First Steps Towards Learning from Game Annotations

Christian Wirth and **Johannes Fürnkranz** [1]

Abstract.
Most of the research in the area of evaluation function learning is focused on self-play. However in many domains, like chess, expert feedback is amply available in the form of annotated games. This feedback comes usually in the form of qualitative information due to the inability of humans to determine precise utility values for game states. We are presenting a first step towards integrating this qualitative feedback into evaluation function learning by reformulating it in terms of preferences. We extract preferences from large-scale database for annotated chess games and use them for calculating the feature weights of a heuristic chess position evaluation function. This is achieved by extracting the feature weights out of the linear kernel from a learned SVMRANK model, based upon the given preference relations. We evaluate the resulting function by creating multiple heuristics based upon different sized subsets of the trainings data and compare them in a tournament scenario. Although our results did not yield a better chess playing program, the results confirm that preferences derived from game annotations may be used to learn chess evaluation functions.

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 (e.g., "A treatment $\boldsymbol{a}$ is more effective than treatment $\boldsymbol{b}$), and cannot be expressed numerically without selecting arbitrary values. This is due to the fact that humans are not able to determine a precise utility value of an option, but are typically able to compare the quality of two options. The emerging field of preference learning tries to make this information usable in the field of machine learning, by introducing concepts and methods for applying qualitative preferences to a wide variety of learning problems [12].

In the game of chess, qualitative human feedback is amply available in the form of game notations. For example, the company *Chessbase*[3] specializes in the collection and distribution of chess databases. Their largest database contains annotations for over 66000 games, according to the official page. However, this rich source of information about the game has so far been ignored in the literature on machine learning in chess [21, 11]. Much of the work in this area has concentrated on the application of reinforcement learning algorithms to learn meaningful evaluation functions [3, 4, 7]. These approaches have all been modeled after the success of *TD-Gammon* [24], a learning system that uses temporal-difference learning [22] for training a game evaluation function [23]. However, all these algorithms were trained exclusively on self-play, entirely ignoring human feedback that is readily available in annotated game databases.

[1] TU Darmstadt, Germany, [cwirth, juffi]@ke.tu-darmstadt.de

In this paper, we report the results of a first study that aims at learning a heuristic function for chess based on a large amount of qualitative feedback from experts available in annotated game database. In particular, we show how preferences can be extracted from chess databases, and show how state-of-the-art ranking algorithms can be used to successfully learn an evaluation function. The learning setup is based on the methodology used in [16], where it has been used for learning evaluation functions from move preferences of chess players of different strengths. However, to our knowledge this is the first work that reports on results from learning evaluation functions from game annotations.

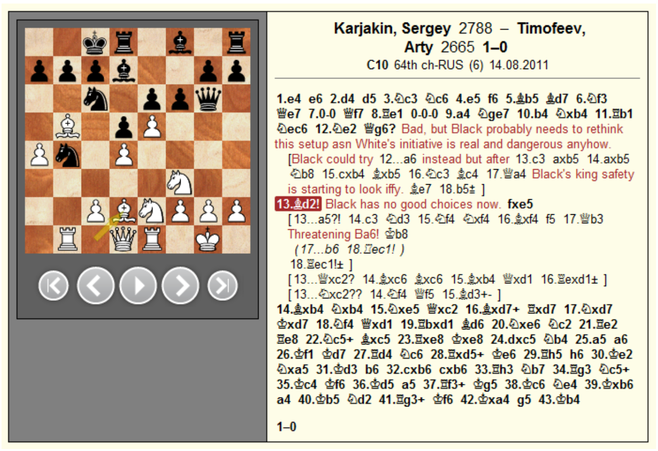
In Section 2, we are explaining which information can be contained in annotations for chess games, especially concerning *portable game notation* files with *numeric annotation glyphes* [8]. A widely available data format. Section 3 details the object ranking by preferences method in general, as well as how to extract the preference information. In our experimental setup (Section 5), we are training a SVM with the preference data, based upon the state feature values given by a strong chess engine. This enables the creation of a new heuristic evaluation function by using the learned (linear) SVM model. The quality of the resulting function is evaluated in a chess engine tournament. Section 7 is concluding the paper and gives a short overview over possible further 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 frequently 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., They have been produced without any time constraints, and the annotators can resort to any means they deem necessary for improving their judgement (such as consulting colleagues, books, or computers). Thus, these annotations are usually of a high quality.

Annotated chess games are amply available, not only in chess books or magazines. Chess databases, such as those provided by companies like *Chessbase*[3], are storing millions of games, many of them annotated. Chess players of all strengths use them regularly to study the game or to prepare against their next opponent.

Chess annotators use a standardized set of symbols for annotating moves and positions, which have been popularized by the Chess Informant book series. *Portable game notation* (PGN) files are chess games recorded in *standard algebraic notation* with optional *numeric annotation glyphes* (NAG) [8]. Those annotation symbols can be divided into three major categories: *move*, *position* and *time* evaluation.



Karjakin, Sergey 2788 – **Timofeev, Arty** 2665 **1–0**
C10 64th ch-RUS (6) 14.08.2011

1.e4 e6 2.d4 d5 3.♘c3 ♘c6 4.e5 f6 5.♗b5 ♗d7 6.♘f3 ♕e7 7.0-0 ♕f7 8.♖e1 0-0-0 9.a4 ♘ge7 10.b4 ♘xb4 11.♖b1 ♘ec6 12.♘e2 ♕g6? Bad, but Black probably needs to rethink this setup asn White's initiative is real and dangerous anyhow.
[Black could try 12...a6 instead but after 13.c3 axb5 14.axb5 ♘b8 15.cxb4 ♗xb5 16.♘c3 ♗c4 17.♕a4 Black's king safety is starting to look iffy. ♗e7 18.b5±]
13.♗d2! Black has no good choices now. **fxe5**
[13...a5?! 14.c3 ♘d3 15.♘f4 ♘xf4 16.♗xf4 f5 17.♕b3 Threatening Ba6! ♔b8
(*17...b6 18.♖ec1!*)
18.♖ec1!±]
[13...♕xc2? 14.♗xc6 ♗xc6 15.♗xb4 ♕xd1 16.♖exd1±]
[13...♘xc2?? 14.♘f4 ♕f5 15.♗d3+-]
14.♗xb4 ♘xb4 15.♘xe5 ♕xc2 16.♗xd7+ ♖xd7 17.♘xd7 ♔xd7 18.♘f4 ♕xd1 19.♖bxd1 ♗d6 20.♘xe6 ♘c2 21.♖e2 ♖e8 22.♘c5+ ♗xc5 23.♖xe8 ♔xe8 24.dxc5 ♘b4 25.a5 a6 26.♔f1 ♔d7 27.♖d4 ♘c6 28.♖xd5+ ♔e6 29.♖h5 h6 30.♔e2 ♘xa5 31.♔d3 b6 32.cxb6 cxb6 33.♖h3 ♘b7 34.♖g3 ♘c5+ 35.♔c4 ♔f6 36.♔d5 a5 37.♖f3+ ♔g5 38.♔c6 ♘e4 39.♔xb6 a4 40.♔b5 ♘d2 41.♖g3+ ♔f6 42.♔xa4 g5 43.♔b4

1–0

Figure 1. An annotated chess game (screen-shot taken from `http://chessbase.com/`).

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 (!?),
- 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 ($\pm$),
- white has a slight advantage ($\pm$),
- equal chances for both sides ($=$),
- black has a slight advantage ($\mp$),
- black has a moderate advantage ($\mp$),
- black has a decisive advantage ($-+$),
- the evaluation is unclear (∞).

time evaluation: Each move can be annotated with a symbol indicating a time constraint that arose at this move. This information is not used in our experiments.

In addition to annotating games with NAG symbols, annotators can also add textual comments and move variations to the game, i.e., in addition to the moves that have actually been played in the course of the game, an annotator provides alternative lines of play. Those are usually suggestions in the form of short move chains that are leading to more promising states than the move chain used in the real game. Variations can also have NAG symbols, and may contain subvariations.

Figure 1 shows an example for an annotated chessgame. The left-hand side shows the game position after the 13th move of white. Here, black is in a difficult position after the mistake he made. (12...♕g6?). From the suggested moves, 13...a5?! is the best, but even here white has the upper hand at the end of the variation (18.♖ec1!±), as well as in the end of the suggested move chain starting with 13...♕×c2 . On the other hand, 13...♘×c2?? is an even worse choice, ending in a position that is clearly lost for black ($+-$).

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 $\pm$. However, it is clear that positions with evaluation $\pm$ are preferable to positions with evaluation $\pm$ or worse ($=$, $\mp$, $\mp$, $-+$).

Also note that the feedback for positions typically applies to the entire sequence of moves that has been played up to reaching this position (a *trajectory* in reinforcement learning terminology). The qualitative position evaluations may be viewed as providing an evaluation of the trajectory that lead to this particular position, whereas the qualitative move evaluations may be viewed as evaluations of the expected value of a trajectory that starts at this point.

However, even though there is a certain correlation between these two types of annotations (good moves tend to lead to better positions and bad moves tend to lead to worse positions), they are not interchangable. 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.

3 Learning an Evaluation Function from Preferences

For learning the mentioned SVM model, it is required to formulate the task as a binary classification problem. We are showing how this can be done by using preference learning.

3.1 Preference Learning

Preference learning is about inducing predictive preference models from empirical data. This establishes a connection between machine learning and research fields like preference modeling or decision making. Especially “learning to rank by preferences” is deemed promising by the community. Preference learning can be applied to *label ranking*, by defining preferences over a set of labels concerning a specific set of objects [25]. But it is also possible to define preference directly over a set of objects, for creating a ranking of those objects [14]. Preferences themselves are constraints that can be violated, which leads to higher flexibility concerning the solving process, opposed to hard constraints. These constraints can be described via a *utility function* or *preference relations*. [12] We are only considering preference relations in this work, because they can be represented in a qualitative manner.

Object Ranking is about learning how to order a subset of objects out of a (potentially infinite) reference set $\mathcal{Z}$. Those objects $\mathbf{z} \in \mathcal{Z}$ are usually given as a vector of attribute/value pairs, but this is no necessary property. The trainings data is given in the form of rankings, which is decomposed into a finite set of pairwise preferences $\mathbf{z_i} \succ \mathbf{z_j}$. The object ranker is then learning a ranking function $f(\cdot)$ which returns a (ranked) permutation of a given object set. [12]

3.2 States and Actions

In chess, we are searching for the best action $\boldsymbol{a} \in \mathbf{A}$ for a state $\boldsymbol{s} \in \mathbf{S}$. For game tree exploration concerns or suboptimal play, it can also be required to determine the expected quality of an suboptimal action

$\boldsymbol{a'} \in \mathbf{A}$. When defining this quality in a relative way, as opposed to an absolute value, we are searching for a rank. Because of the high amount of legal states in chess (roughly 10^{50} states [2]), it is not feasible to learn those ranking functions directly. Considering the chess transition function $f : \boldsymbol{S} \times \boldsymbol{A_s} \to \boldsymbol{S_s}$, with $\mathbf{S_s} \subset \mathbf{S}$ as the set of states that can be reached from $\mathbf{s}$ by an action $\boldsymbol{a} \in \mathbf{A_s}$ possible in $\boldsymbol{s}$, we can rewrite the problem as the search for a ranking for all $\boldsymbol{s} \in \mathbf{S_s}$. This ranking is also not dependent on the current state $\mathbf{s}$, because the state/action history is not relevant for a chess state (excluding the *fifty-moves* and the *threefold repetition* draw rules). This reduces the problem to a *object ranking* problem over all $\boldsymbol{s} \in \mathbf{S}$.

3.3 SVM-based ranking

Following [16], we can use state preferences of the form $\boldsymbol{s}_i \succ \boldsymbol{s}_j$ for training the SVMRank ranking support vector machine proposed by [13].[2] Its key idea is to reinterpret the preference statements as constraints on the evaluation function, i.e.,

$$\boldsymbol{s}_i \succ \boldsymbol{s}_j \Leftrightarrow h(\boldsymbol{s}_i) > h(\boldsymbol{s}_j).$$

If the function h is a linear, i.e., it is a weighted sum

$$h(\boldsymbol{s}) = \sum_f w_f \cdot f(\boldsymbol{s})$$

of features f, the latter part is equivalent to

$$\begin{aligned} h(\boldsymbol{s}_i - \boldsymbol{s}_j) &= \sum_f w_f \cdot f(\boldsymbol{s}_i - \boldsymbol{s}_j) \\ &= \sum_f w_f \cdot (f(\boldsymbol{s}_i) - f(\boldsymbol{s}_j)) > 0 \end{aligned}$$

Thus, essentially, the training of the ranking SVM corresponds to the training of a classification SVM on the pairwise differences $\boldsymbol{s}_i - \boldsymbol{s}_j$ between positions $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$. The pairwise ranking information can thus be converted to binary training data in the form of a feature distance vector $\overrightarrow{A}$ with the preference relation $r \in \{<, >\}$ as the binary class vector.

4 Generating Preference Data from Game Annotations

The training data that are needed for an object ranking algorithm like SVMRank can be generated from game annotations of the type discussed in Section 2. For our first experiments, we only focused on move preferences, and ignored state preferences.

Our algorithm for generating preferences from move annotations is sketched in Algorithm 1: a given list of games $\boldsymbol{G}$ in PGN format is parsed, and triplets $(\boldsymbol{s}, \boldsymbol{a}, \mathbf{n}), \mathbf{n} \in \mathbf{N}_{\boldsymbol{s},\boldsymbol{a}}$ with $\mathbf{N}_{\boldsymbol{s},\boldsymbol{a}}$ being the list of NAG in $(\boldsymbol{s}, \boldsymbol{a})$ are created for each occurrences of a NAG symbol. A state is represented by its Forsyth-Edwards Notation (FEN). It is a serialized representation of the game board, capturing all data that is required to uniquely identify a chess state [8]. Actions are saved in the Long Algebraic Notation (LAN). After collecting this data for every game, all triples containing the same FEN state are compared. The NAG symbols are checked against a static relation list and a pairwise preference relation for the attached actions is created, if possible. The static relation table contains entries like $\mathbf{n_1}$:?? $\mathbf{n_2}$:!? $\to \mathbf{n_1} < \mathbf{n_2}$. In rare cases, we may get multiple conflicting annotations for a pair $(\boldsymbol{s}, \boldsymbol{a})$, which are then ignored.

[2] Available from `http://svmlight.joachims.org`.

Algorithm 1 Preference Generation

```
Require: list of games $G$, initial position $s_0$
 1: $triples \leftarrow \emptyset$, $prefs \leftarrow \emptyset$, $seen \leftarrow \emptyset$
 2: for all $g \in G$ do
 3:     $s \leftarrow s_0$
 4:     for all $(a, N_{s,a}) \in g$ do
 5:         $s \leftarrow \textsc{Move}(s, a)$
 6:         for all $n \in N_{s,a}$ do
 7:             $triples \leftarrow triples \cup \{(s, a, n)\}$
 8:             $seen \leftarrow seen \cup \{s\}$
 9:         end for
10:     end for
11: end for
12: for all $s' \in seen$ do
13:     for all $N_{\boldsymbol{s'},\boldsymbol{a}}, N_{\boldsymbol{s'},\boldsymbol{a'}} \in triples$ with $\boldsymbol{a} \neq \boldsymbol{a'}$ do
14:         $r \leftarrow \textsc{Relation}(N_{\boldsymbol{s'},\boldsymbol{a}}, N_{\boldsymbol{s'},\boldsymbol{a'}})$
15:         if $r \neq \varnothing$ then
16:             $s_1 \leftarrow \textsc{Move}(s', a)$, $s_2 \leftarrow \textsc{Move}(s', a')$
17:             $prefs \leftarrow prefs \cup \{(s_1, s_2, r)\}$
18:         end if
19:     end for
20: end for
21: return $prefs$
```

When applying this algorithm to the example given in Figure 1, it would yield the following action preferences: ($\boldsymbol{s}$, ♕×c2 $\succ$ ♘×c2),($\boldsymbol{s}$,a5 $\succ$ ♘×c2),($\boldsymbol{s}$,a5 $\succ$ ♕×c2) with $\boldsymbol{s}$ being the state shown in the example.

In a last step, action preferences $(\boldsymbol{s}, \boldsymbol{a}_1 \succ \boldsymbol{a}_2)$ are converted to state preferences by applying $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ to $\boldsymbol{s}$, resulting in state preferences $\boldsymbol{s}_1 \succ \boldsymbol{s}_2$, where $\boldsymbol{s}_i = \textsc{Move}(\boldsymbol{s}, \boldsymbol{a}_i)$. A practical problem is, that annotated moves are usually not leading to a stable state to which the qualitative evaluation can be directly applied. For example, in the middle of an exchange sequence, the first player will be behind by one piece after the initial move but may gain a significant advantage after a short chain of moves. For this reason, preferences are not applied to the positions $\boldsymbol{s}_i$, but to quiet positions that result from a fixed-depth search starting in $\boldsymbol{s}_i$ (we use depth 7), followed by a quiescence search. The positions $\bar{\boldsymbol{s}}_i$ at the leaves of these searches are then used in the state preferences.

Additionally, most variations added to the PGN data are also move chains and not single moves, hence we are applying the suggested move chain to the state and not only the first, single move. This is implemented in step 16 of algorithm 1.

5 Experimental Setup

For showing the usefulness of preference data, we are training a SVM model based on preference data generated from annotated chess games (Section 5.1), and employ it in the strong open source chess engine Cuckoo (Section 5.2). All states are represented by the heuristic features created by the position evaluation function. Training a linear kernel model allows us to simply extract the feature weights for the linear sum function. The quality of the preferences can now be analyzed by comparing the playing strength of our reweighted 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. A single value for every piece type.
pawn bonus	1	Bonus for pawns that have passed the enemy pawn line , while also considering its distance to the enemy king.
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.

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

5.1 ChessBase

As a data source we are using the *Mega Database 2012*, provided by *Chessbase*.[3] To the authors' knowledge, it is the largest database of professionally annotated chess games available. The annotations are commonly, but not exclusively provided by chess grandmasters. In this first study, we only considered action preferences, and ignored state preferences, mostly because of complexity considerations.

In the more than 5 million games contained in the database, we identified 86,767 annotated games with 1.67 million annotated moves in total. 343,634 NAG symbols occurred pairwise concerning the same state, but different moves. Out of these, the preference generation process yielded 271,925 preferences with 190,143 being unambiguous and not equal. The rest are incomparable symbol pairs, and were ignored in our data generation process.

5.2 CUCKOO Chess Engine

We used the CUCKOO chess engine[4] for our experiments, because of its combination of high playing strength[5] and good modifiability. It facilitates BitBoards [19, 1] as state representation and NegaScout [18] as search algorithm.

Most state of the art chess engines are using a heuristic position evaluation function, while searching for the best, currently reachable position with enhanced Alpha-Beta search algorithms like NegaScout. For performance reasons, evaluation functions are commonly linear sums over abstract, manually constructed features. Usually, features like material difference or usefulness of pieces in their current position are used. Table 1 shows the 16 features shown by CUCKOO. We used these features for describing a state.

The CUCKOO Chess Engine was used in a single thread configuration. All experiments haven been executed on systems with 2 cores or more, ensuring independence of the available computing power for each player.

5.3 Training Data

In our experimental setup, we are creating the object preferences as described in Section 4. The pairwise preference data is used as training data for SVMRANK, which is an optimized implementation of the SVM based ranker described in 2.4, which can handle pairwise preference data directly [13]. The feature weights can now be extracted out of the SVMRANK model and be applied to the CUCKOO chess engine.

The features have not been standardized or normalized, because they are already internally normalized to a pico-pawn scale, hence no significant improvement in classification accuracy was expected. This was also confirmed in experiments.

Annotators can disagree concerning the exact quality of a move, but the same relative outcome is expected when comparing two moves. E.g. an annotator may use $\mathbf{n_1}$:? instead of $\mathbf{n_1}$:??, but not $\mathbf{n_1}$:!! if the consensus is $\mathbf{n_1} < \mathbf{n_2}$, $\mathbf{n_2}$:!?. Tests confirmed the expected low amount of directly contradicting preferences ($< 0.2\%$), but it is still possible for subsets to indicate a different valuation of features.

We created 6 different engines, based upon different training set sizes. 5%, 10%, 25%, 50%, 75% and 100% randomly sampled elements of the available preference data have been used to create the different engines. The results have been generated by averaging over three all-against-all tourneys, including the player with the original feature weighting as upper bound and a random player as baseline. The random engine is picking new random weights for each position evaluation. The distribution for those weights is a uniform distribution, bounded by the min/max values observed within all learned SVMRANK models. Each pairing played 100 games with a 5min timeframe and no increments.

5.4 Evaluation

All results are reported in terms of Elo ratings [9], which is the commonly used rating system for rating chess players. It not only considers the absolute percentage of won games, but also takes the strength of the opponent into account. A rating difference of a 100 points approximately means that the stronger player has an expected win rate of $5/8$. It also enables the reporting of upper and lower bounds for the playing strength of each player. For calculating the Elo values, a base Elo of 2600 was used, because this it the rating for the Cuckoo Chess Engine as reported by the *Computer Chess Rating List*[6]. 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.

[3] http://www.chessbase.com/
[4] http://web.comhem.se/petero2home/javachess/
[5] http://www.computerchess.org.uk/ccrl/

[6] http://www.computerchess.org.uk/ccrl/

6 Results

6.1 Predictive Accuracy

We first compared the predictive accuracy of different classifiers on the binary classification problem of learning a preference relation from the collected preference set. The binary classification accuracy a can be compared to the average amount of swapped pairs over all pairs metric e of the original ranking problem by $a = 1 - e$. The *Weka*[7] implementation of all classifiers was used, if not stated otherwise.

Table 2 shows that multilayer perceptrons and random forests yielded the best results, whereas LIBLINEAR and SVMRANK performed the worst. This seems to indicate that a non-linear combination of the base features is able to yield a better performance than the linear combination that is used in the chess program.

We can also see the performance of the original position evaluation function of CUCKOO, which is a linear function that assigns a uniform weight to all features. IT is somewhat higher than the trained linear functions, but considerably below the best non-linear functions.

Classifier	Accuracy
MULTILAYER PERCEPTRON [5]	0.6871
RANDOM FOREST [6]	0.6864
NAIVE BAYES TREE [15]	0.6799
J48 [17]	0.6719
PEGASOS [20]	0.6651
LIBLINEAR[8][10]	0.6521
SVMRANK[9][13]	0.6505
CUCKOO	0.6620

Table 2. Comparison of the predictive performance of different classifiers and the CUCKOO chess engine (10-fold CV).

6.2 Playing Strength

For evaluating the playing strength we were limited to using a linear evaluation function because only those could be easily plugged into the chess program. We chose evaluation functions learned by SVMRANK. Figure 2 shows the development of the rating over the percentage of used preferences in the training data. It is clearly recognizable in that an increase in the amount of used preference data is leading to an improved chess engine, which we take as evidence that the game annotations provide useful information for learning an evaluation function. The playing strength is clearly above the random baseline, which reached an average Elo rating of 2332 ± 32, but well below the original player and its average Elo rating of 2966 ± 43.

6.3 Stability

The player that was trained on 5% of the data is a clear outlier, resulting from the comparably high variance in the training data at this point. The variance of the feature weights at this setting is shown in Figure 3.

However, most features are showing convergence and a mostly stable average value. Figure 4 shows the development of the feature

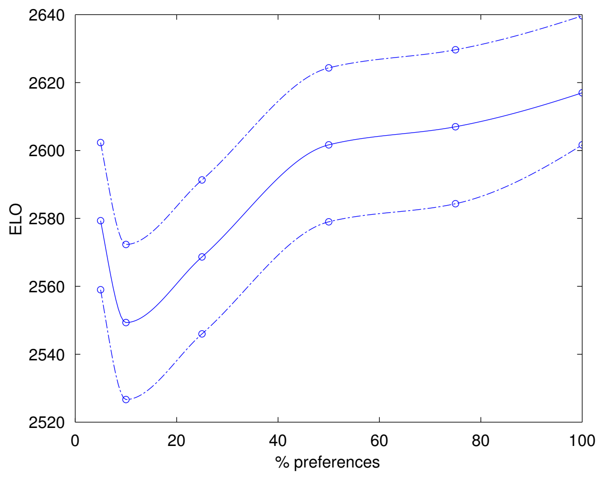


Figure 2. Learning curve, measured in Elo rating.

values (average, standard deviation and min/max values) for all features. The 10 features in the left and the middle graph are quite stable, whereas the features in the right graph are rather unstable. For the features *castleBonus* and *bishopB*, a possible explanation could be the sparsity of these values. The feature value difference for these values is 0 in 84.6% and 99.7%, respectively, of all training examples.

7 Conclusion

This paper presented the results of a preliminary study that uses expert feedback in the form of game annotations for the automated construction of an evaluation function for the game of chess. It was shown how annotated chess games can be used for the creation of preference data. This is especially interesting because of the

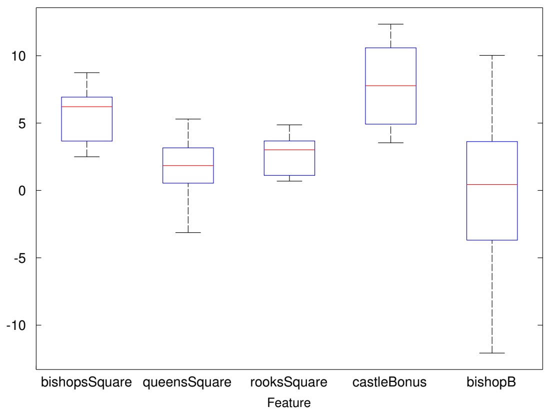


Figure 3. Learned weight for the 5 most variant features, based on 10 different 5% samplings

[7] http://www.cs.waikato.ac.nz/ml/weka/
[8] http://www.csie.ntu.edu.tw/~cjlin/liblinear/
[9] http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

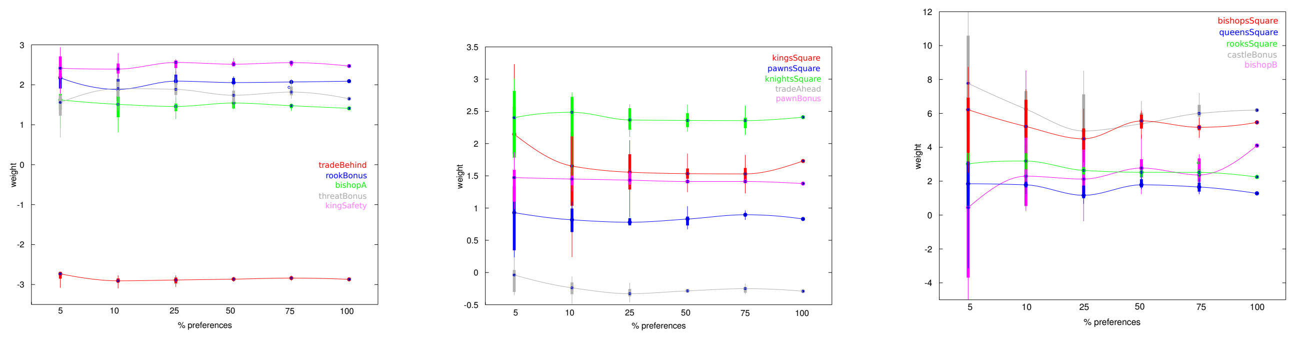


Figure 4. Average and variance for the feature weights, averaged over 10 samples per subset size. Weights are scaled to *materialDifference*$= 1$.

widespread availability of annotated chess games, which enables the creation of large-scale datasets.

Following the approach shown by Paulsen et al [16], the preferences have been successfully used to learn the feature weights for an position evaluation function. It can be observed, that the playing strength of the chess engine is scaling with the amount of trainings data. This is a first step towards using qualitative feedback in game playing scenarios.

However, alhthough we can observe a correlation with the amount of seen preferences and the playing strength of the learned players, their overall strength was not able to reach the strength of the original player. We still have to investigate the reasons for this, but it should be noted that the original feature weighting is outperforming the learned weights. Thus, SVMRANK was only able to find suboptimal feature weights.

Moreover, in this work we have essentially ignored state preferences and focused on action preferences. The reason for this was pragmatic, because action preferences relate to a single state, whereas state preferences can be widely compared, even across multiple games. For example, every position evaluated with $+-$ can be considered to be better than every position evaluated with $=$, all of which can, in turn, be considered to be preferred over positions that are evaluated with $-+$. This approach gives rise to a vast number of preferences. One could consider to only apply this to positions of the same game, because different annotators may have a different calibration of the used symbols. This would also reduce the complexity. These issues are currently under investigation.

REFERENCES

[1] G. M. Adel'son-Vel'skii, V. L. Arlazarov, A. R. Bitman, A. A. Zhivotovskii, and A. V. Uskov, 'Programming a computer to play chess', *Russian Mathematical Surveys*, **25**(2), 221, (1970).

[2] V. Allis, *Searching for Solutions in Games and Artificial Intelligence*, Ph.D. dissertation, University of Limburg, The Netherlands, 1994.

[3] J. Baxter, A. Tridgell, and L. Weaver, 'Learning to play chess using temporal differences', *Machine Learning*, **40**(3), 243–263, (September 2000).

[4] D. F. Beal and M. C. Smith, 'Temporal difference learning applied to game playing and the results of application to Shogi', *Theoretical Computer Science*, **252**(1-2), 105–119, (2001). Special Issue on Papers from the Computers and Games 1998 Conference.

[5] C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, UK, 1995.

[6] L. Breiman, 'Random forests', *Machine Learning*, **45**(1), 5–32, (2001).

[7] S. Droste and J. Fürnkranz, 'Learning the piece values for three chess variants', *International Computer Games Association Journal*, **31**(4), 209–233, (2008).

[8] S. J. Edwards. Portable game notation, 1994. accessed on 14.06.2012.

[9] A. E. Elo, *The Rating of Chessplayers, Past and Present*, Arco, New York, 2nd edn., 1978.

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, 'Liblinear: A library for large linear classification', *Journal of Machine Learning Research*, **9**, 1871–1874, (2008).

[11] J. Fürnkranz, 'Machine learning in computer chess: The next generation', *International Computer Chess Association Journal*, **19**(3), 147–161, (1996).

[12] J. Fürnkranz and E. Hüllermeier (eds.), *Preference Learning*, Springer-Verlag, 2010.

[13] T. Joachims, '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).

[14] T. Kamishima, H. Kazawa, and S. Akaho, 'A survey and empirical comparison of object ranking methods', In Fürnkranz and Hüllermeier [12], 181–201.

[15] R. Kohavi, 'Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid', in *Proceedings of the 2nd International Conference On Knowledge Discovery And Data Mining*, pp. 202–207. AAAI Press, (1996).

[16] P. Paulsen and J. Fürnkranz, 'A moderately successful attempt to train chess evaluation functions of different strengths'. In C. Thurau, K. Driessens, and O. Missura (eds.) *Proceedings of the ICML-10 Workshop on Machine Learning and Games*, Haifa, Israel, (2010).

[17] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.

[18] A. Reinefeld, 'An improvement to the scout tree-search algorithm', *International Computer Chess Association Journal*, **6**(4), 4–14, (December 1983).

[19] A. L. Samuel, 'Some studies in machine learning using the game of checkers', *IBM Journal on Research and Development*, **3**, 210–229, (1959).

[20] Y. Singer and N. Srebro, 'Pegasos: Primal estimated sub-gradient solver for SVM', In Z. Ghahramani (ed.) *Proceedings of the 24th International Conference on Machine Learning (ICML-07)*, pp. 807–814, (2007).

[21] S. S. Skiena, 'An overview of machine learning in computer chess', *International Computer Chess Association Journal*, **9**(1), 20–28, (1986).

[22] R. S. Sutton, 'Learning to predict by the methods of temporal differences', *Machine Learning*, **3**, 9–44, (1988).

[23] G. Tesauro, 'Practical issues in temporal difference learning', *Machine Learning*, **8**, 257–278, (1992).

[24] G. Tesauro, 'Programming backgammon using self-teaching neural nets', *Artificial Intelligence*, **134**(1-2), 181–199, (January 2002). Special Issue on Games, Computers and Artificial Intelligence.

[25] S. Vembu and T. Gärtner, 'Label ranking algorithms: A survey', In Fürnkranz and Hüllermeier [12], 45–64.