How Hard is the Correct Coding of An Easy Endgame[*]

Jean-Christophe WEILL

Université Paris 8 – Vincennes
Département d'informatique – Institut d'Intelligence Artificielle
2, Rue de la Liberté
93526 Saint-Denis (FRANCE)

Abstract

This paper concerns the correct coding of an endgame evaluation function to be used in a regular chess program (i.e. using a non modified α-β search). After showing why it is necessary to implement such knowledge, it is shown how the author, a non chess expert, and a computer inductively learnt classification concepts.

1 Introduction

One of the most famous positions for testing the ability of chess programs in the endgame is Réti's following position shown in figure 1. Today's chess programs easily find the solution which is 1. ♔g7 ♔b6 2. ♔f6 h4 3. ♔e5 h3 4. ♔d6 h2 5. c7 h1♕ 6. c8♕ and claims that the position is a draw, for example Chess Genius reports after depth 17 that the evaluation is -0.06 for white.

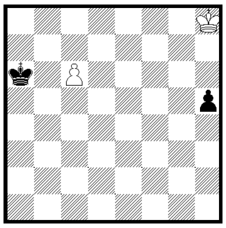

Figure 1: **R. Réti** Kagan's Neueste Schachnachrichten 1921 *WTM*

Does that mean that chess programs are able to understand this position? No, if they are not able to evaluate the resulting King and Queen against King and Queen endgame. Let us look at one other example depicted by figure 2. The complete solution is

[*]to appear in "Advances in Computer Chess 7",(J.v.d Herik and J. Uiterwijk Eds.) Ellis Horwood 1994

1. ♕×b2 ♔×b2 2. c4 a5 3. c5 a4 4. c6 a3 5. c7 a2 6. c8=♕ a1=♕ 7. ♕b7 (♕b8) ♔a3 8. ♕a6 (♕a7) ♔b2 (♔b4) 9. ♕b5 (♕b6) ♔a3 10. ♕a5 ♔b3 11. ♕×a1 ♔c4 (♔b4) 12. ♕e5 ♔b4 13. ♔d3 (♕d5) ♔a3 (♔b3) 14. ♔c3 (♕b8 ♕b5) ♔a4 (♔a2) 15. ♔c4 (♕f5 ♕g5 ♕h5 ♕d5) ♔ move 16. ♕ mate .

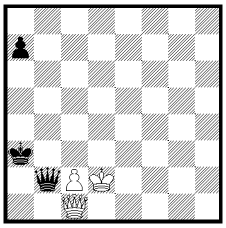

Figure 2: **J.-C. Weill** *WTM*

We designed this position so that chess programs would have to correctly evaluate the resulting KQKQ endgame in order to produce a correct evaluation of the root position. The best move is roughly forced to have all chess programs show the best continuation and thus to be able to say whether or not they correctly evaluate the KQKQ endgame.

Program	Node per secs	Evaluation	Depth	Principal Variation
Chess Genius	15000	0.30	11	Qxb2 Kxb2 c4 a5 c5 a4 c6 a3 c7 a2
Mchess 1.64	6000	0.11	8	Qxb2 Kxb2 c4 a5 c5 a4 c6 a3 c7 a2 c8=Q a1=Q
Fritz 2	20000	0.16	12	Qxb2 Kxb2 c4 a5 c5 a4 c6 a3
Cumulus 2.1	13000	0.06	10	Qxb2 Kxb2 c4 a5 c5 a4 c6 a3
Cray Blitz	920000	19.33	17	Qxb2 Kxb2 c4 a5 c5 a4 c6 a3 c7 a2 c8=Q a1=Q Qb8 Ka3 Qa7 Kb2 Qb6 Ka3 Qa5 Kb3 Qxa1 Kb4 Kc2

Figure 3: Results of the experiment for some of the best PC based chess programs, (Computer 486DX 50Mhz) after 3 minutes of search. Cray Blitz was running on a C90 [Hyatt, 1993]

Figure 3 shows the results for various chess program. It is shown that even the best PC programs are not able to evaluate correctly this position, and that with more computing power, the solution is reachable[Hyatt, 1993].

In this position, we did not mind that the programs would not correctly evaluate the root position (since all chess program chose ♕×b2) but there exist positions leading to KQKQ endgame which a bad evaluation leads to a bad choice of move. Figure 4 gives such a position.

To put the necessary knowledge with minimal modification in a regular chess program (i.e. using a minimax search algorithm with a point value evaluation function), it is necessary to implement a correct evaluation function for this endgame.[1]

To do this, following the example of [Beal and Clarke, 1980] and [Quinlan, 1984], we must construct the endgame database and then implement an evaluation function which is proven to be correct against the database.

[1] Of course, one can modify his program to have a special search for each particular endgame[Herik, 1983b, Herik, 1983a, Barth and Barth, 1992], but this is not our purpose.

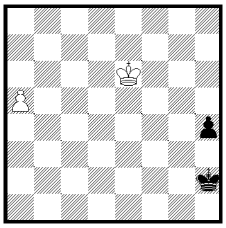

Figure 4: **L. Proke's 1937** *WTM* [Villeneuve, 1984] The solution is 1.Kf5 h3 2. Kg4 Kg2 3. a6 h2 4. a7 h1=Q 5. a8=Q +- , promoting first for White leads to a draw.

2 Some Properties of KQKQ

2.1 The Database

We constructed the database for the KQKQ endgame using the standard back-up procedure, the retrograde analysis [Herik and Herschberg, 1985, Thompson, 1986].

The array representing the database contains about 2 million entries. The player to lose is supposed to be black. Due to symmetry the black king stays in an octant (squares a1-d1-d4) triangle. If the black king is on the diagional a1-h8, then the white king must stay in the (a1-h1-h8) triangle. If in turn, the white king is on the diagonal, the white queen must be in the preceding triangle, and if the white queen is on the diagonal, the black queen must also be in the preceding triangle.

This allows us a unique entry for each group of symmetrical positions. Thus, an entry in the database represents 8 real board positions or 4 real board positions if all the pieces are on the diagonal.

We chose the following convention:

- A position is said to be losing with a distance 0, if the side to move has no queen and is not stalemate or if the side to move is mate and the opponent has still its queen.
- A position is said to be winning with a distance 0, if the side to move has a queen and the other side has lost its queen.

This implies that distance 1 winning positions are positions where the side to move can mate in one or capture the queen in one (resulting into a non stalemate position).

In the KQKQ endgame, there are 9324672 (1165815 entries in database) legal positions including 3612 (462) King against King , 144508 (18081) KQK and 227556 (28518) KKQ positions.

Figure 5 shows the distributions of winning an losing positions against the distance to win. The maximum distance to win is 10.

It is shown that 55, 8% of the legal positions are drawn, 41, 6% of the legal positions are winning and only 2, 1% of the legal positions are losing.

The high number of drawn positions compared to other endgames [Jansen, 1992] explains the fact that most human chess players considered it as an uninteresting drawing endgame[Tarrasch, 1980]. But it has some supporters: In his book "Practical Chess Endings", P. Keres wrote

distance	*Losing positions*		*Winning positions*	
	board	database	board	database
0	207304	25961	144508	18081
1	26108	3276	3541464	442753
2	4664	583	169108	21146
3	2216	277	19320	2415
4	712	89	4608	576
5	144	18	1264	158
6	144	18	440	55
7	152	19	552	69
8	56	7	240	30
9	24	3	56	7
10			40	5

Figure 5: Distribution of winning and losing position for the player to move

> It may seem a little mad to mention such an ending for there is apparently no win unless the opponent's queen is immediately captured. However, Victory can in fact be achieved in certain positions which must be know thoroughly.

Most of the winning positions are wins in 0 or 1 and there remain only $2,1\%$ wins in at least 2 moves.

2.2 Some Properties

For the remainder of this paper, we will only examine database positions, i.e. positions where the black king (the losing side) is in the octant (a1-d1-d4) triangle.

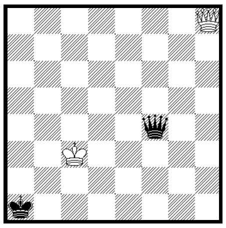

Figure 6: **V. Kantorowitch 1952** Black to move and lose [Villeneuve, 1984]

From the analysis of the database positions, we see that there are only 181 winning positions where the white king is in check and 22 losing positions where the black king is not in check. The maximun losing distance for those positions is 5 and is obtained only by the well know Kantorowitch position[Keres, 1973, Villeneuve, 1984] given in figure 6. This position is one of the exceptions to the rule "If you have to move and you can check without sacrificing the Queen, you have at least a draw" given in [Barth and Barth, 1992].

3 The Construction of a Correct Evaluation Function for KQKQ

We will now consider the paradigm of regular computer chess programs in which the search is a minimax equivalent search, the evaluation is a point value evaluation function and is applied at the leaves of the tree and in positions where the king is not in check.

3.1 First Attempt

Considering the low number of losing positions where the white king is in check, the author tried to construct an evaluation function for positions:

- Where the king is not in check
- Where no capture give a win (capture of the queen)

The excluded positions should be evaluated correctly by the quiescence search.

Since the evaluation of non checking losing positions is quite easy (you only have to know if your position is in the 22 listed in the appendix), the author tried to construct an evaluation function to decide whether or not a position is winning for the side to move.

The first attempt at this was using Roycroft's depthcharts[Roycroft and Beal, 1991] but if it was seen that it is more or less easy to evaluate positions where the black king is not on the first rank, other configurations of the black king appear to be very intricate. (At least for the author, as he's not a chess player.)

Then we decided to use ID3 like algorithms (summarized in figure 7). After this, It was planned to use ID5R[Utgoff, 1988] or IDL[Van de Velde, 1990] to produce a smaller tree. The main difference is that those algorithms incrementaly build the decision tree, and that ID3 builds every time a new decision TREE from scratch given a set WINDOW.

The author tried to code with ID3 the predicates for Winning positions where the king is not in check, and where the black queen cannot be taken directly but found that the rules became too complex or too time consuming.

3.2 2nd Attempt

Then, the author chose to try to construct an evaluation function for *all* losing positions, i.e. even the positions where the king is in check.

Following[Beal and Clarke, 1980], another scheme for the evaluation of the position was chosen:

- Check if the position is losing by comparing it with the 22 positions listed in the appendix.
- Do a one ply lookahead search to see if we can play a move that leads to a losing position for the other side.

This was a clash with our previous goal of not having to do a search but we know by experience that a one ply lookahead search is not too time consuming[2].

Then we used ID3 to elaborate the test set that will be incorporated into the decision tree.

[2] This scheme is used by many chess programs in order to evaluate King and Pawn against King positions

1. The user provides TEST, a set of predicates that return a boolean value given a KQKQ positions.

2. Define WINDOW to be a random sample of positions from the database to be discriminated

3. Define TREE to be the decision tree based on positions in WINDOW and the set TEST.

4. While TREE misclassifies any position to be discriminated:

   (a) Define WRONG to be a random sample of instances from the set of database positions to be discriminated that would be misclassified by TREE

   (b) Add all instances in WRONG to WINDOW

   (c) Define TREE to be the decision tree based on positions in WINDOW and the set TEST.

5. Stop, learned rule is TREE.

The decision trees at step 3 and 4c are built by choosing a good predicate that partitions the set of positions into smaller sets for which decision subtrees are constructed recursively. To determine which predicate should be the test attribute for a node, the algorithm applies an entropy information measure[Quinlan, 1979].
When a *clash* occurs, i.e. when there are two positions P_1 and P_2 with different class values and $\forall \mathcal{P} \in \text{TEST}, \mathcal{P}(P_1) = \mathcal{P}(P_2)$, then the two positions are given to the user who should produce a new predicate or recode an old predicate in order to discriminate the two positions.[Shapiro and Niblett, 1983]

Figure 7: The ID3 algorithm [Quinlan, 1984, Utgoff, 1988]

3.2.1 Losing in 0 move positions

The ID3 algorithm constructed the following tree to discriminate losing in 0 positions from winning or drawing positions(i.e. the result in losing in a least 1 move positions is not defined):

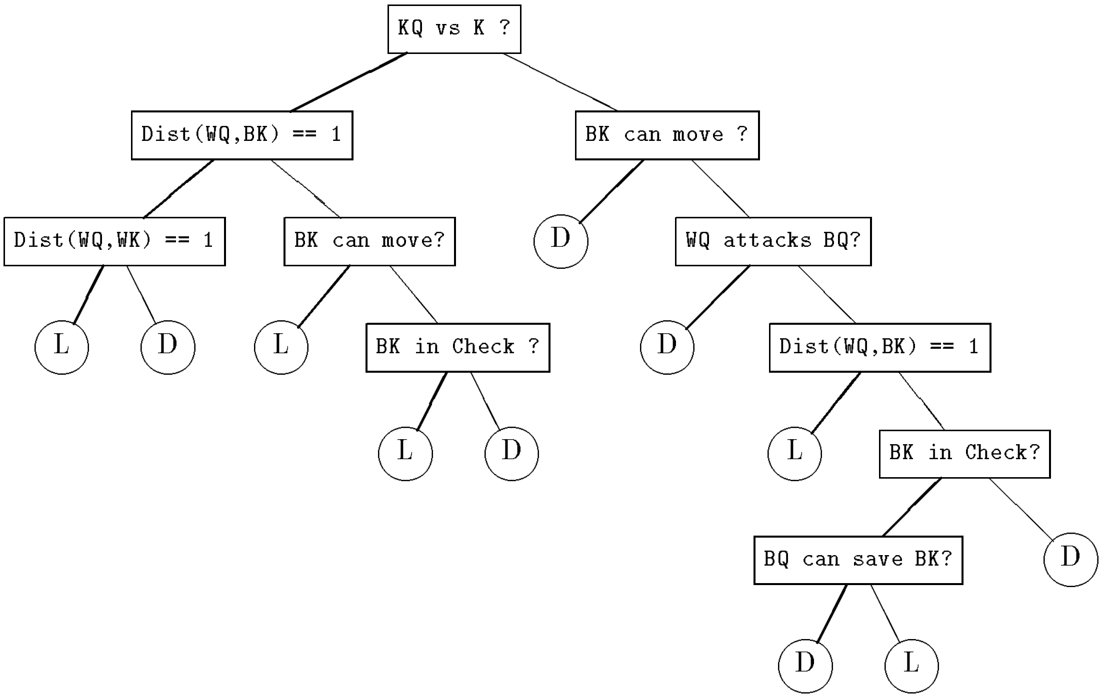

Figure 8: The Complete Decision Tree to discriminate Losing in 0 moves positions from Winning or Drawing positions. Thicker lines show *True* part of the predicates. All losing in 0 moves positions are going to a **L** leaf and all drawing positions are going to a **D** leaf.

This is a decision tree which sees if the king is checkmate without generating any move.[3] The most difficult predicates here were

BK can move checks if there is a square which is empty or occupied by a White piece and which is not attacked by a White piece near the Black King.

BQ can save BK It is a predicate that is true if the white Queen attacks the black King and the black Queen attacks the square near the black King in the direction of the white Queen.

3.3 Losing in 1 move positions

After this first easy success, we (the computer and the author) go on for losing in 1 positions.

Here the predicates become more involved. Some predicates deal with generalities:

Black King is pinned over the Black Queen

[3]This tree can be compared with the optimal tree provided by the IDL algorithm such as given in the appendix

Black King and Black Queen are on the same diagonal This predicate is useful combined with the previous predicate to indicate that the black King can't save the Black Queen if their distance is more than 2.

Kings are in opposition The BK is on the first rank and the White King is on the same column and on the third rank.

Other predicates deal with the possible moves of the king:

King can't move up This predicate indicates whether or not when the Black King is on the first rank, it can go on the second rank without being mated on the next move. In the general case, it returns *False* if the king can go to one of the second rank squares. But it appeared that the user modified the predicate in order to deal with clashes by adapting the predicate in the case where the BK is on A1 or B1 and the BK cannot go to B2 and C2:

1. If the White Queen and the White King attack B2 and the Black Queen does not, the predicate returns *True*.
2. If The White king controls the Square B2 and B1 and the White queen attacks A4 and the Black Queen does not attack A4 nor A3 then the predicate returns *True*.
3. If the White king is on B3 and the Black Queen does not attack C2 then the predicate returns *True* if the White Queen can go on the column A without being captured by the Black Queen.

King can move right (or left) Those two predicates indicate whether a Black King on the first rank can move right (or left) whithout being mated on the next White Move, the definition is given for a right move:

1. If the BK cannot move legally to the right, return *False*.
2. If the BQ is on the first rank and two columns to the right and the WK attacks the square just upper the BK, return *False*.
3. If the WQ and WK attack the square just up the right square and the BQ does not, return *False*.
4. If the WK controls the two squares up and up the right square of the BK and the BQ does not attack the square just 2 and 3 columns right the BK square and the WQ attacks the square 3 columns right the BK then return *False*.
5. If the WK controls the two squares of the 2nd rank on the columns of the BK and to the right of BK. If either the WK attacks the square of the second rank two column right from the BK or either the BQ is on this square and either The BQ does not attack the BK and the WQ can go safely to the left of the BK or either the BQ does not attack the square two columns right from the BK and the WQ can go safely to the right of that square return *False*.

BK can save BQ This predicate used in conjunction with the predicate **Black King is pinned over the Black Queen** returns *True* when the Black King and the Black Queen are on the same rank and the Black King can go near to the Black Queen.

Other predicates deal with static positions:

Black king on a1 and White king on c2 The predicate returns $True$ if

- The White Queen attacks D5 and the Black Queen does not attack D5 nor A5.
- The White Queen attacks B1 and the Black Queen does not attack B1 nor B3.
- The White Queen attacks a square on the A4-A8 semi column which is not attacked by the Black Queen.

The resulting decision Tree used 22 different predicates. The size of the tree was 82 nodes (41 leaves and 41 internal nodes).

4 The Results

We also defined good sets of predicates for "Losing in 2 moves positions", "Losing in 3 moves" and finally "Losing in more than 3 moves". Each time, it became more and more complex to determine global patterns and in the end the predicates became like testing if the position is in a set of positions.

Then, putting all these results together and calling ID3 again, we obtained a decision tree using 46 different predicates and 258 nodes.[4]

At this point, we have a decision tree which gives us whether a position is losing or not. An evaluation function such as $15 - 2 * dist(WQ, BK) - dist(BK, WK)$ is enough to denote that we know that it is losing, and that statically the distance to conversion or mate is proportional to the distance between the White Queen and the Black King.

5 conclusion

This paper shows how a non chess expert succeeded in coding a correct endgame evaluation function. It is shown that even if this endgame is considered to be an easy one, the predicates which were coded became quite involved.

It is not sure that this scheme would be feasible on a harder endgame. Even if it were, the definition of the predicates would become much more complex.

What should be necessary to code complex endgames is to have a specific and general enough language to express predicates on positions and to that could efficiently coded into a chess program.

It would be great if the computer chess community could share decision trees in the same maneer as we share endgame databases. The only thing we need is to design a language for predicates and then an interface to chess programs.

6 Acknowledgements

I would like to thank Marc-François Baudot for one more time taking time to proof read this article and for his search of traces of KQKQ endgame in chess books.

[4]Since then, the author had implemement ID5R, the resulting tree uses the same predicates and have 240 nodes. It was planned to use IDL but the author implementation of that algorithm gave a tree with more than 1000 nodes !

References

[Barth and Barth, 1992] Barth, W. and Barth, S. (1992). Validating a range of endgame programs. *ICCA Journal*, 15(3):132–139.

[Beal and Clarke, 1980] Beal, D. and Clarke, M. (1980). The construction of economical and correct algorithms for king and pawn against king. In Clarke, M., editor, *Advances in Computer Chess*, volume 2, pages 1–30. Edinburg University Press.

[Herik, 1983a] Herik, H. v. d. (1983a). *Computerschaak, Schaakwerelden Kunstmatige Intelligentie.* PhD thesis, Delft University of Technology.

[Herik, 1983b] Herik, H. v. d. (1983b). Strategy in chess endgames. In Bramer, M., editor, *Computer Game-Playing*, pages 87–105. Ellis Horwood Limited.

[Herik and Herschberg, 1985] Herik, H. v. d. and Herschberg, I. (1985). The construction of an omniscient endgame data base. *ICCA Journal*, 8(2):66–87.

[Hyatt, 1993] Hyatt, R. (1993). Personal communication. e-mail.

[Jansen, 1992] Jansen, P. (1992). *Using Knowledge about the Opponent in Game-Tree Search.* PhD thesis, Carnegie-Mellon University, Pittsburgh, PA.

[Keres, 1973] Keres, P. (1973). *Practical Chess Endings.* Batsford.

[Quinlan, 1979] Quinlan, J. (1979). Induction over large data bases. Heuristic Programming Project HPP-79-14, Stanford University.

[Quinlan, 1984] Quinlan, J. (1984). Learning efficient classification procedures and their application to chess end games. In Michalski, R., Carbonel, J., and Mitchell, T., editors, *Machine Learning*, volume 1, pages 463–482. Springer Verlag.

[Roycroft and Beal, 1991] Roycroft, J. and Beal, D. (1991). To make dumb endgame databases speak. In Beal, D., editor, *Advances in Computer Chess*, volume 6, pages 149–159. Ellis Horwood.

[Shapiro and Niblett, 1983] Shapiro, A. and Niblett, T. (1983). Automatic induction of classification rules for a chess endgame. In Clarke, M., editor, *Advances in Computer Chess*, volume 3, pages 73–92. Pergamon Press.

[Tarrasch, 1980] Tarrasch, S. (1980). *Trait pratique du jeu d' checs.* Fayot.

[Thompson, 1986] Thompson, K. (1986). Retrograde analysis of certain endgames. *ICCA Journal*, 9(3):131–139.

[Utgoff, 1988] Utgoff, P. E. (1988). Id5: An incremental id3. In *Proc. of the 5th Int'l Workshop on Machine Learning*, pages 107–120.

[Van de Velde, 1990] Van de Velde, W. (1990). Incremental induction of topologically minimal trees. In *Proc. of the 7th Int'l Workshop on Machine Learning*, pages 66–74.

[Villeneuve, 1984] Villeneuve, A. (1984). *Les Finales*, volume 2. Garnier, Paris.

A The 22 lost positions where the king is not in check

distance to lose	*White*		*Black*	
	King	Queen	King	Queen
1	b3	d2	b1	a1
	b3	e2	b1	c1
	c2	b7	a1	d8
2	b3	e2	b1	a1
	b3	f2	b1	a1
	b3	g2	b1	a1
	b3	h2	b1	a1
	d3	a2	d1	a1
	c3	f2	c1	a1
	c1	c4	a1	d7
	c2	c4	a1	d8
	c2	c5	a1	d8
	c2	c6	a1	d8
	d3	g2	d1	e1
	b3	g2	b1	h4
3	b3	e3	b1	h4
	c3	d4	a1	h5
4	c1	c5	a1	d8
	c3	e5	a1	g4
	c3	d4	a1	g5
	c3	d4	a1	h6
5	c3	h8	a1	f4

B Losing in 0 moves decision tree provided by IDL