Towards optimal play Fanorona

Heriniaina Andry RABOANARY*, Toky Hajatiana RABOANARY†, Julien Amédée RABOANARY‡
Department of Computer Science
Institut Supérieur Polytechnique de Madagascar
Antananarivo – Madagascar
Email: {*andry.raboanary, †toky.raboanary, ‡julien.raboanary}@ispm-edu.com

***Abstract*—When a game can be played at perfect-play level by an Artificial Intelligence engine, it is possible to reach the theoretical value of a position. If the opponent is a "*perfect player*" as well, there is nothing more that could be done. But, when playing against human player, we can try to achieve more than the theoretical value. What we propose here are some techniques, in particular an algorithm named WINIMAX, to increase the win probability of a perfect play engine against a human player.**

Keywords—Fanorona, MT Framework, Endgame Databases, Proof-number search, Monte-Carlo Tree Search.

I. Introduction

FANORONA is the malagasy national board game, whose principles are much more like the checkers game but with its own specifity. In fact, Fanorona is more than just a Game for malagasy people. It has a very large cultural value.

It has been solved in [1] to be a draw, when both players play perfectly. The solving was mostly powered by the proof-number search and endgame databases, which can be used for game play as well.

Now, it is safe to say that most of mid-game positions can be played at a *perfect play* level in tournament conditions just combining these two tools. Even, once the game has been solved, *perfect play* is not that much exciting anymore.

Nevertheless, there is still some improvements that can be done to increase the winning rate against human imperfect players. Moreover, our experiments showed that, when reaching perfect play level, our program tends to "draw" positions that it would have won before.

To show our work, the this paper is then divided into four parts:

- A brief description of Fanorona
- The exisiting algorithms and tools being used
- Perfect play and beyond.
- Experiments and Results

II. A brief Description of FANORONA

Fanorona is a two player, zero-sum and perfect information board game whose complexity has been classified to be between Checkers and Chess.

The following is a very short description of Fanorona, a more detailed description can be found in [2].

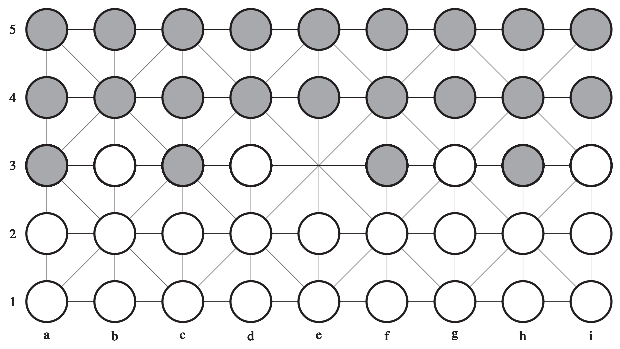

Figure 1. Fanorona Board and initial position

A. The board

The most well-known variant of the FANORONA is played on a 9×5[1] board, named *Akalana* which is illustrated in the Figure 1. It has similarities with *Alquerque*'s board. Each intersection is a spot to place a stone and each line represent movement that can be followed by a stone. There are two different types of intersection: *strong* ones and *weak* ones. An intersection is told *strong* when there are two diagonals which intersect (for example: e3 and h4 in Figure 1).

Each player has 22 stones at the start position as shown in Figure 1 and the goal in the game is to capture all opponent's stones.

B. Game Rules

The two players play moves alternatively. White plays first.

A move is made of one or more movements of a single stone. Each movement consists of moving a stone along the lines from an intersection to another next to it. According to the situation, there might be capture or not. We distinguish two types of moves: *capture moves* and *shuffle moves*.

1) Capture moves: A player can capture his opponent's stone by *approach* or *withdrawal*. When an opponent stone is captured, all the stones in the entire continuous set of adjacent opponent stones on the line are captured as well.

If both withdrawal and approach are available simultaneously, the moving player must choose which kind of capture he wants.

If, after a capture movement, capture is still available for the moving stone, the player may continue to capture with this same stone.

[1]9 columns and 5 rows

While capturing, the moving stone may not:

- return to an intersection that has already been visited during the move;
- be moved successively to the same direction.

When there is a capture move available, the player *must* play a capturing move.

2) Shuffle moves: When no capture moves are available, the player may just move one stone along the line to an adjacent empty intersection. Shuffle moves are also called *paika*.

C. Complexity Analysis

It is important to quantify the game's complexity as it allows us to know how difficult a game is and also to know which techniques are suitable for the game. According to [1], Fanorona has a game-tree complexity of

$$\log(10.3^{32.84}) \approx 40 \tag{1}$$

and a state-space complexity of

$$\log\left(\sum_{w=0}^{22}\sum_{b=0}^{22}\binom{45}{w}\times\binom{45-w}{b}\right) \approx 21.46 \tag{2}$$

It comes out that Fanorona is more difficult than *checkers* and *awari* and is easier than *chess* and *go* [3].

III. The existing algorithms and tools being used

A. The opening book

The opening books are a well known technique for board games. [4]

In the opening book, we store some positions at the start of the game with the best moves for each of them. Ours have more than 100 positions. It took several months to compute it.

In our case the opening book is computed by verifying (using our AI engine) the opening proposed by past and present human experts and computing the best move for non documented opening situations.

B. The proof-number search algorithm

The proof-number search algorithm was first initiated by Victor Allis in [5]. It is a general algorithm that is able to determine if a given goal can be achieved. It is a best-first algorithm that needs all the nodes to be stored in memory.

More precisely, we mainly use the PN^2 variant as it needs much less memory than the standard PN.

The pn-search has proved its efficiency on Fanorona [1] as it takes advantage of the variable branching factor of Fanorona.

C. Rote learning

The main principle of rote learning is to memorize encountered results to save time for later occurrences by just retrieving stored results in order to increase perfect-play performance. [4]

The program's performance increases as it gains experience.

We store in the hard disk all encountered and *solved* positions, indexing and paging techniques are implemented to increase the performance of storage and retrieval.

D. Endgame databases

Since Fanorona is a converging game, endgame databases are an extremely powerful tool to increase the game-play performance, as it is for chess and checkers. One big part in the implementation of the endgame databases is the optimization of the retrieval and the storage.

We use 7-piece endgame databases, which are used in [1] and [6]. They were computed by retrograde analysis.

E. The MINIMAX Algorithm

The minimax algorithm is the basic approach to create the Artificial Intelligence Engine for any Zero-Sum, perfect Information, two players games. The most popular variant is the Alpha-Beta pruning [7].

In practice, we do not use the simple MINIMAX, but it is used in this paper to ease the comprehension of our methods.

F. The MT Framework

The Memory Test (MT) framework [8] is a high level game-tree search algorithm which is based on null-window searches and transposition tables. It gives a best-first behavior of the depth-first, classical Alpha-Beta algorithm.

G. Monte-Carlo Tree Search

Monte-Carlo is a flexible stochastic evaluation for game-tree search [9]. It is an algorithm about game tree search that is based on *pseudo-random* repeated simulations of a full-game. A simulation is made of four main steps:

- *Selection:* where a given node is chosen according to its exploitation and exploration factors. Selection is repeated along the tree until the leaf is reached.
- *Expansion:* on the leaf, one node is added.
- *Simulation:* from the new created node, simulate a full game, by hypothetical *pseudo-random* moves (or actions) until the end of the game.
- *Back-propagation:* Update all the values on the tree according to the outcome of the simulated game

It is an important tool that we use to accelerate the endgame once a wining position achieved.

IV. Perfect play and Beyond

A. The perfect play engine

When we encounter a position not in the *opening book*, we use proof-number search to find out if a quick-win could be achieved. If not, proof-number search is applied once again to determine which move is needed to keep the draw. In either case, the move leading to the most-proving node is played.

All encountered nodes that have been solved are stored in a database as described in III-C to be recalled later during the search. Moreover, we use the endgame databases (III-D) to retrieve the theoretical value of any situation with 7 pieces or less.

B. Accelerating the convergence

Once a winning position achieved, we arrive quickly in endgame because of the forced capture rule. Once in endgame, a problem appears. The computer plays mostly aimless moves when reaching a winning (even draw) endgame position. In fact, the endgame databases that we use don't contain informations concerning the distance to mate because of storage issues.

For winning positions, the only matter is to have a quick win. The tree search in that case is aimed to find a quick path to mate, not necessarily the shortest, but a quick one. We use the Monte-Carlo Tree Search as engine in this case, with an evaluation function based on the two metrics below. We note that it is the extreme priority to keep the winning value, so in the .

Two main metrics are used to evaluate whether a given position is near or far the mate:

- *the number of stones on the board:* the less stone we have on the board, the more we are close to the actual end of the game.
- *the distance between stones:* when stones are close to themselves, they are more likely to capture each other and reduce the stones on the board.

The distance here is the length of the shortest path from a stone to another.

The Manhattan distance is an upper bound of the distance between two stones. A precise evaluation of this distance is given when considering the diagonals.

If we suppose that A and B are two stones with the respective coordinates (x_A, y_A) and (x_B, y_B) the distance between A and B is given by the algorithm on Figure 2.

C. Increasing the win probability

1) The problem: In this subsection, we assume that the current position's theoretical value is *draw*.

All search techniques used in IV-A are based on the hypothesis that the opponent is a perfect player as well. As risen in [10], perfect computing is not necessarily the best way to play against human opponents.

If we play that way, we loose any ambition to get more than the theoretical value of the current position.

```
**function** DISTANCE($x_A$,$y_A$,$x_B$,$y_B$)
    $d_x \leftarrow \mid x_A - x_B \mid$
    $d_y \leftarrow \mid y_A - y_B \mid$
    $\delta_{min} \leftarrow \min(d_x, d_y)$
    $\delta_{max} \leftarrow \max(d_x, d_y)$
    **if** $\delta_{max} = \delta_{min}$ **then return** $\delta_{max} + (x_A + y_A) \bmod 2$
    **else return** $\delta_{max}$
    **end if**
**end function**
```

Figure 2. Algorithm computing the distance between two stones $A(x_A, y_A)$ and $B(x_B, y_B)$

2) Our Solution Proposal: In fact, even with the approach described in IV-A win is possible, when the opponent makes a mistake. The idea here is to lead the human opponent to a position where it is very difficult for him to keep the draw [11].

We assume that we don't have any information about how the human opponent would play, we will admit that all moves would have equal probabilities to be played.

If it is the human player's turn to play on a given position and there is N possible moves. Among these N moves, let us admit that there are n_{draw} moves leading to draw and n_{win} moves leading to win (*for us*[2]). The probability for the human player to choose a loosing move is given by

$$p_{win} = \frac{n_{win}}{N} \tag{3}$$

.

The goal here is to lead the human player into a situation with a very high value of p_{win}, in other words, we are going to *maximize* p_{win} and the opponent wants to *minimize* it (on purpose, or not).

For that, we use a minimax search algorithm (more precisely the MT framework) with an evaluation function calculating the value of p_{win}, which is illustrated on Figure 4 where square nodes represent computer's turn to play and circular ones, the opponent's turn to play.

The algorithm, named WINIMAX is described in the Figure 3 and illustrated in the Figure 4. In practice, we do not implement it as presented here, but the algorithm is shown in its MINIMAX form to ease the comprehension.

In addition, several techniques are combined in the ISWin function, such as lookup in both databases (rote learning databases and endgame databases) and the proof-number search when the position encountered during the search.

V. Experiments and Results

A. The experiments

To quantify the quality of our algorithm, we measure both the effectiveness and the efficiency. In all our experiments, two versions of the program are being compared :

- the one implemented without WINIMAX
- the one with WINIMAX included

[2]the AI engine

```
function WINIMAX($nd$,$d$,$ismax$)
    if $d = 0$ or TERM($nd$) then
        if $\neg ismax$ and TERM($nd$) then
            return EVAL($nd$,$ismax$)
        else
            return WINIMAX($nd$,1,FALSE)
        end if
    else
        if $ismax$ then
            $val \leftarrow -\infty$
            for all child $c$ of $nd$ do
                $val \leftarrow \max(val$,WINIMAX($c$,$d-1$,FALSE))
            end for
        else
            $val \leftarrow +\infty$
            for all child $c$ of $nd$ do
                $val \leftarrow \min(val$,WINIMAX($c$,$d-1$,TRUE))
            end for
        end if
        return $val$
    end if
end function
function EVAL($nd$,$ismax$)
    if TERM($nd$) then
        if ISWIN($nd$,$ismax$) then
            return $1.0$
        else
            if ISWIN($nd$,$\neg ismax$) then
                return $-\infty$
            else
                return $0.0$
            end if
        end if
    else
        $N \leftarrow 0$
        $n_{win} \leftarrow 0$
        for all child $c$ of $nd$ do
            $N \leftarrow N + 1$
            if ISWIN($c$,$ismax$) then
                $n_{win} \leftarrow n_{win} + 1$
            else
                if ISWIN($c$,$\neg ismax$) then
                    return $-\infty$
                end if
            end if
        end for
        return $\frac{n_{win}}{N}$
    end if
end function
function ISWIN($node$,$maximizing$)
    returns TRUE if $node$ is a win for $maximizing$
end function
```

Figure 3. Algorithm trying to maximize the win probability based on MINIMAX and the computing of p_{win}

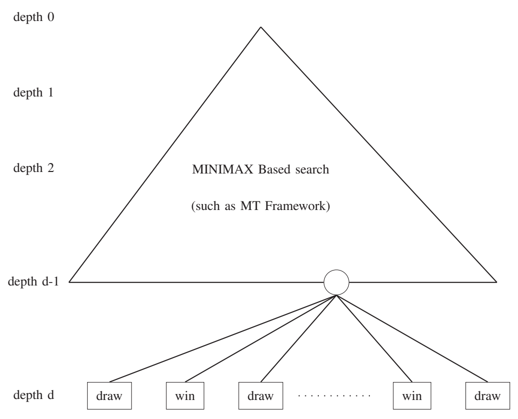

Figure 4. a schematic view of the WINIMAX algorithm

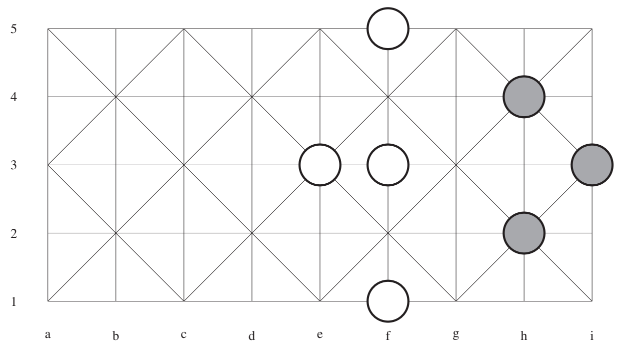

Figure 5. the 4vs3 position

We will measure the effectiveness of our algorithm with the winning rate given by $W_{rate} = \frac{n_{win}}{n_{games}}$. and the efficiency by the overhead due to WINIMAX. We ran two experiments, then :

- 100 games without using the WINIMAX against 25 different human players
- 100 games using the WINIMAX against 25 different human players

We ran those two experiments for two different test positions :

- the starting position
- the classic 4vs3 drawn position shown in Figure 5

Each player plays eight games(four against each AI engine), then. Two of them playing white(one for each position), and two others, playing black(one for each position).

We used the JAVA programming language, and ran the program on a republic of gamers G750JH, with a core i7-4700HQ processor and 16 gigabytes of RAM. The databases (endgame and rote learning) are stored in the included solid state drive.

The 25 human players have different levels, ranging from beginner to advanced.

position	without winimax	with winimax	increase rate
starting	22	47	1.1367
4vs3	11	16	0.4545
total	33	63	0.9091

Table I. THE NUMBER OF WINS WITH AND WITHOUT WINIMAX

position	without winimax	with winimax	overhead
starting	126	167	0.3253
4vs3	24	31	0.2917

Table II. THE OVERHEAD IN THINKING TIME DUE TO WINIMAX

B. The results and discussions

The Table I gives the number of wins (out of 100 games) during the experiments; whereas the Table II gives the (rounded) average thinking time (in seconds) per move during the experiments and the average overhead due to WINIMAX.

As we can see in Table I, WINIMAX has a positive impact on the number of wins, as there is nearly twice more won positions (0.9091 increase rate, in average). However, in the 4vs3 position, we see much less impact from the WINIMAX, it might be because of the 4vs3 is a classic well known position that is easy to keep drawn.

Indeed, in the 4vs3 position, the AI engine can only win if the player does not know the position or he takes some risks trying to "trap" the computer and gets "trapped" himself.

Moreover, results show that there might be some improvements that can be done, because we only have 31.5% of won positions, in average, despite using WINIMAX.

We can also see from Table II that there is some overhead related to the WINIMAX. In the 4vs3 position (both with and without WINIMAX), the average thinking time is so much less than in the full-length game, it is due to the ability to use the endgame databases from this position and any of the possible encountered positions in this game.

VI. Conclusion

Combining different techniques, we created a more-than perfect play engine, in a sense that, it uses successfully some techniques to achieve more than the theoretical value.

We can say that we made a step forward towards the optimal play since the WINIMAX algorithm proposed here is proven to be effective to achieve more wins. To obtain better results, we can try to model the opponent [12]–[14] such that we override the "equiprobability" assumption in IV-C2.

Regarding the performance, our engine is close to tournament conditions (1 minute per move) and a few positions still weren't played "perfectly" due to time limitations in our current experiments but were fixed afterwards. The evolution will take "automatically" place with time as more solved positions will be stored. Some fine tuning and performance improvements can be done to the WINIMAX as well. Increasing the size of the endgame database and/or the opening book.

Acknowledgment

We give thanks to ISPM (Institut Supérieur Polytechnique de Madagascar) for financially supporting our researches. We would like to thank as well Dr. Maarten Schadd for online discussions about Artificial Intelligence, Fanorona and machine learning.

References

[1] M. P. D. Schadd, M. H. M. Winands, J. W. H. M. Uiterwijk, H. van den Herik, and M. H. J. Bergsma, "Best Play in Fanorona leads to Draw," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 369–387, 2008.

[2] M. P. D. Schadd, "*Solving Fanorona*," Master's thesis, Universiteit Maastricht, Maastricht, The Netherlands, 2006.

[3] M. Heule and L. Rothkrantz, "Solving games: Dependence of applicable solving procedures," *Science of Computer Programming*, vol. 67, no. 1, pp. 105 – 124, 2007, special Issue on Aspects of Game Programming. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642307000573

[4] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM JOURNAL OF RESEARCH AND DEVELOPMENT*, pp. 71–105, 1959.

[5] L. V. Allis, "Searching for solutions in games and artificial intelligence," Ph.D. dissertation, University of Limburg, Maasricht, The Netherlands, 1994.

[6] H. RABOANARY, J. RABOANARY, and T. RABOANARY, "Mining human-friendly knowledge from endgame databases with a genetic algorithm," *INDIAN OCEAN REVIEW OF SCIENCE AND TECHNOLOGY, ISSN: 2312-1874*, 2014.

[7] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," 1975.

[8] E. Universiteit, R. Op, G. V. De, R. Magnificus, P. D. J. Schaeffer, P. D. I. R. Dekker, A. Plaat, and A. Plaat, "Research - re: Search & re-search," 1996.

[9] I. S. Guillaume Chaslot, Sander Bakkes and P. Spronck, "Monte-carlo tree search: A new framework for game ai," Department of Knowledge Engineering, Maastricht University, Tech. Rep., 2008.

[10] E. B. Baum and W. D. Smith, "Best play for imperfect players and game tree search; part i - theory," Tech. Rep., 1995.

[11] H. A. Raboanary, "Playing fanorona beyond perfect-play level using persuasive dialogue," in *Proceedings of African Conference on Software Engineering and Applied Computing - Doctoral Symposium*. IEEE, ISBN 978-0-620-51684-6, 2011.

[12] D. Carmel, D. Carmel, S. Markovitch, and S. Markovitch, "Learning models of opponent's strategy in game playing," in *In Proceedings of the AAAI Fall Symposium on Games: Planning and Learning*. The AAAI Press, 1993, pp. 140–147.

[13] D. Carmel and S. Markovitch, "Incorporating opponent models into adversary search," in *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*. AAAI, 1996, pp. 120–125.

[14] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron, "Improved opponent modeling in poker." AAAI Press, 2000, pp. 493–499.