). *One Jump Ahead : Challenging Human Supremacy in Checkers*. Springer-Verlag. ISBN 0387949305. [21]

Schaeffer, J. and Herik, H.J. van den (2002). Games, computers, and artificial intelligence. *Artificial Intelligence, 134*, pp. 1–2 (Jan. 2002), 1–7. [1]

Schaeffer, J. and Lake, R. (1996). Solving the Game of Checkers. *Games of No Chance (ed. K.J. Nowakowski)*, Vol. 29, pp. 119–133. MSRI Publications, Berkeley, USA. [2]

Schaeffer, J., Bjornsson, Y., Burch, N., Lake, R., Lu, P., and Sutphen, S. (2003). Building the Checkers 10-piece Endgame Databases. *Technical report*, University of Alberta, Edmonton, AlbertaCanada. [22]

Schaeffer, J., Bjornsson, Y., Burch, N., Kishimoto, A., M.Mller, Lake, R., Lu, P., and Sutphen, S. (2005). Solving Checkers. *IJCAI-05 proceedings*, pp. 292–297. [2]

Seo, M., Iida, H., and Uiterwijk, J.W.H.M. (2001). The PN*-search algorithm: application to Tsume-Shogi. *Artificial Intelligence 129*, pp. 253–277. [17]

Shannon, C.E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, Vol. 41, No. 7, pp. 256–257. [1]

Ströhlein, T. (1970). Untersuchungen über kombinatorische Spiele. *Dissertation, Fakultät für Allgemeine Wissenschaften der Technischen Hochschule München.* [21]

Turing, A.M. (1953). Digital Computers Applied to Games. *Faster than Thought(ed. B.V. Bowden)*, pp. 286–297. Pitman, London. [1]

Uiterwijk, J.W.H.M. and Herik, H.J. van den (2000). The Advantage of the Initiative. *Information Sciences*, Vol. 122, No. 1, pp. 43–58. [33, 34]

Uiterwijk, J.W.H.M., Allis, L.V., and Herik, H.J. van den (1989). A Knowledge-Based Approach to Connect-Four. *Heuristic Programming in Artificial Intelligence: the first computer olympiad (eds. D.N.L. Levy and D.F. Beal)*, pp. 113–133. Ellis Horwood Limited, Chichester, UK. [13]

Winands, M.H.M. (2000). Analysis and Implementation of Lines of Action. M.Sc. thesis, Universiteit Maastricht, The Netherlands. [12]

Winands, M.H.M. and Uiterwijk, J.W.H.M (2001). PN, PN2 and PN* in Lines of Action. *The CMG Sixth Computer Olympiad Computer-Games Workshop Proceedings* (ed. J.W.H.M. Uiterwijk). Technical Reports in Computer Science CS 01-04. Universiteit Maastricht, Maastricht, The Netherlands. [17]

Winands, M.H.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2004). An effective two-level proof-number search algorithm. *Theoretical Computer Science*, Vol. 313, pp. 511–525. [19]

Zobrist, A.L. (1970). A New Hashing Method with Application for Game Playing. *Technical report 88*, Computer Science Department, The University of Wisconsin,Madison, WI, USA. Reprinted in (1990) *ICCA Journal*,Vol. 13,No. 2, pp. 69-73. [20]

Appendix A

Algorithms

A.1 Alpha-Beta pseudo-code

```
function evaluate(node, alpha, beta)
begin
    if is_leaf(node)
        return heuristic_value(node)
    endif
    if minimizing_node(node)
        foreach child of node
            beta := min (beta, evaluate (child, alpha, beta))
            if beta < = alpha
                return alpha
            endif
        endforeach
        return beta
    endif
    if maximizing_node(node)
        foreach child of node
            alpha := max (alpha, evaluate (child, alpha, beta))
            if beta < = alpha
                return beta
            endif
        endforeach
            return alpha
    endif
end
```

A.2 Proof-Number search pseudo-code

```
function select_most_proving_node(J)
begin
    while is_an_internal_node(J) do
        if max_node(J) then
            J:=leftmost_son_with_equal_proof_number(J)
        else
            J:=leftmost_son_with_equal_disproof_number(J)
        endif
    done
    return J
end

    procedure update_proof_numbers(J)
    begin
        while j≠NIL do
            if max_node(J) then
                proof(J):= min_{j∈sons(J)} proof (j)
                disproof(J) := Σ_{j∈sons(J)} disproof(j)
            else
                proof(J):= Σ_{j∈sons(J)} proof(j)
                disproof(J) := min_{j∈sons(J)} disproof (j)
            endif
            j:=father(J)
        done
    end

    procedure main
    begin
        while not (root.get_proof_number()=0 or root.get_disproof_number()=0) do
            j = select_most_proving_node(root)
            j.expand_node()
            update_proof_numbers(j)
        done
    end
```

A.3 Retrograde analysis pseudo-code

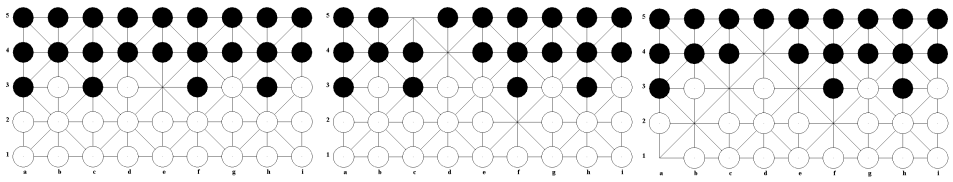
```
procedure retrostep
begin
    for (i=0,i<all_Boards,i++)
        B := make_board_from_index(i)
        if not already_solved_in_DB(B)
            Children := create_children(B)
            Children_Values := check_values_in_DB(Children)
            if found_Loose(Children_Values)
                assign_win_in_DB(i)
            endif
            if all_Win(Children_Values)
                assign_loose_in_DB(i)
                endif
        endif
    endfor
end

    procedure main
begin
    lastcount=-1
    thiscount=0
    while lastcount≠thiscount
        retrostep
        lastcount:=thiscount
        thiscount:=count_how_many_solved_in_DB()
    done
end
```

Appendix B

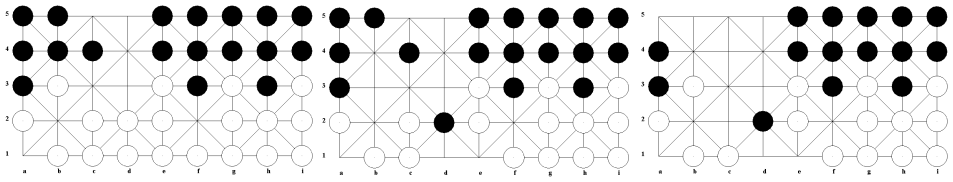
Detailed results

B.1 Example game

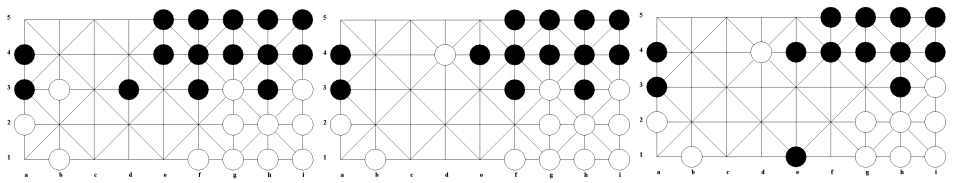
The game showed in Figures B.1 and B.2 was played in a FanoRona tournament organized by the Universiteit Maastricht in the course Intelligent Search Techniques. Both players used alpha-beta search with iterative deepening and transposition tables.



(a) Start (b) f2-e3A (c) c3-d4W-c5W

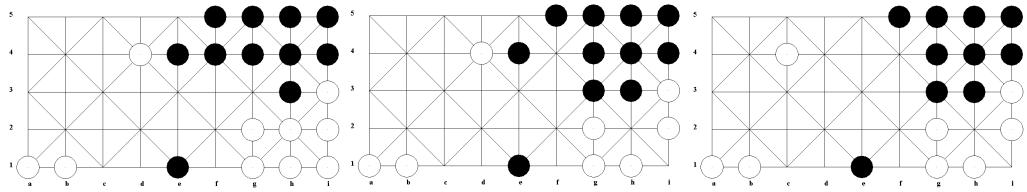


(d) d3-d4A-e3W (e) b4-c3A-d3W-d2A (f) c2-c3A-b4A-b3W

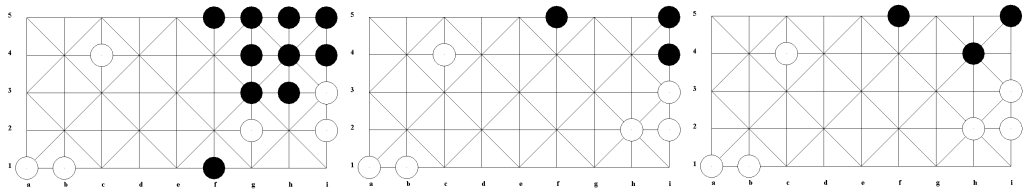


(g) d2-c2W-c3W-d3A (h) b3-c3A-d4A (i) f3-f2A-e1W

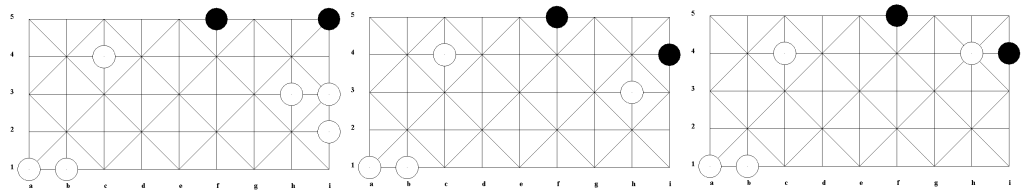
Figure B.1: An example FanoRona game (Part1).



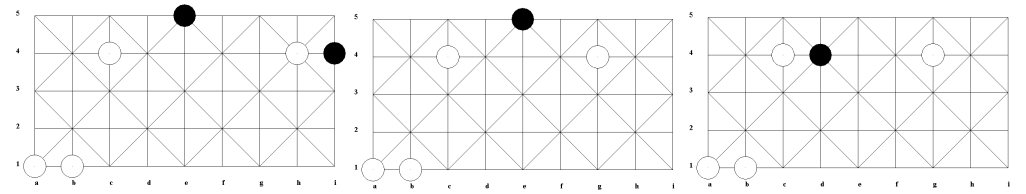
(a) a2-a1W (b) f4-g3A (c) d4-c4W



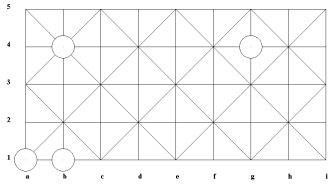
(d) e1-f1A (e) g2-g1W-h1W-h2A (f) i4-h4



(g) h2-h3A (h) i5-i4A (i) h3-h4A



(j) f5-e5 (k) h4-g4W (l) e5-d4



(m) c4-b4W

Figure B.2: An example FanoRona game (Part2).

B.2 Position with high branching factor

The position shown in Figure B.3 has 414 possible capturing moves for Black.

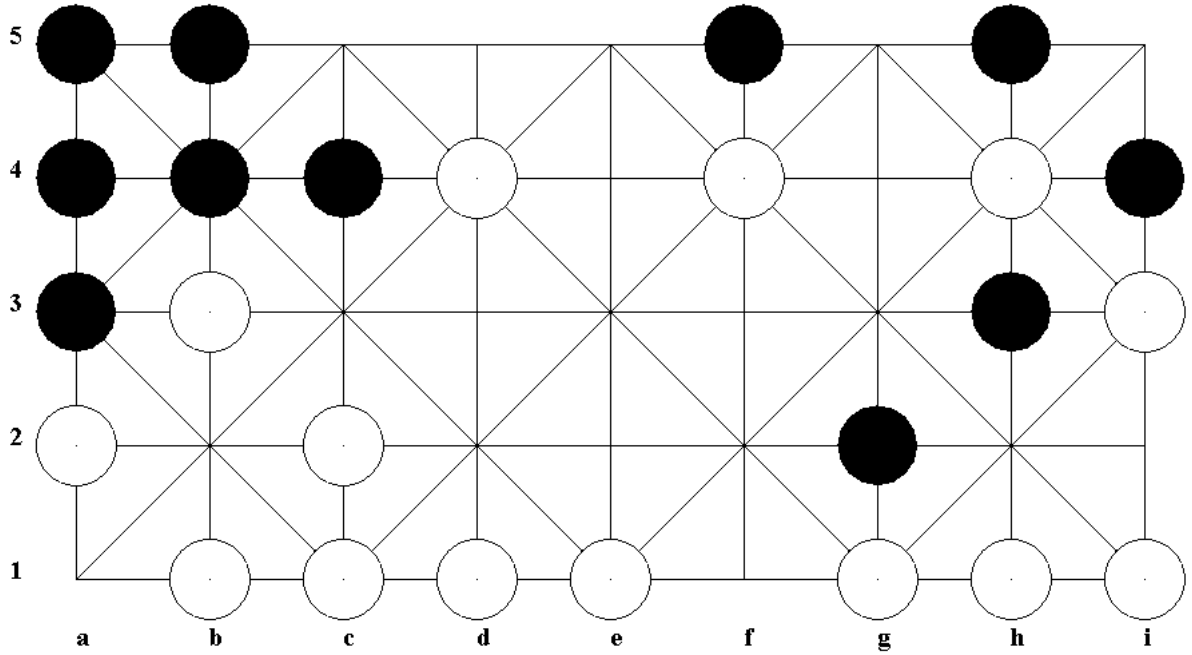


Figure B.3: Black can play 414 different capturing moves.

B.3 Black wins in 44 moves

The position shown in Figure B.4 on the 7×5 board is a position where Black has a mate in 44 moves. If both players are playing optimally then we get the moves shown. The optimal strategy for White is defined as postponing the loss as long as possible. It is White's turn to move.

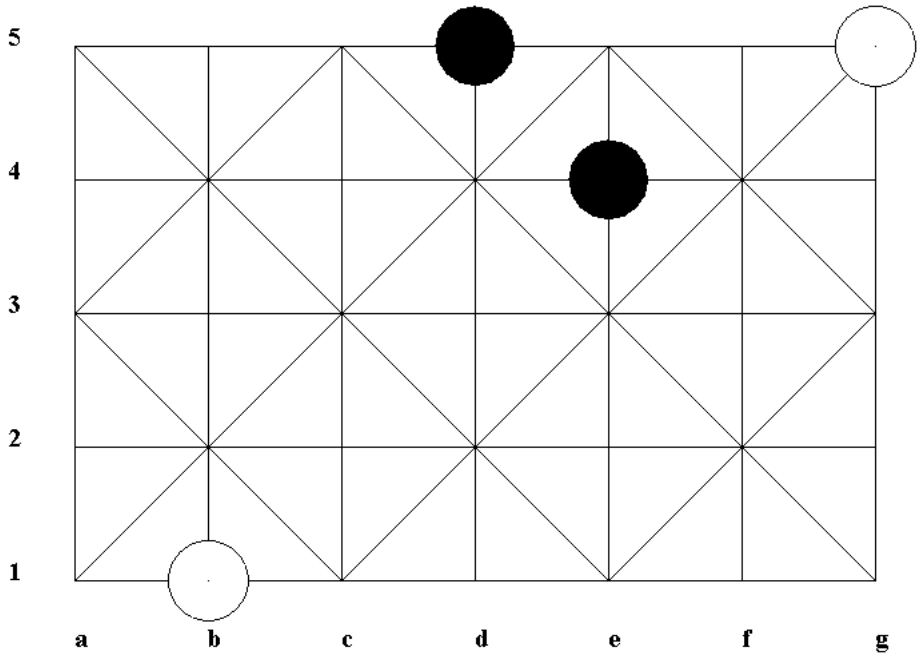
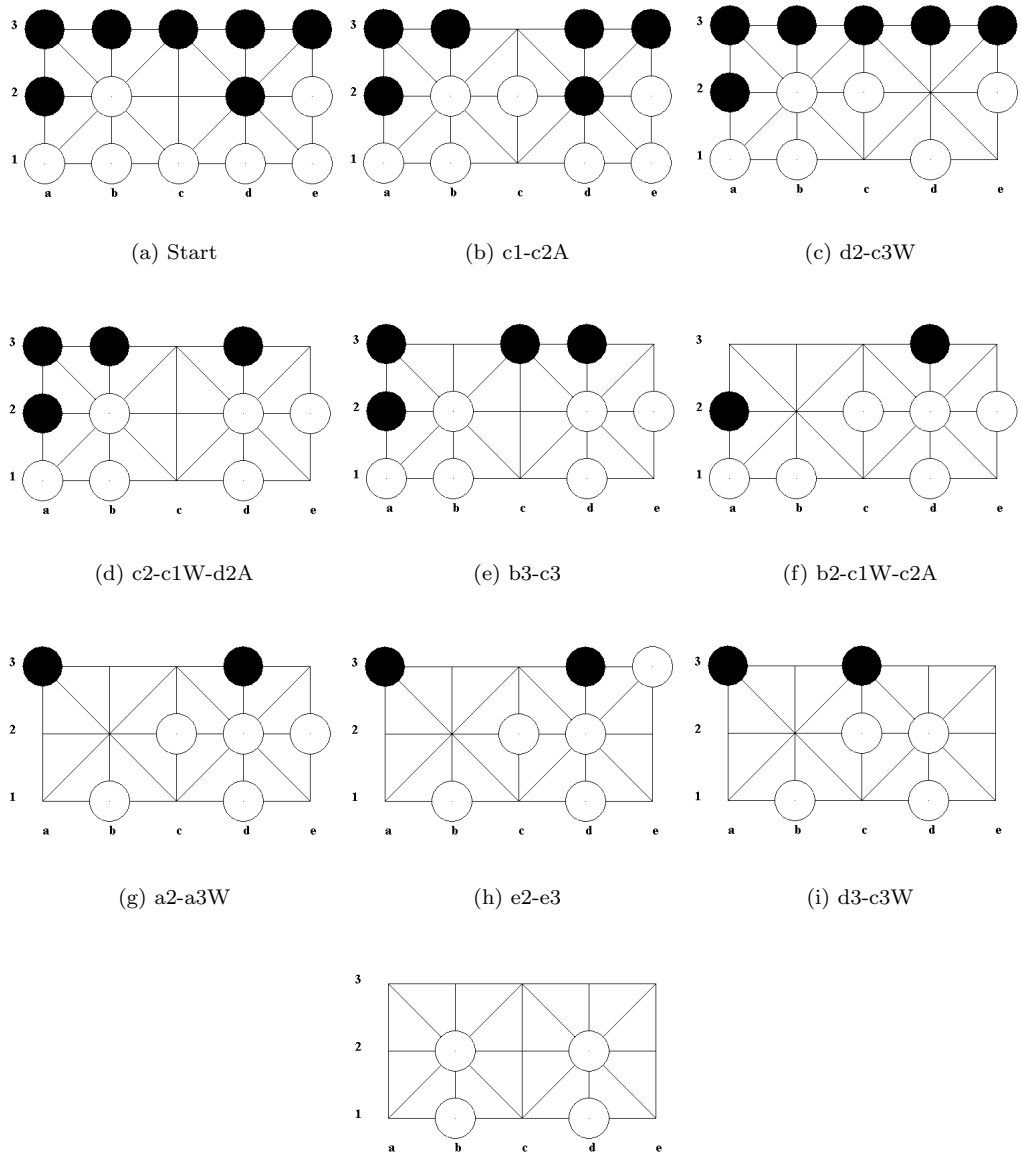


Figure B.4: Black has a mate in 44 moves, White to move.

1.b1-c1 e4-d4 2.c1-d1 d4-c3 3.g5-f4 d5-c5 4.f4-e4 c5-b5 5.e4-f4 b5-a5 6.f4-e4 a5-a4 7.e4-f4 a4-a3 8.f4-e4 a3-a2 9.e4-f4 a2-b2 10.f4-e4 b2-a3 11.e4-f4 a3-b3 12.f4-g5 b3-b2 13.g5-f4 b2-c2 14.d1-d2 c2-b2W 15.f4-f3 b2-b1 16.f3-f4 b1-c1 17.f4-f5 c3-d3 18.f5-e5 c1-d2 19.e5-e4 d2-e3 20.e4-e5W d3-d4 21.e5-f5 d4-e3 22.f5-e5 e3-e4A

B.4 Optimal play 5×3



(a) Start (b) c1-c2A (c) d2-c3W

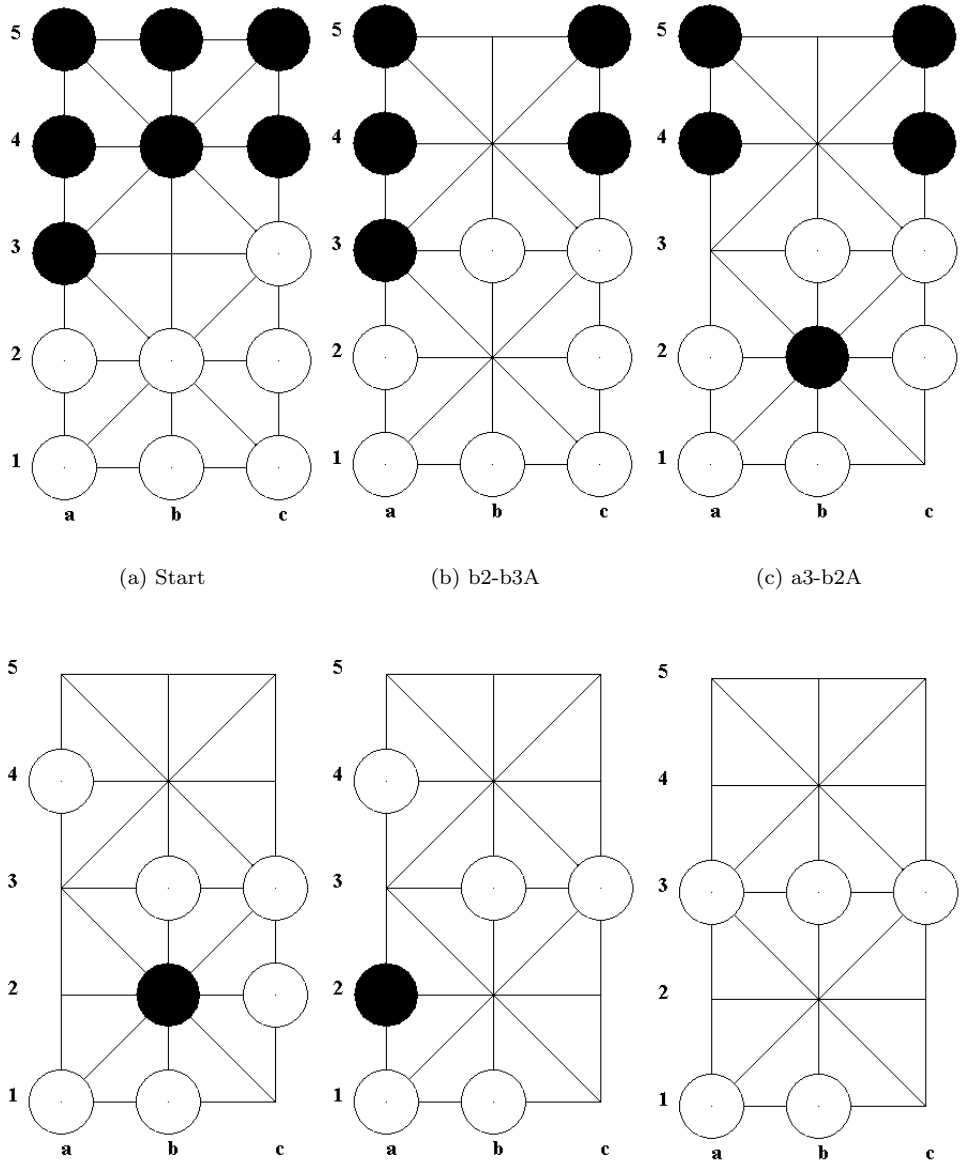
(d) c2-c1W-d2A (e) b3-c3 (f) b2-c1W-c2A

(g) a2-a3W (h) e2-e3 (i) d3-c3W

(j) c2-c1W-b2A

Figure B.5: White can force a win on the 5x3 board.

B.5 Optimal play 3×5

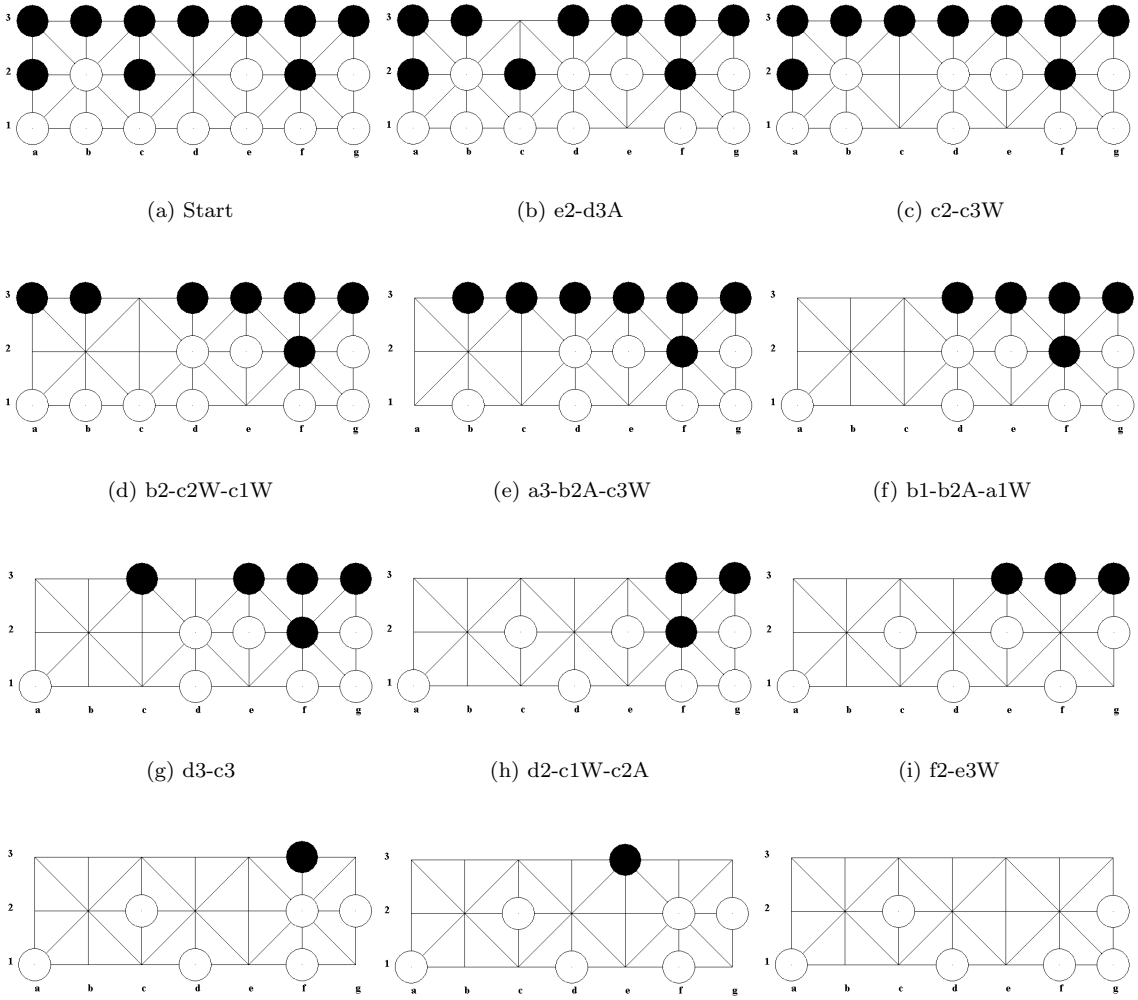


(a) Start (b) b2-b3A (c) a3-b2A

(d) a2-a3A-b4A-a4W (e) b2-a2W (f) a4-a3A

Figure B.6: White can force a win on the 3x5 board.

B.6 Optimal play 7×3



(a) Start (b) e2-d3A (c) c2-c3W

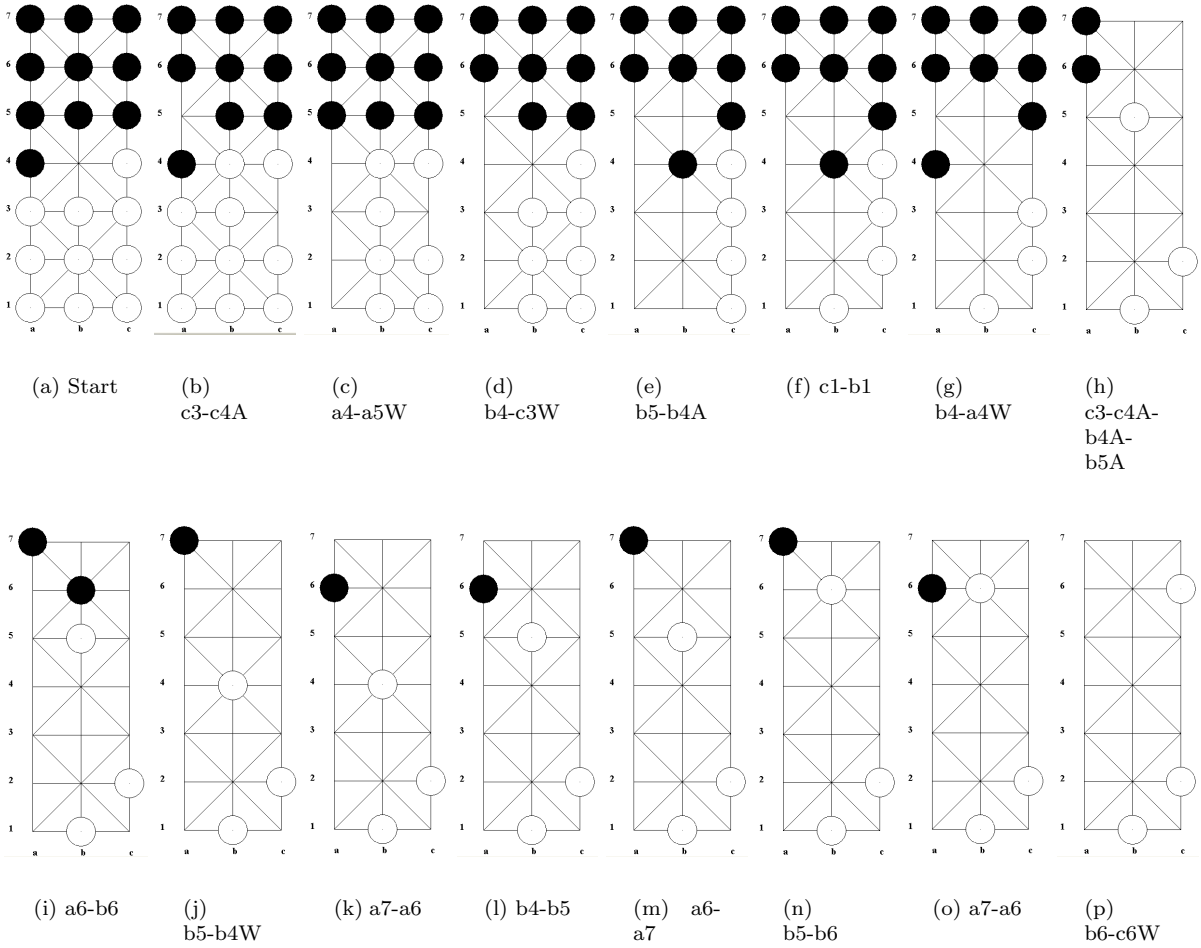
(d) b2-c2W-c1W (e) a3-b2A-c3W (f) b1-b2A-a1W

(g) d3-c3 (h) d2-c1W-c2A (i) f2-e3W

(j) e2-e1W-f2A (k) f3-e3 (l) f2-g1W

Figure B.7: White can force a win on the 7x3 board.

B.7 Optimal play 3×7



(a) Start (b) c3-c4A (c) a4-a5W (d) b4-c3W (e) b5-b4A (f) c1-b1 (g) b4-a4W (h) c3-c4A-b4A-b5A

(i) a6-b6 (j) b5-b4W (k) a7-a6 (l) b4-b5 (m) a6-a7 (n) b5-b6 (o) a7-a6 (p) b6-c6W

Figure B.8: White can force a win on the 3x7 board.