Genetically programmed strategies for chess endgame

Conference Paper · January 2006

DOI: 10.1145/1143997.1144144 · Source: DBLP

4 authors:

N. Lassabe
The French Aerospace Lab ONERA
15 PUBLICATIONS **191** CITATIONS
SEE PROFILE

Stéphane Sanchez
Toulouse 1 Capitole University
29 PUBLICATIONS **152** CITATIONS
SEE PROFILE

Hervé Luga
Institut de Recherche en Informatique de Toulo…
88 PUBLICATIONS **306** CITATIONS
SEE PROFILE

Yves Duthen
Institut de Recherche en Informatique de Toulo…
123 PUBLICATIONS **484** CITATIONS
SEE PROFILE

Genetically Programmed Strategies For Chess Endgame

Nicolas Lassabe, Stéphane Sanchez Hervé Luga and Yves Duthen
IRIT/UT1
21 allé de Brienne
31042 Toulouse, France
[lassabe,sanchez,luga,duthen]@irit.fr

ABSTRACT

Classical chess engines exhaustively explore moving possibilities from a chessboard configuration to choose what the next best move to play is. In this article we present a new method to solve chess endgames without using Brute-Force algorithms or endgame tables. We are proposing to use Genetic Programming to combine elementary chess patterns defined by a chess expert. We apply this method specifically to the classical King-Rook-King endgame. We show that computed strategies are both effective and generic for they manage to win against several opponents (human players and artificial ones such as the chess engine CRAFTY). Besides, the method allows to propose strategies that are clearly readable and useable for a purpose such as teaching chess.

Categories and Subject Descriptors

Algorithm [**Genetic Programming**]

Keywords

chess, Genetic Programming, evolving strategies

1. INTRODUCTION

One of the first challenges of computer science, and more specifically Artificial Intelligence, is to create chess engines. But, from quite simple rules and a limited chessboard size, chess game generates a great complexity and chess engines have been conceived in several ways.

The first one is to evaluate and to store, for each chessboard configuration what the best move to play is. This simplistic and idealistic approach of the problem seems technically difficult for it is commonly said that there are more possible chess games than atoms in the universe[1] [17] . Despite this, chess engines commonly use such tables of best moves but only for endgame situations with less than five pieces on the chessboards and only a few with six pieces on chessboard. Even under such restricted conditions, the needed storage size for each table is often several gigabytes.

A second approach is what is commonly called Brute-Force method[2]. It consists in calculating the best next move considering exhaustively the possibilities of the opponent from the actual chessboard configuration. Here the problem is no longer storage but the exponential growth of search space. Nevertheless, over the years, algorithms to search through trees of moves, and evaluation functions to estimate the correctness of a move, have improved to such an extent that modern chess engines play almost like chess grandmasters. Still, this approach has a flaw. Even if chess engines based on Brute-Force algorithms play near perfectly, they only mathematically compute the next best move: it is impossible to qualitatively know why such move is preferred to another one. The method to choose a move is not the one used by a real chess player.

A third approach to conceive artificial chess engines is to try and solve the chess game considering it and understanding it as a chess grandmaster does. This induces a cognitive study of a chess game that began with the appearance of the first artificial chess engines. This study shows that chess players elaborate strategies splitting up the configuration of chessboards in to specific patterns. Modelling these patterns and putting them together could be a good way of creating chess engines that use understandable and readable strategies instead of using Brute-Force algorithms or large endgame tables such those used by Nalimov's ones [12]. Several works already propose such an approach to specifically solve chess endgames. They usually highlight the difficulty of creating strategies that are both effective end generic enough to win all the configurations of a specific chess endgame.

This paper refers to this third approach as we propose to use Genetic Programming to automatically create an artificial chess player that plays using specific patterns and moves proposed by an associate chess expert. In order to implement and evaluate the performance of the method we will only for this time focus on a classic endgame, the King-Rook-King endgame (KRK). Genetic Programming allows to put patterns and moves together in order to compute generic strategies that can be effective against several opponents and in various starting configurations of a KRK endgame. We will also show in the results section that com-

[1]The number of chess games is approximately 10^{123}, that of legal positions is between 10^{43} and 10^{50} as a comparison, the number of atoms in the Universe is estimated to be betwee 4×10^{78} and 6×10^{79}

[2]See Appendix

puted strategies are realistic and clearly readable by any chess player. Last, to evaluate the quality of computed strategies, we will discuss games played by our artificial player against a perfect player using Nalimov's tables and compare them to those played by a referring chess engine, CRAFTY by Hyatt[3] [14], against the same perfect opponent.

2. BACKGROUND

Chess players need to develop a tactical and strategic sense to master this game and about ten years are generally necessary to become an international master. If the chess players devoted a lot of time to their passion, very early in the history of computer sciences, researchers also spent a lot of time trying to create chess programs able to play like chess grandmasters.

The first article on artificial chess engines was written by Shannon in 1950 [17]. At the time, they thought that resolving this game would be fast and so they could prove the power of artificial intelligence. In 1957, Herbert Simon said that within ten years, a digital computer would be the world chess champion. One year later, he discovered the alpha-beta pruning algorithm for chess [13]. Since then, performance of chess engines has never cease improving along with the improvement of alpha-beta, of its heuristics and of its evaluation functions [15]. Over the past years, research in this domain has focused on ways to improve evaluation functions. The most original ones is proposing to optimize the evaluation function using genetic algorithms [9], to replace it by artificial neural networks or to build it automatically using genetic programming [1]. In this perspective, genetic programming [10] has been mainly used in two ways. The first one is the automatic production of new heuristics for alpha-beta pruning [7], and the second is to find new evaluation functions to estimate the value of potential next moves [5, 8]. Though these uses of GP were original and effective, they still rely on an exhaustive exploration of possible moves to play in order to choose the best next one. A third approach is to use Genetic Programming to classify endgames situations according to the remaining moves to checkmate the opponent. This classification is done using standard endgames tables [6].

One inconvenience of such an approach (based on evaluation of moves and alpha-beta pruning) is that it is bound to the complexity of the alpha-beta trees (36 moves per position on the average means 36^n according to depth n). Indeed, if this approach is effective in the case of a chess game, it is no longer viable in the case of the Go game where there are about 300 available moves per situation [3]. Still, this method, commonly called Brute-Force, has proved its worth thanks to the growing processing capacities of modern computers: in 1997, DEEP BLUE defeated the world chess champion, Garry Kasparov, in six classical game match held in New-York [16].

This kind of victory of a computer over a human master has led to make chess one of the most used games to promote artificial intelligence and computer science. Nevertheless, chess engines are not able to strategically plan moves or to explain why they compute such a combination of moves. They just exhaustively consider moving possibilities and eliminating the least interesting ones in order to play the most promising move. On the contrary, human players have the capacity to focus only on a few available moves without exhaustively exploring the moving possibilities.

To prove this, Herbert Simon and others have studied the cognitive mechanisms of chess players. Using psychological tests, they have highlighted the importance of learning and the systematic use of patterns by chess players [18]. From this study, work about the solving of chess endgames using specific patterns has been pursued. Weill shows that it is possible to create by hand strategies for chess endgames by building decision trees using patterns [19]. The same year, Bain [2] proposed to solve a KRK endgame by learning logical rules from the extraction of patterns from KRK endgame tables. Similar work about solving the KRK endgame was carried out [11] but learned strategies came from human versus human games and they were modified by hand when they were unable to propose the next move.

3. EVOLVING STRATEGIES FOR KRK

In this work, our main purpose is to automatically generate strategies that can checkmate the Black King in every possible configuration of KRK endgame. Our concern is not to use endgame tables or classical algorithms bases on trees of moves and evaluation functions anylonger, but to build strategies as a human chess player does. That is to say by combining elementary chess patterns to build a winning strategy. This work differs from previous studies about endgames solving due to a fact that we do not try to extract patterns or playing rules from known endgame tables such as Nalimov's or from recorded played games, but we intend to propose a chess engine that learns how to play effectively using patterns proposed by chess experts.

The learning technique that we have chosen to apply is Genetic Programming (GP), by which computer programs can evolve [10]. A prime advantage of using GP is that we do not need to estimate topologies of evolved programs in advance or to determine precisely what the operators (or predicates) are to be used to compute a correct strategy. In GP, we start with an initial set of operators and functions and we let the system evolves according to the fitness function. A secondary advantage of GP is that computed programs are often readable. This was important to us for we wanted computed endgames strategies to be easily readable and understandable in order to reproduce them while playing, or to be used as a support for chess teaching. Last, using GP, it is no longer necessary to evaluate the correctness of every move played, but it is possible to generate an effective strategy with a fitness function that only considers the outcome of games played during evaluation.

3.1 The King-Rook-King endgame

In the particular case of chess endgames, a player usually follows specific algorithms or strategies. While involving few pieces, solving endgames is not trivial. Indeed, pieces involved have more freedom and so there are many available moves per situations. While chess engines usually perform a wide-search through endgame tables to determine the correct solution, human players are able to exclusively perform a deep-search to determine a wining strategy. This is a real advantage to solve endgames where planning a good strategy

[3]Crafty v19.13 is a direct descendent of CrayBlitz, the World Computer Champion from 1983 to 1989 : Elo of 2617 SSDF rating list (source code: ftp://ftp.cis.uab.edu/pub/hyatt)

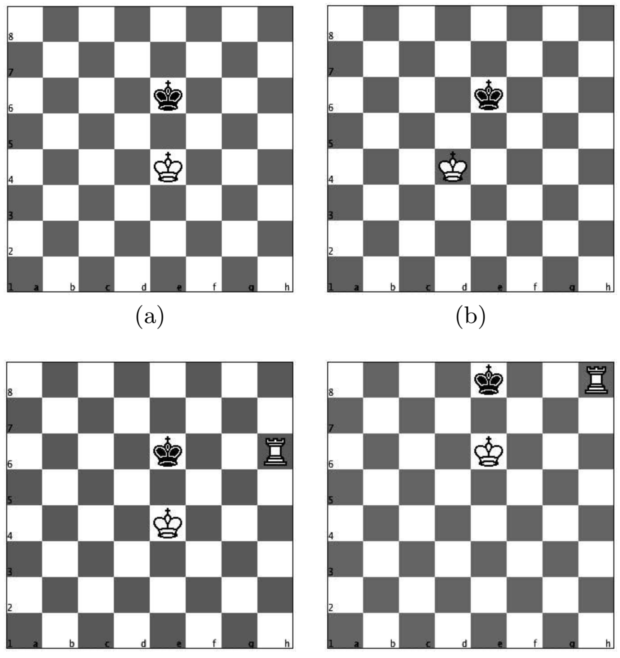

(a) (b) (c) (d)

Figure 1: Patterns: (a) Opposition: the Kings are in front of each other, separated by one field. (b) Near opposition: the Kings are almost in front of each other (c) Lateral check: there is opposition and the Rook laterally checks the King. (d) Checkmate: the King can not flee behind anymore.

is necessary. This phase is less tactical than the mid-game period (where chess engines are excellent) and it is crucial to anticipate the future of the game much further than an acceptable deep-search in a tree of moves. In the case of the King-Rook-King (KRK) endgame, a good coordination of King and Rook moves is necessary to avoid stalemate and realize the checkmate. Players use solving methods that are not easily implemented and usually based on patterns that are specific to endgames.

In a KRK endgame, the players easily understand that it is necessary to beat the defending King back to the edge of the board in order to checkmate. While several more or less effective methods exist, the strategy used remains the same. In order to beat back the defending King, and to eventually checkmate, attacking King must be in direct opposition to the defending one and the Rook must check the defending King on its side (Fig. 1). Players must repeat this operation until defending King reaches the edge of the board. To do that, they analyze the board configuration to determine specific patterns in order to choose their best next move. These patterns are usually simple, but their combination can produce complex strategies.

3.2 Patterns definitions

Chess players do not choose the next move considering all the possibilities, or counting how many fields a piece can cross. They have a specific objective that can be split up in patterns reachable by specific moves [18]. While playing, they usually try to determine which pattern they can obtain next, and how good enough it will be for their current strat-

Predicates
`boolean kingsInOpposition()` Are the Kings in front of each other and separated by one fields?
`boolean kingsCloseToOpposition()` Are the Kings almost in opposition?
`boolean kingOnEdge()` Is the Black King on the edge of the board?
`boolean oneEmptyFieldToBlackKing()` Does the Black King have only one possible move left?
`boolean distanceBKWK3()` Does the distance between the Kings equal three fields?
`boolean noEmptyFieldToBlackKing ()` In the current configuration, can the Black King move?
`boolean possibleControlOfEscapeLine()` Can the White Rook block the Black King?
`boolean possibleLateralCheck()` Can the White Rook check the Black King?
`boolean controlOfLineBeetweenKings()` Does the White Rook control a line between the Kings?
`boolean threatenedRook()` Is the Black King in a position to capture the White Rook?
`boolean protectedRook()` Is the White King protecting the White Rook?
`boolean possibleProtectionOfRook()` Can the White King protect the White Rook?
`boolean possibleProtectionOfRookWithOpposition()` Can the White King protect the White Rook by getting in opposition with the Black King?

White King moves
`protectRook()` move the White King to protect the Rook.
`nearlyOpposition()` move the White King to be close to opposition.
`distantOpposition()` move the White King toward the Black King.

White Rook moves
`controlEscapeLine ()` move the White Rook to limit the movement freedom of the Black King.
`lateralMove()` move the White Rook away from the Black King along the controlled line.
`controlLineBeetweenKings()` move the White Rook to control a line between the Kings.
`lateralCheck()` move the White Rook to check the Black King.
`moveAwayToFreeBK()` move the White Rook away from the Black King to avoid stalemate.

Table 1: Set of predicates and moves used to generate strategies

states	bonus	penalty
Rook is lost	0	5
impossible move	0	3
draw by repetition	0	2
game exceed 50 moves	0	1
King is stalemate	200	1
King is checkmate	400	1

Table 2: Bonus and penalties used by the fitness.

egy. In the particular case of KRK endgames, the discussions with the International Grandmaster Laurent Fressinet[4] 2625 elo [4] have isolated 13 predicates (Tab. 1) that determine a current situation of the chess board, and 8 specific moves to play.

3.3 Genetic encoding and genetic operators

Using GP, we will build binary trees with operators (predicates) as nodes and functions (moves) as leaves. Each tree is a chromosome and it defines a unique strategy that is evaluated by performing several reference games. As each defined predicate can be considered as a conditional statement, we defined 13 operators. Each operator can be read as an IF THEN ELSE statement. Each one of the 8 moves is modeled as a function. Transition between two nodes is considered as a logical AND.

Crossover is similar to the usual operator used in genetic programming [10]. Mutation substitutes a randomly selected node to another predicate, modifies a function (leaf) or replaces a part of the actual tree with a randomly generated one.

3.4 Fitness

Fitness evaluation should allow getting winner strategies for endgames of a King Rook against a King. That is to say that the best programs must be able to checkmate. We also want computed programs to be the most effective in most chessboard configurations. That is to say that they must win every game they play. Thirdly, computed programs must win a game as fast as possible. That is to say that they win a game with a minimal sequence of moves. Last, we want to favor the shortest computed programs (the smallest computed trees). Fitness value F (Equ. 1) of a computed program is a weighted sum of a sum of partial fitness functions f_i (Equ. 2) and an evaluation of the size of the program tree N. A partial fitness f_i is computed for a unique chess game i amongst a set of chosen games (that will ensure that the program is effective against various starting configurations). Each partial fitness function is firstly based on how many moves M_i it is necessary to play to end one game. This partial value is then modified according to how the game ends. If the King or the Rook checkmate their opponent, or if the opponent King achieves stalemate (the game ends in a draw but the King and the Rook manage to push out the opponent King to the edge and are close to checkmate), genetically computed programs are considered as good ones. In these cases, we add a bonus B_i to the partial fitness function (Tab. 2). If the game ends in a draw because there are three similar chessboard configurations, or if the program does not play any valid moves, or if the Rook is lost (and so it is impossible to checkmate), the computed programs are considered as bad and the partial fitness value is divided by an according penalty P_i. In the case of a draw if the computed program exceeds 50 moves without any capture, we can not conclude whether the program is good or bad, so no bonus or penalty is applied.

[4]Laurent Fressinet is in the FIDE world top 100 chess players from 2002.

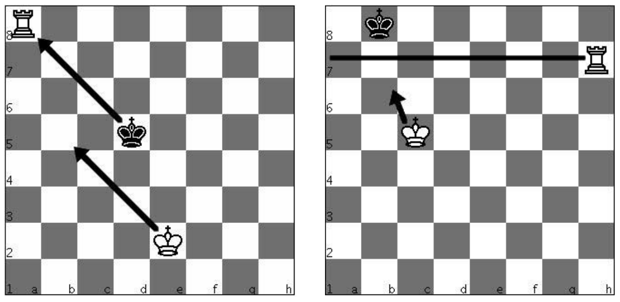

(a) (b)

Figure 2: (a) The White Rook attracts the Black King. (b) The White Rook blocks the Black King.

$$F = \alpha \sum_{i=1}^{n} f_i + \beta(1 - \frac{N}{maxN}) \quad (1)$$

$$f_i = \frac{(1 - \frac{M_i}{maxM}) + B_i}{P_i} \quad (2)$$

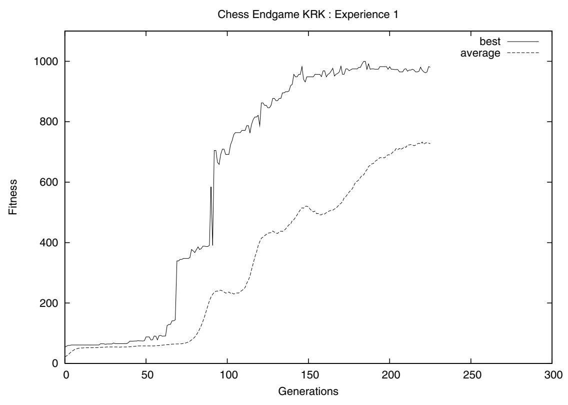
$maxN$: Maximal count of allowed nodes.
$maxM$: Maximal count of allowed moves.

4. RESULTS

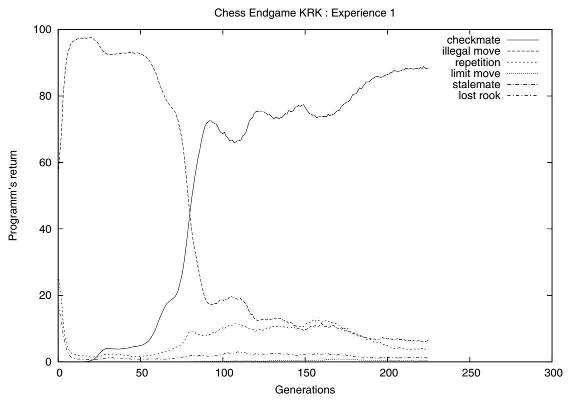
Chess programs to solve the KRK endgame have been generated using four experiments. The goal of each experiment is to generate the strategy of the white King and Rook to checkmate the Black King. Experiments differ from each other by the playing strategy applied to the defending King. The evaluation is done by making each generated program to play 17 reference games that represent the most relevant cases of the KRK endgame. The side to play at the beginning of a reference game can be White or Black according to the game itself. For all experiments, the mutation rate is 5% and the crossover rate is 65% (after several trials, these parameters are those that give best convergence to our problem). The population of programs to evolve consists of 10000 individuals.

4.1 First experiment

In the first experiment, the Black King's strategy is aggressive as it tries to capture the opponent Rook. After a couple of minutes, a suited strategy to checkmate the Black King is computed: instead of pushing out the opponent King to the edge of the board (the usual strategy for the White side), the computed strategy consists in attracting the Black King by putting the White Rook in one corner of the chess board (Fig. 2). Once the Black King is near the corner, the White Rook blocks it against the edge by controlling the second line from the edge. Then, White King only has to approach and to checkmate the opponent King. The best white computed strategy wins all the 17 reference games (Fig. 3). But, as the Black King's strategy is pretty dumb and unrealistic, the relevant result for this experiment is only to

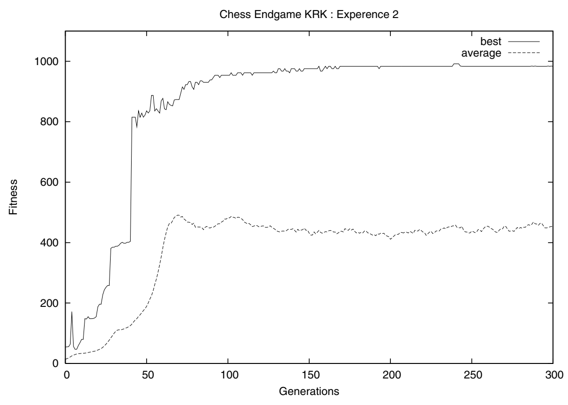


(a)

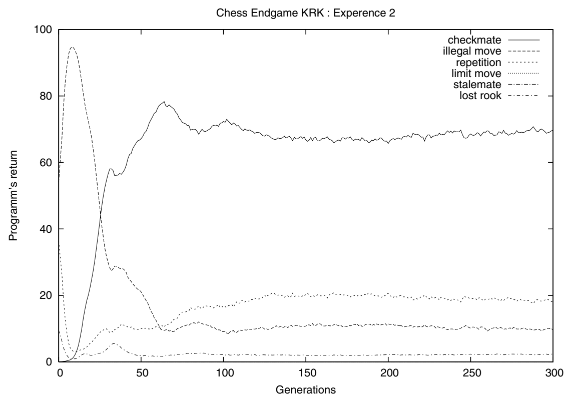


(b)

Figure 3: (a) The fitness evolution. (b) Before the 50th generation, strategies played mainly illegal moves and a lot of Rooks were lost. After the 50th generation, number of checkmates grows quickly, most Rooks are saved and illegal moves are mostly avoided.



(a)



(b)

Figure 4: (a) The fitness evolution. (b) Before the 30th generation, the strategies played in majority illegal moves and a lot of Rooks are lost. After the 30th generation, the number of checkmates grows quickly, most Rooks are saved and illegal moves are mostly avoided.

show how good our method is to compute an effective strategy against a specific one, and to compute a strategy that is effective in all the reference games.

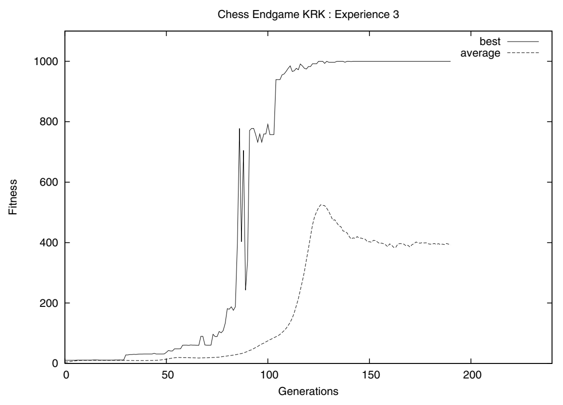
4.2 Second experiment

In the second experiment, the Black King has a better strategy: it does not follow the White Rook anylonger but it will try to capture it if it is close to it. If White Rook is not close to it, it will run away from it, trying to stay in the center of the chess board. Against this strategy, an effective White strategy is computed in a little less than an hour (Fig. 4). White King and the Rook manage to checkmate the Black King using a realistic strategy: the White Rook and the King cooperate to push the Black King against one edge of the chess board, and the White King also protects White Rook if necessary. However, computed strategies are no longer effective if we slightly the modify Black King's strategy (the White side is unable to checkmate). This experiment confirmed the correct convergence of the method toward a suited winning strategy, but we could not obtain a generic strategy for the KRK endgame, that is to say a strategy that checkmates any opponent.

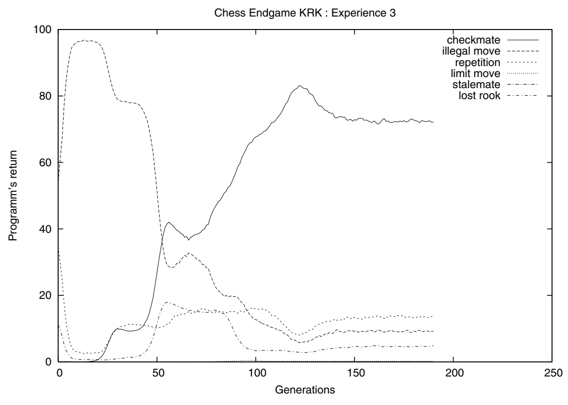
4.3 Third experiment

In third experiment, each computed strategy is still evaluated for the 17 reference games, but against 4 different Black Kings strategies. So, each computed strategy is evaluated with 68 games. The common point of four strategies is that the Black King tries to stay in the center of the chess board. They differ from each other by the following facts: the Black King does not seek to break opposition to the White King and it prefers to try and stay close to the center of the chessboard considering the Euclidian distance to the center. The Black King avoids opposition to the White King and, if necessary, moves away from the center of the chessboard, considering Euclidian distance to the center. The Black King does not seek to break opposition to the White King and it prefers to try and stay close to the center of the chessboard considering how many moves are necessary to reach the center. The Black King avoids opposition to the White King and, if necessary, moves away from the center of the chessboard, considering how many moves are necessary to reach the center.

An effective strategy to checkmate the Black King and to win all the 68 games is computed in about 3 hours (Fig. 5).

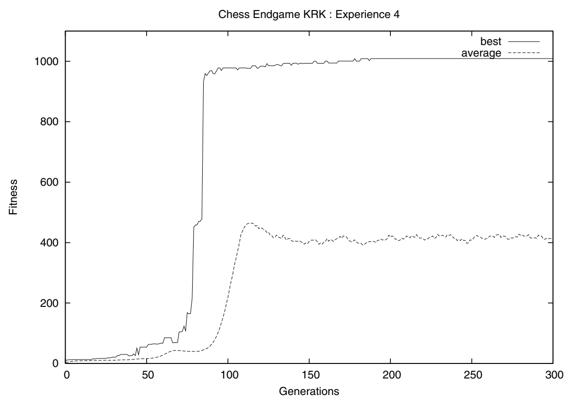


(a)

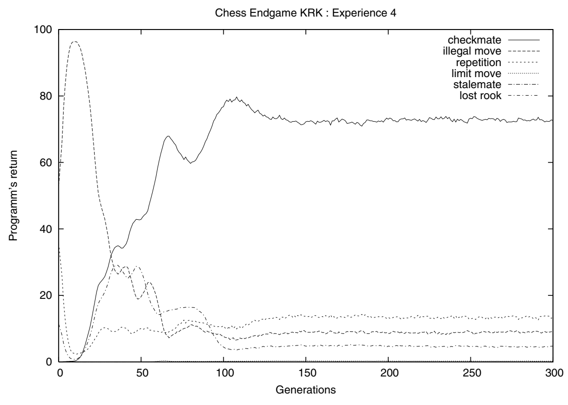


(b)

Figure 5: (a) The fitness evolution. (b) Before the 50th generation, strategies played in majority illegal moves and a lot of Rooks are lost. After the 50th generation, the number of checkmates grows quickly, most Rooks are saved and illegal moves are mostly avoided.



(a)



(b)

Figure 6: (a) The fitness evolution. (b) Before the 30th generation, strategies play in majority illegal moves and a lot of Rooks are lost. After 30th generation, the number of checkmates grows quickly, most Rooks are saved and illegal moves are mostly avoided.

The strategy proposed is really close to the one usually used by the chess expert. But, although it is effective against the dumb strategy of the first experiment, further tests using slightly different starting configurations of the chessboard, or against a human player with different strategies, show that the computed strategy is not totally generic. Indeed, the best computed strategy only wins 96% of the new 68 played games.

4.4 Fourth experiment

In the fourth experiment, we add a fifth Black King's strategy to the previous four. This strategy consists in using Nalimov's tables to make the Black King always play the best move. As in the previous experiment, an effective strategy is computed in about 3 hours (Fig. 6). This strategy wins all the 85 games used for evaluation and it manages to win all over games it plays (different starting positions, different strategies, against human players).

4.4.1 *A computed strategy from fourth experiment*

Figure (Fig. 7) shows the tree generated that represents one of the best computed strategies from the fourth experiment.

In the left part of the tree, the Kings are in opposition. This side is used both to push the Black King back to the edge of the chessboard and to checkmate it: if lateral check is not possible, the White Rook will be moved to make it possible (1).

In the right part of the tree, the Kings are not in opposition. If the Black King attacks the White Rook then the White King protects the White Rook by taking opposition (2). This action is really effective because the White King can protect the White Rook and goes on pushing the Black King back without moving the White Rook.

If the White Rook is not attacked, and this kings are not near the opposition, the White King gets closer to the Black one (3). If the kings are close, the White Rook controls the lines around the Black King to prevent it from escaping (4).

After the analysis of the computed strategies, the chess expert confirms that they are really close to those used by good chess players. Besides, he appreciates the fact that they are concise, readable and understandable. It is also noticeable that some of the concepts he formulated are not used by the computed strategies. Our method only uses the most relevant ones. Last, playing the KRK endgame against

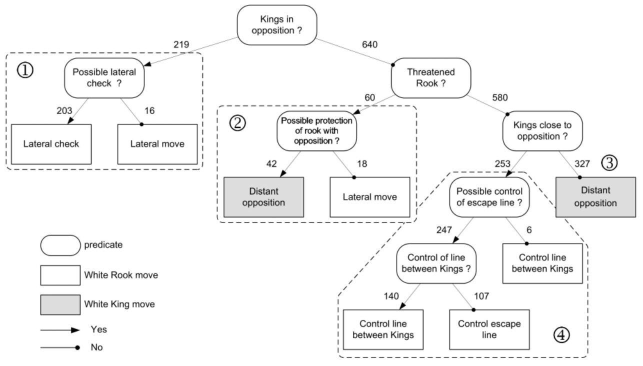


Figure 7: Representation of one of the best strategies for the KRK endgame. The predicates are represented by circles and the moves by squares. The number indicates how many times each branch of the tree was visited. All parts of the tree are useful.

several computed strategies, and systematically losing the game, tends to confirm that computed strategies are generic and work against any strategy for any starting configuration.

4.4.2 *Qualitative evaluation of best computed strategies*

Nalimov's tables are reference tables for chess endgames that involve less than 6 pieces on the chessboard. They allow to know which move it is best to play in each configuration for both the Black and White sides. The best move is the one that leads the attacker to the fastest checkmate or the one that leads the defender to the longest survival before checkmate or to stalemate. To evaluate the performance of the best computed strategies, we perform for each of them the 17 reference games. In each game, the White side (the King and the Rook) is attacking and played by a computed strategy while the Black King is perfectly defending using Nalimov's tables. Figure (Fig. 8) shows the average moves needed by ten computed strategies to checkmate the Black King. In theory, it is impossible to win a game in fewer moves than a game played between two players using Nalimov's tables. The graph shows that our computed strategies manage to be perfect, or near perfection, in most of the games played, except for 4 starting configurations when they badly fail to reach the minimal count of moves. In this case, our first analysis is that the patterns used (or predicates) may not be precise enough to identify really the specific configurations of the chessboard. To complete this evaluation, we performed the same 17 games between the CRAFTY chess engine as the White side and a player using Nalimov's tables. In games that require less than eight moves to checkmate, CRAFTY is as effective as a perfect attacker: it can go through its entire research tree to find the best way to checkmate. For games that require more than eight moves to checkmate, CRAFTY must take its move decision by considering only its evaluation function and it becomes less effective than the computed strategies: a lot of played moves are unnecessary and do not effectively lead the endgame to checkmate. On the contrary, computed strategies almost managed to play moves that induce progress to checkmate. Besides, for the latter, the CRAFTY reflecion time increased to about 4 minutes, while that of any of the

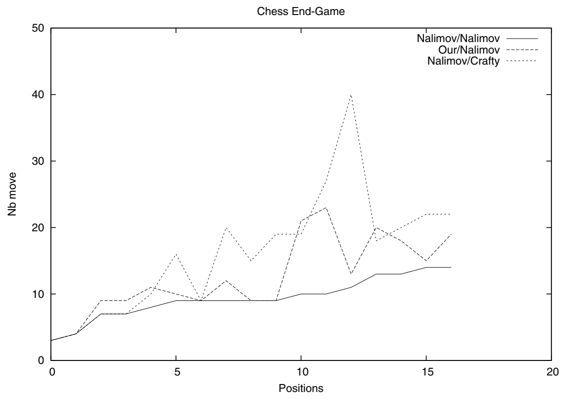


Figure 8: Average count of moves needed to win each of the 17 referring games

computed strategies remained below 1 second. Last, it is important to notice that computed strategies checkmated their 17 opponents in 11 moves on average, against 9 if the attacker played perfectly (Nalimov) and against 16 in the case of CRAFTY.

5. CONCLUSION

In this paper we presented a method to solve chess endgames without using classic Brute-Force algorithms or huge chess endgame tables. Instead, we used Genetic Programming to generate automatically winning strategies from a set of elementary chess patterns. The computed strategies for the first considered endgame, the KRK endgame, showed promising results. Indeed, we managed to produce generic strategies that can be effective against the tactics of every opponent they encountered and this for every starting game configuration. Besides, Laurent Fressinet's evaluation confirms that they are both concise and realistic in the way they play the KRK endgame. Last, the secondary goal of this project, producing strategies that are fully readable and that allow to understand how such an endgame must be played, is fulfilled.

However, two problems arose from this work. Firstly, computed strategies for KRK endgame do not manage to be optimally effective. While they seem better than the one used by a chess engine such as CRAFTY, it is still possible to refine the definition of patterns in order to avoid the less effective moves that remain in some games. Secondly, it seems pretty obvious that the defined patterns are really specific to KRK endgames. It might be necessary to evaluate the validity of those patterns in other endgames involving Rooks and Kings.

This work about the generation of chess strategies using genetic programing is still in progress. We are tring currently to model with our chess expert a new set of patterns to generate strategies for other endgames such as King-Knight-Bishop against King. This endgame requires at least 30 moves to checkmate and, though known strategies exist, they are usually not well mastered even by chess masters. To compute effective strategies for this endgame could allow to teach this endgame more effectively and to understand more precisely the utility of each move played.

6. ACKNOWLEDGMENTS

We wish to thank International Grandmaster Laurent Fressinet, Sosthène Kramo, Samy Abbas and Jérome Barrio for helpful and constructive comments.

7. REFERENCES

[1] M. Autonès, A. Beck, P. Camacho, N. Lassabe, H. Luga, and F. Scharffe. Evaluation of chess position by modular neural network generated by genetic algorithm. In *EuroGP*, pages 1–10, 2004.

[2] M. Bain and S. Muggleton. Learning optimal chess strategies. In *Machine Intelligence 13*, pages 291–309, 1994.

[3] J. Burmeister and J. Wiles. The challenge of go as a domain for ai research: a comparison between go an chess. In *Proceedings of the Third Australian and New Zealand Conference on Intelligent Information Systems IEEE Conference on Evolutionary Computation*, volume volume 2, 1995.

[4] A. E. Elo. *The Rating of Chessplayers, Past and Present*. Arco Pub., New York, 2nd edition, 1978.

[5] G. J. Ferrer and W. N. Martin. Using genetic programming to evolve board evaluation functions for a boardgame. In *1995 IEEE Conference on Evolutionary Computation*, volume 2, page 747, Perth, Australia, 29 - 1 1995. IEEE Press.

[6] D. Gleich. Machine learning in computer chess: Genetic programmig and krk, 2003.

[7] R. Groß, K. Albrecht, W. Kantschik, and W. Banzhaf. Evolving chess playing programs. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 740–747. Morgan Kaufmann Publishers, 2002.

[8] A. Hauptman and M. Sipper. GP-endchess: Using genetic programming to evolve chess endgame players. In *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 120–131, Lausanne, Switzerland, 30 Mar. - 1 Apr. 2005. Springer.

[9] G. Kendall and G. Whitwell. An evolutionary approach for the tuning of a chess evaluation function using population dynamics. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 995–1002. IEEE Press, 27-30 2001.

[10] J. R. Koza. *Genetic Programming: On the programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Mass., 1992.

[11] E. Morales. On learning how to play. In *Advances in Computer Chess 8*, pages 235–250. Universiteit Maastricht, 1997.

[12] E. V. Nalimov, G. Haworth, and E. A. Heinz. Space-efficient indexing of endgame databases for chess. *ICGA Journal*, Vol. 23(No. 3):148–162, 2000.

[13] A. Newell, J. Shaw, and H. Simon. Chess-playing programs and the problem of complexity. *IBM Journal of Research and Development*, 2:320–335, 1958.

[14] A. E. G. Robert M. Hyatt, Harry L. Nelson. Cray blitz. In *in Computers, Chess, and Cognition*, pages 111–130. Springer-Verlag, 1990.

[15] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[16] Y. Seirawan, H. Simon, and T. Munakata. The implications of kasparov vs. deep blue. *Commun. ACM 40(8)*, pages 21–25, 1997.

[17] C. Shannon. Programming a computer for playing chess. *Phil. Mag.*, 41:256–275, 1950.

[18] H. Simon and W. Chase. Skill in chess. *American Scientist*, 61:394–403, 1973.

[19] J.-C. Weill. How hard is the correct coding of an easy endgame. In H. J. v. d. Herik, I. S. Herschberg, and J. W. H. M. Uiterwijk, editors, *Advances in Computer Chess 7*, pages 163–176. University of Limburg, 1994.

APPENDIX

(Glossary of chess terms : www.arkangles.com)

Alpha-Beta pruning. A technique used by computer programmers to cut down on the number of possible moves a computer has to evaluate before choosing the best move.

Check. The act of attacKing the opponent's King. The opponent must get out of check on the next move, either by moving the King, capturing the attacKing piece, or moving another piece between the King and the attacKing piece.

Checkmate. Threatening the capture of the enemy King such that it cannot escape. This wins the game for the attacKing side.

Draw. A game that ends in a tie, where each player is awarded half a point. A draw occurs when 1) there's not enough material to force mate; 2) there is a stalemate; 3) a 3-time repetition of position has been reached

Elo rating. An internationally accepted mathematical system for ranKing chess players, created by Arpad Elo. International Grandmasters are typically in the range 2500 to 2700, world champions often over 2700. The standard deviation is 200 points. The scale is such that a player at 1800 would be expected to beat one at 1600 by the same margin as a player at 2600 against one at 2400.

Endgame. The final phase of the game when there are few pieces left on the board. The endgame generally starts when the immediate goal is to promote a pawn.

Fifty move rule. A game can be drawn when fifty moves have been made by each player without a capture or pawn advancement.

Opposition. An endgame term meaning the King not forced to move. Where the two Kings stand on the same file or diagonal with an odd number of squares between them, the player that doesn't have to move is said to "have the opposition." This is important in King and pawn endings as the player who can secure the opposition can effectively guard certain spaces or drive the opposing King back.

Positional. A move, series of moves, plan, or playing style concerned with exploiting small advantages.

Strategy. The formation and execution of an overall plan.

Tactics. Traps threats, and plans based on the calculation of combinations or variations. A position where many combinational ideas are present is a tactical position.

Transposition. Reaching an identical position from a different sequence of moves.