Preference Learning from Annotated Game Databases

Christian Wirth and Johannes Fürnkranz

Knowledge Engineering, Technische Universität Darmstadt, Germany
{cwirth,fuernkranz}@ke.tu-darmstadt.de

Abstract. In chess, as well as many other domains, expert feedback is amply available in the form of annotated games. This feedback usually comes in the form of qualitative information because human annotators find it hard to determine precise utility values for game states. Therefore, it is more reasonable to use those annotations for a preference based learning setup, where it is not required to determine values for the qualitative symbols. We show how game annotations can be used for learning a utility function by translating them to preferences.
We evaluate the resulting function by creating multiple heuristics based upon different sized subsets of the training data and compare them in a tournament scenario. The results show that learning from game annotations is possible, but our learned functions did not quite reach the performance of the original, manually tuned function.The reason for this failure seems to lie in the fact that human annotators only annotate "interesting" positions, so that it is hard to learn basic information, such as material advantage from game annotations alone.

1 Introduction

For many problems, human experts are able to demonstrate good judgment about the quality of certain courses of actions or solution attempts. Typically, this information is of qualitative nature, and cannot be expressed numerically without selecting arbitrary values. Particularly well-studied form of qualitative knowledge are so-called *pairwise comparisons* or *preferences*. Humans are often not able to determine a precise utility value of an option, but are typically able to compare the quality of two options (e.g., "Treatment $\boldsymbol{a}$ is more effective than treatment $\boldsymbol{b}$"). Thurstone's *Law of Comparative Judgment* essentially states that such pairwise comparisons correspond to an internal, unknown utility scale [14]. Recovering this hidden information from such qualitative preference is studied in various areas such as ranking theory [12] or voting theory [3] Most recently, the emerging field of preference learning [6] studies how such qualitative information can be used in a wide variety of machine learning problems.

In the game of chess, qualitative human feedback is amply available in the form of game notations.

We show how this information can be used in combination with state-of-the-art ranking algorithms to successfully learn an evaluation function. The learning setup is based on the methodology used in Paulsen and Fürnkranz [13], where it has been used for learning evaluation functions from move preferences of chess players of different strengths. This paper briefly summarizes the key results, for details we refer to Wirth and Fürnkranz [18].

In Section 2, we discuss the information than can typically be found in annotated chess games. Section 3 shows how to extract preference information from such data and how to use this information for learning an evaluation function via an object ranking algorithm. In our experimental setup (Section 4), we evaluate the proposed approach with an large scale tournament, discussing the results in Section 5. Section 6 concludes the paper and discusses open questions and possible future work.

2 Game Annotations in Chess

Chess is a game of great interest, which has generated a large amount of literature that analyzes the game. Particularly popular are game annotations, which are regularly published after important or interesting games have been played in tournaments. These annotations reflect the analysis of a particular game by a (typically) strong professional chess player.

Annotated chess games are amply available. For example, the largest database distributed by the company *Chessbase*[1], contains over five million games, more than 66,000 of which are annotated.

Chess players annotate games with alternative lines of play and/or textual descriptions of the evaluation of certain lines or positions. An open format for storing chess games and annotations is the so-called *portable game notation* (PGN) [4].

Most importantly, however, typical events in a chess game can be encoded with a standardized set of symbols. There is a great variety of these *numeric annotation glyphs* (NAG), referring to properties of the positions (e.g., attacking chances, pawn structures, etc.), the moves (e.g., forced moves, better alternatives, etc.), or to external properties of the game (such as time constraints). In this work, we will focus on the most commonly used symbols, which annotate the quality of moves and positions:

– *move evaluation:* Each move can be annotated with a symbol indicating its quality. Six symbols are commonly used:
  • very poor move (`??`),
  • poor move (`?`),
  • speculative move (`?!`),
  • interesting move (`!?`),

[1] `http://www.chessbase.com/`

    - good move (!),
    - very good move (!!).
  - *position evaluation:* Each move can be annotated with a symbol indicating the quality of the position it is leading to:
    - white has a decisive advantage (+−),
    - white has a moderate advantage (±),
    - white has a slight advantage (⩲),
    - equal chances for both sides (=),
    - black has a slight advantage (⩱),
    - black has a moderate advantage (∓),
    - black has a decisive advantage (−+),
    - the evaluation is unclear (∞).

We will denote the set annotation symbols with $\mathcal{A} = \mathcal{A}_P \cup \mathcal{A}_M$ where $\mathcal{A}_P$ are position annotations and $\mathcal{A}_M$ are move annotations. $\mathcal{A}(\mathbf{m})$ are the annotations associated with a given move $\mathbf{m}$ in the annotated game. Move and position evaluations can be organized into a partial order which we denote with the symbol $\sqsupset$. The move evaluations can be ordered as

$$+- \ \sqsupset \ \pm \ \sqsupset \ ⩲ \ \sqsupset \ = \ \sqsupset \ ⩱ \ \sqsupset \ \mp \ \sqsupset \ -+$$

and the position evaluations as

$$!! \ \sqsupset \ ! \ \sqsupset \ !? \ \sqsupset \ ?! \ \sqsupset \ ? \ \sqsupset \ ??.$$

Note that, even though there is a certain correlation between position and move annotations (good moves tend to lead to better positions and bad moves tend to lead to worse positions), they are not interchangeable. A very good move may be the only move that saves the player from imminent doom, but must not necessarily lead to a very good position. Conversely, a bad move may be a move that misses a chance to mate the opponent right away, but the resulting position may still be good for the player. For this reason, $\sqsupset$ is partial in the sense that it is only defined on $\mathcal{A}_M \times \mathcal{A}_M$ and $\mathcal{A}_P \times \mathcal{A}_P$, but not on $\mathcal{A}_M \times \mathcal{A}_P$.

In addition to annotating games with NAG symbols, annotators can also add textual comments and variations. These complement the moves that were actually played in the game with alternative lines of play that could have happened or that illustrate the assessment of the annotator. Typically, such variations are short move sequences that lead to more promising states than the moves played in the actual game. Variations can also have NAG symbols, and may contain subvariations.

It is important to note that this feedback is of qualitative nature, i.e., it is not clear what the expected reward is in terms of, e.g., percentage of won games from a position with evaluation ±. However, according to the above-mentioned relation $\sqsupset$, it is clear that positions with evaluation ± are preferable to positions with evaluation ⩲ or worse (=, ⩱, ∓, −+).

As we will see in the following, we will collect preference statements over positions. For this, we have to uniquely identify chess positions. Chess positions can be efficiently represented in the *Forsyth-Edwards Notation* (FEN), which is a serialized, textual representation of the game board, capturing all data that is required to uniquely identify a chess state.

3 Learning an Evaluation Function from Annotated Games

Our approach of learning from qualitative game annotations is based on the idea of transforming the notation symbols into preference statements between pairs of positions, which we describe in more detail in sections 3.2 Each such preference may then be viewed as a constraint on a utility function for chess positions, which can be learned with state-of-the-art machine learning algorithms such as support-vector machines (Section 3.4). First, however, let us briefly recapitulate a few key notions in the field of preference learning.

3.1 Preference Learning

Preference learning [6] is a newly emerging area that is concerned with the induction of predictive preference models from empirical data. It establishes a connection between machine learning and research fields like preference modeling or decision making.

One can distinguish two main types of preference learning problems: in *label ranking*, the task is to learn which of a set of labeled options are preferred in a given context [9, 15]. *Object ranking* is an alternative setting, where the preferences are defined over a set of objects, which are not designated by labels but characterized via features, and the task is to learn a ranking function for arbitrary objects from this domain [11]. More precisely, the task is to learn a utility function $f(\cdot)$ that allows to order a subset of objects out of a (potentially infinite) reference set $\mathcal{Z}$. The training data is given in the form of rankings between subsets of these objects $\mathbf{z} \in \mathcal{Z}$, which are usually specified as a vector of values for a given set of features. These rankings over objects can be decomposed into a finite set of pairwise preferences $\mathbf{z_i} \succ \mathbf{z_j}$ specifying that object $\mathbf{z_i}$ should have a higher utility than object $\mathbf{z_j}$, i.e., that

$$\mathbf{z_i} \succ \mathbf{z_j} \Leftrightarrow f(\mathbf{z_i}) > f(\mathbf{z_j}). \tag{1}$$

The task of the object ranking algorithm is to learn the utility function $f(\cdot)$ which can then be used to rank any given subset of objects in $\mathcal{Z}$.

3.2 Generating Preferences from Annotations

The training data that are needed for an object ranking algorithm can be generated from game annotations of the type discussed in Section 2.

First, we parse each game in the database $\mathcal{G}$, replaying all moves so that we are able to generate all occurring positions. For each move $\mathbf{m}$ that has a set

Algorithm 1 Preference generation from position annotations

Require: database of games $\mathcal{G}$, initial position $\boldsymbol{p}_0$

```
 1: $prefs \leftarrow \emptyset$,
 2: for all $\mathbf{g} \in \mathcal{G}$ do
 3:     $pairs \leftarrow \emptyset$,
 4:     $seen \leftarrow \emptyset$
 5:     $\boldsymbol{p} \leftarrow \boldsymbol{p}_0$
 6:     for all $\mathbf{m} \in \mathbf{g}$ do
 7:         $\boldsymbol{p} \leftarrow \textsc{Move}(\boldsymbol{p}, \mathbf{m})$
 8:         if $\exists \mathbf{a} \in \mathcal{A}_P(\mathbf{m})$ then
 9:             $pairs \leftarrow pairs \cup \{(\boldsymbol{p}, \mathbf{a})\}$
10:             $seen \leftarrow seen \cup \{\boldsymbol{p}\}$
11:         end if
12:     end for
13:     for all $\boldsymbol{p}' \in seen$ do
14:         for all $\boldsymbol{p}'' \in seen$ do
15:             for all $(\boldsymbol{p}', \mathbf{a}'), (\boldsymbol{p}'', \mathbf{a}'') \in pairs$ do
16:                 if $\mathbf{a}' \sqsupset \mathbf{a}''$ then
17:                     $prefs \leftarrow prefs \cup \{\boldsymbol{p}' \succ \boldsymbol{p}''\})\}$
18:                 end if
19:             end for
20:         end for
21:     end for
22: end for
```

of associated position annotation symbols $\mathcal{A}_P(\mathbf{m})$ we associate the annotation symbols with the position $\boldsymbol{p}$ that results after playing each move.

Algorithm 1 shows the algorithm for generating state preferences. The first loop collects for each game a list of all positions that have a position annotation. In a second loop, all encountered annotated positions are compared. For all pairs of positions $(\boldsymbol{p}', \boldsymbol{p}'')$ where $\boldsymbol{p}'$ received a higher evaluation than $\boldsymbol{p}''$, a corresponding position preference is generated. Position $\boldsymbol{p}'$ receives a higher evaluation than $\boldsymbol{p}''$ if the associated position evaluation $\mathbf{a}'$ prefers white to a stronger degree than the annotation $\mathbf{a}''$ associated with $\boldsymbol{p}''$. (Position evaluations are always viewed from the white player)

As can be seen, we only compare position preferences within the same game and not across games. This has several reasons. One is, of course, that a comparison of all annotated position pairs in the entire database would be computationally infeasible. However, it is also not clear whether this would be a sensible thing to do.

For example, it is possible that different annotators reach different conclusions for the same position. A decisive advantage ($-+$,$+-$) for one player may only be annotated as moderate advantage ($\mp$,$\pm$) by another annotator. For this reason, we only compare annotations within the same annotated game.

Concerning move preferences, we follow a comparable procedure. As already mentioned, the quality of a move depends on the possibilities given in a certain state, hence move preferences are restricted to moves applied to the same state. This means when ever we encounter two moves for the same state, annotated with different symbols, we create a preference according to the ranking given by $\mathcal{A}_M$. By applying the move to the state, we can convert the move preferences to position preferences. This means the preference is then given by a relation between to states directly following a given state by playing different moves.

3.3 Position Evaluations in Computer Chess

In many cases, not only a NAG annotation is available, but also a suggested move sequence that should be followed afterwards. This usually means that not the position reached after the move is preferable, but the position at the end of the line. Consequently, in such cases we also considered following these variations until the end of the line.

It should also be noted that chess engines are usually only evaluating quiet positions, where no capturing move is obviously preferred. For example, when the opponent initiated a piece-trading move sequence, the trade should be completed before evaluating the position. This is why such a setup has been suggested in previous works on reinforcement learning in chess [2]. Hence, we further extended the algorithm so that it may also perform a quiescence search.

3.4 SVM-based ranking

Once we have a set of position preferences of the form $\boldsymbol{p}_i \succ \boldsymbol{p}_j$, we can use them for training a ranking support vector machine, as proposed in [10]. As stated above (1), the key idea is to reinterpret the preference statements as constraints on an evaluation function. If this function f is linear, i.e., it is a weighted sum

$$f(\boldsymbol{p}) = \sum_{\phi} w_{\phi} \cdot \phi(\boldsymbol{p}) \tag{2}$$

of features ϕ, the latter part is equivalent to

$$\begin{aligned} f(\boldsymbol{p}_i - \boldsymbol{p}_j) &= \sum_{\phi} w_{\phi} \cdot \phi(\boldsymbol{p}_i - \boldsymbol{p}_j) \\ &= \sum_{\phi} w_{\phi} \cdot \big(\phi(\boldsymbol{p}_i) - \phi(\boldsymbol{p}_j)\big) > 0 \end{aligned} \tag{3}$$

This essentially corresponds to the training of a classifier on the pairwise differences $\bar{\boldsymbol{p}}_{ij} = \boldsymbol{p}_i - \boldsymbol{p}_j$ between positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ with an unbiased hyperplane. The pairwise ranking information can thus be converted to binary training data in the form of feature distance vectors $\bar{\boldsymbol{p}}_{ij}$.

We selected SVMs for our experiments because they learn a linear evaluation function, which can be easily plugged into the chess program, and which can

Table 1. Features used in the linear evaluation function of the CUCKOO chess engine.

Feature Type	# Features	Description
material difference	1	Difference in the sum of all piece values per player.
piece square	6	Position dependent piece values by static piece/square tables.
pawn bonus	1	Bonus for pawns that have passed the enemy pawn line.
trade bonus	2	Bonus for following the "when ahead trade pieces, when behind trade pawns" rules.
castle bonus	1	Evaluates the castling possibility.
rook bonus	1	Bonus for rooks on (half-) open files.
bishops scores	2	Evaluating the bishops position by attack possibilities, if trapped and relative positioning.
threat bonus	1	Difference in the sum of all piece values under attack.
king safety	1	Evaluates the kings position relative to the rooks.

be quickly evaluated. This is important for a good performance of the chess program because the more positions it can evaluate the deeper it can search. Finally, the default evaluation function of the chess program is also linear, so that it should, in principle, be possible to achieve a good performance with learned linear functions.

3.5 Related Work

The basic idea of learning from annotated games was first introduced by [8], where a hill-climbing approach was used to optimize Kendall's τ rank correlation measure.

Our approach to using a preference-based object ranking approach for learning an evaluation function essentially follows [13, 16], where a similar algorithm was applied to the problem of learning evaluation functions of different playing strengths. In that case, the training preferences were obtained by assuming that the move actually played by a player had been preferred over all other moves that had not been played.

Of course, approaches to learning from (expert) demonstration, most notably inverse reinforcement learning [1], are also applicable to learning from chess databases, but our focus is on learning from annotations. IRL assumes (near) perfect demonstrations, which is hardly available in many domains, as opposed to the partial, qualitative evaluations that can be found in the game annotations.

4 Experimental Setup

To investigate the usefulness of preference data, we train chess evaluation functions based on preferences generated from annotated chess games (Section 4.1), and employ them in the strong open source chess engine CUCKOO[2]. All states are

[2] `http://web.comhem.se/petero2home/javachess/`

represented by the heuristic features created by the position evaluation function. CUCKOOO's evaluation function uses the 16 features, shown in Table 1.

Training a linear kernel model allows us to simply extract the feature weights for the linear sum function. The quality of the preferences can then be analyzed by comparing the playing strength of our re-weighted chess engine. In this section, we describe the basic experimental setup, the results of the experiments will be describe in the subsequent section 5.

4.1 Chess Database

As a data source we used *Mega Database 2012*, which is available from *Chessbase*. To the authors' knowledge, this is the largest database of professionally annotated chess games available. The annotations are commonly, but not exclusively, provided by chess grandmasters.

In the more than 5 million games contained in the database, we found 86,767 annotated games,[3] from which we computed 5.05 million position preferences and 190,000 move preferences using the algorithms sketched in section 3.

4.2 Evaluation

The learned evaluation functions have been evaluated in several round robin tournaments, where each learned function plays multiple games against all other functions. All tournaments are played using a time control of two seconds per move.

All results are reported in terms of Elo ratings [5], which is the commonly used rating system for rating chess players. It also allows reporting upper and lower bounds for the playing strength of each player. For calculating the Elo values, a base value of 2600 was used, because this is the rating for CUCKOO as reported by the *Computer Chess Rating List*[4]. It should be noted that computer engine Elo ratings are not directly comparable to human Elo ratings, because they are typically estimated on independent player pools, and thus only reflect relative strengths within each pool.

5 Results

In this section, we report the results from the explained approach. Upon analyzing these results, we noticed that a possible reason for the weak performance could be that annotated positions are not representative for all positions that may be encountered during this game. We investigate this hypothesis in Section 5.2, where we show how the results can be improved by adding positions where one side has a very clear, decisive advantage.

[3] In addition to the 66k games with textual annotations, there are a few more that only contain a few annotation symbols, which we included, but which are not officially counted as "annotated".

[4] `http://www.computerchess.org.uk/ccrl/`

5.1 Learning from Preferences

We generated both move and position preferences for training set sizes of 10%, 25%, 50%, and 100% of all games. We divided the data into k non-overlapping folds, each holding $\frac{1}{k}$-th of the data (e.g., 4 folds with 25%), and trained an evaluation function for each fold. The players that represent different values of k then, before each game, randomly selected one of the evaluation functions for their size. The second and third column of Table 2 show the number of preferences in the training data for each configuration, averaged over all folds of the same size (we will return to columns four and five in Section 5.2).

We first optimized various parametrizations of the experimental setup. In particular, we optimized the C-parameter of the SVM by testing three values: $C = 1$, $C = 0.01$ and $C = .00001$.

Our second objective was to compare different ways of incorporating position evaluations. As described in Section 3.3, one can either directly use the position to which the annotation is attached, attach the annotation to the end of the line provided by the annotator, or perform a quiescence search as is commonly done in reinforcement learning of evaluation functions.

It turns out that following the annotated side lines, but without using quiescence search performs best in every instance. Hence, we decided to run all subsequent experiments with this setup.

These optimal parameter settings were then used in a large tournament that compares the performance of different types of preference information. Figure 1 shows the results. Note that the absolute Elo values presented there are not comparable with the baseline value, because they are estimated on a different player pool. This means the new evaluation function does not outperform the original one. For the moment, only consider the right-most three groups in each graph, namely those representing learning from position preferences, learning from move preferences and learning from both. Clearly, learning from only move preferences yields much worse results than learning from position preferences. Moreover, adding move preferences to the position preferences does not further increase the performance.

Clearly, one has to keep in mind that the number of move preferences is considerably smaller than the number of position preferences, so that the values are already different with respect to training set sizes. However, we also see from the

Table 2. Average Number of Generated Preferences (in Millions).

	Preferences			
#games	Move	Position	Augmented	Surrender
10%	0.02M	0.50M	0.34M	0.05M
25%	0.04M	1.26M	0.85M	0.12M
50%	0.09M	2.53M	1.71M	0.24M
100%	0.19M	5.05M	3.41M	0.48M

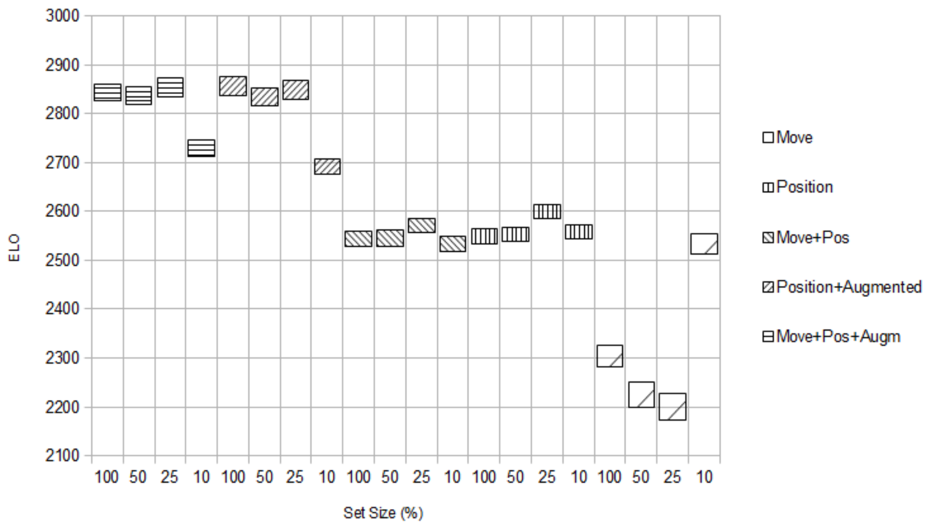

Fig. 1. Comparison of players learned from different types of preference information for four training set sizes with the square height as confidence interval.

results that learning from position preferences is already saturated in the sense that there are only minor performance varations between the different training set sizes, i.e., more position preferences do not further improve performance. The same seems to be the case when adding move preferences to the position preferences. Thus, we are inclined to conclude that the move preferences do not add qualitatively different information but just add more of the same information.

5.2 Augmenting Position Preferences

After a first batch of experiments, we noticed a considerable performance gap between the performance of the original Cuckoo chess program and the one using our learned evaluation function. In particular, we noticed that fundamental concepts like the material value of the different pieces tends to be underestimated by the learned heuristics, resulting in a lower playing strength.

A possible reason for this could be that human players tend to only annotate "interesting" positions, i.e., positions where the evaluation is not entirely clear. Thus, the program may miss picking up that, e.g., it is in general really bad to be a queen down because such positions are too clear to be annotated.

In order to test this hypothesis, we tried to augment our preference data with preferences that involve a very clear material advantage for one side. For each position annotated as a decisive advantage (+−, −+, we generated a new position by removing a random piece, excluding pawns and kings, of the inferior side. The idea is that if we remove a piece from an already lost position, the resulting position should be extremely bad. Thus, such positions are hopefully more extreme than those that are usually annotated in game databases. This technique increased the amount of state preferences by 60%-70%, as can be seen from the fourth column of Table 2.

We compared the augmented preference sets to non-augmented sets in a tournament with 100 games per pairing. Figure 1 shows the results. We can clearly see that the augmented preference sets always outperform all other settings. We can also observe a similar saturation as with position preferences, but here it only occurs after 25% of the preferences have been seen. The learned function is still not able to reach the performance of the original one.

Finally, we also tried to enhance the amount of preferences even further by including position where a player resigned the game, because most chess games do not end with checkmate. We inserted such final position at the top or bottom of the preference order $\sqsupset$, with the reasoning that these positions are so hopeless that their qualitative evaluation should be worse than $-+$ (or better than $+-$ respectively).[5]

Including such surrenders increased the amount of available position preferences by ca. 5% (cf. column five in Table 2). However, these additions did not further increase the performance.

6 Conclusion

We have been able to confirm, that useful information can be learned from annotations but that the result does not reach the performance of a hand-tuned function. We also showed that this is not a problem of insufficient training data.

One problem that we identified is that human annotators only tend to focus on close and interesting positions. We could show that an augmentation of the training information with artificial preferences that result from generating bad positions by removing pieces from already losing positions leads to a significant increase in the quality of the learned function, confirmed by an evaluation of the resulting feature weights. Thus, we may conclude that important training information is missing from human annotations. One could also speculate that the observed performance differences might be due to different scales used by different annotators. However, we did not generate preferences across different games, only within individual games. Thus, we cannot suffer from this problem as long as each expert is consistent with herself.

We conclude that annotations should be complemented with other phases of learning (e.g., phases of traditional, reinforcement-based self-play).

This is also what we plan to explore in the near future. We are currently thinking of ways for combining learning from game annotations with preference-based reinforcement learning algorithms that are currently under investigation [7, 17]. In that way, the information learned from game annotations can be fine-tuned with the agent's own experience (or vice versa), a strategy that seems typical for human players.

[5] We used a few crude heuristics for excluding cases where a game was decided by time controls or similar.

References

[1] Abbeel, P., Ng, A.Y.: Inverse reinforcement learning. In: Encyclopedia of Machine Learning, pp. 554–558. Springer-Verlag (2010)

[2] Baxter, J., Tridgell, A., Weaver, L.: Knightcap: A chess program that learns by combining td(lambda) with game-tree search. In: Proceedings of the 15th International Conference on Machine Learning, (ICML-98). pp. 28–36. Morgan Kaufmann (1998)

[3] Coughlin, P.J.: Probabilistic Voting Theory. Cambridge University Press (2008)

[4] Edwards, S.J.: Portable game notation (1994), `http://www6.chessclub.com/help/PGN-spec`, accessed on 14.06.2012

[5] Elo, A.E.: The Rating of Chessplayers, Past and Present. Arco, 2nd edn. (1978)

[6] Fürnkranz, J., Hüllermeier, E. (eds.): Preference Learning. Springer-Verlag (2010)

[7] Fürnkranz, J., Hüllermeier, E., Cheng, W., Park, S.H.: Preference-based reinforcement learning: A formal framework and a policy iteration algorithm. Machine Learning 89, 123–156 (2012), Special Issue of Selected Papers from ECML PKDD 2011

[8] Gomboc, D., Marsland, T.A., Buro, M.: Evaluation function tuning via ordinal correlation. In: Advances in Computer Games, IFIP, vol. 135, pp. 1–18. Springer US (2004)

[9] Hüllermeier, E., Fürnkranz, J., Cheng, W., Brinker, K.: Label ranking by learning pairwise preferences. Artificial Intelligence 172, 1897–1916 (2008)

[10] Joachims, T.: Optimizing search engines using clickthrough data. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02). pp. 133–142. ACM Press (2002)

[11] Kamishima, T., Kazawa, H., Akaho, S.: A survey and empirical comparison of object ranking methods. In: [6], pp. 181–201

[12] Marden, J.I.: Analyzing and Modeling Rank Data. CRC Press (1995)

[13] Paulsen, P., Fürnkranz, J.: A moderately successful attempt to train chess evaluation functions of different strengths. In: Proceedings of the ICML-10 Workshop on Machine Learning and Games (2010)

[14] Thurstone, L.L.: A law of comparative judgement. Psychological Review 34, 278–286 (1927)

[15] Vembu, S., Gärtner, T.: Label ranking algorithms: A survey. In: [6], pp. 45–64

[16] Wirth, C., Fürnkranz, J.: First steps towards learning from game annotations. In: Workshop Proceedings - Preference Learning: Problems and Applications in AI at ECAI 2012. pp. 53–58 (2012)

[17] Wirth, C., Fürnkranz, J.: EPMC: Every visit preference Monte Carlo for reinforcement learning. In: Proceedings of the 5th Asian Conference on Machine Learning, (ACML-13). JMLR Proceedings, vol. 29, pp. 483–497. JMLR.org (2013)

[18] Wirth, C., Fürnkranz, J.: On learning from game annotations. IEEE Transactions on Computational Intelligence and AI in Games (2014), in press