Learning Through the KRKa2 Chess Ending

Alejandro González Romero and René Alquézar

Departament de Llenguatges I Sistemes Informàtics, Universitat Politècnica de Catalunya,
Jordi Girona Salgado 1-3, Mòdul C6, 08034 Barcelona, Spain
{aromero,alquezar}@lsi.upc.es

Abstract. The chess ending (KRKa2) has been studied using decision trees, neural networks and human reasoning to build a classifier for this ending, and for the discovery of convenient chess attributes. These chess attributes will serve for testing new ideas in planning. The idea is to investigate whether good automatically learnt policies for a planning problem can be generated using training examples along with evolutionary algorithms. The training examples, used as input to the learning algorithm, describe specific descriptions of a number of solved instances in one domain; then to improve the learnt policies obtained from the training examples, the policies should evolve. We believe that the domain of games is well-suited for testing these new ideas in planning.

1 Introduction

Chess skill requires an ability to form plans, generalise from examples, learn from experience and to recognise the important features of a position such as a "strong" square or weak pawn structure ([1], [2]). The fact that attempts to incorporate, say, planning into chess-playing programs have only been partially successful so far is an indication that the game of chess provides a serious and complex, yet well-defined and self-contained testbed for Artificial Intelligence theories and techniques ([3], [4]).

A classifier for the ending KRKa2 (King and Rook vs. King and pawn on a2) has been built with the aid of neural networks, decision trees and human reasoning. For space reasons, only the problem of deciding whether Black wins or not is presented, although we have solved also the problem of deciding whether White wins or not. The previous three techniques also allowed the discovery of a set of useful short predicates. They will serve to construct training examples for an evolutionary learning algorithm [5] that will learn a policy set to solve this domain.

2 The KRKa2 Chess Ending Domain

The ending KRKa2 (King and Rook against King and pawn on a2) was first studied by Alen Shapiro in 1983 [6]. Given a legal position with Black (the pawn's side) to play, Shapiro considers the problem of deciding whether Black wins or not. The problem of deciding whether the rook's side wins or draws is not treated in his work. We consider the same ending with significant modifications: It's White's (the rook's side) turn to play and is determined if White wins, Black wins or it's a draw. See diagram 1.

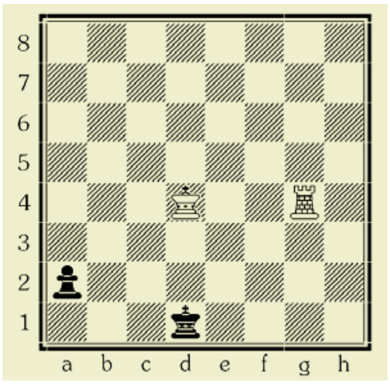


Diagram 1. Comparison between the two works: In Shapiro's work it's Black's turn; Black wins by queening the pawn and thus checking the white King. In the present work it is White's turn; White wins but the process is not trivial: *1. Kc4! Kc2(1... Kc1 2. Kb3 a1+(Knight promotion) 3. Kc3 Kb14.Rg2+ –) 2. Rg2+ Kb1 3. Kb3 a1+(Knight promotion) 4. Kc3 and White wins since Black's knight is trapped*

Another important difference between the present work and Shapiro's work is that in our case besides building a classifier we started to work in the construction of automatically built policy sets [5]. To this end, a set of short predicates is needed to conform the LHS (left hand side) and RHS (right hand side) of our rules. To increment substantially the learning chances of our learning system, it is very important that these short predicates be as good as possible. Thus to deduce them we used neural networks, decision trees and human reasoning.

As in Shapiro(1987) [7] the problem has been tackled using decision trees, but neural networks have also been employed. In both techniques a training set, consisting of a subset of chess positions, has been used to induce the classification rules. Then another subset of chess positions (the test set) has been classified with the learned rules (or the learned function in case of NN). The results were compared with the true classification giving an average accuracy of the rules.

2.1 Some Definitions and Useful Notation

For convenience some chess concepts will be mentioned, they serve to build be attributes for the construction of training and test examples.

Hidden check: It consists of just one move; for our purposes it will be moving the WK permitting with this move a check by the rook.

Horizontal check: It is a check given along a row by the rook, it can be a rook move or a king move in case of a hidden check.

Vertical check: It is a check given along a column by the rook, it can be a rook move or a king move in case of a hidden check.

The *distance function* is calculated by the minimal number of king moves that takes to go from one square to another.

Let us introduce some useful notation:

U: set of squares different from a2.
x: cartesian product. Thus AxC={ (a,c): $a \in A$ and $c \in C$ }.
+: union.
-: difference of sets. Thus A-C= { $x \in A$: $x \notin C$ }
A symbol as {defgh} means the set of squares of U on columns d, e, f, g, h.
A'=U–A, that is, the complement of the set A in U.

The symbols i, j, k denote numbers between 1 and 8; the symbols α, β, γ will denote letters between a and h. Let us define n(a)=1, n(b)=2, n(c)=3, n(d)=4, n(e)=5, n(f)=6, n(g)=7, n(h)=8; $\beta - \alpha = n(\beta) - n(\alpha)$; $\alpha < \beta$ if $n(\alpha) < n(\beta)$. We will denote by β–1 the letter preceding β and by β–2 the letter preceding β–1. The distance between two squares αi and βj is given by $d(\alpha i, \beta j) = \max\{|j - i|, |\beta - \alpha|\}$, which is the number of king moves needed to go from αi to βj.

We define the segments $[\alpha i, \beta i] = \{(\gamma i): \alpha \leq \gamma \leq \beta\}$ and $[\alpha i, \alpha j] = \{(\alpha k): i \leq k \leq j\}$ that refer respectively to row and column segments. Also [d4,h8] will denote the diagonal segment $\{(\alpha k): 4 \leq n(\alpha) = k\}$.

It is assumed that (*) different pieces occupy different squares, the black king is not in check and the distance between the kings is greater than one (d(WK,BK) > 1). One must eliminate from the positions specified below those that do not satisfy (*).

3 Results

3.1 Classifying Positions and Choosing Attributes Using NN and DT

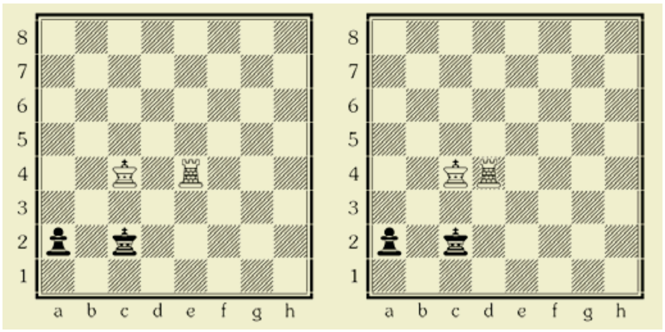
There were several tries of sets of attributes in order to find the attributes that are more useful to the learning algorithm. A first try consisting of 21 attributes had the following results.

Using 21 attributes, 80 training examples and a program that induces decision trees, a decision tree was obtained. The tree was tested using 150 test examples; its percentage of right answers was 71.6%. In the same set of 80 training examples, a feed-forward neural network was employed. A 10-fold cross-validation was utilised, each fold contained 80 training examples and 15 test examples, and the network had one hidden layer of 15 units. The right hit performance was 63.75%; using sequential backward feature selection (*backselection*) the percentage of right answers increased to 65.75% and five attributes remained in the best line.

More training examples were added, up to 140, and using 150 test examples, the decision tree method had a performance of 75% correct answers, while the neural network scored 71.33%, using 15 hidden units, 10-fold cross-validation, each fold composed with 140 training examples and 15 test examples, and backselection. Again five attributes remained in the best line, two of them coincide with those obtained using the 80 training examples.

To improve classification results the number of attributes was reduced to eleven, although the attributes had to be more complicated. In this case, we obtained the following results. Using 140 training examples and 150 test examples, the decision tree method had a performance of 92% correct answers. The feed-forward neural network scored 85%, using 15 hidden units, 10-fold cross-validation, each fold composed with 140 training examples and 15 test examples. After using backselection the NN method reached 90%. The best line of attributes was: 1), 2), 3), 5), 6) and 10). Those attributes were:

1. White can win in one or two moves:
   a) Checkmating in the next move.
   b) Capturing the pawn in the next move.
   c) Giving a *skewer* in the next move. (A *skewer* will be defined as a check followed by a safe pawn capture with the rook).
   d) Giving checkmate in two moves.
   e) Threatening the pawn in the first move and capturing it in the 2nd move.
2. White can draw in any of the following cases:
   a) The WK is in check, the rook is threatened and is not attacking the pawn.
   b) The BK is in a stalemate position and it is not possible to make a move that changes the "state" of the position, that is, no matter what White moves, the BK will remain in a stalemate position.
   c) There is a *pseudoskewer position*. This is a position such that it is possible to move a *pseudoskewer* (a check with the rook, followed by an exchange of the rook for the pawn. In diagram 2, Re2+ followed by Rxa2 is a pseudoskewer).
   d) There is an *A position*. (an *A position* is one in which the rook or the WK can be moved threatening a pseudoskewer). Diagram 3, illustrates an A position, for example, Rh4 would threaten the pseudoskewer (Rh2+ followed by Rxa2).



Diagrams 2 and 3. Show a *pseudoskewer position* and an *A position* respectively

   e) The rook can check the BK creating an A position; we could name it as an almost A position. See diagram 3.
   f) It is possible to give perpetual check.
3. The rook can guard the promotion square.
   a) And at the same time threaten the pawn. (The rook can be moved to column "a" and nothing interferes between it and the pawn).
   b) Without threatening the pawn. (The rook can be moved to row "1" and nothing interferes between it and the pawn).

5. The rook or the King can be moved threatening mate or skewer. It is possible to check BK and then give a skewer or threaten mate or skewer.
6. The rook can check (horizontally or vertically) and afterwards protect the promotion square.
10. d(WK,a2) ≤ 2.

These results confirm certain key concepts to build proper attributes, such as: giving a vertical check with the rook and the distance between the white king and the pawn. Other useful concepts were found, for example what Shapiro calls a delay skewer, i.e. moving the rook or the white king threatening skewer in the next move.

3.2 Position Classifier

Another approach to tackle the main problem is to divide the main problem into two smaller problems. The first one consists of deciding if in the chess position given, Black (the pawn's side) wins or not. If Black does not win, the second problem would arise, which consist of deciding when White (the rook's side) wins or draws. The advantage of this strategy is that it reduces the size of the tree, and increases the accuracy of the classification.

An important result of our research is the position classifier that can serve to classify random positions for the learning algorithm. Unfortunately, due to space reasons, we only present part of the classifier, which determines for every legal position of the KRKa2 ending with White to play, whether Black wins or not. This part was obtained using human reasoning for the construction of the attributes and a decision tree program for the tree construction.

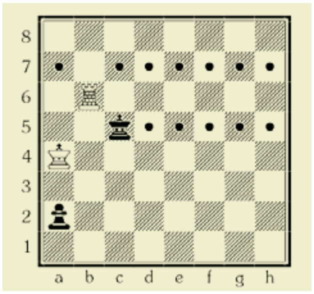
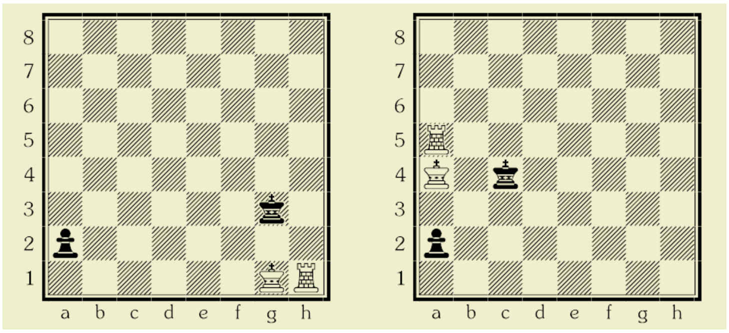


Diagram 4. Some ab positions (won by Black). The black dot denotes Black king positions

Eight attributes (to determine whether Black wins or not):

1. (ab position)
   (BK, WK, R) belongs to {(γk,ai,bj) : 4≤i≤6, i≤j, i≤k, |j-k|=1}- {(ak,ai,bj) : j=i+1}
2. (WK against the ropes)
   (BK, WK, R) belongs to {(ci,ai,aj) : 4≤i, j=i+1} +{(α3,α1,β1) : d≤α, β=α+1}
3. (BK in northwest) BK=γk γ=a or b k=6, 7 or 8
   (WK,R) belongs to {(αi,αj) : 2≤i<j, e≤α≤h, |j-k|=1} + {(di,dj) : 4≤i<j, i≠5, |j-k|=1} + {(ci,cj) : 4≤i<j, |j-k|=1} + ({defgh}+[c4,c6])x{bj : |j-k|=1} + {(a5,a8)}

4. (BK in southeast) BK=γk γ=f,g or h k=1 or 2
   (WK,R) belongs to {(ei, hi) : i=3-k} + {(α3,β3) : d≤α<β+(1-k), α≠f, |β-γ|=1|}
   +{(α4,β4) : d≤α<β, α≠e, |β-γ|=1} + {(αi,βi) : 5≤ i, b≤α<β, |β-γ|=1}



Diagrams 5 and 6. Two "against the ropes" positions (won by Black)

5. (BK on row 2) BK=γ2
   (WK,R) belongs to {(αi, βi) : 5≤i, b≤α<β, |β-γ|=1}
6. (BK in south) BK=γk γ=d or e k=1 or 2
   (WK,R) belongs to {(αi,βi) : d≤α<β, i=n(α), |β-γ|=1}
7. (BK on column a) BK=ak
   (WK,R) belongs to {(αi,bj) : αi ∊ [d1,h1]+[d4,h8], |j-k|=1, 3≤j}
   +{(αi,αj) : 4≤n(α)=i<j, |j-k|=1}
8. (BK on column b) BK=bk
   (WK,R) belongs to {(αi,αj) : e≤α≤h, 2≤i<j, |j-k|=1} + {(d4,dj) : 4<j, |j-k|=1)

Using the above eight attributes, seventeen examples (not stated for space reasons) and the decision tree induction program, a decision tree was obtained that can determine whether a position is won by Black or not. The resulting decision tree is functionally equivalent to the following If statement:

If (ab position) or (against the ropes WK) or (BK in northwest) or (BK in southeast) or (BK on row 2) or (BK in south) or (BK in column a) or (BK in column b) then Black wins; if not then Black does not win.

3.3 Discovery of Attributes for Learning Policy Sets

Finally with the aid of neural networks, decision trees, and human reasoning a set of useful short predicates was discovered. They will serve to construct training examples that are used as an input to an evolutionary learning algorithm which will learn a policy set that solves this domain. Although much work remains to be done, we believe that this method to automatically construct plans [5] will work for many domains.

Let WK=αi, R=βj, and BK=γk. Then:

|β-γ|=1: Rook and black king are on adjacent columns (and therefore a horizontal (=lateral) check at βk is not possible unless it is supported by the white king, that is, d(WK,βk)=1).

|j-k|=1: Rook and black king are on adjacent rows (and therefore a vertical check at γj is not possible unless it is supported by the white king, that is, d(WK,γj)=1).

i=j and α < β (i can be 1, 2, ...or 8): Rook on the right of the white king on row i.

α=β and i < j (α can be a, b, ...or h): Rook above the white king on column α.

β=b: Rook on column b (so it cannot move to b1 which is protected by the pawn).

Concepts regarding distance: d(WK,p)>n, d(BK,p)>n (n=1,2,...). Note, for example, that d(WK,p)=1 is equivalent to NOT (d(WK,p)>1) which is the negation of a concept.

a=γ<β and i<=j: (WK a-blocks the rook), 1=i<j and γ<β: (WK 1-blocks the rook).

R ∈ {2}+{a} and WK ∉ [p,R]: Rook attacks pawn.

i=1 or i=n(α): The white King would be in check if Black could queen.

These short predicates would serve as primitive predicates to the learning algorithm. To be precise the above short predicates would play the role of concepts that would be placed in the left hand side of the rule.

A list of possible actions (the right hand side of the rule) would be: R x P (Capture the pawn with the Rook), WK x P (Capture the pawn with the King), *Rook to a* (move the rook to column a), *Rook to 1* (move the rook to row 1), *Rook to 2* (move the rook to row 2), give a horizontal check, give a vertical check, etc.

An example of a policy set for the KRKa2 domain would be:

π₁ = `(R ∈ {2}+{a} and WK ∉ [p,R]) ^ d(BK,P) > 1 → R x P`

π₂ = `¬ d(WK,P) > 1 ^ d(BK,P) > 1 → WK x P`

Rule **π₁** says: if the Rook is attacking the pawn and BK's distance to the pawn is greater than 1 then capture the pawn with the Rook. Rule **π₂** says: if the WK's distance to the pawn is NOT greater than 1 (d(WK,P) = 1) and BK's distance to the pawn is greater than one then capture the pawn with the white king.

4 Conclusions

In [3] Fürnkranz states: "A significant body of research has gone into the task of learning to classify positions from a given chess endgame as wins or no wins. Endgame databases for KPK, KPa7KR, or KRKN have become standard benchmark problems for induction algorithms. Research in this area is also important for computer chess, because if endgame positions could be correctly classified with a few rules, this would save many resources compared to the alternative of storing all positions of a variety of endgames. Besides, many of the endgame databases that are now available are not thoroughly understood by human experts". We believe that these ideas hold true for many domains.

In the KRKa2 ending when deciding whether a position is won by Black or not, using the eight attributes, we noticed that in order to obtain a perfect classifier (one that correctly classifies every position), it is convenient, whenever possible, to build attributes that are sufficient conditions, that is, those attributes that might classify by themselves. For example: 1) *If (ab position) then Black wins*.

In every planning domain it is necessary to have some human knowledge about it in order to build a solver, in our case conformed of policy sets. Unfortunately this was not the case for the KRKa2 domain. Shapiro (1987) [7] writes: "In relation to its complexity the textbook treatment of KPKR (King and Pawn vs. King and Rook) is even more sketchy than that of KPK. Indeed the particular subset of KPKR that was chosen for this experiment (Ka7KR) is not mentioned at all in any chess ending book". In our case the lack of chess literature for this ending increases even more the difficulty of the problem. Recall that it is harder to analyze the ending when it is the Rook's side turn (KRKa2). Therefore we had to deduce this knowledge using human reasoning, neural networks and decision trees.

From the results of the system that makes policy sets for the Blocks World [5] and the construction of the KRKa2 classifiers, we believe that it is possible to build a system that constructs good policy sets for the KRKa2 domain. Since the KRKa2 domain involves more attributes (chess concepts) and more actions than the Blocks World, it constitutes a much better testbed for the evolutionary learning algorithm [5]. In fact the main idea is to prove that the learning algorithm can serve for many domains, but it is convenient to pass the KRKa2 barrier first.

References

1. Bain, M.E.: Learning Logical Exceptions in Chess. Ph. D. thesis, Department of Statistics and Modelling Science, University of Strathclyde, Scotland (1994)
2. De Groot, A. and Gobet, F.: Perception and memory in chess. Heuristics of the professional eye. Assen: Van Gorcum, Assen, The Netherlands. (With R.W. Jongman) (1996)
3. Fürnkranz, J.: Machine learning in computer chess: The next generation. International Computer Chess Association Journal, 19(3), (1996)
4. Khardon, R.: Learning action strategies for planning domains. Artificial Intelligence, 113, (1999)
5. González, A.: Learning Policy Sets using Evolutionary Computation. http://www.lsi.upc.es/ ~alquezar/gonzalez2003.ps (2003)
6. Shapiro, A.D. and Niblett T.: Automatic induction of classification rules for a chess endgame. In M. R. B. Clarke (Ed.), Advances in Computer Chess 3, Oxford: Pergamon (1983) 73-92
7. Shapiro, A. D.: Structured Induction in Expert Systems. Turing Institute Press. Addison-Wesley (1987)