Learning Positional Features for Annotating Chess Games: A Case Study

Matej Guid, Martin Možina, Jana Krivec, Aleksander Sadikov, and Ivan Bratko

Artificial Intelligence Laboratory, Faculty of Computer and Information Science,
University of Ljubljana, Slovenia

Abstract. By developing an intelligent computer system that will provide commentary of chess moves in a comprehensible, user-friendly and instructive way, we are trying to use the power demonstrated by the current chess engines for tutoring chess and for annotating chess games. In this paper, we point out certain differences between the computer programs which are specialized for playing chess and our program which is aimed at providing quality commentary. Through a case study, we present an application of argument-based machine learning, which combines the techniques of machine learning and expert knowledge, to the construction of more complex positional features, in order to provide our annotating system with an ability to comment on various positional intricacies of positions in the game of chess.

1 Introduction

The ever stronger chess programs are dangerous opponents to human grandmasters - already surpassing them in many aspects. In spite of that, their capabilities to *explain* why certain moves are good or bad in a language understandable to humans are very limited. So far, very little attention was paid to an automatic intelligent annotation of chess games and consequently the progress made in this field is negligible in comparison to the enormous progress in the power of chess engines, which we have witnessed in the last decades. The typical "commentary" in the form of best continuations and their numerical evaluations can hardly be of much help to the chess-player who would like to learn the important concepts that are hidden behind the suggested moves.

For several years, the scientific research in this field was limited only to chess endgames, and the demonstrated concepts all had a common weakness - the inability to practically extend annotations to the entire game of chess[1]. Finally, we introduced a new approach, which utilizes the power demonstrated by the current chess engines to provide commentary of chess games during all the phases of the game and besides the ability of annotating tactical positions also enables commenting on various strategic concepts in given positions. Our approach to annotating chess games automatically was introduced in Sadikov *et al.* [1]. The main idea is to use the chess engine's evaluation function's features to describe

[1] An interested reader can find an overview of the related work in Sadikov *et al.* [1].

the changes in the position when a move (or a sequence of moves) is made. These elementary features can later be combined to form higher-level concepts understandable to humans. The descriptions can be used both for the purpose of tutoring chess by providing knowledge-based feedback to students, and for the purpose of annotating chess games.

As we intend to demonstrate, evaluation functions of the programs that are aimed to *play* chess successfully do not need to be aware of all the concepts that would otherwise be useful for commenting on chess games. Hence, introducing additional positional features to the evaluation function of our annotating software turned out to be very important to obtain comprehensible, user-friendly and instructive annotations.

Introducing new knowledge into the evaluation function of the program requires knowledge elicitation from a chess expert, which proved to be a difficult task. Computer researchers usually find it difficult to elicit useful knowledge from human experts. This *knowledge acquisition bottleneck* occurs because the knowledge is intuitive. The experts are quite capable of using their domain knowledge, but they find it much harder to formalise it and describe it systematically. Overcoming the knowledge acquisition bottleneck was of crucial importance and also the key motivation for introducing machine learning to construct additional positional features. In this paper, we present an application of a recent approach to machine learning - Argument Based Machine Learning (ABML, [2]) - by a case study in learning the positional feature bad bishop in a form suitable for annotating chess games. Such features may not always be suitable for evaluation functions of chess-playing programs, where time spent on heuristic search is very important (besides, appropriate weights of these features should be determined). Nevertheless, they could serve well for annotation purposes.

The paper is organized as follows. In Section 2, we point out certain differences between positional features of a program which is specialized for playing chess and the program which is intended for providing quality commentary. In Section 3, through a case study "*The Bad Bishop*", we describe the construction process of a more complex positional feature that enables our annotating software to detect and comment on bad bishops. Section 4 concludes the paper and describes our future work.

2 Positional Features for Annotating Chess Games

The programs that are aimed to *play* chess successfully do not need to be aware of all the concepts that would otherwise be useful for giving instructive annotations. There is always a dilemma how much knowledge to implement into evaluation functions of the programs in order to achieve best tournament performances. The more knowledge means less time for efficient search and vice versa. It is commonly known that some programs have more knowledgeable evaluation functions, while others rely more on efficient search algorithms that allow them to reach higher search depths.

To illustrate our points, we will introduce a concept of *the bad bishop*. Watson [3] gives the following definition as traditional one: A bishop that is on the same colour of squares as its own pawns is bad, since its mobility is restricted by its own pawns and it does not defend the squares in front of these pawns. Moreover, he puts forward that centralisation of these pawns is the main factor in deciding whether the bishop is bad or not. In the middlegame, he continues, the most important in this aspect are d and e pawns, followed by c and f pawns, while the rest of the pawns on the same colour of a square as the bishop, are irrelevant (up to the endgame, where they might again become an important factor for determining the goodness of the bishop).

The example in Fig. 1 is taken from the classic book by Aaron Nimzovich, *The Blockade* [4]. The black bishop on c8 is bad, since its activity is significantly hindered by its own pawns. Furthermore, these pawns are blockaded by the pieces of his opponent, which makes it even harder for black to activate the bishop.

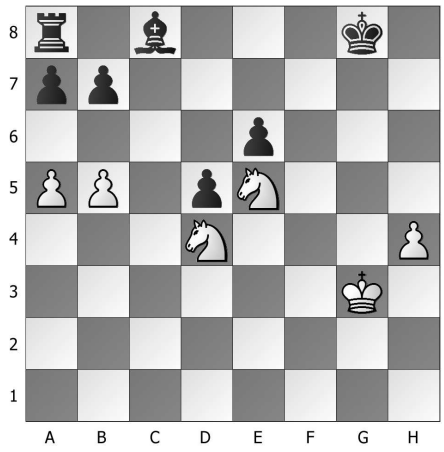

Fig. 1. Classical example of a bad bishop.

Chess program CRAFTY, for example, has several positional features that are associated with the goodness of the bishops, but they are insufficient to fully describe this concept. They apply to both bishops for one side at the same time, i.e., the values for both bishops are represented by one feature only. Even if we arrange to obtain the feature values for each bishop separately, these positional features are still not appropriate to describe the goodness of a bishop, with the aim to annotate chess games in an instructive way.

Table 1 shows the most relevant CRAFTY's positional features for describing a bad bishop. As it becomes clear from descriptions of these features and their deficiencies from the purpose of describing bad bishops, CRAFTY clearly could not be aware of such a concept. For example, if we move pawns e6 and d5 to g6 and h7 (preserving the value of BLACK_BISHOP_PLUS_PAWNS_ON_COLOR - since pawns on the same colour of the square as the bishop carry the same penalty, regardless of their position) and the rook from a8 to d7 (the value of

Table 1. Some CRAFTY's positional features that have a potential for describing bad bishops and their deficiencies for doing so successfully.

Feature	Description	Deficiency for annotating
BLACK_BISHOP _PLUS_PAWNS _ON_COLOR	a number of own pawns that are on the same colour of the square as the bishop	all such pawns count the same, regardless of their position and how badly they restrict the bishop
BLACK_BISHOPS _POSITION	an evaluation of the bishop's position based on predefined values for particular squares	such predefined value is not the actual value of the bishop's placement in a particular position
BLACK_BISHOPS _MOBILITY	the number of squares that the bishop attacks	the number of attacked squares and the actual bishop's mobility are not necessarily the same thing

BLACK_BISHOPS_MOBILITY even decreases, as the bishop is attacking one square less), the bishop clearly would not be bad, but in CRAFTY's view it would be even worse than in the given position.

In order to introduce a new positional feature (say BAD_BISHOP) that would allow commenting on such complex concepts, as is the concept of the bad bishop, it is therefore essential first to obtain additional (simple) positional features, and then combining them into some kind of rules that would allow to obtain the value of the new (more complex) positional feature BAD_BISHOP.

It should be noted that when annotating chess games, it is not necessary to comment on a particular feature each time it occurs. When the annotator is not sure about some feature in the position, it is better to say nothing at all than giving wrong comments.

2.1 The Static Nature of Positional Features

Positional features are static in their nature - they describe the state of their purposed issue for the current position only. It is heuristic search that enables them to fulfil their purpose - contributing to the program finding the best moves. For example, in the position in Fig. 1, if we moved the knight from e5 to h1, and decided that black is to move, black would easily solve all his problems by playing e6-e5, chasing the other white knight away and freeing both of his pieces. The positional features from Table 1, among others, would contribute to deciding for this freeing move, since the values of all three attributes become more desirable soon along the principal variation (e.g., after e6-e5 and Bc8-f5, there are less pawns on bishop's square colour, and the bishop itself is placed on a square with a higher predefined value and also attacks more squares). Although the mentioned positional features are not suitable for commenting on the bad bishop, they nevertheless help it to become a good one.

It is also desirable for positional features for annotating chess games to be of static nature. For example, it is up to the chess engine to determine whether the freeing move e6-e5 is possible or not (e.g., in case of white king on f4 and the e5 knight still on h1 it would drop at least a pawn).

3 Case Study: The Bad Bishop

In this case study, we demonstrate the construction of a static positional feature, BAD_BISHOP (with possible values *yes* or *no*), which was designed for commenting on bad bishops (possibly combined with some heuristic search).

In our domain, it turns out to be extremely difficult for a chess expert to define appropriate rules, using typical positional features, for the program to be able to recognise complex concepts, such as *the bad bishop*. Our domain experts[2] defined the rules, using CRAFTY's positional features only, which in their opinion described bad bishops in the best possible way (considering the constraint of having only CRAFTY's features at disposal). The rules were of the following type:

```
IF (|BLACK_BISHOP_PLUS_PAWNS_ON_COLOR| > X)
AND (|BLACK_BISHOPS_MOBILITY| < Y) THEN BAD_BISHOP = yes
```

Three such rules were given, depending on the number of black pawns in the position. The positional features and the values for X and Y were determined, after the experts got acquainted with the exact meaning of CRAFTY's positional features and observed their values in several positions of various types. However, after examining the outcome of these rules on various chess positions, it turned out that the rules performed rather poorly, which was the key motivation for introducing machine learning into the system's development.

3.1 The Learning Dataset

The learning dataset consisted of middlegame positions[3] from real chess games, where the black player has only one bishop. Based on the aforementioned expert-crafted rules, positions were obtained automatically from a large database of chess games. The bishops were a subject of evaluation by the experts.

When chess experts comment on concepts such as the bad bishop, they also have dynamic aspects of a position in mind. Therefore, assessing bishops "statically" is slightly counter-intuitive from the chess-player's point of view. After a careful deliberation, the following rules were chosen for determining a bad bishop from the static point of view.

The bishop is *bad* from the static point of view in some position, if:

1. its improvement or exchange would notably change the evaluation of the position in favour of the player possessing it,
2. the pawn structure, especially the one of the player with this bishop, notably limits its chances for taking an active part in the game,
3. its mobility in this position is limited or not important for the evaluation.

[2] The chess expertise was provided by WGM Jana Krivec and FM Matej Guid.

[3] While the concept of the bad bishop hardly applies to the early opening phase, different rules for determining bad bishops apply in the endgames (see Watson's definition in Section 2). In all positions, quiescence criterion was satisfied.

These rules seem to be in line with the traditional definition of the bad bishop, and in the experts' opinion lead to sensible classification - in positions where assessment from the static point of view differs from the one obtained from the usual (dynamic) point of view, it seems very likely that possible implementation of heuristic search, using the newly obtained positional feature BAD_BISHOP (such search could enable the program to realise whether the bishop is more than just temporarily bad and thus worth commenting on), would lead to sensible judgement on whether to comment on the bad bishop or not.

The learning dataset consisted of 200 positions[4]. We deliberately included notably more bishops labelled as "bad" by the initial rules given by the experts, due to our expectations (based on the unsuitability of CRAFTY's positional features) that many of the bishops labelled as "bad" will not be assessed so after the experts' examination of the positions. After examination by the experts, 80 examples in the dataset obtained the class value *yes* ("bad") and 120 examples obtained the class value *no* ("not bad").[5] It turned out that only 59% of the examples were correctly classified by the expert-crafted rules.

3.2 Machine Learning

As the expert-crafted rules scored only 59% classification accuracy on our dataset, which is clearly insufficient for annotating purposes, there is a clear motivation for the use of machine learning. However, as classification accuracy equally penalises false positives ("not bad" classified as "bad") and false negatives, we should also use precision, which measures the percentage of true "bad" bishops among ones that were classified as "bad". Remember, falsely commenting is worse than not commenting at all.

From the many available machine learning methods we decided to take only those that produce understandable models, as it will be useful later to be able to give an explanation why a bishop is bad and not only labeling it as such. We chose standard machine learning methods given in Table 2. We also give accuracy and precision results of these methods on our learning set.

Table 2. The machine learning methods' performance with CRAFTY's features.

Method	**Classification accuracy**	**Precision**
Decision trees (C4.5)	71%	64%
Logistic regression	80%	76%
Rule learning (CN2)[6]	73%	75%

All the accuracy and precision results were obtained through 10-fold cross validation. All the methods achieve better accuracies than expert given rules, but they are still too inaccurate for commenting purposes.

[4] This number was chosen as the most feasible one, considering limited available time of the experts. The quality of the final model implies that the number of selected positions in the learning dataset was high enough.

[5] An interested reader will find the learning dataset, and some other interesting details associated with this paper at the first author's website: http://www.ailab.si/matej.

3.3 Argument Based Machine Learning

Argument Based Machine Learning (ABML, [2]) is machine learning extended with some concepts from argumentation. Argumentation is a branch of artificial intelligence that analyses reasoning where arguments for and against a certain claim are produced and evaluated [5].

Arguments are used in ABML to enhance learning examples. Each argument is attached to a single learning example only, while one example can have several arguments. There are two types of arguments; positive arguments are used to explain (or argue) why a certain learning example is in the class as given, and negative arguments are used to explain why it should not be in the class as given. We used only positive arguments in this work, as negatives were not required. Examples with attached arguments are called *argumented examples*.

Arguments are usually provided by domain experts who find it natural to articulate their knowledge in this manner. While it is generally accepted that giving domain knowledge usually poses a problem, in ABML they need to focus on one specific case only at a time and provide knowledge that seems relevant for this case and does not have to be valid for the whole domain. The idea can be easily illustrated with the task of commenting on chess games. It would be hard to talk about chess moves in general to decide precisely when they are good or bad. However, if an expert is asked to comment on particular move in a given position, he or she will be able to offer an explanation and provide relevant elements of this position. Naturally, in a new position the same argument could be incorrect.

An ABML method is required to induce a theory that uses given arguments to explain the examples. Thus, arguments constrain the combinatorial search among possible hypotheses, and also direct the search towards hypotheses that are more comprehensible in the light of expert's background knowledge. If an ABML method is used on normal examples only (without arguments), then it should act the same as a normal machine learning method. We used the method ABCN2 [2], an argument-based extension of the well known method CN2 [6], that learns a set of unordered probabilistic rules from argumented examples. In ABCN2, the theory (a set of rules) is said to explain the examples using given arguments, when there exists at least one rule for each argumented example that contains at least one positive argument in the condition part[7].

In addition to rules, we need an inference mechanism to enable reasoning about new cases. Given the nature of the domain, we decided to learn only rules for "bad" bishop and classify a new example as "bad" whenever at least one of the learned rules triggered.

Asking experts to give arguments to the whole learning set is not likely to be feasible, since it requires too much time and effort. The following loop describes an iterative process for acquiring arguments and new attributes from experts:

[7] Due to space limitations, we only roughly described some of the properties of ABML and ABCN2 (see [2] or/and the website http://www.ailab.si/martin/abml for precise details).

1. Learn a set of rules.
2. Search for problematic cases in the data set; these are the examples that are misclassified by the induced rules.
3. If no problematic examples are found, stop the process.
4. Select a problematic example and present it to experts. If the case is a position with a "bad" bishop, then experts are asked to explain why this bishop is "bad". If it is a "not bad" bishop position, then we search for the culpable rule predicting "bad" and ask experts to explain an example with the class value *yes* ("bad") from the set of examples covered only by this rule. In the latter case, experts need to be careful to provide reasons that are not true in the problematic position. Problematic positions with a "not bad" bishop are called *counter-examples*.
5. Experts have three possibilities of responding to the presented case:
   (a) They can give reasons why bishop is "bad". Reasons are added to the example in the data set.
   (b) If they cannot explain it with given attributes, they can introduce a new attribute (or improve an existing one), which is then added to the domain.
   (c) Experts can decide that this bishop is actually "good" and thus the class of the example needs to be changed.

   If experts are unable to explain the example, we select another one.
6. Return to step 1.

Table 3 shows the three rules induced in the first iteration of the aforementioned process, where only CRAFTY's positional features were used and no arguments have been given yet. Condition part of a rule is the conjunction of the features indicated in the corresponding column. In the cases with no threshold specified, the feature is not part of the corresponding rule. For example, the rule #1 is: "IF BLACK_BISHOP_MOBILITY > -12 THEN BAD_BISHOP = yes".[8]

Table 3. The rules for BAD_BISHOP = *yes*, after the first iteration of the process.

Positional feature	**#1**	**#2**	**#3**
BLACK_BISHOPS_MOBILITY	> -12	> -18	> -18
BISHOP_PLUS_PAWN_ON_COLOR			> 12
BLACK_BISHOP_POSITION		> 4	
positive examples ("bad")	40	44	67
negative examples ("not bad")	4	7	25

The distribution of positive and negative examples covered by each of the rules speaks about the relatively poor quality of these rules - especially in the last rule.

[8] The negative values of BLACK_BISHOP_MOBILITY are the consequence of CRAFTY using negative values for describing features that are good for black. The more squares this bishop is attacking, the more negative this value. For each attacked square, the feature's value is decreased by -4. For example, the value of -12 means that the bishop is attacking three squares.

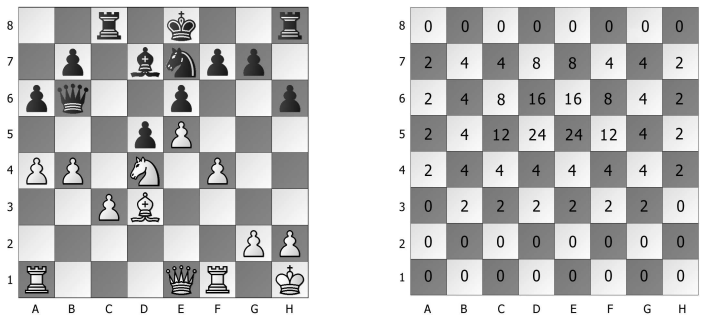

Fig. 2. The experts were asked the question: "*Why is the black bishop bad?*" They used their domain knowledge to provide the following answer: "*The black bishop is bad, since a lot of black pawns are on the same colour as the bishop. Especially the central pawns notably limit its chances for taking an active part in the game.*" The need for the attribute BAD_PAWNS was identified. The experts designed a look-up table with predefined values for the pawns that are on the colour of the square of the bishop in order to assign weights to such pawns.

Figure 2 (left) shows the first problematic example selected by our algorithm. The experts were asked to describe why the black bishop is bad. Based on their answer, another attribute, BAD_PAWNS, was added into the domain. The experts designed a look-up table (right) with predefined values for the pawns that are on the colour of the square of the bishop in order to assign weights to such pawns. According to Watson's definition, centralisation of the pawns has been taken into account. Several other attributes that were added at later stages (see Table 4) used this look-up table to heuristically assess an influence of such bad pawns on the evaluation of the bishop.

Table 4. The new attributes, and iterations when they were added to the domain.

Attribute	**Description**	**It.**
BAD_PAWNS	pawns on the colour of the square of the bishop - weighted according to their squares (*bad pawns*)	2
BAD_PAWNS_AHEAD	bad pawns ahead of the bishop	3
BLOCKED_DIAGONAL	bad pawns that block the bishop's (front) diagonals	4
BLOCKED_BAD_PAWNS	bad pawns, blocked by opponent's pawns or pieces	5
IMPROVED_BISHOP _MOBILITY	number of squares accessible to the bishop, taking into account only pawns of both opponents	6
BLOCKED_PAWNS _BLOCK_DIAGONAL	bad pawns, blocked by opponent's pawns or pieces, that block the bishop's (front) diagonals	12

Table 4 presents the list of attributes that were added to the domain during the process. To give an example of how the values of these new attributes are obtained, we will calculate the value of the attribute BAD_PAWNS_AHEAD for the position in Fig. 2. This attribute provides an assessment of pawns on the same colour of square as the bishop that are in front of it. There are three such pawns

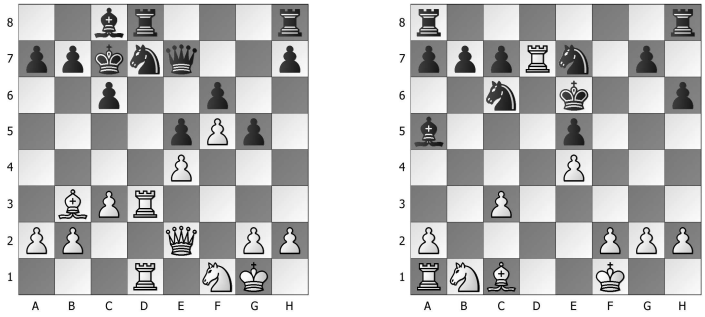

Fig. 3. After iteration 6, the expert gave the following description why the bishop is bad in position on the left: "*The bishop is bad, because, taking the pawn structure into account, only one square is accessible to it.*" The argument "IMPROVED_BISHOP_MOBILITY=low" was added to this position, based on this description. However, in the next iteration, the machine learning method selected the position on the right, where the bishop is classified as "not bad", as the counter-example. After the expert's examination, the following significant difference between the two positions was determined: in the position on the right, there are no bad pawns ahead of the bishop. Based on that, the argument to the position on the left was improved to "IMPROVED_BISHOP_MOBILITY=low AND BAD_PAWNS_AHEAD=high".

in that position: e6, d5, and a6. For each of these pawns their corresponding values are obtained from the look-up table, that is, 16, 24, and 2, respectively. The sum of these values ($16 + 24 + 2 = 42$) represents the value of the attribute BAD_PAWNS_AHEAD in that position.

Figure 3 shows an example how the argument given to some particular position could be improved by the expert, using some help by machine learning method, which automatically suggests him or her appropriate counter-example. The counter-examples are another effective feature for overcoming the knowledge acquisition bottleneck.

Table 5. The rules for BAD_BISHOP = *yes*, obtained after the 14^{th} (final) iteration.

Positional feature	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**
BAD_PAWNS					> 14		> 32
BAD_PAWNS_AHEAD	> 20	> 18		> 26	> 28	> 12	
BLOCKED_DIAGONAL	> 4		> 16				> 16
BLOCKED_BAD_PAWNS				> 0			
IMPROVED_BISHOP_MOBILITY		< 3	< 4	< 4		< 2	< 5
BLOCKED_PAWNS _BLOCK_DIAGONAL					> 0		
BLACK_BISHOPS_MOBILITY	$<$ -15						
positive examples ("bad")	46	46	42	38	38	36	31
negative examples ("not bad")	0	0	0	0	0	0	0

The final rules at the end of the process are presented in Table 5 (for the interpretation of this presentation, see the description before Table 3). The ob-

tained rules for the new positional feature BAD_BISHOP only cover positive examples, have a pure distribution (no misclassified examples), and also appear sensible to the experts.

Particularly valuable is that the rules enable not only commenting on whether a bishop is bad, but also *why* it is bad. Formulation of the explanations is provided in the expert module of our annotating system. For example, if the rule #2 from Table 5 triggers in a particular position, the following comment could be given: "*Black bishop is bad, since black pawns on the same colour of squares ahead of it, and pawns of both opponents restrict its mobility.*"

The machine learning methods that were used on original CRAFTY's positional feature values were again tested on the same data, supplemented with the newly obtained attribute values. All the accuracy and precision results were again obtained through 10-fold cross validation.

Table 6. The machine learning methods' performance with the data, supplemented with the newly obtained attribute values.

Method	**Classification accuracy**	**Precision**
Decision trees (C4.5)	85%	85%
Logistic regression	89%	91%
Rule learning (CN2)	91%	94%
Rule learning with arguments (ABCN2)	94%	96%

The results are presented in Table 6. They suggest that the performance of other algorithms could also be improved by adding appropriate additional attributes. However, using arguments (as with the method ABCN2), besides stimulating the expert to identify the need for useful additional attributes, also guides the method towards appropriate combinations of attributes, which is likely to lead to even more accurate models.

4 Summary and Conclusions

We investigated a particular aspect in the development of a chess annotating software - the ability of making intelligent comments on the positional aspects of a chess game. This task is made more difficult by the fact that the strength of the chess playing programs mainly comes from search and not from subtle positional knowledge which is necessary for generating positional comments. Therefore, components of a chess program's evaluation function are not sufficient for making in-depth positional comments. Defining deep positional patterns requires powerful knowledge-elicitation methods. Our study suggests that argument-based machine learning enables such a method.

Our approach to the generation of positional comments makes use of elements of a chess evaluation function. However, more sophisticated positional patterns have to be introduced in addition to the features contained in an evaluation function. Defining such sophisticated positional patterns is often a difficult knowledge-elicitation task.

In the presented case study, we considered the elicitation of the well-known chess concept of the bad bishop. There is a general agreement in the chess literature and among chess players about the intuition behind this concept. However, formalise it in a way that would enable an annotating system to reliably decide whether a bishop in a given position is bad turned out to be beyond the practical ability of our chess experts (a master and a woman grandmaster). The introduction of sophisticated positional concepts such as the bad bishop turned out to be an intricate knowledge-elicitation problem.

To alleviate the knowledge-elicitation problem, we then employed machine learning from examples of good and bad bishops. We tried several standard machine learning techniques to induce definitions of the bad bishop from examples. This did not produce really satisfactory results. Finally, we successfully applied the recently developed approach of Argument Based Machine Learning (ABML). The efficacy of ABML comes from its unique ability to make use of expert's arguments (or justifications) for selected example cases. This approach is natural and effective for the expert because he or she can concentrate on explaining concrete cases, and does not have to construct general rules, which is much more difficult. The method also leads the expert to think about new relevant descriptive features, thereby improving the description language that the learning program uses. In our case study, the initial repertoire of the attributes taken directly from CRAFTY's evaluation function was thus extended by another six attributes that the experts coined when explaining critical cases selected automatically by the ABML-based knowledge-elicitation process.

Our future work will be associated with further improvements of our annotation software. We intend to implement several additional positional features into its evaluation function, in order to make the commentary more instructive. In particular, the expert module of our annotation system, which provides the user with a commentary of chess games, based on learned or manually-crafted positional features, and possibly with more detailed explanations about particular features of chess positions, requires further attention. As part of future work, we intend to apply this knowledge-acquisition method to the formalisation of other positional concepts of fuzzy nature, such as weak or strong pawn structures, "harmony among the pieces", etc.

References

1. Sadikov, A., Možina, M., Guid, M., Krivec, J., Bratko, I.: Automated chess tutor. In: Proceedings of the 5th International Conference on Computers and Games. (2006)
2. Možina, M., Žabkar, J., Bratko, I.: Argument based machine learning. Artificial Intelligence **171**(10/15) (2007) 922–937
3. Watson, J.: Secrets of Modern Chess Strategy. Gambit Publications (1999)
4. Nimzovich, A.: The Blockade. Hardinge Simpole Limited (2006)
5. Prakken, H., Vreeswijk, G.: Logics for Defeasible Argumentation. In: Handbook of Philosophical Logic, second edition. Volume 4. Kluwer Academic Publishers, Dordrecht etc (2002) 218–319
6. Clark, P., Boswell, R.: Rule induction with CN2: Some recent improvements. In: EWSL. (1991) 151–163