# Chapter 4

# Proof-Number Search with Endgame Databases

This chapter is an updated and abridged version of the following publications:

1. Schadd, M.P.D., Winands, M.H.M., Bergsma, M.H.J., Uiterwijk, J.W. H.M., and Herik, H.J. van den. (2007a). Fanorona is a Draw. *ICGA Journal*, Vol. 30, No. 1, pp. 43–45.
2. Schadd, M.P.D., Winands, M.H.M., Uiterwijk, J.W.H.M., Herik, H.J. van den, and Bergsma, M.H.J. (2007b). Best Play in Fanorona Leads to Draw. *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)* (eds., P. Wang *et al.*), pp. 635–642, World Scientific Publishing Co. Pte. Ltd., Singapore.
3. Schadd, M.P.D., Winands, M.H.M., Uiterwijk, J.W.H.M., Herik, H.J. van den, and Bergsma, M.H.J. (2008d). Best Play in Fanorona Leads to Draw. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 369–387.

This chapter answers the second research question by weakly solving the two-player deterministic perfect-information board game Fanorona. We propose to use a search-based approach that establishes the game-theoretic value of Fanorona. The search-based approach is a combination of a mate-solver (Proof-Number (PN) search) and pre-constructed endgame databases. We examine the tradeoff between time spent on backward search (i.e., creating the endgame databases) and time invested in forward search (i.e., mate-solvers using the created endgame databases). Based on the analysis of the game, solving Fanorona and its variants is possible by a well-chosen combination of PN search and endgame databases. Moreover, the behavior of the PN search is investigated when solving different variants of the game.

The chapter is organized as follows. Section 4.1 discusses the notion of solving games. Next, Section 4.2 provides background information and the rules of the game of Fanorona. Subsequently, Section 4.3 presents various characteristics of the game. Section 4.4 discusses the application of a retrograde analysis in Fanorona.

The proof-number search variant, $PN^2$, is explained in Section 4.5. Then, Section 4.6 presents the results of a well-tuned combination of $PN^2$ search and the corresponding databases. Section 4.7 describes the verification of the results. Finally, Section 4.8 provides the conclusions and gives future research topics.

## 4.1 Solving Games

Already for half a century, building strong game-playing programs is one of the main targets of AI researchers. The principal aim is to witness the "intelligence" of computers. A second aim is to establish the *game-theoretic value* of a game, i.e., the outcome of the game when all participants play optimally (Van den Herik, Uiterwijk, and Van Rijswijk, 2002). The game-theoretic value indicates whether a game is won, lost, or drawn from the perspective of the player who has to move first.

Pursuing the second aim is an exciting task; there, game solvers are looking for new techniques and new achievements in research. For solving a game, three "layered" definitions exist (cf. Allis, 1994).

- **Ultra-weakly solved**: For the initial position, the game-theoretic value has been determined.
- **Weakly solved**: For the initial position, a strategy has been determined to obtain at least the game-theoretic value of the game.
- **Strongly solved**: For all legal positions, a strategy has been determined to obtain at least the game-theoretic value of the position.

In the last 20 years, quite a number of games have been solved (cf. Van den Herik *et al.*, 2002). Below, we provide examples for each definition. (1) The game of Hex is an instance of an ultra-weakly solved game (Anshelevich, 2000). The proof was published by Nash (1952). (2) The following games were weakly solved: Qubic (Patashnik, 1980; Allis and Schoo, 1992), Go-Moku (Allis, 1994; Allis, Van den Herik, and Huntjes, 1996), Nine Men's Morris (Gasser, 1995), Domineering (Breuker, Uiterwijk, and Van den Herik, 2000; Bullock, 2002), and Renju (Wágner and Virág, 2001). Checkers is the latest addition to this list (Schaeffer *et al.*, 2007; Schaeffer, 2007). (3) Recently, we saw considerable development in "layer" three: Kalah (Irving, Donkers, and Uiterwijk, 2000), Awari (Romein and Bal, 2003) and, Connect-Four (Tromp, 2008) were strongly solved.

Different methods can be used for solving games, which we divide into three classes: knowledge methods, special search methods and endgame databases.[1]

(1) Knowledge methods are used to prune the search tree. Allis (1988) applied knowledge rules in Connect-Four that made it possible to determine the game-theoretic value for non-terminal positions. Using these rules the game of Connect-Four was solved.

(2) Special search methods (mate-solvers) are used to search the space efficiently. For instance, Threat-Space Search (Allis, Van den Herik, and Huntjes, 1993) was

[1] It is possible to define different classes. Heule and Rothkrantz (2007) use five classes to distinguish solving methods.

introduced in Go-Moku. This technique searches as if the opponent is allowed to play all counter moves to a threat at once. Using this technique, a series of threat sequences to win the game can be established in such that it is indifferent what the opponent plays. The technique reduces the size of the search tree in such a way that it was possible to solve the game (Allis *et al.*, 1996). Other examples of mate-solvers are Proof-Number Search (Allis *et al.*, 1994) and λ-search (Thomson, 2000).

(3) Endgame databases are used to store the game-theoretic value of each possible endgame position. Retrograde analysis (Ströhlein, 1970) is a method to solve endgame positions iteratively, starting with the terminal ones. For instance, Romein and Bal (2003) were able to solve every possible Awari position using parallel retrograde analysis, thus solving the game strongly.

## 4.2 Fanorona

Fanorona is a board game with its roots in Madagascar (Montgomery, 1886). It is derived from the game "Alquerque", which might be over 3,000 years old. Below, we explain the rules of Fanorona. The explanation is based on the rules given in Bell (1980), and in Chauvicourt and Chauvicourt (1980). There exist some variations on the rules, but we will focus on the most common variant. The goal of the game is to capture all opponent pieces. The game is a draw if neither player succeeds. Fanorona has three standard versions: *Fanoron-Telo*, *Fanoron-Dimyand*, and *Fanoron-Tsivy*. The difference between these versions is the board size. Fanoron-Telo is played on a 3×3 board and the difficulty of this game can be compared to that of Tic-Tac-Toe. Fanoron-Dimyand is played on a 5×5 board and Fanoron-Tsivy is played on a 5×9 board. We call Fanoron-Tsivy from now on *Fanorona*, because it is the best-known board size and the main subject of this chapter.

This section is organized as follows. Subsection 4.2.1 introduces the board on which Fanorona is played. Next, Subsection 4.2.2 explains the movement rules. Finally, Subsection 4.2.3 describes the ending conditions of Fanorona.

### 4.2.1 Board

The Fanorona board consists of lines and intersections. A line represents the path that a piece can move along during the game. There are weak and strong intersections. On a weak intersection, it is only possible to move a piece horizontally and vertically, while on a strong intersection, it is also possible to move a piece diagonally. Figure 4.1 shows a weak intersection at **e2** and a strong intersection at **e3**. A piece can only move from one intersection to an adjacent intersection. In the initial position, each player has 22 pieces. They are placed as shown in Figure 4.1. Players move alternately; White plays first. Figure 4.2 shows two non-standard board sizes, the 5×7 and the 7×5 board. We remark that a game played on the 7×5 board is totally different from a game played on a 5×7 board. Rotating the 7×5 board to a 5×7 board creates a Fanorona game with a different 5×7 initial position (e.g., left Black, right White). It is standard in Fanorona that the white pieces are below the black pieces.

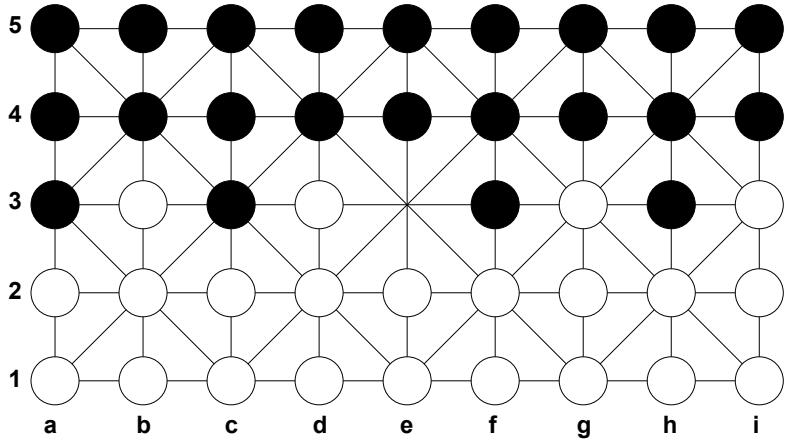
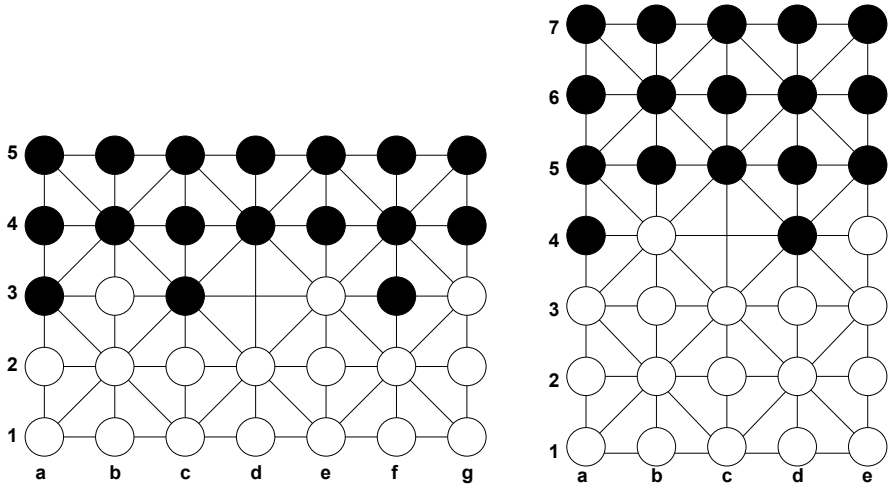



Figure 4.1: The initial position of a Fanorona game.




(a) 5x7: Initial position (b) 7x5: Initial position

Figure 4.2: Initial positions for the 5×7 and 7×5 board.

### 4.2.2 Movement

We distinguish two kinds of moves, capturing and non-capturing moves. Capturing moves are *obliged* and have to be played, if possible, above non-capturing (*paika*) moves. We start explaining capturing moves because thereafter describing paika moves is straightforward.

Capturing implies removing one or more pieces of the opponent. It can be done in two different ways, either (1) by approach or (2) by withdrawal. An *approach* is the movement of the capturing piece to an intersection adjacent to an opponent piece

provided that the opponent piece is situated on the continuation of the capturing piece's movement line. A *withdrawal* works analogously as an approach but the difference is that the movement is away from the opponent piece. When an opponent piece is captured, all opponent pieces in line with that piece and directly behind that piece (i.e., there is no interruption by an empty intersection or an own piece) are captured as well.

Figure 4.3 shows how the capturing mechanism works. We start with a straightforward capturing move. In the given position, White can capture Black's piece on **d2** by moving the white piece from **b2** to **c2**. By this move, Black's piece on **e2** is captured as well. **g2** is not captured because there is no black piece on **f2**. This is called *capturing by approach* because the white piece is moved towards the black piece on **d2**.

White can also *capture by withdrawal* if the white piece is moved from **f4** to **e4**. This is allowed because the white piece is moving away from the black piece on **g4**. The piece on **i4** is not captured because there is a white piece on **h4** interrupting the line.

White cannot capture **c4** with **f4** because for a capturing move the own piece has to be next to the one captured after movement: **f4** is too far away to capture **c4**. In order to allow capturing, the piece has to be moved to an intersection adjacent to the captured piece, if it is approached.

Also, White cannot capture **c4** with **b2** (moving diagonally) because **c4** is not on the extension of a movement line from **b2**. Only pieces can be captured that are located in the extension of the movement line of the capturing piece. Thus, capturing "around a corner" is not allowed.

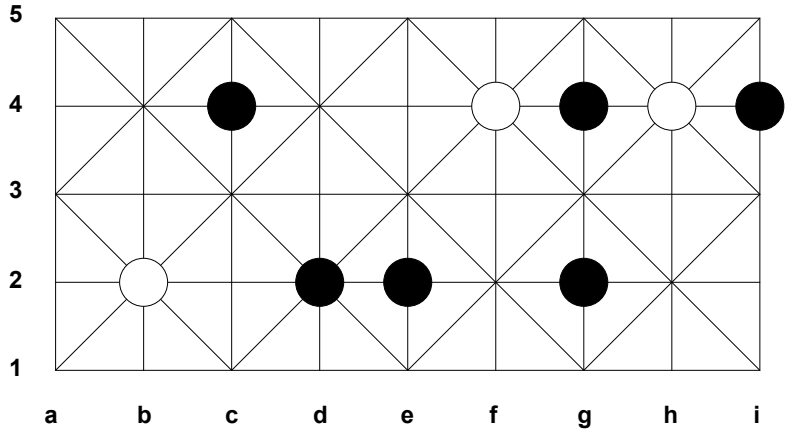



Figure 4.3: An example position during a Fanorona game.

To denote a capturing by approach, we define the following notation: **b2-c2A** meaning that the piece on **b2** moves to **c2** and approaches (A) the piece on **d2**. For a withdrawal, the letter "W" is used. If a piece is moved without capturing any opponent piece, no letter "A" or "W" is used.

As in checkers, it is allowed to continue capturing with the same piece as long as possible. We call this extended capturing. Figure 4.3 shows that White can capture **c4** with the move: **b2-c2A-c3A**. (Even if a move consists of multiple movements of one single piece it still counts as a single move.) Although a player is obliged to prefer a capturing move above a non-capturing move, the player may stop capturing after any number of opponent pieces are captured. This rule is different from the checkers rule where stopping a capturing sequence is not permitted. For instance, in Figure 4.3, White is allowed to stop early and play only **b2-c2A**.

There are three more rules concerning capturing pieces. (1) It is not allowed to capture by approach and withdrawal at the same time. This is the case at the initial position shown in Figure 4.1 where White could play **d3-e3** as an approach or a withdrawal. In such a situation, the player has to choose whether the current move is an approach or a withdrawal. (2) 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 4.3. White is not allowed to play: **f4-e4W-d4A** because the white piece would move twice in a row in the same direction. A player is allowed to play a capturing movement in a previously chosen direction if a capturing movement in another direction is done in between. The last movement direction of the capturing move in the previous turn (i.e., before the last opponent move) does not influence possible capturing directions in the current turn. (3) The current capturing piece is not allowed to arrive on the same intersection twice during a capturing sequence. In Figure 4.3, White is not allowed to play **f4-e4W-e3A-f4W** because of this rule.

If no capturing move can be played, a non-capturing move is allowed. This is called a paika move and consists of moving one piece along a line to an adjacent intersection. White is not allowed to play the paika move **b2-b1** in the position depicted in Figure 4.3 because capturing is possible.

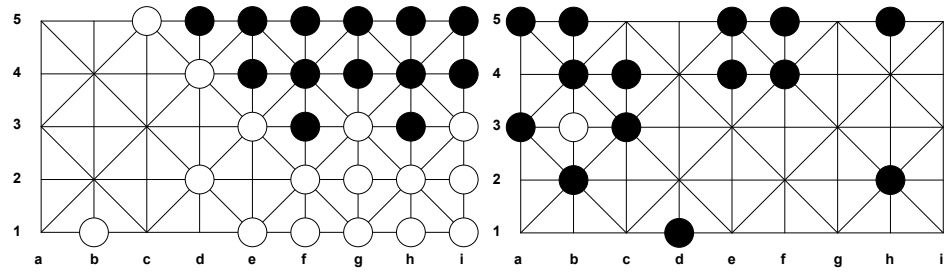
### 4.2.3 End of the Game

The player who first captures all opponent pieces wins the game. The game is a draw if no player is able to win. In practice this is achieved by repetition of the same position with the same player to move (Schadd, 2006).

There does not exist any documentation stating what the outcome of the game would be if a player is not able to move. If this occurs during a game, we define that a player who is not able to move forfeits the game. However, this situation rarely occurs and it is unlikely that the game-theoretic value would change if another outcome would be defined in this situation. This situation was encountered a few times when we solved Fanorona and its smaller variants. Two positions that were encountered during the search are depicted in Figure 4.4.

## 4.3 Analyzing Fanorona

Two important indicators for establishing the questions (a) whether games can be solved and (b) which methods may be successful are (1) the game-tree complexity and (2) the state-space complexity.




(a) No moves for Black (b) No moves for White

Figure 4.4: Two positions without legal moves for one side encountered during the search.

An approximation for the game-tree complexity can be computed by performing a self-play experiment. 10,000 games were played by two $\alpha\beta$ players, which performed a 4-ply deep search with a greedy evaluation function. The evaluation function consists of material difference and a random factor. By using these players, the average branching factor may be approximated accurately. It is still possible however, that the approximated average game length deviates significantly off the real average game length. On the one hand, in games where humans know early that the result is a draw, the $\alpha\beta$ players may shuffle pieces around until a repetition occurs. Only after the first repetition of a position, the game is counted as a draw. On the other hand, the $\alpha\beta$ players may make mistakes which leads to faster capturing of pieces. Therefore, the average game length is somewhat unreliable. We determined that the average game length of Fanorona is 43.81 moves (i.e., plies) and the average branching factor is 11.19 moves. This gives us a game-tree complexity of approximately $10^{46}$. The state-space complexity has been computed as $10^{21}$ by using Formula 4.1 (explained in the next section). These numbers are comparable to those of checkers, which has a game-tree complexity and state-space complexity of $10^{31}$ and $10^{20}$, respectively (Allis, 1994; Schaeffer *et al.*, 2007).

A typical game of Fanorona can be divided into two parts. In the first part of the game, mostly capturing moves are played until most pieces are captured. In the second part, the endgame, mainly paika moves are played. Figure 4.5 shows the ratio of capturing moves and paika moves as a function of the number of pieces on the board. This figure is based on 10,000 games played by $\alpha\beta$ players. In the initial position, 44 pieces are on the board.

Figure 4.5 shows that with 19 or more pieces on the board, a capturing move is to be expected. We define the endgame as the part of the game where more paika than capturing moves are played. Figure 4.5 indicates that the endgame starts when there are fewer than 13 pieces on the board.

Figure 4.6 shows the average branching factor as a function of the number of pieces on the board, based on the same 10,000 games. We see in the figure that when the game enters the endgame the branching factor increases again. The reason for the increase of the average number of possible moves in the endgame is the occurrence of paika moves. The combination of a long endgame and a local maximum of the branching factor results in an immense solution tree.

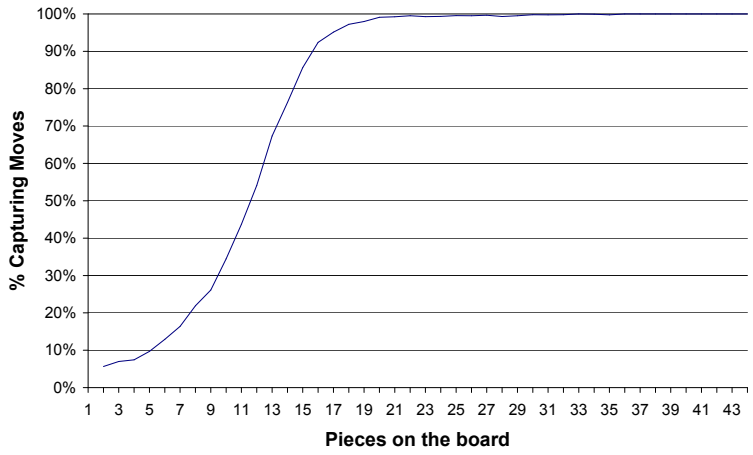



Figure 4.5: Ratio of capturing moves and paika moves as a function of the number of pieces on the board.

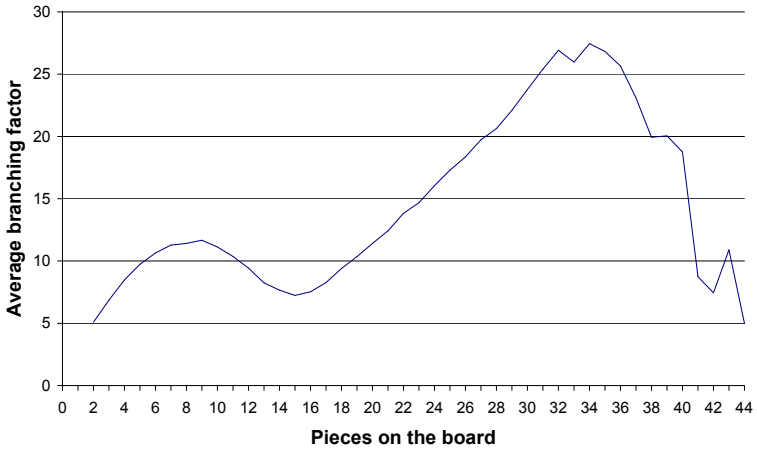



Figure 4.6: The average branching factor as a function of the number of pieces on the board.

Figure 4.7 shows the average number of pieces on the board as a function of the move number. Here we see that the number of pieces decreases fast at the start of the game. After 15 or more moves, the number of pieces only decreases slowly, implying that the endgame has started.

In order to cope well with the difference between both parts, different methods have been selected. For the endgame part, retrograde analysis has been selected to create endgame databases (Ströhlein, 1970; Van den Herik and Herschberg, 1985). There are three reasons, why endgame databases are a valuable tool for solving Fanorona. (1) Fanorona is a converging game (Van den Herik *et al.*, 2002). (2) A large part of the game consists of positions with only a few pieces on the board.

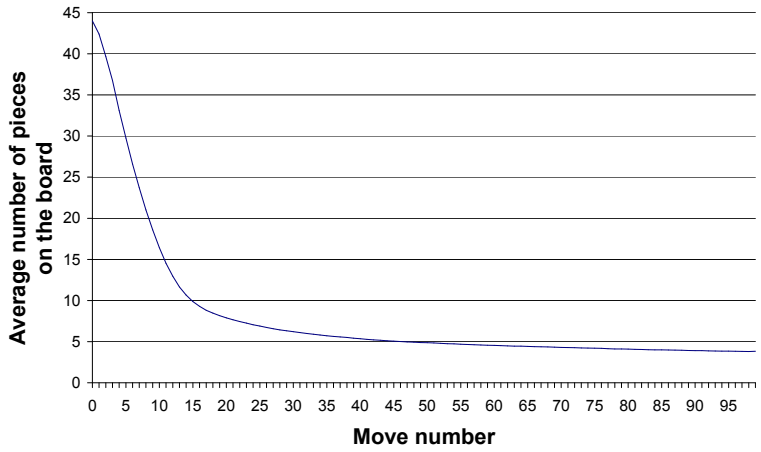



Figure 4.7: The average number of pieces as a function of the move number.

Table 4.1: Estimated complexities with increasing database size.

| Database size | 0 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Average Game Length | 43.81 | 42.60 | 40.23 | 36.23 | 31.62 |
| Average Branching Factor | 11.19 | 11.37 | 11.63 | 11.98 | 12.31 |
| $\text{Log}_{10}$ Game-Tree Complexity | 45.96 | 44.97 | 42.88 | 39.07 | 34.48 |
| Database size | 6 | 7 | 8 | 9 | 10 |
| Average Game Length | 26.99 | 22.98 | 19.74 | 17.27 | 15.44 |
| Average Branching Factor | 12.59 | 12.82 | 13.02 | 13.26 | 13.53 |
| $\text{Log}_{10}$ Game-Tree Complexity | 29.70 | 25.45 | 22.01 | 19.39 | 17.47 |

Fanorona has an average game length of 44 plies. Already after 21 plies there are on average fewer than 8 pieces on the board. (3) The endgame is not trivial. The branching factor has a local maximum in the endgame (see Figure 4.6) and there the game is converging more slowly.

Because of the long endgame where mostly paika moves are played, one would expect that an endgame database would decrease the size of the solution tree substantially. The expectation can be tested by doing a simulation. This was done in the following way.

A *virtual database* was used to finish a game early. For instance, if we assume that a 2-piece database is available, the game is stopped when a position with two pieces is reached. This is a terminal node because the outcome of the game can be retrieved from the virtual database. Using this approach, we can estimate the change of both the average game length and the branching factor as a direct consequence of using larger databases. Table 4.1 shows that the virtual database was able to reduce the game-tree complexity substantially. In this experiment, we see that the game-tree complexity decreases by a factor of more than 100 on average when the database size increases by one piece.

Because of the large number of positions, it is not possible to create an endgame database up to the initial position. Therefore, for the first part of the game, a dedicated search method is required. PN search (Allis *et al.*, 1994) has been chosen because the method is efficient for non-uniform trees. A non-uniform tree can be the result of many forced moves (e.g., capturing in Fanorona). Moreover, the use of an endgame database in the search tree makes the search tree non-uniform.

During the search the most-promising lines in the tree (i.e., lines where relatively the weakest resistance is expected) are examined first because PN search uses a best-first tree traversal to find the solution tree. The combination of endgame databases and the fact that Fanorona converges fast may make PN search an efficient technique for this game. In the next two sections, we describe the two methods, retrograde analysis and PN search, in detail.

## 4.4 Retrograde Analysis

Retrograde analysis is a method to create omniscient endgame databases for games (Ströhlein, 1970; Van den Herik and Herschberg, 1985). Such endgame databases proved to be vital to the strength of computer programs in quite a number of games (Schaeffer, 1997; Heinz, 1999). For instance, in the game of checkers endgame databases have been built for up to 10 pieces remaining on the board (Schaeffer *et al.*, 2003). The more board positions are stored in the endgame database, the earlier the search process can be terminated. Moreover, the method makes a deeper search possible in the middle game. Besides improving the playing strength of a program in the middle game, endgame databases can be used to solve a game as well. For instance, Romein and Bal (2003) solved the game of Awari by storing the complete state space in a database.

A requirement for retrograde analysis is an index function. This function has to be a one-to-one mapping. Making such a function efficient can save a significant amount of space in the database (Dekker, Van den Herik, and Herschberg, 1990; Lake, Schaeffer, and Lu, 1994). The function we used consists of two parts: (1) a function to transform the pieces on the board to a number, independent of the color of a piece and (2) a function to convert the order of black and white pieces to a number. The combined index function identifies uniquely all possible board positions disregarding symmetry. Such an index function is called gap-less. A gap-less function uses each index between the minimum and the maximum index. It is also invertible so that given an index, the corresponding board position can be computed.

Formula 4.1 shows the index function used. We denote the total number of pieces on the board by $M$ and the position of a piece $i$ by $S_i$ (which is a number between 0 and 44). We define $W$ as the total number of white pieces and $W_i$ is the place of white piece $i$ in the sequence of white and black pieces on the board (which is a number between 0 and $M-1$).

$$index = \sum_{i=1}^{M} \binom{S_i}{i} + \binom{45}{M} \times \sum_{i=1}^{W} \binom{W_i}{i} \tag{4.1}$$

Table 4.2: Database sizes up to 15 pieces.

| # of Pieces | # of Positions |
|---|---|
| 2 | 1,980 |
| 3 | 85,140 |
| 4 | 2,085,930 |
| 5 | 36,652,770 |
| 6 | 504,993,720 |
| 7 | 5,717,832,120 |
| 8 | 54,750,511,530 |
| 9 | 451,943,198,850 |
| 10 | 3,260,371,406,292 |
| 11 | 20,768,119,231,860 |
| 12 | 117,743,529,024,030 |
| 13 | 597,920,852,078,550 |
| 14 | 2,733,686,209,314,720 |
| 15 | 11,299,926,066,685,300 |

Table 4.2 shows the size of each database if the above index function is used.[2] Moreover, each position uses 2 bits of space on the hard disk, indicating that the position is a win, draw, or loss. So, for instance, we may state that the computation of the 9-piece database would be feasible on a regular desktop machine, where a total of 119.3 GB of hard disk space would be required.

For speeding up the creation process, paging was implemented. Paging is a technique that stores frequently used parts of the database in memory without writing them to the hard disk (Lake *et al.*, 1994). When data from a page is requested, the information can be retrieved from memory and no hard-disk access is required.

During this research all databases with 7 or fewer pieces were computed for Fanorona. The computation took two to three months and was performed on a regular desktop PC with a Pentium IV 3.0 GHz processor and 256 MB RAM of memory.

Table 4.3 shows the number of wins, draws, and losses in the databases for the 5×9 variant. Symmetric positions have been removed. *a*-*b* denotes that the player to move has *a* pieces and its opponent *b* pieces. Table 4.3 indicates that the player to move has an advantage. One might suspect that a player who is to move and has more pieces than the opponent would win the game. The results are that (1) in the 4-1 database the player to move cannot lose any position, (2) in the 5-1 and 6-1 databases the player to move can win every position. So, an advantage of more than three pieces is required for a forced win.

We have created databases for the smaller Fanorona boards as well. We have computed (1) all endgame databases up to 7 pieces for the 5×7 board, (2) all endgame databases up to 9 pieces for the 5×5 board, and (3) all endgame databases up to 8 pieces for the 3×9 board.

[2]Please note that $\binom{x}{y} = 0$ if $y > x$.

Table 4.3: Number of won, drawn, and lost positions in the databases for the 5×9 board. *a-b* indicates that the player with *a* stones is to move.

| Db. | 1-1 | 2-1 | 1-2 | 3-1 | 2-2 |
|---|---|---|---|---|---|
| Win | 158 | 10,366 | 717 | 149,458 | 127,756 |
| Draw | 334 | 398 | 3,231 | 91 | 79,012 |
| Loss | 26 | 6 | 6,822 | 1 | 17,386 |
| Db. | 1-3 | 4-1 | 3-2 | 2-3 | 1-4 |
| Win | 4,188 | 1,529,142 | 2,711,327 | 774,043 | 19,814 |
| Draw | 15,875 | 12 | 327,836 | 1,252,162 | 88,187 |
| Loss | 129,487 | 0 | 18,297 | 1,031,255 | 1,421,153 |
| Db. | 5-1 | 4-2 | 3-3 | 2-4 | 1-5 |
| Win | 12,223,788 | 30,095,407 | 24,137,779 | 4,180,200 | 81,728 |
| Draw | 0 | 426,350 | 13,955,354 | 7,926,733 | 391,405 |
| Loss | 0 | 32,491 | 2,644,731 | 18,447,315 | 11,750,655 |
| Db. | 6-1 | 5-2 | 4-3 | 3-4 | 2-5 |
| Win | 79,431,164 | 237,393,018 | 344,370,238 | 145,408,435 | 18,659,090 |
| Draw | 0 | 774,868 | 46,020,564 | 170,633,688 | 41,896,491 |
| Loss | 0 | 108,614 | 6,724,158 | 81,072,837 | 177,720,919 |
| Db. | 1-6 | | | | |
| Win | 302,021 | | | | |
| Draw | 1,509,775 | | | | |
| Loss | 77,619,368 | | | | |

## 4.5 Proof-Number Search

This section discusses the search procedure applied for weakly solving the game. The method used is Proof-Number (PN) search; it is briefly described in Subsection 4.5.1. A variant of PN search, called PN$^2$, is explained in Subsection 4.5.2.

### 4.5.1 PN Search

PN search is a best-first search algorithm especially suited for finding the game-theoretic value of a game (Allis *et al.*, 1994). Its aim is to prove the true value of the root of a tree. A tree can have three values: *true*, *false*, or *unknown*. In the case of a forced win, the tree is *proven* and its value is *true*. In the case of a forced loss or draw, the tree is *disproven* and its value is *false*. Otherwise, the value of the tree is *unknown*. In contrast to other best-first algorithms, PN search does not require a domain-dependent heuristic evaluation function to determine the most-promising node to be expanded next. In PN search, this node is usually called the *most-proving* node. PN search selects the most-proving node using two criteria: (1) the shape of the search tree (the branching factor of every internal node) and (2) the values of the leaves. These two criteria enable PN search to treat game trees with a non-uniform branching factor efficiently (Allis *et al.*, 1994). PN search exploits an irregular search tree by using a "least-work-first" strategy. An irregular

tree can be caused by capturing or blocking the opponent. Endgame databases can also influence the shape of the tree. For a number of application domains with a highly irregular tree structure, such as chess mating problems or the games of Qubic and Fanorona, PN search is more efficient than $\alpha\beta$-like algorithms (Plaat, 1996).

### 4.5.2 PN$^2$ Search

A disadvantage of PN search is that the whole search tree has to be stored in memory. Therefore, we use PN$^2$ as an algorithm to reduce memory requirements in PN search (Allis, 1994; Breuker, Uiterwijk, and Van den Herik, 2001b). PN$^2$ consists of two levels of PN search. The first level consists of a PN search ($pn_1$), which calls a PN search at the second level ($pn_2$) for an evaluation of the most-proving node of the pn$_1$-search tree. This pn$_2$ search is bound by a maximum number of nodes $N$ to be stored in memory. In our implementation (analogously to Allis, 1994), $N$ is equal to the size of the $pn_1$ tree. The pn$_2$ search is stopped when the number of nodes stored in memory exceeds $N$ or the subtree is (dis)proven. After completion of the pn$_2$ search, the children of the root of the pn$_2$-search tree are preserved, but subtrees are removed from memory.[3]

The maximum size of the subtree for determining the PN numbers is an important factor for PN$^2$ search. With a larger tree, the search would be more directed but would take more time to compute. Allis (1994) proposed a variable size for the $pn_2$ tree that is dependent on the size of the $pn_1$ tree. Breuker *et al.* (2001b) used a fraction function to determine the size of the $pn_2$ tree. We did not use the latter approach because the fraction function is not efficient for large problems. We set the maximum size of the $pn_2$ tree equal to the size of the $pn_1$ tree.

## 4.6 Experiments and Results

This section presents the results obtained during our research. In Subsection 4.6.1, the game-theoretic values of Fanorona and its smaller board variants are given and the optimal solution for the 3×3 board is presented. Subsection 4.6.2 investigates the tradeoff between backward search and forward search. Finally, Subsection 4.6.3 discusses an interesting observation regarding the behavior of the proof and disproof numbers.

### 4.6.1 Solving Fanorona and its Smaller Variants

PN$^2$ search is used in combination with endgame databases to compute the game-theoretic value of Fanorona and its smaller variants. After some tuning, we arrived at the "ideal" combination for this configuration (i.e., 5×9 board on current hardware) viz. PN$^2$ with all databases including 7 or fewer pieces. By doing so, we were able to prove that Fanorona is a draw. For proving this, a total of 130,820,097,938 nodes were created during the search. Solving the initial position of Fanorona took more than a week when using our search-based approach on a computer with an AMD64

[3]This only holds if the root is not proven or disproven.

Table 4.4: The game-theoretic values for different board sizes.

| *Board Size* | *Winner* | *DB Size (Pieces)* | *Nodes* |
|---|---|---|---|
| 3×3 | White | 0 | 122 |
| 3×5 | White | 0 | 2,490 |
| 5×3 | White | 0 | 1,491 |
| 3×7 | White | 0 | 87,210 |
| 7×3 | White | 0 | 172,101 |
| 5×5 | Draw | 9 | 119,354 |
| 3×9 | White | 8 | 105,343 |
| 9×3 | White | 8 | 3,999,481 |
| 5×7 | Black | 7 | 72,826,963 |
| 7×5 | White | 7 | 1,107,756 |
| 5×9 | Draw | 7 | 130,820,097,938 |

2.6 GHz processor and 32 GB RAM of memory. The search proved that both moves **f2-e3A** and **d3-e3A** lead to a draw. Preliminary results suggest that by optimal play the moves **e2-e3A**, **d3-e3W** and **d2-e3A** will lead to a win for Black.

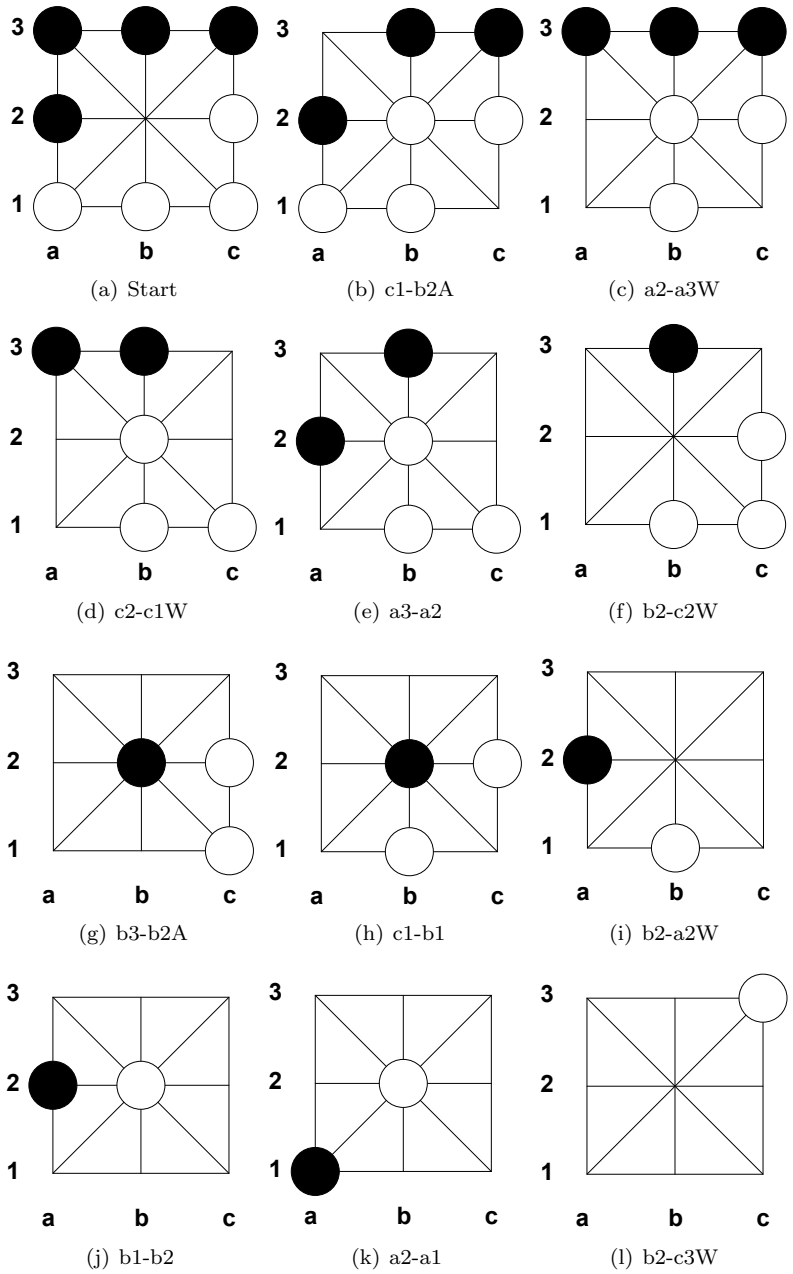
Furthermore, we have solved the smaller variants of the game as well. An overview of the results of Fanorona and all smaller boards is given in Table 4.4. The column labeled *DB size* indicates the maximum number of pieces for the databases used. The column labeled *Nodes* indicates the total number of created nodes.

For smaller boards we remark the following. If we have a look at Table 4.4, we see that all boards with a side equal to size 3 are a win for White. Thus, the starting player can exploit a narrow board and force a win. However, for most boards, with sides of at least size 5, White does not have this advantage anymore.

Table 4.4 shows differences between horizontal and vertical boards. The difference in number of nodes between the 5×7 and 7×5 boards (see Table 4.4 and Figure 4.2) can be explained by the fact that 5×7 is a win for Black (the second player), which is harder to prove. Furthermore, a substantial difference between the 3×9 and 9×3 boards can be observed. We conjecture that the average distance between the players' pieces is larger on vertical boards than on horizontal boards when entering the endgame. This would result in a slower convergence.

In order to provide insight into the strategies behind optimal play we show the solution for the 3×3 board. Figure 4.8 shows the game that is played if both players play optimally.

Below, some interesting positions in this game are discussed. In Subfigure 4.8(b), Black has no choice of movement because there is only one possible capturing move. In Subfigure 4.8(d), Black has two choices. The move **b3-c3** would lead to a loss in 1 because White would answer with **b2-a1W-a2A**. Subfigure 4.8(f) is the most interesting one. Black decides to stop the capturing sequence after the capturing of only one of the two possible pieces. The reason for this is to postpone the eventual loss by one move.




(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 4.8: 3×3 board: White can force a win.

### 4.6.2 Tradeoff between Backward and Forward Search

There exists a tradeoff between time spent on backward search (i.e., creating the endgame databases) and time invested in forward searching (i.e., PN Search using the endgame databases to cut off the search). To demonstrate this, we take the 5×5 variant as example. Table 4.5 shows the time and the number of nodes required to compute different 5×5 databases. The number of positions in the databases and the time to compute the database increases exponentially. This table also shows the amount of time required to solve the 5×5 variant using different endgame databases. Here we see that the effect of an additional database diminishes fast.

Table 4.6 shows the effectiveness of endgame databases for the 7×5 variant. Because the 7×5 board is larger than the 5×5 board, only the databases up to 7 pieces were computed. The effectiveness of endgame databases for the 3×9 variant is presented in Table 4.7. For this variant, all databases with up to 8 pieces were computed. A remarkable phenomenon for this variant is that it is faster to solve with the 3-piece endgame database than with the 4-piece database. The following explanation can be given for this phenomenon. PN search develops the tree into the direction where the weakest resistance is expected. Because the endgame databases has a large influence on the branching factor, it may happen that the endgame database misdirects the PN search. The reason that this variant is solved faster with the 7-piece database than with the 8-piece database is that the problem size has become so small that the hard-drive access becomes the dominant factor.

In order to determine the optimal database size for a Fanorona variant, both the solving time and the time required to compute the endgame databases have to be taken into account. Figure 4.9 shows the total time required for creating the endgame databases and for the PN search for the 5×5, 7×5 and 3×9 variant. For the 5×5 variant the optimal database size is 4 pieces. With fewer pieces, the time

Table 4.5: Effectiveness of endgame databases for 5×5 Fanorona.

| DB size | 0 | 2 | 3 |
|---|---|---|---|
| DB Time(ms) | - | 138 | 1,228 |
| DB Size | - | 600 | 13,800 |
| Solving Time (ms) | 189,919,141 | 31,610,647 | 6,176,044 |
| Solving Nodes | 101,830,539,576 | 18,934,832,800 | 3,546,498,830 |
| DB size | 4 | 5 | 6 |
| DB Time (ms) | 6,547 | 60,927 | 326,104 |
| DB Size | 171,100 | 1,593,900 | 10,980,200 |
| Solving Time (ms) | 9,049 | 2,715 | 1,033 |
| Solving Nodes | 6,323,71 | 1,509,329 | 364,278 |
| DB size | 7 | 8 | 9 |
| DB Time (ms) | 2,104,583 | 8,972,377 | 41,415,589 |
| DB Size | 60,568,200 | 274,720,050 | 1,041,917,250 |
| Solving Time (ms) | 842 | 694 | 577 |
| Solving Nodes | 251,236 | 173,566 | 119,354 |

Table 4.6: Effectiveness of endgame databases for 7×5 Fanorona.

| DB size | 0 | 2 | 3 |
|---|---|---|---|
| DB Time(ms) | - | 141 | 3,062 |
| DB Size | - | 1,190 | 39,270 |
| Solving Time (ms) | 62,371,743 | 29,159,087 | 2,300,224 |
| Solving Nodes | 39,243,004,473 | 20,120,149,985 | 1,685,583,077 |
| DB size | 4 | 5 | 6 |
| DB Time(ms) | 31,350 | 479,801 | 3,858,750 |
| DB Size | 733,040 | 9,738,960 | 100,635,920 |
| Solving Time (ms) | 1,108,075 | 568,325 | 99,164 |
| Solving Nodes | 862,109,429 | 439,220,855 | 72,524,679 |
| DB size | 7 | | |
| DB Time(ms) | 77,294,159 | | |
| DB Size | 847,289,520 | | |
| Solving Time (ms) | 4,132 | | |
| Solving Nodes | 1,107,756 | | |

Table 4.7: Effectiveness of endgame databases for 3×9 Fanorona.

| DB size | 0 | 2 | 3 | 4 |
|---|---|---|---|---|
| DB Time(ms) | - | 129 | 1,836 | 9,629 |
| DB Size | - | 702 | 17,550 | 245,700 |
| Solving Time (ms) | 39,975 | 35,379 | 6,883 | 12,747 |
| Solving Nodes | 33,248,635 | 28,588,852 | 5,885,429 | 10,939,734 |
| DB size | 5 | 6 | 7 | 8 |
| DB Time(ms) | 104,829 | 585,177 | 4,387,274 | 21,854,833 |
| DB Size | 2,421,900 | 18,352,620 | 113,891,780 | 563,899,050 |
| Solving Time (ms) | 2,953 | 1,638 | 997 | 2,159 |
| Solving Nodes | 2,138,924 | 1,055,233 | 233,197 | 105,343 |

required for PN search is the critical factor. With more pieces, the time required for computing the endgame databases is the critical factor. The reason for this is that for larger databases, the majority of the positions in the database are not used during the search. For the 7×5 variant, the optimal database size is 5 pieces, and 3 pieces for the 7×5 variant. These optima are at relatively small databases. We also see that the optima occur when the time required for database construction and the time required for solving are of the same order. Taken into account that for 5×9 Fanorona it took 2 to 3 months to construct the databases and slightly more than a week to solve the game, we suspect that the optimal database size for 5×9 Fanorona is 6 or 7 pieces.

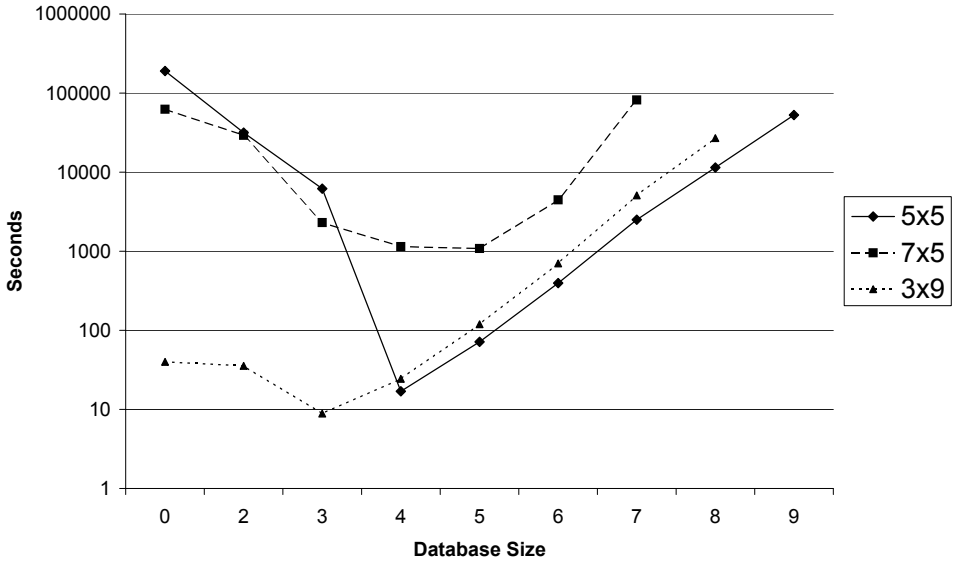



Figure 4.9: Total time required for solving Fanorona variants.

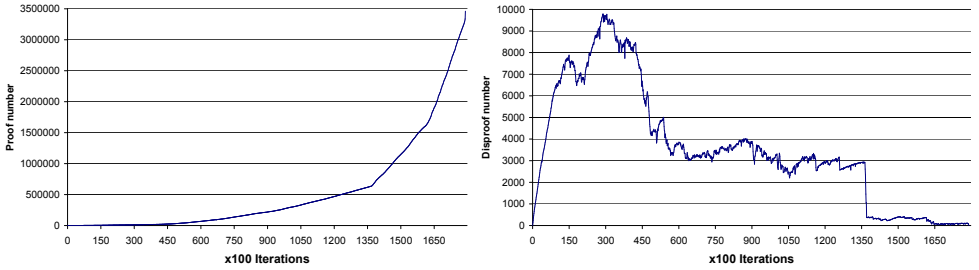
### 4.6.3 Behavior of the PN Search

The behavior of PN search can be expressed by displaying the development of the proof and disproof number at the root node. During this research, we made an interesting observation. Figure 4.10 and Figure 4.11 show the development of proof and disproof number of the root node during the search. Figure 4.10 depicts a search with the goal that White can win the initial position. Figure 4.11 visualizes a search with the goal that White can at least draw the initial position. We found that the development of the proof and disproof number has a similar pattern for smaller boards and looks like a Chi-Square distribution. The number that goes to zero, independent of its nature (proof or disproof number), always reaches its peak in the beginning and has a long tail at the end. A similar pattern can be found when solving the small boards. For instance, this can be seen when solving the 7×3 board (See Figure 4.12). A reason for this may be that Fanorona is not a *sudden-death* game (i.e., all pieces have to be captured to win the game). When proving the win in the sudden-death game Lines of Action, the development of the proof-number is bell-shaped (Winands, 2008).

## 4.7 Correctness

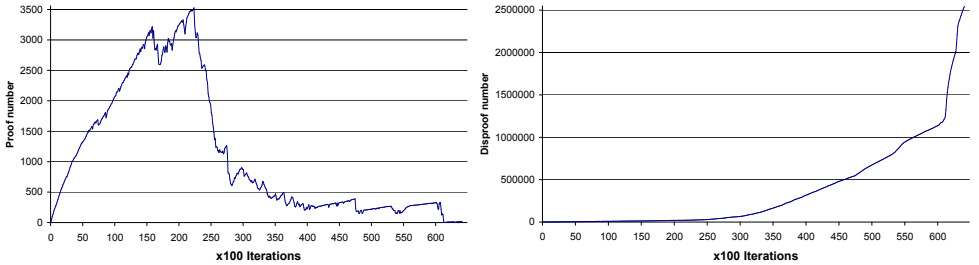
It is important that the obtained results are free of errors. This section gives two arguments why we are convinced that the results presented in this chapter are correct.

First, we verified the code for the retrograde analysis in three ways. (1) We solved



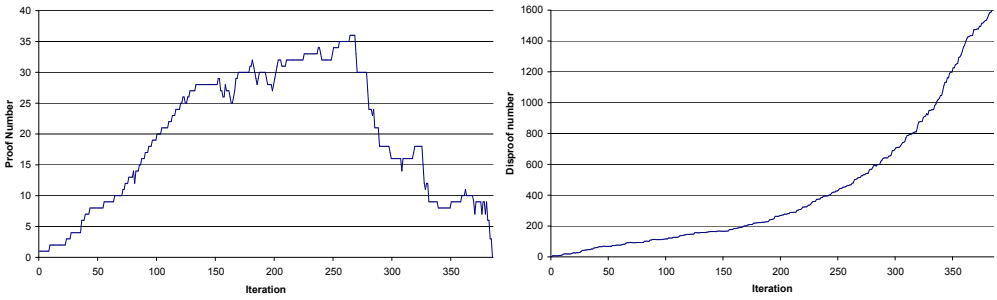

(a) Proof number (b) Disproof number

Figure 4.10: Development of the proof and disproof number when proving that White cannot win the initial 5×9 Fanorona position.




(a) Proof number (b) Disproof number

Figure 4.11: Development of the proof and disproof number when proving that White can at least draw the initial 5×9 Fanorona position.




(a) Proof number (b) Disproof number

Figure 4.12: Development of the proof and disproof number when proving that White can win the initial position on the 7×3 board.

endgame positions with the help of PN search and compared the obtained result with the entry in the database. (2) We did use the database code to construct Lines of Action (LOA) and Surakarta endgame databases, which were verified independently by the programs MIA (Winands, 2004) and SIA (Winands, 2007), respectively. (3) A consistency check was done in order to check for possible bit flips (Schaeffer, 1997). It turns out that no bit flips had occurred.

Second, in order to verify PN search three actions have been taken. (1) The smaller boards were checked manually. This was possible for the 3×3, 5×3, and 3×5 boards. (2) PN search was compared to a standard $\alpha\beta$ search. Non-trivial 5×9 positions were solved by both solvers in order to detect differences. (3) To handle the Graph-History-Interaction (GHI) problem (Palay, 1983; Campbell, 1985), no transposition tables were used during the search. There exist methods that handle the GHI problem with PN search (Breuker *et al.*, 2001a; Kishimoto and Müller, 2004) but these were not implemented.

## 4.8 Chapter Conclusions and Future Research

Our first and main conclusion of this chapter is that the game of Fanorona (played on the 5×9 board) has been weakly solved and is drawn when both players play optimally. This conclusion was arrived through a well-chosen combination of the proof-number (PN) search variant $PN^2$ and endgame databases. Combining these two methods is a relatively new approach for solving a game. Simultaneously to our research, checkers was solved with a similar method (Schaeffer *et al.*, 2007; Schaeffer, 2007). Endgame-database statistics show that (1) the player to move has an advantage and (2) that a draw can often be achieved in spite of having fewer pieces than the opponent. Second, we have seen that the optimal endgame database size for the 3×9, 5×5 and 7×5 are 3, 4 and 5 pieces, respectively. The optimal database size is located at the point where the time required for database construction and the time required for solving are of the same order. From this observation, we conclude that the optimum for Fanorona is 6 or 7 pieces. Third, we may conclude that White is able to force a win on board sizes with one side equal to 3. We conjecture that for boards where both sides have at least size 5, White does not have this advantage for the majority of cases (so, we consider 7×5 as an exception because White still wins). The 9×5 board (please note the inversion) of the game has not been fully weakly solved up to now. Preliminary results suggest that this board is much harder to solve than the 5×9 board. We anticipate that the reason for this is the larger distance to the opponent when entering the endgame (see Subsection 4.6.1).

The game-tree and state-space complexity of Fanorona are somewhat higher than those of checkers. Therefore, the following question may arise: why is this game easier to solve than checkers? The answer lies in the properties of the game (e.g., the decision complexity, Allis, Van den Herik, and Herschberg, 1991). In Fanorona, capturing is almost always possible in the opening and middle game; often a large number of pieces is then captured. Thus, the game converges fast to the endgame, where the endgame databases can take over the best-play procedure. The speed of the game convergence is not represented in the game-tree and state-space complexity.

We are confident that if such a measure would be established, it would be higher for Fanorona than for checkers.

In this chapter, Fanorona has been *weakly* solved. We determined a strategy to achieve the game-theoretic value against any opposition starting from the initial position. Solving Fanorona strongly is a challenging subject for future research. At this moment of time (2010) the 9×5 and all larger boards are unsolved. We believe that for solving these board sizes, larger databases are required. It would be interesting to investigate whether the proof and disproof number show a similar pattern as seen in Subsection 4.6.3 when solving other games.

Finally, the time for solving may be reduced significantly by using Evaluation-Function Based Proof-Number Search (EF-PN) (Winands and Schadd, 2011). EF-PN is a general framework for employing a traditional evaluation function in PN search. The search is directed to branches where the evaluation function indicates a promising situation. For Fanorona, the material on the board may be used as an evaluation function.