

# Using a genetic algorithm for mining patterns from Endgame Databases

Heriniaina Andry RABOANARY  
 Department of Computer Science  
 Institut Supérieur Polytechnique  
 de Madagascar  
 Antananarivo – Madagascar  
 Email: andry.raboanary@gmail.com

Julien Amédée RABOANARY  
 Department of Computer Science  
 Institut Supérieur Polytechnique  
 de Madagascar  
 Antananarivo – Madagascar  
 Email: julien.raboanary@gmail.com

Toky Hajatiana RABOANARY  
 Department of Computer Science  
 Institut Supérieur Polytechnique  
 de Madagascar  
 Antananarivo – Madagascar  
 Email: toky.raboanary@gmail.com

**Abstract**—Tic-tac-toe and Fanorona games are ones that can be played at an extremely high level by computer engines. Endgame databases are powerful tools to create these engines. These Endgame databases contain precious informations about how to play the game. Unfortunately, it is quite impossible for human players to learn the game strategies from endgame databases, which are only made of raw sequences of bits. The work presented here is the use of genetic algorithms to extract human friendly knowledge from these databases.

**Index Terms**—Board games, Fanorona, tic-tac-toe, Endgame Databases, genetic algorithms, Knowledge Discovery.

## I. INTRODUCTION

Solving games and creating high level artificial intelligence engines is an exciting, and challenging task for any game oriented artificial intelligence researcher [1]–[3]. Many board games are now at least weakly solved and played by computer engines, at least, at a world-championship level. These engines contain then the power and the knowledge to play these games almost perfectly, at least at the mid-game and the endgame.

But, it is still quite impossible for human beings, even programmers, to get any knowledge from an artificial intelligence engine. Some of the computer AI<sup>1</sup> strengths such as computing speed, perfect memory could never be acquired by human players. Mostly, in any game AI engine, the part containing the most substantial knowledge is the Endgame Database.

An Endgame Database is a list of any possible positions for a given set of pieces on the board<sup>2</sup> paired with the theoretical outcome of each positions. That is to say, whether a given position is a win, a loss or a draw, assuming perfect play from both players. Endgame Databases are most of the time created by retrograde analysis.

We present here a method of extracting useful knowledge from Databases and Endgame datasets. Our main goal is to extract knowledge for human players to learn *Fanorona*. Our work can be divided into five parts:

- Description of the two games
- Description of the mining process
- Experiments with *tic-tac-toe*
- Experiments with *Fanorona*

<sup>1</sup>Artificial Intelligence

<sup>2</sup>for example 3 white stones against 2 black stones

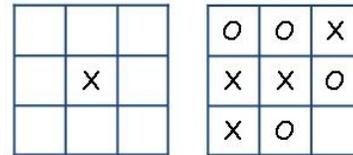


Figure 1. tic-tac-toe sample positions

- General discussions and future research

## II. TIC-TAC-TOE IN A FEW WORDS

Tic-tac-toe is a simple game played by two players: X and O. The board is represented by the figure 1. The goal is to align tree of your own Symbol X or O. Because it is a well-known game, there is no need to develop too much this section.

### A. Rules

Usually, the player X plays first. The two players play alternately, the first player capable of aligning tree of his own symbol wins the game. If such alignment is impossible for both players, the game ends into a draw.

### B. Complexity Analysis

We are now going to perform a complexity analysis of tic-tac-toe game, which quantifies “how difficult it is to play tic-tac-toe”.

1) *Number of moves - Game tree complexity*: There are only 9 ply moves possibles at most, because the board is full afterwards. At the opening, the player X has 9 possibilities; the player O has 8 possibilities; after what X has 7 possibilities left, and so on. The game tree complexity of the tic-tac-toe is then

$$\log(9 \times 8 \times 7 \times \dots \times 2 \times 1) = \log(9!) \approx 5.56 \quad (1)$$

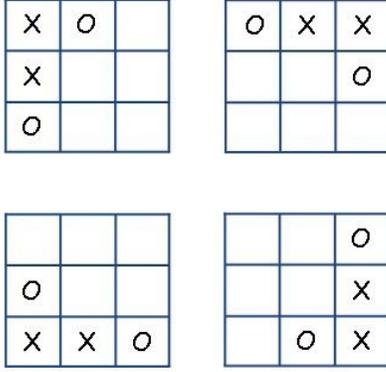


Figure 2. tic-tac-toe rotations-equivalent states

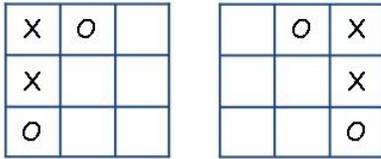


Figure 3. tic-tac-toe Symmetric states

2) *State-space complexity*: Each spot on the tic-tac-toe board can be *blank*, *X* or *O*. Thus, there might be  $3^9$  states possible in the game. But, this is only an upper bound because illegal positions and unreachable positions (such as *X* at every spot) are taken account. Moreover, rotations (figure 2) and symmetries (figure 3) yield many equivalent states. There are thus 765 unique states in tic-tac-toe according to [4]. We have, then, the state-space complexity of

$$\log(765) \approx 2.88 \quad (2)$$

### III. A SHORT DESCRIPTION OF FANORONA

Fanorona is a two player, zero-sum and perfect information board game which 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 [5].

#### A. The board

The most well-known variant of the FANORONA is played on a  $9 \times 5^3$  board, named *Akalana* which is illustrated in the figure 4. 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.

Each player has 22 stones at the start position as shown in figure 4 and the goal in the game is to capture all opponent's stones.

<sup>3</sup>9 columns and 5 rows

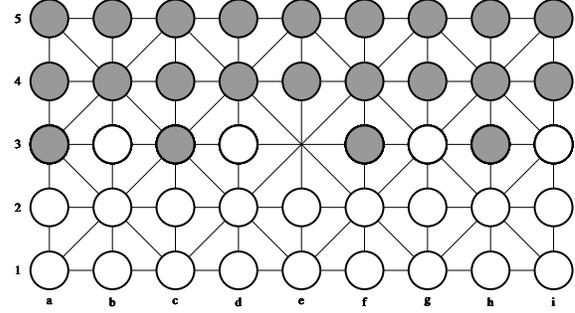


Figure 4. Fanorona Board and initial position

#### B. Game Rules

The two players 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.

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 is a game and also to know which techniques are suitable for the game. According to [6], Fanorona has a game-tree complexity of

$$\log(10.3^{32.84}) \approx 40 \quad (3)$$

and a state-space complexity of

$$\log \left( \sum_{w=0}^{22} \sum_{b=0}^{22} \binom{45}{w} \times \binom{45-w}{b} \right) \approx 21.46 \quad (4)$$

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

#### IV. DESCRIPTION OF THE MINING PROCESS

##### A. Comparing tic-tac-toe and Fanorona

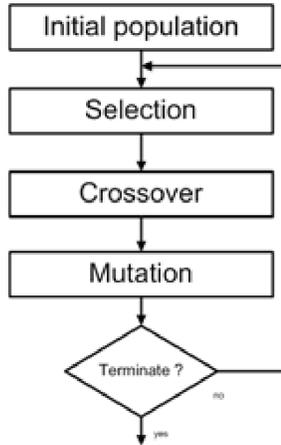
The following table gives us an overview of the complexity of *Fanorona* and *tic-tac-toe* compared to each other, and with other games<sup>4</sup> :

Game	Game-tree	State-space
tic-tac-toe	5	3
checkers	31	18
fanorona	40	21
chess	123	47

It comes out that *Fanorona* is much more difficult than *tic-tac-toe*.

##### B. What is a genetic algorithm ? [7]

A genetic algorithm is a search heuristic based on the process of natural evolution. It is routinely used to generate useful solutions to optimization and search problems. Techniques used here are inspired by natural evolution, such as *inheritance*, *mutation*, *selection*, and *crossover*. The basic process is as follows :



The genetic algorithm process

##### C. Use of JGAP [8]

*JGAP* or *Java Genetic Algorithm Package* is an open source genetic algorithm framework. It provides basic genetic mechanisms that can be easily used to apply evolutionary principles to problem solutions. *JGAP* is a powerful package with many kinds of genetic *operators* included, but it provides as well the possibility to create customized *genes*, *operator*, *natural selector*, ...

##### D. Description of Genes

1) *Binary Arrays*: A gene in our program is made of a low level binary array, which is represented by a *long integer* variable, which can be manipulated with bitwise operations. The size of the binary array is equal to the number of spots in the board.

<sup>4</sup>these are rounded values

- For the *tic-tac-toe* we use an array of 9 bits,
- For the *Fanorona* we use an array of 45 bits.

2) *First configuration*: For our first configuration, we have one gene which is a *binary array* like described in IV-D1. This *gene* indicates the suitable spots for our player.

3) *Second configuration*: For our second configuration, we have two genes, where the first gene is just like described in IV-D2. The second gene is a binary array just like described in IV-D1, which indicates the positions taken by the opponent.

##### E. Description of Operators

1) *The Crossover*: The *crossover operator* that we used is an operator that operates at the bit level, the *swapped* bits are selected randomly. For example, in the following figure, the bits 1 and 3 are *swapped*.

	1	2	3	4	5
Chrom A	0	1	1	0	1
Chrom B	1	1	0	1	1

Chromosomes A and B before crossover

	1	2	3	4	5
Chrom A	1	1	0	0	1
Chrom B	0	1	1	1	1

Chromosomes A and B after crossover

2) *The Mutation*: The *mutation operator* that we used is an operator that operates at the bit level, the *flipped* bits are selected randomly. For example, in the following figure, the bit 4 is changed from 0 to 1 because of the mutation.

	1	2	3	4	5
Chrom A	0	1	1	0	1

↓

	1	2	3	4	5
Chrom A	0	1	1	1	1

Mutation effect over Chromosome A

##### F. Fitness function

The base of our study is to find the set of spots that carries the most information. Our fitness function is then based on the *entropy* function and the *infogain* from the ID3 decision-tree induction algorithm [9]. The information entropy  $E$  of a set  $S$  is given by

$$E(S) = - \sum_{j=1}^n f_s(j) \log_2 f_s(j) \quad (5)$$

where :

- $n$  is the number of different values of the expected outcomes<sup>5</sup> in  $S$
- $f_S(j)$  is the frequency (proportion) of the value  $j$  in the set  $S$ .

### G. Hardware and software specifications

We used personal computer equipped with an Intel Core 2 Quad Q9400 2.8Ghz, with 4GB of RAM. The program is written in JAVA using the Netbeans IDE.

## V. EXPERIMENTS WITH THE TIC-TAC-TOE GAME

As we saw earlier, the tic-tac-toe is an extremely easy game to play. Therefore, there is no need to extract knowledge from any database to learn to play well the game. In fact, our interest in this game is the fact that we can validate and tune the process. Indeed, as human players knowing how to play the game, it is easy for us to validate the output patterns. In that sense, the tic-tac-toe game is used to verify and tune up our learning process.

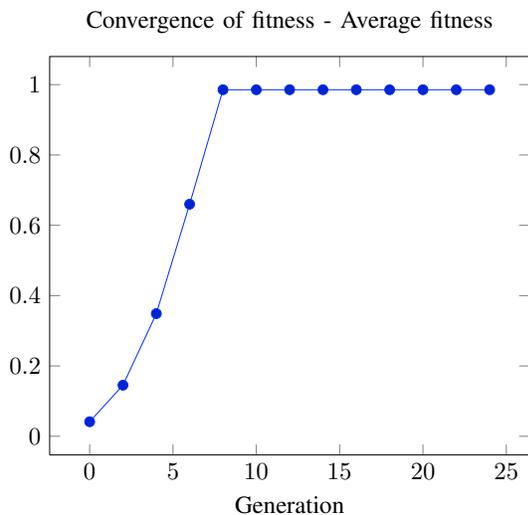
### A. The Dataset

The dataset used for our training came from the UCI repository [10]. It contains 958 instances of data. Each instance is made of 9 attributes corresponding to every position in the tic-tac-toe board; and the class, which has two values : *positive* or *negative* which indicates the theoretical outcome of the board state. *Positive* means win for X.

positive (win for X)	65,3
negative (else)	34,7

### B. Results for the first configuration

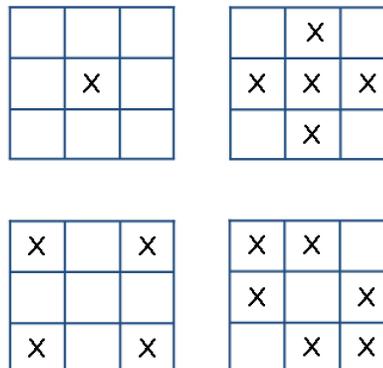
1) *Convergence speed*: The plotting below represents the convergence speed of the average fitness value. The population size used is 1000.



<sup>5</sup>in our case :  $n=2$  (positive, negative) or  $n=3$  (win, loss, draw)

We can conclude that the convergence was extremely fast. This is due to the fact that the state space of the problem is extremely small.

2) *Resulting patterns*: We give in the figure below the four best patterns mined from the dataset. The 'X's indicate the important set of spots that we must consider.



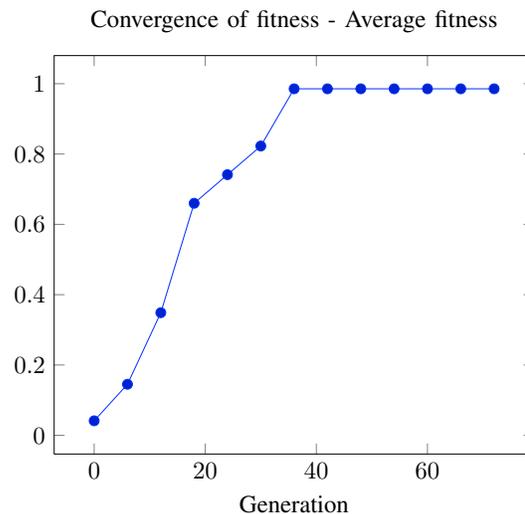
Best patterns throughout generations

We can conclude that we have excellent results. Indeed, each pattern gives us immediately ideas subsequent strategy to apply for achieving easily a win.

### C. Results for the second configuration

In the first configuration, the patterns concern only our positions. But, in the second configuration, a pattern concerns both our positions and our opponent's.

1) *Convergence speed*: The convergence speed is similar than in the first configuration, but slightly slower, as shown in the following:



2) *Resulting patterns*: We give in the figure below the four best patterns mined from the dataset. The 'X's indicate the important set of spots that we must consider when the opponent occupies the 'O's.

X		
O	X	
O	X	X

O	X	
X	X	X
	X	O

O		X
	O	X
X	X	X

X	X	
X	O	X
	X	X

Best patterns throughout generations

While patterns presented in V-B2 are extremely clear and straightforward, the patterns obtained here are not as clear. They may be correct in a mathematical point of view, but, they aren't much useful for learning. Some improvements might be necessary to create a more human-friendly representation.

## VI. EXPERIMENTS WITH THE FANORONA GAME

Fanorona is much difficult than tic-tac-toe. Moreover, there is almost no literature about Fanorona tactics. So, we are going to apply some successful techniques from the tic-tac-toe game in order to acquire knowledge for Fanorona.

### A. Endgame databases

We use the endgame databases used in [6], which are powerful and were useful to weakly solve the game. These databases contain all positions with 7 stones or less.

The following table gives us the distribution<sup>6</sup> of win, draw or loss for each part of the databases.

Db.	1-1	2-1	1-2
Win	158	10,366	717
Draw	334	398	3,231
Loss	26	6	6,822

Db.	3-1	2-2	1-3
Win	149,458	127,756	4,188
Draw	91	79,012	15,85
Loss	1	17,386	129,487

Db.	4-1	3-2	2-3	4-1
Win	1,529,142	2,711,327	774,043	19,814
Draw	12	327,836	1,252,162	88,187
Loss	0	18,297	1,031,255	1,421,153

Db.	5-1	4-2	3-3
Win	12,223,788	30,095,407	24,137,779
Draw	0	426,350	13,995,354
Loss	0	32,491	2,644,731

<sup>6</sup>a-b means a stones versus b stones

Db.	2-4	1-5
Win	4,180,200	81,728
Draw	7,926,733	391,405
Loss	18,447,315	11,750,655

Db.	6-1	5-2	4-3
Win	79,431,164	237,393,018	344,370,238
Draw	0	774,868	46,020,564
Loss	0	108,614	6,724,158

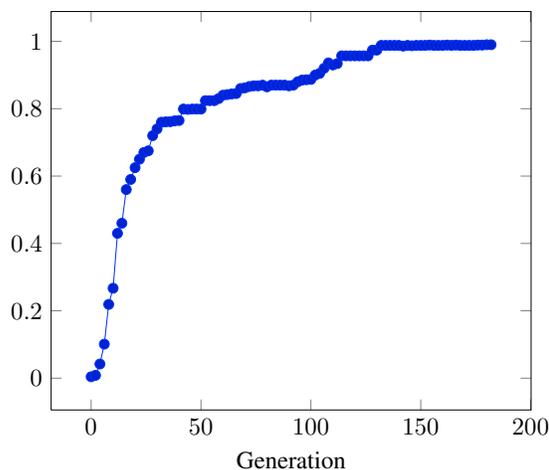
Db.	3-4	2-5	1-6
Win	145,408,435	18,659,090	320,021
Draw	170,633,688	41,896,491	1,509,775
Loss	81,072,837	177,720,919	77,619,368

In total, there are 6,261,651,750 positions in our databases.

### B. Results for the first configuration

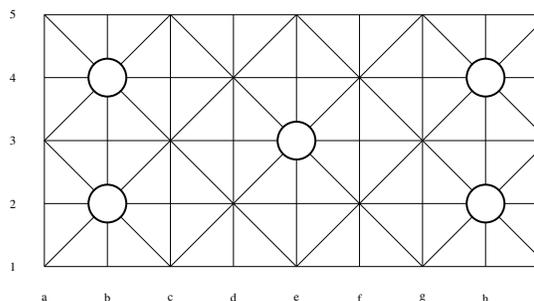
1) *Convergence speed:* We ran our algorithm on the 2-2 database. The plotting below represents the convergence speed of the average fitness value. The population size used is 1000.

Convergence of fitness - Average fitness

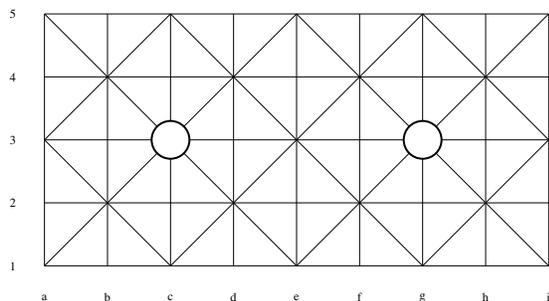


The convergence is much slower than in the tic-tac-toe's experiments.

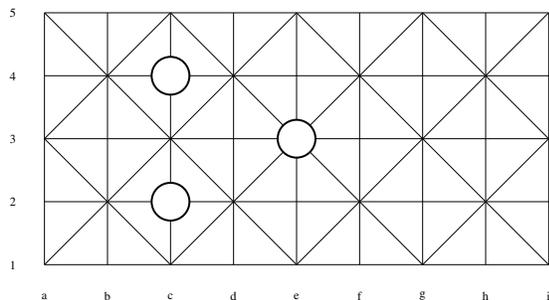
2) *Resulting patterns:* We give in the figure below the five of the best patterns mined from the 2-2 database.



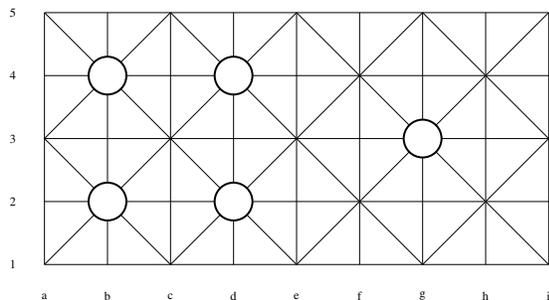
Fanorona pattern 1



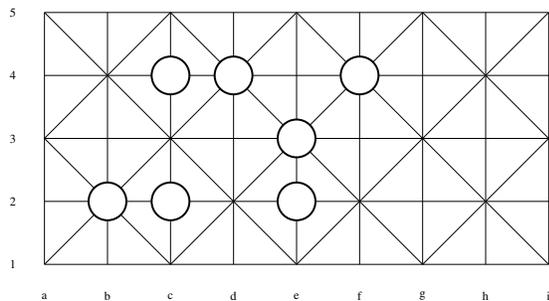
Fanorona pattern 2



Fanorona pattern 3



Fanorona pattern 4



Fanorona pattern 5

The experiment results in some very interesting patterns; because they are in the same time effective and human friendly.

Unfortunately, there are some patterns, such as *pattern 5*, that are very difficult to understand, even if they having a good fitness value. We could for example adjust the fitness value to give priority to human friendly patterns (by adding a bonus to symmetric patterns for example; or adding bonus to patterns

with less stones).

### C. Results for the second configuration

We did not ran the second configuration on the Fanorona because of the relatively poor results from the tic-tac-toe experiments.

## VII. RELATED WORKS

- Quinlan proposed the ID3 algorithm [9], which aimed to create decision-tree based state evaluator.
- Breda in his Ph D Thesis [11] exploited the J48 algorithm, which is a decision-tree induction algorithm, to extract knowledge from chess endgame databases.
- Bhatt and al [4] proposed a customized genetic algorithm to extract strategies for tic-tac-toe.

## VIII. GENERAL DISCUSSIONS AND FUTURE RESEARCH

Our experiments have good results despite some non-human-friendly extracted patterns, like *Fanorona pattern5*. Nevertheless, some human learners might learn better from these kind of patterns. Anyway, to quantify these perceptions, we might need to do some experiments on human learners.

A this state, some patterns might need to be interpreted by some advanced human players.

Moreover, patterns could be used as features in decision-tree induction algorithms, such as ID3 [9] and J48 [11], which can create logic based rules.

## IX. CONCLUSION

Our main goal in this paper is to get any human-readable knowledge from Fanorona endgame databases. Tic-tac-toe was used as test-bench for our method. In this work, we had an overview of the two games, followed by the results and discussions.

We got some excellent results, that will be helpful. We sometimes, had some drawbacks that we will overcome by creating better structured patterns, and applying some other algorithms.

The current work may open other directions for other games, as other researchers in might be interested in doing similar approaches.

## ACKNOWLEDGEMENT

We give thanks to ISPM (Institut Supérieur Polytechnique de Madagascar) for financially supporting our researches.

## REFERENCES

- [1] H. A. Raboanary, "Playing fanorona beyond perfect-play level using persuasive dialogue," in *Proceedings of African Conference on Software Engineering and Applied Computing*. IEEE, ISBN 978-0-620-51684-6, 2011.
- [2] L. V. Allis, "Searching for solutions in games and artificial intelligence," Ph.D. dissertation, University of Limburg, Maasricht, The Netherlands, 1994.
- [3] J. Schaeffer, Y. Björnsson, N. Burch, A. Kishimoto, M. Mller, R. Lake, P. Lu, and S. Sutphen, "Solving checkers," in *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005, pp. 292–297.
- [4] A. Bhatt, P. Varshney, and K. Deb, "In search of no-loss strategies for the game of tic-tac-toe using a customized genetic algorithm."

- [5] M. P. D. Schadd, “*Solving Fanorona*,” Master’s thesis, Universiteit Maastricht, Maastricht, The Netherlands, 2006.
- [6] 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.
- [7] J. R. Koza, “Genetic programming,” 1997.
- [8] K. Meffert and al, “Jgap - java genetic algorithms and genetic programming package,” URL: <http://jgap.sf.net>.
- [9] J. R. Quinlan, “Induction of decision trees,” *Mach. Learn*, pp. 81–106, 1986.
- [10] D. W. Aha, “Tic-tac-toe endgame data set,” UCI Machine Learning Repository - <http://archive.ics.uci.edu>, 1991.
- [11] G. Breda, “Krk chess endgame database knowledge extraction and compression,” 2006.