Inductive Inference of Chess Player Strategy

Anthony R. Jansen, David L. Dowe, and Graham E. Farr

School of Computer Science and Software Engineering,
Monash University,
Clayton, Victoria 3800
Australia
{tonyj, dld, gfarr}@csse.monash.edu.au

Abstract. We investigate the problem of inferring, from records of chess games, some aspects of the strategy used to play the games. Initially, game records are generated from self-play by two simple chess programs, one of which does a one-ply search while the other does a four-ply quiescent search. In each case, we are able to infer, from just the game records, good estimates of the weights used in the evaluation function. The approach is then applied to grandmaster games. Our one-ply and quiescent four-ply programs are now drastic simplifications of the true strategy used. Nonetheless, using inferred weights for these hypothetical models, we are still able to achieve some success (as measured by compression rates for the games) in predicting moves made by the players.

1 Introduction

Most research on Computer Game Playing has concentrated on getting programs to play games as well as possible. This is usually achieved by the use of appropriate search algorithms together with some explicit coding of human knowledge about strategies for the game (Shannon, 1950), although learning methods have also been used (Samuel, 1959, draughts/checkers; recent chess examples include Thrun, 1995, Morales, 1996).

In this paper we go in an opposite direction. Starting from records of games played, we wish to infer something about the strategy used. This is clearly a machine learning problem, but different to problems of improving play by learning from experience. We approach the problem in two steps. First, we attempt to infer a strategy from records of chess games generated by a simple chess program. Here the actual algorithm is known, so this provides an important test of the feasibility of our approach.

We then attempt to infer aspects of the strategy used by human chess players, based on the success of the first step. This is the primary motivation behind this research. There is much unknown about the algorithms humans use, although there have been attempts by psychologists and others to understand how humans play the game (for example, Newell and Simon, 1972). The works of de Groot (1978), Frey (1977), and Levy and Newborn (1991) reveal the contrast between current game playing strategies used in computer programs, and the techniques that humans use.

Despite the amount of research exploring computer chess playing, little work appears to have been done along the lines reported here. The approach of Carmel and Markovitch (1993) tries to learn the strategy used by the opponent during play, but does not use records of games. Muggleton (1988) takes expert descriptions of how to play some specific chess endgame positions and does 'sequence induction'. The aim there is to infer a finite automaton which encodes a more general strategy for that phase of that particular type of endgame. Our approach is to begin with the result of a strategy, in the form of records of games played, and then to infer aspects of that strategy. Work in a similar vein has been done by Walczak (1992), in which an Inductive Adversary Modeler was developed that can infer the perceptual chunks used by a chess player based on game records.

In the next Section we introduce a simple game player model, its implementation, and the results obtained by doing inference with it. The work done there provides evidence of the validity of our approach. We then introduce a more advanced game player model, and discuss the results of our attempts at inference. Finally, we discuss the conclusions and future directions.

2 Simple one-ply player model

2.1 Description of the Model

The simple player model is based on a 1-ply search. Positions at this depth are evaluated in the following manner. The player uses an *evaluation function* $v(_)$ of the form

$$v(P) = \sum_{k=1}^{n} \lambda_k a_k(P), \tag{1}$$

where

- the $a_k(_)$ are *attributes* (or *advisors*), i.e. functions which report on some specific aspect of a position. The total number of attributes is n. Typical attributes for chess would be
  - $a_1(P)$ = the *material balance* of the position (where here, Queen = 9, Rook = 5, Bishop = Knight = 3, Pawn = 1), being the difference between the player's and opponent's total material;
  - $a_2(P)$ = the *mobility balance*, i.e. the difference between the numbers of moves available to the player and the opponent.
- the λ_k are the *weights* attached to the attributes.

The two attributes described here are the only ones used for the simple model. For each (and hence for the evaluation function as a whole), a positive value is good for the player, while a negative value is good for the opponent, and reversal of roles negates the values.

A key feature of the model is that the player chooses *probabilistically* from the moves available. This random element serves several purposes. Firstly, people do not always choose the same move from the same position, and may err. Also, the model can never account for all possible considerations a player uses to determine

what move will be played. This will naturally result in uncertainty when choosing a move. Probabilistic choice of moves also allows statistical inference to be used. A consequence of probabilistic move choice is that we are not attempting to develop a model that selects the correct move in every position. Rather, we wish the correct move to have a high probability of being played. While it is desirable that the correct move be the most probable, the model will still be successful if the correct move tends to have a high probability.

The model also assumes that the player strategy is consistent – and so variation in strategy based on things like the playing style of the opponent, or whether the game is in the opening, middle or end phase, is ignored.

We now describe how the player moves under this model.

1. Input: position P.
2. Generate all positions $P_1, P_2, \ldots, P_m$, to which the player can move from position P. (We can think here of the positions $P_1, P_2, \ldots, P_m$ as corresponding to the possible **moves**, numbered $1, 2, \ldots, m$ respectively, which the player can make from the position P.)
3. Compute the evaluation $v(P_i)$ of each position P_i, $i = 1, \ldots, m$.
4. Choose a move probabilistically from $\{1, \ldots, m\}$ according to
$$\Pr(\text{move } i) = e^{v(P_i)}/(e^{v(P_1)} + \ldots + e^{v(P_m)}). \tag{2}$$
This multivariate logistic function maps position values, which range over $(-\infty, \infty)$, to the interval $[0, 1]$ in a natural and elegant way.
Let the chosen move be i^*.
5. Make the necessary move to the chosen position, P_{i^*}.

From our database of game records, we have a large set $\mathcal{P}$ of positions and for each position we are given the move that was made. It is these choices of move which are the data from which we seek to infer a model. We regard the positions themselves as prior information and do not seek to explain them.

Likelihood function Consider for simplicity a single position P, from which one can move to any of $(P_i : i = 1, \ldots, m)$. The observed data is the index i^* of the position P_{i^*} to which the player chose to move. Our model is $\theta = (\lambda_1, \ldots, \lambda_n)$. The likelihood function for this single data item i^* is
$$f_P(i^*|\theta) = \frac{e^{v(P_{i^*})}}{\sum_{i=1}^{m} e^{v(P_i)}}.$$

The actual data we have is a whole set of moves, one for each position in $\mathcal{P}$. For each $P \in \mathcal{P}$, denote the move chosen by $i^*(P)$. Then our likelihood function is
$$\prod_{P \in \mathcal{P}} f_P(i^*(P) \mid \theta) = \prod_{P \in \mathcal{P}} \frac{e^{v(P_{i^*(P)})}}{\Sigma_i \; e^{v(P_i)}} \tag{3}$$

The simple game player model described determines the probability of each move being played based on the weights of the attributes that are being considered. To infer the weights, a Maximum Likelihood approach was used with a simple and robust hill climbing algorithm.

2.2 Inference Results

Generated Data To test the validity of the inference technique, a simple chess program was written which, in each position, essentially did a 1-ply search, and pseudorandomly chose a move based on the λ values that the program was given, according to Equation 2. To keep things simple, the player model only considers two attributes ($n = 2$): material balance and mobility balance. Using several sets of λ values (from Equation 1), databases of up to 1000 games were generated by the chess program playing against itself.

A question of initial interest was how large a database needs to be to allow successful inference to take place. Databases of various sizes were created considering the single attribute, material balance. The attribute weight was set at $\lambda_{MAT} = 3.0$. Inference was then conducted on the different sized databases. For a database containing only one game, the inferred λ_{MAT} value was 4.11; for a 10 game database, 3.28; for 100 games, 3.20; for 1000 games, 3.09 was inferred.

The inference results show that as the size of the database increases, the inferred value of λ_{MAT} gets closer to the correct value. For the database of 1000 games the inference performs well, although it still overestimates slightly.

To test our approach further, inference was tried on other databases of generated data, each consisting of 1000 games, with two attributes (material balance and mobility balance). In Table 1, the weights that were used to generate the games for the databases are given (as Actual Values), as are the inferred weights.

	Data Set 1		Data Set 2	
Attribute Weights	λ_{MAT}	λ_{MOB}	λ_{MAT}	λ_{MOB}
Actual Values	5.0	0.5	10.0	1.0
Inferred Values	5.15	0.51	10.23	1.02

Table 1. Inference results from 1000 game databases using the simple model.

The results show that the inference method performs well on both of these databases. Note that the inference method again slightly overfits the data, with no parameter values being underestimated.

Grandmaster Databases Since the inference results from the generated data indicate that accurate inference is possible, it seemed worth trying our approach on grandmaster data. Although such players do not conform to our simple model, it may still be interesting to see what attribute weights are inferred for them. The same two attributes (material and mobility) are used.

The two grandmasters selected are Robert J. Fischer and Garry Kasparov. Since inference seems to perform better with larger amounts of data, the entire available database for each player is used. The results can be seen in Table 2, which shows the number of games in each grandmaster database and the inferred weights.

Player	No. Games	λ_{MAT}	λ_{MOB}	I-Random	I-Inferred	Comp (%)
Fischer	732	0.510	0.021	146047	138383	94.8
Kasparov	1030	0.458	0.025	191936	183021	95.4

Table 2. Inference results from grandmaster databases using the simple model.

These weights, however, give no indication of how well the model can predict the move made in a given position. Probabilistic prediction of information corresponds to compression of data. This fact has been exploited in the work of Althöfer (1991), where chess databases have been compressed by using a chess program to predict the most likely move to be played by a grandmaster in a given position. Here we also use data compression as a measure of predictive accuracy. The information content (in bits) of a move can be readily calculated from the probability of that move being played, using

$$I(\text{move } i) = -\log_2 Pr(\text{move } i)$$

The last three columns in Table 2 give details of the information content of the two grandmaster databases. The first value, I-Random, gives the amount of data (in bits) when the move choice is completely random, with no strategy assumed and each move equally likely. This is just uncompressed data. The next value, I-Inferred, indicates the size of the data (in bits) when compressed using the results of the Maximum Likelihood inference. The inferred weights will mean that all moves are no longer equally likely in a given position, and thus the information content will be different. The final column, Comp (%), indicates the compression achieved by giving the size of the post-compression data as a percentage of the original random information. For both grandmasters, the inferred values compress the data to about 95% of its original size. Considering the simplicity of the player model being used, this is an encouraging result.

The successful implementation and inference of the simple game player strategy in this Section has shown that even a simple model can be applied to grandmaster games with positive results. The next Section looks at a more advanced game player model that uses more aspects of the strategies that humans use.

3 A more advanced player model

3.1 Description of the model

In the previous Section we introduced a very simple player model which we used primarily to confirm the validity of our approach. The success with this model leads us to consider a more advanced model. The advanced player model takes into account more aspects used by human chess players, such as considering more position attributes, searching deeper than 1-ply, and searching some lines deeper than others. There are still many aspects missing in our advanced model, such as adjusting attribute weights depending on whether the game is in the

opening, middle or end phase. However, it is a significant improvement on the simple player model.

The advanced player model is based on a quiescent 4-ply search, which works as follows. All positions at depth 1-ply are generated. Further search is done only if a capture is possible, a promotion is possible, a check can be played, or the player is already in check. Maximum search depth is 4-ply, but of course not all lines are explored to this depth.

As with the simple model, the player chooses the move probabilistically. Since the number of dimensions in the search space depends on the number of attributes, we use just four attributes ($n = 4$). We realize that both humans and computer programs generally examine many more attributes, but we have chosen this constraint in order to make the search space for our inference more tractable. The attributes used are the balance of *material*, *mobility*, *control of the centre 16 squares* and *number of attacks* (see, for example, Levy and Newborn, 1991). Attacks against pieces of equal or lesser value are considered to be worth only one tenth of an attack against a more valuable piece.

Likelihood function The evaluation of a single position in this model is the same as the evaluation used for the simple game player model. Each attribute is given a weighting, i.e. a λ value. The multiple ply search is also probabilistic, again just as the simple 1-ply model was. The probability of each move from the root position is calculated in the same manner as was used in Section 2. However, the evaluation for each of the 1-ply positions is now determined by considering positions at deeper levels in the search tree.

A probabilistic variation of the minimax algorithm is applied to determine the probabilities of each move available from the root position. This is similar to the way that the minimax algorithm is applied to games such as backgammon, which involve both strategy and chance. The full likelihood function for the 4-ply quiescent search is quite complex, and is contained in the Appendix.

For the inference, a Maximum Likelihood approach is again used along with a simple and robust hill climbing algorithm. It should be noted that for our 4-ply quiescent model, a typical search to determine the probabilities of the available moves needs to evaluate over 2000 positions, with the evaluation of each position depending on the four weights that need to be inferred. Rather than repeatedly searching and evaluating all of the positions on each iteration of the hill climbing algorithm, this is done only once initially with the resulting attribute values from each position stored in a search tree database. This data requires additional information about where in a given search tree each position arose. As a result, the inference programs have very large memory requirments, especially when all the moves from several games are being considered. For example, encoding all of the information from five games requires over 150 megabytes of memory. This memory expense is justified by the speed increase obtained as a result of only generating the search tree data once.

3.2 Inference Results

Generated Data As with the simple player model, we first try inferring weights from databases created using the player model.

A chess program based on this player model was used to create databases of games that used the four attributes defined by the model with a 4-ply quiescent search. The game records were generated by the program playing against itself. Again, a pseudo-random choice between possible moves was made, based on probabilities obtained from the likelihood function. The coefficients were selected to try and make the program play as well as possible, given that it is still much simpler than human play. The level of play achieved by this model is quite weak, but it is competitive against a human beginner.

The results using the generated data showed that the Maximum Likelihood inference method worked very well, and are given in Table 3. The λ weights (from Equation 1) used to generate the data are also given (as Actual Values). Each inference is from the data of a single player in a single game, with five games being considered. The Table also gives the information content (in bits) of the data both when the move choice is random (the data is uncompressed), I-Random, and after the Maximum Likelihood inference, I-Inferred. The amount of compression of the data, Comp (%), as defined previously in Section 2.2, is also given. It can be seen that most of the games have been compressed to less than 50% or their original size. The second last row of the table shows the inference result when both the white and black moves from five games are used for inference.

Colour	No. Moves	λ_{MAT}	λ_{MOB}	λ_{CEN}	λ_{ATT}	I-Random	I-Inferred	Comp (%)
White	126	7.11	0.25	0.49	3.65	464.7	208.5	44.9
Black	126	7.89	0.22	0.62	3.45	598.6	265.2	44.3
White	94	7.49	0.22	0.51	3.56	404.1	160.3	39.7
Black	94	8.13	0.24	0.51	3.89	458.8	169.9	37.0
White	147	7.14	0.25	0.50	3.34	662.6	299.4	45.2
Black	147	7.74	0.25	0.53	3.88	579.0	244.7	42.3
White	30	6.12	0.25	0.50	2.84	160.3	92.8	57.9
Black	30	6.49	0.19	0.29	8.36	147.0	80.5	54.8
White	99	7.75	0.17	0.47	2.98	498.3	190.4	38.2
Black	99	8.09	0.24	0.69	4.10	479.5	107.5	22.4
W & B	5 Games	7.53	0.21	0.52	3.31	3879.4	1442.1	37.2
Actual Values		7.0	0.2	0.5	3.0			

Table 3. Inference results from generated data using the advanced model.

An interesting result about the inference with the advanced player model is that very good results are obtained using the data from a single game. This appears to be because each position in the search tree contributes to the data available, and the inference for a single move has such a large search tree that

accurate inference is possible without needing to consider too many database positions.

This is a welcome result, because the memory requirement for a single chess game is already very large. However, the inference on five games also gave an excellent result, and the compression achieved over five games is superior to the average compression achieved for a single game. As a final point, unlike with the simple game player model, not all inferred weights overfit the correct values.

Grandmaster Databases After successful inference on the generated data, the next step was to apply the advanced player model to do inference from grandmaster games. Again we used games by Fischer and Kasparov.

Table 4 shows the results of the inferences on two games by Fischer and two games by Kasparov. For each player, we chose the longest game the player won with both white (W) and black (B). The games were: Fischer–Taimanov, Vancouver, 1971, 89 moves; Sherwin–Fischer, USA Championship, 1966, 100 moves; Kasparov-Karpov, 1990, 102 moves; Kamsky–Kasparov, New York, 1989, 107 moves. The inferred weights are given, as is the information in the data before and after inference, and the compression achieved. The last row gives an inference using as data the 11 games played by Fischer in the 1963 USA Championships.

Player	No. Moves	λ_{MAT}	λ_{MOB}	λ_{CEN}	λ_{ATT}	I-Random	I-Inferred	Comp (%)
Kasparov	102 (W)	1.20	0.09	0.10	3.19	481.4	370.8	77.0
Kasparov	107 (B)	0.82	0.18	0.09	1.81	443.7	328.7	74.1
Fischer	89 (W)	1.31	0.00	0.06	0.89	385.5	307.6	79.8
Fischer	100 (B)	1.61	-0.06	0.09	2.82	416.7	339.1	81.4
Fischer	11 Games	1.24	0.04	0.09	1.15	1995.3	1555.3	77.9

Table 4. Inference results from grandmaster games using the advanced model.

Where the simple 1-ply model was used, we managed to compress the grandmaster data to about 95% of its original size (see Table 2). The results in Table 4 show significant improvement, with compression ranging from approximately 81% to only 74% of the original data. This demonstrates that the advanced player model certainly helps in the inference of aspects of the strategy used by these players. It is also interesting to note that Kasparov's games compressed better than those of Fischer. This could indicate that the attributes considered by the model are more in keeping with Kasparov's evaluation of a position than that of Fischer.

The final results examine compression using different training and test data. Table 5 shows the results of taking λ values inferred from one set of data taken from grandmaster games, and using it to compress other grandmaster games. The training data used was Fischer's 11 games played in the 1963 USA Championships. This has been used to compress the same games that were used in

Table 4, whose original Maximum Likelihood compression results are listed again for comparison. The final two columns in Table 5 are the information content of the games after the strategy inferred from the training data is applied to them, I-Training, and the subsequent compression achieved.

Player	No. Moves	I-Random	I-Inferred	Comp (%)	I-Training	Comp (%)
Kasparov	102 (W)	481.1	370.8	77.0	380.2	79.0
Kasparov	107 (B)	443.7	328.7	74.1	345.0	77.8
Fischer	89 (W)	385.5	307.6	79.8	312.1	81.0
Fischer	100 (B)	416.7	339.1	81.4	348.1	83.5

Table 5. Inference results using the advanced model with different training data.

The results show that the compression achieved using strategies obtained from training data that are different to the test data, is nearly as good as when strategies inferred directly from the test data are used. This is expected, since all grandmasters try to play the best possible move in a given chess position. This result shows that general strategies can be inferred from large databases, and then successfully be applied to specific instances.

4 Conclusions and Future Work

The goal of this research was to infer something about the strategies used by chess programs and, ultimately, humans based on the records of games played. Initially, a simple 1-ply model was developed to confirm the validity of the concept. The results indicated that successful inference was possible over both data generated using the simple 1-ply model, and grandmaster databases. The amount by which the data was compressed was used throughout as a measure of the modelling accuracy of the inferred strategy.

This led to the development of a more advanced player model that used a 4-ply quiescent search. The results show that excellent compression was achieved for data generated using the advanced 4-ply quiescent model (on average, data was compressed to less than 50% of its original size), and that grandmaster games could also be significantly compressed. It was also possible to compress grandmaster games using a strategy inferred from a different grandmaster.

The results indicate that it is possible to infer something about the strategies used by chess players, and that the more advanced the model is, the better is the predictive accuracy of the inferred strategy. Inductive inference of strategies based solely on information contained in databases can thus allow successful learning and prediction.

There are several directions for future work. These include developing more advanced game player models that better mimic the way humans really play chess and looking at different inference methods, such as Minimum Message Length (MML). Also, the power of the inference could be investigated further

by going beyond data compression as a performance measure and determining if specific questions about game records can be answered. For example, given that strategies have been inferred for several grandmasters, can this information be used to predict who the players are in a particular chess game? Finally, these techniques can also be applied to domains other than chess and game playing.

References

Althöfer, I. (1991). Data compression using an intelligent generator: the storage of chess games as an example, *Artificial Intelligence, 52*, 109–113.

Carmel, D. and Markovitch, S. (1993). *Learning models of opponent's strategy in game playing*, (CIS Report #9318), Israel Institute of Technology.

de Groot, A.D. (1978). *Thought and Choice in Chess*, (2nd ed.), The Hague: Mouton.

Frey, P.W. (1977). *Chess Skill in Man and Machine*, New York: Springer-Verlag.

Levy, D. and Newborn, M. (1991). *How Computers Play Chess*, New York: W.H Freeman & Company.

Morales, E.M. (1996). Learning playing strategies in chess, *Computational Intelligence, 12*(1), 65–87.

Muggleton, S.H. (1988). Inductive acquisition of chess strategies. In J.E. Hayes, D. Michie & J. Richards (Eds.), *Machine Intelligence 11: Logic and the Acquisition of Knowledge* (pp. 375–389). Oxford.

Newell, A. and Simon, H. A. (1972). *Human Problem Solving*, Englewood Cliffs, N.J.: Prentice-Hall.

Samuel, A.L. (1959). Some studies in machine learning using the game of checkers, *IBM Journal*, July.

Shannon, C.E. (1950). Programming a computer for playing chess, *Philosophical Magazine, 41*, 256–275.

Thrun, S. (1995). Learning to play the game of chess. In Tesauro, G., Touretzky, D. and Leen, T. (Eds.), *Advances in Neural Information Processing Systems 7*, Massachusetts: MIT Press.

Walczak, S. (1992). Pattern-Based Tactical Planning. *International Journal of Pattern Recognition and Artificial Intelligence, 6*, 955–988.

Appendix. Quiescent 4-ply Search Likelihood Function

Some definitions:

$P_{i_1 i_2 \ldots i_d}$ = position resulting from P after sequence of moves $i_1, i_2, \ldots, i_d$.

$$\sigma_h = \sigma_h(P) = \begin{cases} +1, \text{ if Black is to move in a position } h \text{ ply distant from } P; \\ -1, \text{ if White is to move in a position } h \text{ ply distant from } P. \end{cases}$$

So, if White is to move in position P, then $\sigma_1 = +1$, $\sigma_2 = -1$, $\sigma_3 = +1$, ...

k_h = absolute value of won position at depth h from P
$= 600 - 10h$
(a little arbitrarily, but so as to make early checkmates better than late ones);

For $h \geq 1$, define

$$v_h(P_{i_1 i_2 \ldots i_h}) = \begin{cases} 0, & \text{if } P_{i_1 i_2 \ldots i_h} \text{ is a Draw of some kind, e.g. Stalemate;} \\ k_h, & \text{if } P_{i_1 i_2 \ldots i_h} \text{ is a won position (i.e. checkmate) for White;} \\ -k_h, & \text{if } P_{i_1 i_2 \ldots i_h} \text{ is a won position for Black;} \\ \sigma_h(P) v(P_{i_1 i_2 \ldots i_h}), & \text{if } \textit{either } P_{i_1 i_2 \ldots i_h} \text{ is quiescent } \textit{or} \text{ it is a terminal non-won position, i.e. } h = d; \\ \dfrac{\sum_{i_{h+1}} v_{h+1}(P_{i_1 i_2 \ldots i_{h+1}}) \exp\left(\sigma_{h+1}(P) v_{h+1}(P_{i_1 i_2 \ldots i_{h+1}})\right)}{\sum_{i_{h+1}} \exp\left(\sigma_{h+1}(P) v_{h+1}(P_{i_1 i_2 \ldots i_{h+1}})\right)}, & \text{if } h < d \text{ and } P_{i_1 i_2 \ldots i_h} \text{ is neither won nor quiescent.} \end{cases}$$

Thus $v_h(_)$ assigns values to positions which are h-ply distant from P. Recall that $v(_)$ is the ordinary evaluation function. The summations are over all moves i_{h+1} that can be made in position $P_{i_1 i_2 \ldots i_h}$. Note that $v_d(_)$ never equals the last of the possibilities listed above, since all non-won non-quiescent positions at d-ply are evaluated using $v(_)$.

The move from position P is chosen probabilistically according to

$$\Pr(\text{move } i^*(P)) = \frac{\exp\left(\sigma_1(P) v_1(P_{i^*})\right)}{\sum_{i=1}^{m} \exp\left(\sigma_1(P) v_1(P_i)\right)}.$$

In the implementation reported in Section 3 of this paper, we use $d = 4$, and P itself is always fully searched to at least depth 1 (so is treated as non-quiescent).

This gives our likelihood for a single position P. For a set of positions, $\mathcal{P}$, we use the likelihood

$$\prod_{P \in \mathcal{P}} \Pr(\text{move } i^*(P)).$$