Available online at www.sciencedirect.com
ScienceDirect
Pattern Recognition Letters 29 (2008) 1154–1159
Pattern Recognition Letters
www.elsevier.com/locate/patrec

BayesChess: A computer chess program based on Bayesian networks ☆

Antonio Fernández *, Antonio Salmerón

Department of Statistics and Applied Mathematics, University of Almería, Almería, Spain

Available online 6 July 2007

Abstract

In this paper, we introduce a chess program able to adapt its game strategy to its opponent, as well as to adapt the evaluation function that guides the search process according to its playing experience. The adaptive and learning abilities have been implemented through Bayesian networks. We show how the program learns through an experiment consisting on a series of games that point out that the results improve after the learning stage.

Keywords: Bayesian networks; Adaptive learning; Computer chess

1. Introduction

Bayesian networks are known as an appropriate tool for modeling in scenarios where a high number of variables take part and there is uncertainty associated to their values (Gámez et al., 2004; Jensen, 2001). One of the problems in which the use of Bayesian networks is specially important is the *classification or pattern recognition* problem. This is connected to the construction of systems able to adapt themselves to the user, since it is necessary to determine the kind of user in order to act in consequence.

In this paper, we describe a computer chess program able to adapt itself to the user and adjust its game strategy according to the user's style. Furthermore, it learns from its own experience, by refining the evaluation function used in the search through the tree of movements. These functionalities have been implemented using Bayesian networks. More precisely, we have used classification-oriented Bayesian networks, based on the *Naive Bayes* structure, specially appropriate in problems involving a high number of variables and with a learning database with limited size.

Our aim is not to achieve a really competitive program as Deep Blue (Newborn, 1997) or Fritz (Muller, 2002), but rather to test the suitability of Bayesian networks for constructing adaptive systems. We have chosen computer chess because it has some features that can be properly handled using adaptive systems:

- The program constantly interacts with the user and it has to respond to his/her actions.
- The impossibility of calculating all the possible moves motivates the use of heuristics.
- The validity of the used heuristics can be tested according to the obtained results.
- There are different playing styles or strategies that can be adopted during a game.

Therefore, the aim of this work is the use of Bayesian networks to provide the program with the ability of being adaptive. More precisely, we have focused on:

(1) Refining the search heuristic, according the program's playing experience, using a Bayesian network.
(2) Use of a Bayesian network to classify the user's behaviour and adopt the appropriate strategy.

☆ Supported by the Spanish Ministry of Education and Science, project TIN2004-06204-C03-01 and FEDER funds.
* Corresponding author. Fax: +34 950015167.
E-mail addresses: afalvarez@ual.es (A. Fernández), antonio.salmeron@ual.es (A. Salmerón).

0167-8655/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2007.06.013

We believe that the adaptation capability would be a valuable feature for sophisticated chess programs like Fritz. The last human–machine competitions have shown that commercial computer chess programs are able to beat almost any professional chess player, which means that computers have reached a really remarkable playing strength. However, many human players still think that computer chess programs play in a monotone and boring way, and it is motivating a lack of interest in purchasing those kind of programs. The feature of adapting to the user's style, in order not to show a monotone behaviour, can be an appealing feature for this software.

The rest of the paper is organised as follows: In Section 2, we review some basic concepts on Bayesian networks and classification. The design of the playing engine and the search heuristics is described in Section 3. The automatic update of the heuristic is explained in Section 4, while the adaptation to the user's behaviour is the aim of Section 5. The experiments carried out to evaluate the learning process is described in Section 6, and the paper ends with conclusions in Section 7.

2. Bayesian networks and classification

Consider a problem defined by a set of variables $\boldsymbol{X} = \{X_1, \ldots, X_n\}$. A Bayesian network (Jensen, 2001; Pearl, 1988) is a directed acyclic graph where each node represents a problem variable and has an associate probability distribution of the variable it contains given its parent in the graph. The presence of an arc between two variables expresses the existence of dependence between them, which is quantified by the conditional distribution assigned to the nodes. From a computational point of view, an important property of Bayesian networks is that the joint distribution over the variables in the network factorises according to the concept of d-separation (Pearl, 1988) as follows:

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | pa(x_i)), \tag{1}$$

where $pa(X_i)$ denotes the set of parents of variable X_i and $pa(x_i)$ is a configuration of values for them. This factorisation implies that the joint distribution for all the variables in the network can be specified with an important reduction of space requirements. For instance, the joint distribution over the variables in the network displayed in Fig. 1, assuming that all the variables are binary, would require the storage of $2^5 - 1 = 31$ values, while making use of the factorisation, the same information can be represented using just 11 values (see Table 1).

A Bayesian network can be used for classification purposes if it contains a class variable, C, and a set of feature variables $X_1, \ldots, X_n$, so that an object with observed features $x_1, \ldots, x_n$ will be classified as belonging to class c^* obtained as

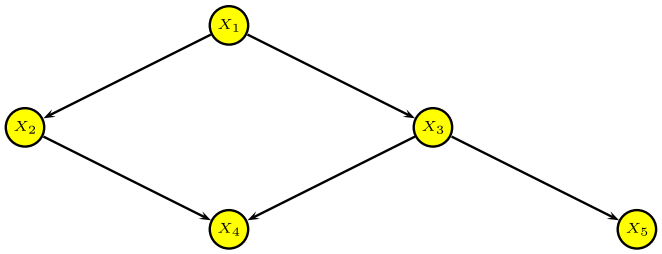

Fig. 1. A sample Bayesian network.

Table 1
Example of factorised distribution for the network in Fig. 1

$p(X_1 = 0) = 0.20$	$p(X_2 = 0 \mid X_1 = 0) = 0.80$
$p(X_2 = 0 \mid X_1 = 1) = 0.80$	$p(X_3 = 0 \mid X_1 = 0) = 0.20$
$p(X_3 = 0 \mid X_1 = 1) = 0.05$	$p(X_4 = 0 \mid X_2 = 0, X_3 = 0) = 0.80$
$p(X_4 = 0 \mid X_2 = 1, X_3 = 0) = 0.80$	$p(X_4 = 0 \mid X_2 = 0, X_3 = 1) = 0.80$
$p(X_4 = 0 \mid X_2 = 1, X_3 = 1) = 0.05$	$p(X_5 = 0 \mid X_3 = 0) = 0.80$
$p(X_5 = 0 \mid X_3 = 1) = 0.60$	

$$c^* = \arg\max_{c \in \Omega_C} p(c | x_1, \ldots, x_n), \tag{2}$$

where Ω_C denotes the set of possible values of class variable C.

Note that $p(c|x_1, \ldots, x_n)$ is proportional to $p(c) \times p(x_1, \ldots, x_n|c)$, and therefore, solving the classification problem would require to specify a distribution over the n feature variables for each value of the class. The associate cost can be very high. However, using the factorisation determined by the network, the cost is reduced. The extreme case is the so-called *Naive Bayes* model (see, for instance Duda et al., 2001), where it is assumed that the feature variables are independent given the class. This fact is represented by an structure as the one displayed in Fig. 2.

The strong independence assumption beneath this model is compensated by the reduction in the number of parameters to be estimated, since in this case it holds that

$$p(c | x_1, \ldots, x_n) = p(c) \prod_{i=1}^{n} p(x_i | c) \tag{3}$$

and thus, instead of one n-dimensional conditional distribution, we have n one-dimensional conditional distributions.

3. Design of the chess playing engine

The game of chess has been deeply studied by Artificial Intelligence. In this work, we have considered a playing

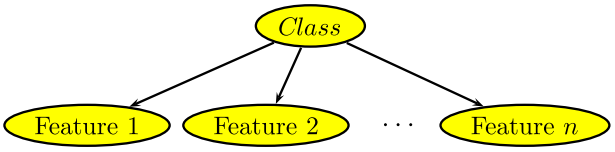

Fig. 2. Structure of a Naive Bayes classifier.

engine based on the mini–max algorithm with alpha–beta pruning. There are more sophisticated search algorithms oriented to chess, but they are outside the scope of this work. However, the search heuristic is actually relevant since, as we will describe later, it will be updated using a Bayesian network learnt from the experience of the program itself.

The heuristic we have chosen is based upon two issues: material (the pieces on the board) and the location of each piece (depending on the square where a piece is placed, it can be more or less valuable). Additionally, we have also given importance to the fact of setting the opponent's king under check, as it drastically reduces the number of possible moves.

The evaluation of the material on the board is carried out by assigning a score to each piece. We have chosen the most usual found in chess programs, which is displayed in Table 2. The king is not associated with any particular score, since it must be present in any valid configuration of pieces on the board.

Regarding the evaluation of the position of each piece, we have used an 8×8 matrix for each piece, so that each cell contains the value which is added to the heuristic in the case that the corresponding piece is placed on its corresponding square. Table 3 shows an example of this kind of matrix, for the case of a white knight. Notice that the location of the knight in central squares is encouraged, since it increases its scope.

In overall, the heuristic function is defined in terms of 838 parameters, that correspond to the value for each piece on the board, the value of setting the opponent's king under check and the number stored in the 8×8 matrices. More precisely, there are five parameters indicating the value of each piece (pawn, queen, rook, knight and bishop – the king is not evaluated, as it must always be on the board), 1 parameter for controlling whether the king is under check, 64 parameters for evaluating the location of each piece on each square on the board (i.e., a total of 786 parameters, corresponding to 64 squares $\times$ 6 pieces each colour $\times$ 2 colours) and finally, 64 more parameters that are used to evaluate the position of the king on the board during the end-game. This last situation is considered separately because the behaviour of the king should be different depending on the game stage. In general, it is not recommendable that the king advances during the opening, but it can be a good idea during the end-game, since it can support the advance of the pawns.

Table 2
Score for the pieces employed by our proposed heuristic

Piece	Pawn	Bishop	Knight	Rook	Queen
Score	100	300	300	500	900

Table 3
Weights associated to the location of a white knight

-10	-10	-10	-10	-10	-10	-10	-10
-10	0	0	0	0	0	0	-10
-10	0	5	5	5	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	10	10	5	0	-10
-10	0	5	5	5	5	0	-10
-10	0	0	0	0	0	0	-10
-10	-30	-10	-10	-10	-10	-30	-10

4. Automatic update of the heuristic

In this section, we describe the process of refinement of the parameters in the heuristic defined in Section 3. Along with the development of machine learning techniques in the decade of the eighties, their application to computer chess was considered, but the conclusion was that they could only be applied in a marginal way, as for extraction of patterns from openings books (Skiena, 1986). However, afterwards some applications of classification techniques were developed, mainly for the evaluation of positions from end-games (Fürnkranz, 1996).

With the aim of updating the parameters in the heuristic, we have considered a Bayesian network based on the Naive Bayes structure, but with the difference that instead of one class variable, in this case there are two of them: the *current game stage* (opening, middle-game or end-game) and the *result* (win, lose, draw). As feature variables, we have included all the parameters in the heuristic as described in Section 3, which means that the network has 840 variables arranged as shown in Fig. 3. The high number of variables is the fact that motivates the use of a structure similar to the Naive Bayes model, since the use of a more complex structure would increase the time spent to evaluate the heuristic, slowing down the exploration of the search tree. The drawback of this choice is that the independence assumption can be little realistic, but this is somehow compensated by the reduction on the number of free parameters that have to be estimated from data.

The parameters in the Bayesian network are initially estimated from a database generated making BayesChess play against itself, employing one of the players the heuristic as defined before, and the other one using a randomly perturbed heuristic, where the value of each variable is ran-

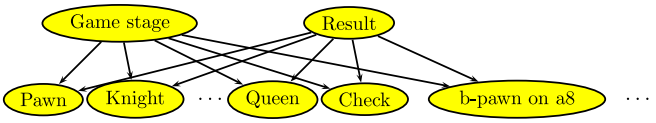

Fig. 3. Structure of the network used to learn the heuristic.

Table 4
Sample games database

Game stage	Pawn	Knight	Bishop	Rook	Queen	Check	Pawn a8	Pawn b8	...	Result
Opening	120	180	240	700	540	42	−6	0	...	Lost
Mid	120	360	360	600	1260	36	3	0	...	Lost
End	120	420	180	400	720	42	−3	3	...	Lost
Opening	140	360	420	300	1080	18	0	6	...	Win
Mid	100	300	360	500	1260	42	3	0	...	Win
End	80	180	240	400	900	42	6	6	...	Win
Opening	120	420	420	400	720	18	−6	3	...	Draw
Mid	140	300	360	500	1260	42	3	0	...	Draw
End	120	420	180	600	900	30	0	6	...	Draw

domly increased or decreased a 20%, 40% or kept to its initial value. Table 4 shows the format of the database with some sample records. We can see how there is a record for each stage in a game, containing the value of the parameters used by the random heuristic, ending with the result of the game.

Each probability table in this Bayesian network requires the estimation of 45 values, since for each one of the five possible values of each variable, we must consider the game stage and the result (actually, only 36 values are required, as the remaining nine can be computed from the restriction that the probabilities in each column must sum up to 1). Table 5 shows an example of the probability table for variable *Pawn*.

The learning process is not limited beforehand: It depends on the number of games recorded in the database. Therefore, the more games we have the more accurate the parameter estimation will be. Once the initial training is concluded, BayesChess can adopt the learnt heuristic and, from them on, refine it with new games, now against human opponents.

After the Bayesian network has been constructed, BayesChess uses it to choose the parameters in the heuristic. The selection is carried out by instantiating both class variables (game stage and result) and computing the configuration of parameters that maximise the probability of the instantiated values. In order to determine in which stage the game is, we have considered that the opening comprises the first 10 moves, while the end-game is reached when there are no queens or the number of pieces on the board is lower than 10. Otherwise, we consider that the stage is the middle-game. Regarding variable *result*, it can be used to determine the playing style that BayesChess will adopt. For instance, if we instantiate that variable to value *win*, BayesChess will choose the configuration of values for the parameters that maximise the probability of winning, even if that probability is lower than the probability of not winning (losing + draw). It means that the program adopts an aggressive strategy. On the other hand, we can choose to minimise the probability of losing, i.e., maximising the sum of the probabilities of winning or reaching a draw. This corresponds to a conservative strategy. The way in which these configurations are obtained is through abductive inference (de Campos et al., 2003; Nilsson, 1998). In the particular case of the network used by BayesChess, the configurations can be easily obtained, since the configuration that maximises the probability of a given instantiation is obtained by taking the value for each individual variable with higher probability, due to the factorisation given in Eq. (3).

Table 5
Example of probability table for variable *Pawn*

Pawn	Game stage Result	O W	O L	O D	M W	M L	M D	E W	E L	E D
	60	0.2	0.1	0.3	0.3	0.2	0.3	0.2	0.3	0.2
	80	0.3	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.2
	100	0.1	0.1	0.2	0.4	0.2	0.1	0.1	0.1	0.2
	120	0.1	0.2	0.4	0.1	0.3	0.1	0.3	0.2	0.3
	140	0.3	0.5	0.1	0.2	0.1	0.4	0.2	0.3	0.1

5. Adaptation to the opponent's situation

Now we will discuss how to make the heuristic adapt the strategy to the opponent style. With this aim we have considered three types of playing styles that the user can be employing: attacking, positional or mixed.

We have implemented a Naive Bayes classifier to determine the way in which the opponent is playing in a given moment, taking as basis these features of the opponent: first move, situation of castles (opposed or not), number of pieces beyond the third row (except pawns) and the number of pawns advanced towards the king. The structure of the classifier is depicted in Fig. 4.

The classifier has been trained using a database of games from four well-known professional players, corresponding

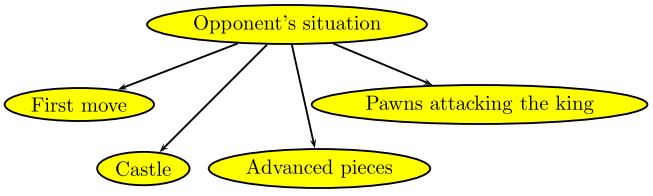

Fig. 4. Structure of the classifier used to determine the opponent's style.

Table 6
Sample database for learning the classifier of the opponent's style

First move	Castle	Advanced pieces	Pawns towards king	Style
e4	Equal	2	1	Attacking
Cf6	Opposed	0	2	Attacking
Cf3	Equal	0	1	Mixed
d4	Equal	1	0	Positional
c5	Equal	0	0	Attacking
c4	Opposed	1	2	Positional
Other_b	Equal	1	1	Mixed

the considered styles. In all the games, the feature variables have been measured and included in the database. More precisely, we have selected 2708 games by Robert Fischer and Gary Kasparov as examples of attacking style, 3078 by Anatoli Karpov as positional play and 649 games by Miguel Illescas as examples of mixed player. Table 6 describes the format of the database.

Using this classifier, BayesChess determines the opponent's strategy by instantiating the feature variables and computing the value of variable *opponent situation* with highest probability.

5.1. The process of adapting to the opponent

Once the opponent style is determined, BayesChess decides its own strategy using the Bayesian network that contains the parameters of the heuristic in this way:

- When the opponent's style is *attacking*, it chooses the parameters in order to *minimise the probability of losing*.
- When the opponent's style is *positional*, it chooses the parameters in order to *maximise the probability of winning*.
- When the opponent's style is classified as *mixed*, it *randomly* chooses one of the former two strategies.

6. Experiments

We have carried out two experiments in order to evaluate the learning of the heuristic, in both of them using a database with 2000 games of the initial heuristic against a random one. In both experiments, the heuristic is incrementally learnt. The version of BayesChess that we used in these experiments is available at http://elvira.ual.es/algra.

The experiment consisted of 11 matches of 200 games between BayesChess with random heuristic against itself with the learnt heuristic with different number of games in the learning database. We can see in Fig. 5 how the learnt heuristic improves its results as the number of games increases.

The second experiment consisted of evaluating the score assigned by the heuristic to a given position, more precisely to the position shown in Fig. 6, with different number of

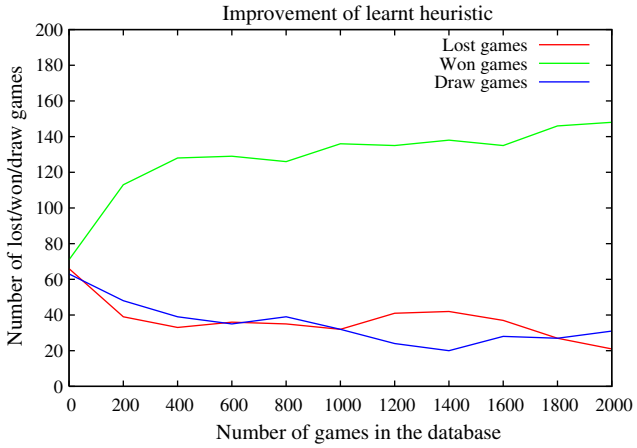

Fig. 5. Evolution of the results of playing the learnt heuristic against the random one.

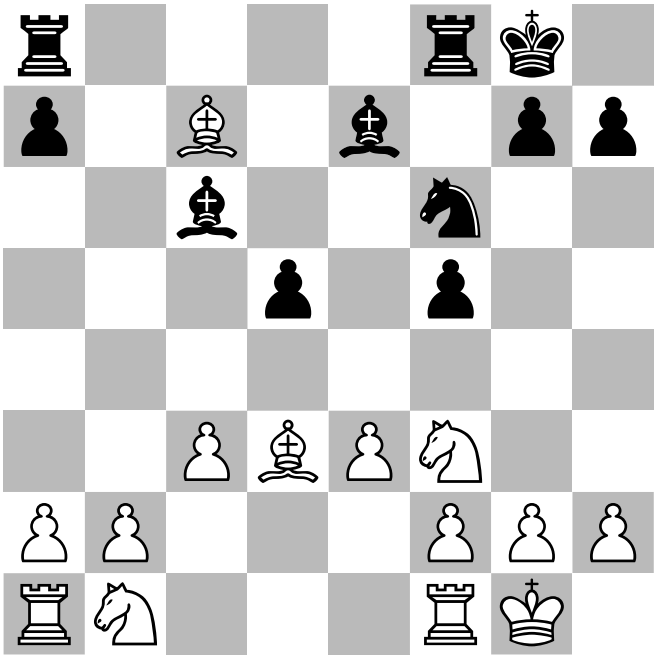

Fig. 6. Sample board for the second experiment.

games in the database. It can be seen that in Fig. 6, the white player has one knight and two pawns above the black player, and therefore, the evaluation of the position should be around 500 points of advantage for white. Fig. 7 shows how the learnt heuristic actually approaches to that value when the database grows.

In both experiments, it can be observed that the performance of BayesChess improves very quickly as the number of games used to learn the heuristic increases. However, once the training database reaches a certain size (around 400 games in the experiments), the behaviour improves much more slowly. We think that this is due to the kind of Bayesian network structure that we use to update the heuristic, which probably reaches close to its maximum

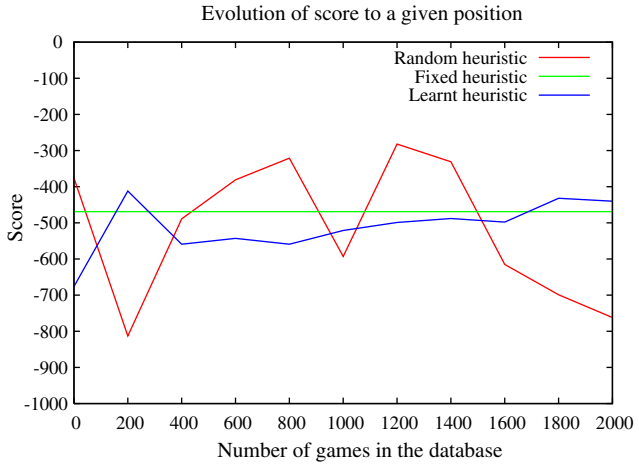

Fig. 7. Evolution of the heuristic as the size of the database grows.

accuracy quickly and after that point it is only slightly refined. This suggests that a more complex structure could be used when the training database is large, in order to reach a higher degree of refinement.

7. Conclusions

In this paper, we have introduced BayesChess, a computer chess program that adapts its behaviour according to its opponent and its previous experience. The results of the experiments carried out suggest that after learning, the results improve, and the heuristic is adjusted with more accurate parameter values.

We think that the use of Bayesian networks is an added value in the construction of adaptive systems. In cases like BayesChess, where the number of variables involved is very high, they allow to carry out the necessary inferences efficiently, using restricted network topologies as the Naive Bayes.

Not only chess, but also other computer games that require the machine to make decisions, can obtain benefits from the use of Bayesian networks in the way described in this paper. An immediate example is the game of checkers, but it must be taken into account that the complexity of that game is much lower and the heuristic would not contain so many variables.

In the next future we plan to improve BayesChess by refining our implementation of mini–max and by introducing an end-game classifier. In this sense, there are databases with data about typical positions in end-games with rooks and pawns, classified as winning, loser, or oriented to draw, and that can be used to train a classifier (Blake and Merz, 1998).

References

Blake, C.L., Merz, C.J., 1998. UCI repository of machine learning databases, University of California, Irvine, Dept. of Information and Computer Sciences. <http://www.ics.uci.edu/mlearn/MLRepository.html>.

de Campos, L.M., Gámez, J.A., Moral, S., 2003. Partial abductive inference in Bayesian networks by using probability trees. In: Proc. 5th Internat. Conf. on Enterprise Information Systems (ICEIS'03), Angers, pp. 83–91.

Duda, R.O., Hart, P.E., Stork, D.G., 2001. Pattern Classification. Wiley Interscience.

Fürnkranz, J., 1996. Machine learning in computer chess: The next generation. ICCA J. 9 (3), 147–161.

Gámez, J.A., Moral, S., Salmerón, A., 2004. Advances in Bayesian Networks. Springer, Berlin,Germany.

Jensen, Finn V., 2001. Bayesian Networks and Decision Graphs. Springer.

Muller, K., 2002. The clash of the titans: Kramnik-FRITZ Bahrain. ICGA J. 25, 233–238.

Newborn, M., 1997. Kasparov vs. Deep Blue: Computer Chess Comes of Age. Springer-Verlag.

Nilsson, D., 1998. An efficient algorithm for finding the *M* most probable configurations in Bayesian networks. Statist. Comput. 9, 159–173.

Pearl, J., 1988. Probabilistic Reasoning in Intelligent Systems. Morgan-Kaufmann, San Mateo.

Skiena, S.S., 1986. An overview of machine learning in computer chess. ICCA J. 9 (1), 20–28.