EVALUATION FUNCTION TUNING VIA ORDINAL CORRELATION

D. Gomboc, T. A. Marsland, M. Buro
Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada
{dave,tony,mburo}@cs.ualberta.ca, http://www.cs.ualberta.ca/~games/

Abstract Heuristic search effectiveness depends directly upon the quality of heuristic evaluations of states in the search space. We show why ordinal correlation is relevant to heuristic search, present a metric for assessing the quality of a static evaluation function, and apply it to learn feature weights for a computer chess program.

Keywords: ordinal correlation, Kendall's τ (tau), static evaluation function, heuristic search, computer chess

1. Introduction

Inspiration for this research came while reflecting on how evaluation functions for today's computer chess programs are usually developed. Typically, evaluation functions are refined over many years, based upon careful observation of their performance. During this time, engine authors will tweak feature weights repeatedly by hand in search of proper balance between terms. This ad hoc process is used because the principal way to measure the utility of changes to a program is to play many games against other programs and interpret the results. The process of evaluation function development would be considerably assisted by the presence of a metric that could reliably indicate a tuning improvement. But what would such a metric be like?

The critical operation of minimax game-tree searches (Shannon, 1950) and all its derivatives (Marsland, 1983; Plaat, 1996) is the asking of a single question: is position B better than position A? Note that it is not "How much better?", but simply "Is it better?". In minimax, instead of propagating values one could propagate the positions instead, and, as humans do, choose between them directly without using values as an intermediary.

Consequently, we need only pairwise comparisons that tell us whether B is preferable to A. Plausibly, then, the metric we seek will assess how well an evaluation function orders positions in relation to each other, without placing importance on the relative differences in the values of the assessed positions – that is, it will be ordinal in nature.

While at shallow depths some resemblance between positions compared by a minimax-based search will be evident, this does not hold true at the search depths typically reached today. The positions that are being compared are frequently completely different in character, suggesting that our mystery metric ought to compare pairs of positions not merely from local pockets of the search space but globally.

Consideration was also given to harnessing the great deal of recorded experience of human chess for developing a static evaluation function. Researchers have tried to make their machines play designated moves from test positions, but we focus on judgments about the relative worth of positions, reasoning that if these are correct then strong moves will emerge as a consequence. But how does one compute a correlation between the (ordinal) human assessment symbols, given in Table 1, with machine assessments? A literature review identified that a statistical measure known as Kendall's τ might be exactly what is needed.

After a brief overview of prior work on the automated tuning of static evaluation functions, we describe Kendall's τ, and our novel algorithm to implement it efficiently. We then discuss the materials used for our experiments, followed by details of our software implementation. Experimental results are provided in Section 6. After drawing some conclusions, we suggest further investigations to the interested researcher.

symbol	meaning
+–	white is winning
±	white has a clear advantage
⩲	white has an edge
=	the position is equal
⩱	black has an edge
∓	black has a clear advantage
–+	black is winning

Table 1. Symbols for chess position assessment.[1]

2. Prior Work

The precursor of modern machine learning in games is the work done by Samuel (1959, 1967). By fixing the value for a checker advantage, while letting other weights float, he iteratively tuned the weights of evaluation

[1] Two other assessment symbols, ∞ (the position is unclear) and =∞ (a player has positional compensation for a material deficit) are also frequently encountered. Unfortunately, the usage of these two symbols is not consistent throughout chess literature. Accordingly, we ignore positions labeled with these assessments.

function features so that the assessments of predecessor positions became more similar to the assessments of successor positions.

Hartmann (1989) developed the “Dap Tap” to determine the relative influence of various evaluation feature categories, or notions, on the outcome of chess games. Using 62,965 positions from grandmaster tournament and match games, he found that “the most important notions yield a clear difference between winners and losers of the games”. Unsurprisingly, the notion of material was predominant; the combination of other notions contribute roughly the same proportion to the win as material did alone. He further concluded that the threshold for one side to possess a decisive advantage is 1.5 pawns.

The DEEP THOUGHT (later DEEP BLUE) team applied least squares fitting to the moves of the winners of 868 grandmaster games to tune their evaluation function parameters as early as 1987 (Nowatzyk, 2000). They found that tuning to maximize agreement between their program’s preferred choice of move and the grandmaster’s was “not really the same thing” as playing more strongly. Amongst other interesting observations, they discovered that conducting deeper searches while tuning led to superior weight vectors being reached.

Tesauro (1995) initially configured a neural network to represent the backgammon state in an efficient manner, and trained it via temporal difference learning (Sutton, 1988). After 300,000 self-play games, the program reached strong amateur level. Subsequent versions also contained hidden units representing specialized backgammon knowledge and used minimax search. TD-GAMMON is now a world-class backgammon player.

Beal and Smith (1997) applied temporal difference learning to determine piece values for a chess program that included material, but not positional, terms. Program versions using weights resulting from five randomized self-play learning trials each won a match versus a sixth program version that used the conventional weights given in most introductory chess texts. They have since extended their reach to include piece-square tables for chess (Beal and Smith, 1999a) and piece values for Shogi (Beal and Smith, 1999b).

Baxter, Tridgell, and Weaver (1998) applied temporal difference learning to the leaves of the principal variations returned by alpha-beta searches to learn feature weights for their program KNIGHTCAP. Through online play against humans, KNIGHTCAP’s skill level improved from beginner to strong master. The authors credit this to: the guidance given to the learner by the varying strength of its pool of opponents, which improved as it did; the exploration of the state space forced by stronger opponents who took advantage of KNIGHTCAP’s mistakes; the initialization of material values to reasonable settings, locating KNIGHTCAP’s weight vector “close in parameter space to many far superior parameter settings”.

Buro (1995) estimated feature weights by performing logistic regression on win/loss/draw-classified Othello positions. The underlying log-linear model is well suited for constructing evaluation functions for approximating winning probabilities. In that application, it was also shown that the evaluation function based on logistic regression can perform better than those based on linear and quadratic discriminant functions. Later, Buro (1999) presented a much superior approach, using linear regression and positions labeled with the final disc differential to optimize the weights of thousands of binary pattern features.

Kendall and Whitwell (2001) evolved intermediate-strength players from a population of poor players by applying crossover and mutation operators to generate new weight vectors, while discarding vectors that performed poorly.

3. Kendall's Tau

Concordance, or agreement, occurs where items are ranked in the same order. Kendall's τ is all about the similarities and differences in the ordering of ordered pairs. Consider two pairs, (x_i, y_i) and (x_k, y_k). Compare both the x values and the y values. Table 2 defines the relationship between the pairs.

relationship between x_i and x_k	relationship between y_i and y_k	relationship between (x_i, y_i) and (x_k, y_k)
$x_i < x_k$	$y_i < y_k$	Concordant
$x_i < x_k$	$y_i > y_k$	Discordant
$x_i > x_k$	$y_i < y_k$	Discordant
$x_i > x_k$	$y_i > y_k$	Concordant
$x_i = x_k$	$y_i \neq y_k$	extra y pair
$x_i \neq x_k$	$y_i = y_k$	extra x pair
$x_i = x_k$	$y_i = y_k$	duplicate pair

Table 2. Relationships between ordered pairs.

Table 3 contains a grid representing ordered pairs of machine and human evaluations. The value in each cell indicates the number of corresponding pairs; blank cells indicate that no such pairs are in the data set. Sample machine and human assessments are on the x- and y-axes, respectively.

To compute τ for a collection of ordered pairs, each ordered pair is compared against all other pairs. The total number of concordant pairs is designated S^+ ("S-positive"). Similarly, the total number of discordant pairs is designated S^- ("S-negative").

Consider the table cell (0.0, =). There are six entries, containing seven data points, located strictly below and to its left; these are concordant pairs and so contribute to S^+. The two discordant pairs, strictly below and to its right, contribute to S^-. We do not consider any cells from above the cell of interest. If we did so, we would end up comparing each pair of ordered pairs twice instead of once. Finally, the 2 contained in the cell indicates that there are two (0.0, =) data points; hence the examination of this cell has produced 7 * 2 = 14 concordant pairs, and 2 * 2 = 4 discordant pairs.

	-1.6	-1.1	-0.7	-0.6	-0.3	-0.1	0.0	0.1	0.2	0.3	0.5	0.9	1.3
+−	1										1		
±									1		1		
⩲				1	1		1			1			1
=				1		1	2	1	2				
⩱			2		1	1							
∓				1						1			
−+	1	1										1	

Table 3. (machine, human) assessments, n = 25.

τ is given by:

$$\tau = \frac{S^{+} - S^{-}}{n(n-1)/2}$$

The denominator equals the number of unique possible comparisons between any two ordered pairs from a collection of *n* ordered pairs.

For the data in Table 3, S^{+} is 162, S^{-} is 83, and *n*, the number of ordered pairs, is 25. τ equals 0.2633; we might also say that the concordance of the data is 0.2633. Possible concordance values range from +1, representing complete agreement in ordering, to -1, representing complete disagreement in ordering. Whenever there are extra or duplicate pairs, the values of +1 and -1 are not achievable.

Cliff (1996) provides a more detailed exposition of Kendall's τ, discussing variations thereof that optionally disregard extra and duplicate pairs. Cliff labels what we call τ as τ_a, and uses it most often, noting that it has the simplest interpretation of the lot.

A straightforward implementation would perform the process illustrated above for each cell of the table. Our novel, algorithmically superior implementation allocates additional memory space, and in successive single passes through the data, applies dynamic programming to compute tables containing the number of data points that are:

either on the same row as or below the current cell;
either on the same column or to the right of the current cell;
either on the same column or to the left of the current cell;
strictly below and to the right of the current cell;
strictly below and to the left of the current cell.

Then, in a final pass, S^{+} and S^{-} are computed by multiplying the number of data points in the current cell by the data in the final two tables listed. It is also possible to use more passes, but less memory, by performing the sweeps to the left and to the right serially instead of in parallel.

There is a better-known ordinal metric in common use: Spearman's ρ, also known as Spearman correlation. In our application, the number of distinct human assessments is constant. Therefore, after initial data processing has identified the unique machine assessments for memory allocation and indexing purposes, τ is computed in time linear in the number of unique machine assessments, which is not possible for ρ. Prototype implementations confirmed that τ was significantly quicker to compute for large data sets.

Not only does τ more directly measure what interests us ("for all pairs of positions (A, B), is position B better than position A?"), it is also more efficient to compute than plausible alternatives. Therefore, we concentrate on τ in this paper.

4. Chess-Related Components

Many chess programs, or chess engines, exist. Some are commercially available; most are hobbyist. For our work, we selected CRAFTY, by Robert Hyatt (1996) of the University of Alabama. CRAFTY is the best chess engine choice for our work for several reasons: the source was readily available to us, facilitating experimentation; it is the strongest such open-source engine today; previous research has already been performed using CRAFTY. We worked with version 19.1 of the program.

4.1 Training Data

To assess the correlation of τ with improved play, we used 649,698 positions from *Chess Informant* 1 through 85 (Sahovski, 1966). These volumes cover the important chess games played between January 1966 and September 2002. This data set was selected because it contains a variety of assessed positions from modern grandmaster play, the assessments are made by qualified individuals, it is accessible in a non-proprietary electronic form, and chess players around the world are familiar with it.

We used a 32,768-position subset for the preliminary feature weight tuning experiments reported here.

4.2 Test Suites

English chess grandmaster John Nunn (1999) developed the Nunn and Nunn II test suites of 10 and 20 positions, respectively. They serve as starting positions for matches between computer chess programs, where the experimenter is interested in the engine's playing skill independent of the quality of its opening book. Nunn selected positions that are approximately balanced, commonly occur in human games, and exhibit variety of play. We refer to these collectively as the "Nunn 30".

Don Dailey, known for his work on StarSocrates and CilkChess, prepared a file of two hundred commonly reached positions, all of which are ten ply from the initial position. We refer to these collectively as the "Dailey 200".

5. Software Implementation

Here we detail some specifics of our implementation. We discuss both alterations made to Crafty and new software written as a platform for our experiments.

5.1 Use of Floating-Point Computation

We modified Crafty so that variables holding machine assessments are declared to be of an aliased type rather than directly as integers. This allows us to choose whether to use floating-point or integer arithmetic via a compilation switch. The use of floating-point computation provides a learning environment where small changes in values can be rewarded. With these modifications, Crafty is slower, but only by a factor of two to three on a typical personal computer. The experiments were performed with this modified version; however, all test matches were performed with the original, integer-based evaluation implementation. Further details can be found in Section 6.

It might strike the reader as odd that we chose to alter Crafty in this manner rather than scaling up all the evaluation function weights. There are significant practical disadvantages to that approach. How would we know that everything had been scaled? It would be easy to miss some value that needed to be changed. How would we identify overflow issues? It might be necessary to switch to a larger integer type. How would we know that we had scaled up the values far enough? It would be frustrating to have to repeat the procedure.

By contrast, the choice of converting to floating-point is safer. Precision and overflow are no longer concerns. Also, by setting the typedef to be a non-arithmetic type we can cause the compiler to emit errors wherever type mismatches exist. Thus, we can be more confident that our experiments rest upon a sound foundation.

5.2 Hill Climbing

We implemented an iteration-based learner, and a hill-climbing algorithm. Other iteration-based algorithms may be substituted for the hill-climbing code if desired. Because we are not working with an analytic function, we measure the gradient empirically.

We multiply $V_{current}$, the current weight of a feature being tuned, by a number fractionally greater than one[1] to get V_{high}, except when $V_{current}$ is near zero, in which case a minimum distance between $V_{current}$ and V_{high} is enforced. V_{low} is then set to be equidistant from $V_{current}$, but in the other direction, so that $V_{current}$ is bracketed between V_{low} and V_{high}. Two test weight vectors are generated: one using V_{high}, the other using V_{low}. All other weights for these test vectors remain the same as in the base vector. This procedure is performed for each weight that is being tuned. For example, when 11 parameters are being learned, 1 + 11 * 2 = 23 vectors are examined per iteration: the base vector, and 22 test vectors.

The three computed concordances related to a weight being tuned ($\tau_{current}$, τ_{low}, and τ_{high}) are then compared. If all three are roughly equal, no change is made: we select $V_{current}$. If $\tau_{current}$ is lower than both τ_{low} and τ_{high}, we choose the V corresponding to the highest τ. If they are in either increasing or decreasing order, we use the slope of test points (V_{low}, τ_{low}) and (V_{high}, τ_{high}) to interpolate a new point. However, to avoid occasional large swings in parameter settings, we bound the maximum change from $V_{current}$. The final case occurs when $\tau_{current}$ is higher than both τ_{low} and τ_{high}. In this case, we apply inverse parabolic interpolation to select the apex of the parabola formed by the three points, in the hope that this will lead us to the highest τ in the region.

Once this procedure has been performed for all of the weights being learned, it is possible to postprocess the weight changes, for instance to normalize them. However, at present we have not found this to be necessary. The chosen values now become the new base vector for the next iteration.

5.3 Automation

A substantial amount of code was written to automate the communication of work and results between multiple, distributed instantiations of CRAFTY and the PostgreSQL database. We implemented placeholder scheduling (Pinchak, 2002) so that learning could occur more rapidly, and without human intervention.

[1] The tuning experiments reported in this paper used 1.01.

5.4 Search Effort Quantum

Traditionally, researchers have used search depth to quantify search effort. For our learning algorithm, doing so would not be appropriate: the amount of effort required to search to a fixed depth varies wildly between positions, and we will be comparing the assessments of these positions. However, because we did not have the dedicated use of computational resources, we could not use search time either. While it is known that chess engines tend to search more nodes per second in the endgame than the middlegame, this difference is insignificant for our short searches because it is dwarfed by the overhead of preparing the engine to search an arbitrary position. Therefore, we chose to quantify search effort by the number of nodes visited.

We instructed CRAFTY to search 16,384 nodes to assess a position. Earlier experiments that directly called the static evaluation or quiescence search routines to form assessments were not successful. When searching 1,024 nodes per position, we had mixed results. Like the DEEP THOUGHT team (Nowatzyk, 2000), we found that larger searches improve the quality of learning. The downside is, of course, the additional processor time required by the learning process.

There are positions in our data set from which CRAFTY does not complete a 1-ply search within 16,384 nodes, because its quiescence search explores many sequences of captures. When this occurs, no evaluation score is available to use. Instead of using either zero or the statically computed evaluation (which is not designed to operate without a quiescence search), we chose to throw away the data point for that particular computation of τ, reducing the position count (n). However, the value of τ for similar data of different population sizes is not necessarily constant. As feature weights are changed, the shape of the search tree for positions may also change. This can cause CRAFTY to not finish a 1-ply search for a position within the node limit where it was previously able to do so, or vice versa. When many transitions in the same direction occur simultaneously, noticeable irregularities are introduced into the learning process. Ignoring the node count limitation until the first ply of search has been completed may be a better strategy.

5.5 Performance

Early experiments were performed using idle time on various machines in our department. Lately, we have had (non-exclusive) access to clusters of personal computer workstations, which is helpful because the task of computing τ for distinct weight vectors within an iteration is trivially parallel. Examining 32,768 positions and computing τ takes about two minutes per weight vector. The cost of computing τ is negligible in comparison, so in the best case, when there are enough nodes available for the concordances of all weight vectors of an iteration to be computed simultaneously, learning proceeds at the rate of 30 iterations per hour.

6. Experimental Results

We demonstrate that concordance between human judgments and machine assessments increases with increasing depth of machine search. This result, combined with knowing that play improves as search depth increases (Thompson, 1982), in turn justifies our attempt to use this concordance as a metric to tune selected feature weights of CRAFTY's static evaluation function.

6.1 Concordance as Machine Search Effort Increases

In Table 4 we computed τ for depths 1 through 7 for *n* = 649,698 positions, performing work equivalent to 211 billion (10^9) comparisons at each depth. S^+ and S^- are reported in billions. As search depth increases, the difference between S^+ and S^-, and therefore τ, also increases. The sum of S^+ and S^- is not constant because at different depths different amounts of extra y-pairs and duplicate pairs are encountered.

depth	$S^+ / 10^9$	$S^- / 10^9$	τ
1	110.374	65.298	0.2136
2	127.113	48.934	0.3704
3	131.384	45.002	0.4093
4	141.496	36.505	0.4975
5	144.168	34.726	0.5186
6	149.517	30.136	0.5656
7	150.977	29.566	0.5753

Table 4. τ computed for various search depths, n = 649,698.

It is difficult to predict how close an agreement might be reached using deeper searches. Two effects come into play: diminishing returns from additional search, and diminishing accuracy of human assessments relative to ever more deeply searched machine assessments. Particularly interesting is the odd-even effect on the change in τ as depth increases. It has long been known that searching to the next depth of an alpha-beta search requires relatively much more effort when that next depth is even than when it is odd (Marsland, 1983). Notably, τ tends to increase more in precisely these cases.

Similar experiments performed using increasing node counts, and increasing wall clock time (on a dedicated machine) with a different, smaller data set also gave increasing concordance, but, as expected, did not exhibit the staggered rise of the increasing depth searches. In sum, these experiments lend credibility to our belief that τ is a direct measure of decision quality.

6.2 Tuning of CRAFTY's Feature Weights

CRAFTY uses centipawns (hundredths of a pawn) as its evaluation function resolution, so experiments were performed by playing CRAFTY as distributed versus CRAFTY with the learned weights rounded to the nearest centipawn. Each program played each position both as White and as Black. The feature weights we tuned are given along with their default values in Table 5.

feature	default value
king safety scaling factor	100
king safety asymmetry scaling factor	-40
king safety tropism scaling factor	100
blocked pawn scaling factor	100
passed pawn scaling factor	100
pawn structure scaling factor	100
bishop	300
knight	300
rook on the seventh rank	30
rook on an open file	24
rook behind a passed pawn	40

Table 5. Tuned features, with CRAFTY'S default values.

The scaling factors were chosen because they act as control knobs for many subterms. Bishop and knight were included because they participate in the most common piece imbalances. Trading a bishop for a knight is common, so it is important to include both to show that one is not learning to be of a certain weight chiefly because of the weight of the other. We also included three of the most important positional terms involving rooks. Material values for the rook and queen are not included because trials showed that they climbed even more quickly than the bishop and knight do, yielding no new insights.

6.2.1 Tuning from Arbitrary Values

Figure 1 illustrates the learning. The 11 parameters were all initialized to 50, where 100 represents both the value of a pawn and the default value of most scaling factors. For ease of interpretation, legend contents are ordered to match up with the vertical ordering of corresponding data at the rightmost point on the x-axis. For instance, bishop is the topmost value, followed by knight, then τ, and so on. τ is measured on the left y-axis in linear scale; weights are measured on the right y-axis in logarithmic scale, for improved visibility of the weight trajectories.

Rapid improvement is made as the bishop and knight weights climb swiftly to about 285, after which τ continues to climb, albeit more slowly. We attribute most of the improvement in τ to the proper determination of weight values for the minor pieces. All the material and positional weights are tuned to reasonable values.

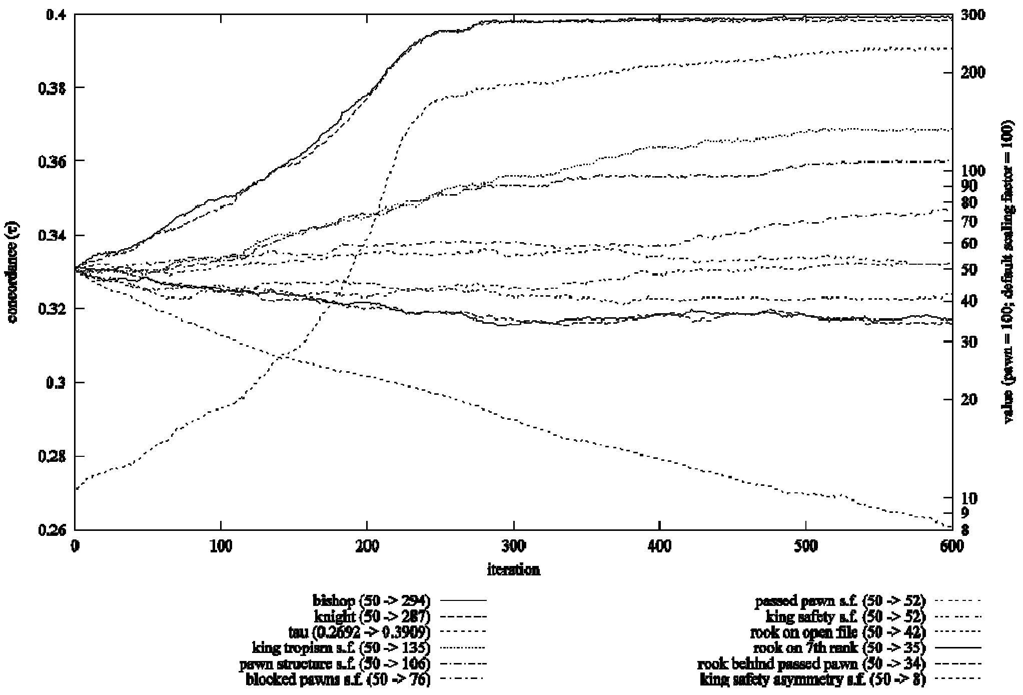

Figure 1. Change in weights from 50 as τ is maximized.

The scaling factors learned are more interesting. The king tropism and pawn structure scaling factors gradually reached, then exceeded CRAFTY's default values of 100. The scaling factors for blocked pawns, passed pawns, and king safety are lower, but not unreasonably so. However, the king safety asymmetry scaling factor dives quickly and relentlessly. CRAFTY's default value for this term is –40; perhaps we should have started it at a lower value to speed convergence.

Tables 6 and 7 contain match results of the weight vectors at specified iterations during the learning illustrated in Figure 1. Each side plays each starting position both as White and as Black, so with the Nunn 30 test, 60 games are played, and with the Dailey 200 test, 400 games are played. Games reaching move 121 were declared drawn.

The play of the tuned program improves dramatically as learning occurs. Of interest is the apparent gradual decline in percentage score for later iterations on the Nunn 30 test suite. The DEEP THOUGHT team (Nowatzyk, 2000) found that their best parameter settings were achieved before reaching maximum agreement with GM players. Perhaps we are also experiencing this phenomenon. We used the Dailey 200 test suite to attempt to confirm that this was a real effect, and found that by this measure too, the weight vectors at iterations 300 and 400 were superior to later ones.

iteration	wins	draws	losses	percentage score
0	3	1	56	5.83
100	3	9	48	12.50
200	14	21	25	40.83
300	21	26	13	56.67
400	19	28	13	55.00
500	18	26	16	51.67
600	18	23	19	49.17

Table 6. Match results (11 weights tuned from 50 vs. default weights), 5 minutes per game, Nunn 30 test suite.

Throughout our experimentation, we have found that our tuned feature weights tend to perform better on the Nunn test suite than the Dailey test suite. Nunn's suite contains positions of particular strategic and tactical complexity. Dailey's suite is largely more staid, and contains positions from much earlier in the game. CRAFTY's default weights appear to be more comfortable with the latter than the former.

iteration	wins	draws	losses	percentage score
0	3	13	384	2.38
100	12	31	357	6.88
200	76	128	196	35.00
300	128	152	120	51.00
400	129	143	128	50.13
500	107	143	150	44.63
600	119	158	123	49.50

Table 7. Match results (11 weights tuned from 50 vs. default weights), 5 minutes per game, Dailey 200 test suite.

We conclude that the learning is able to yield settings that perform comparably to settings tuned by hand over years of games versus grandmasters.

6.2.2 Tuning from CRAFTY's Default Values

We repeated the just-discussed experiment with one change: the feature weights start at CRAFTY's default values rather than at 50. Figure 2 depicts the learning. Note that we have negated the values of the king safety asymmetry scaling factor in the graph so that we could retain the logarithmic scale on the right y-axis, and also for another reason, for which see below.

While most values remain normal, the king safety scaling factor surprisingly rises to almost four times the default value. Meanwhile, the king safety asymmetry scaling factor descends even below –100. The combination indicates a complete lack of regard for the opponent's king safety, but great regard for its own. Table 8 shows that this conservative strategy is by no means an improvement.

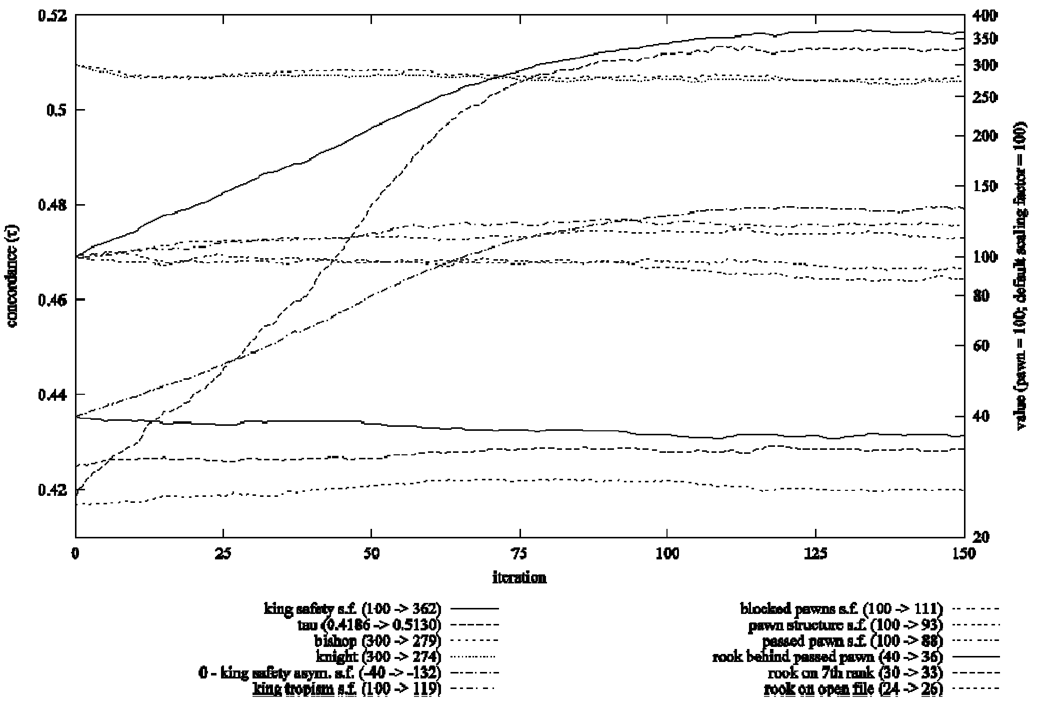

Figure 2. Change in weights from CRAFTY's defaults as τ is maximized.

iteration	wins	draws	losses	percentage score
25	19	23	18	50.83
50	16	31	13	52.50
75	11	32	17	45.00
100	14	28	18	46.67
125	9	23	28	34.17
150	8	35	17	42.50

Table 8. Match results (11 weights tuned from defaults vs. default weights), 5 minutes per game, Nunn 30 test suite.

The most unusual behaviour of the king safety and king safety asymmetry scaling factors deserves specific attention. When the other nine terms are left constant, these two terms behave similarly to how they do when all eleven terms are tuned. In contrast, when these two terms are held constant, no statistically significant performance difference is found between the learned weights and CRAFTY's default weights. When the values of the king safety asymmetry scaling factor are negated as in Figure 2, it becomes visually clear from their trajectories that the two terms are behaving in a codependent manner. More investigation is required to determine the root cause of this behaviour.

7. Conclusion

We have proposed a new procedure for optimizing static evaluation functions based upon globally ordering a multiplicity of positions in a consistent manner. This application of ordinal correlation is fundamentally different from prior evaluation function tuning techniques. We believe it is worth further exploration, and hope it will lead to a new perspective and fresh insights about decision making in game-tree search.

While our initial results show promise, more work is certainly needed. It is important to keep in mind that we tuned feature weights in accordance with human assessments. Doing so may simply not be optimal for computer play. Nonetheless, it is worth noting that having reduced the playing ability of a grandmaster-level program to candidate master strength by significantly altering several important feature weights, the learning algorithm was able to restore the program to grandmaster strength.

7.1 Reflection

Having identified the anomalous behaviour in Figure 2, it is worth looking again at Figure 1. The match results suggest that all productive learning occurred by iteration 400 at the latest, after which a small but perceptible decline appears to occur. The undesirable codependency between the king safety and king safety asymmetry scaling factors also appears to be present in the later iterations of the first experiment.

Furthermore, our training data is small enough (n = 32,768) that overfitting is a consideration. Future learning experiments should use more positions. This may in turn reduce the search effort required per position to tune weights well. Although we are not certain why larger searches improve the quality of learning, as the amount of search used per machine assessment increases, the amount of information gathered about how relative weights interact also increases. On the surface, then, the improvement is not illogical.

While some weights, for instance the positional rook terms, learned nearly identical values in both experiments, other features exhibited more variance. For cases such as the king tropism and blocked pawns scaling factors, it could be that comparable performance may be achieved with a relatively wide range of values.

In our reported experiments, computation of τ was dominated by the search effort to generate machine assessments, enough so that the use of Spearman's ρ (or perhaps even Pearson correlation, notwithstanding our original rationale) may also have been possible. Maximizing these alternative metrics could be tried, at least when the training data contains relatively few positions. Other optimization strategies, for instance genetic algorithms, could also be tried.

It was not originally planned to attempt to maximize τ only upon assessments at a specific level of search effort. Unfortunately, we encountered implementation difficulties, and so reverted to the approach described herein. We had intended to log the node number or time point along with the new score whenever the evaluation of a position changes. This would have, without the use of excessive storage, provided the precise score at any point throughout the search. We would have tuned to maximize the integral of τ over the period of search effort. Implementation of this algorithm would more explicitly reward reaching better evaluations more quickly, improving the likelihood of tuning feature weights and perhaps even search control parameters effectively.

7.2 Future Directions

While our experiments used chess assessments from humans, it is possible to use assessments from deeper searches and/or from a stronger engine, or to tune a static evaluation function for a different domain. Depending on the circumstances, merging consecutively-ordered fine-grained assessments into fewer, larger categories may be desirable. Doing so could even become necessary should the computation of τ dominate the time per iteration, but this is unlikely unless one uses only negligible search to form machine assessments.

Elidan et al. (2002) found that perturbation of training data could assist in escaping local maxima during learning. Our implementation of τ, designed with this finding in mind, allows non-integer weights to be assigned to each cell. Perturbing the weights in an adversarial manner as local maxima are reached, so that positions are weighted slightly more important when generally discordant, and slightly less important when generally concordant, could allow the learner to continue making progress.

It would also be worthwhile to examine positions of maximum disagreement between human and machine assessments, in the hope that study of the resulting positions will identify new features that are not currently present in CRAFTY's evaluation. Via this process, a number of labeling errors would be identified and corrected. However, we do not believe that this would materially affect the outcome of the learning process.

A popular pastime amongst computer chess hobbyists is to attempt to discover feature weight settings that result in play mimicking their favourite human players. By tuning against appropriate training data, e.g., from opening monographs and analyses published in *Chess Informant* and elsewhere that are authored by the player to be mimicked, training an evaluation function to assess positions similarly to how a particular player might actually do so should now be possible.

Producers of top computer chess software play many games against their commercial competitors. They could use our method to model their opponent's evaluation function, then use this model in a minimax (no longer negamax) search. Matches then played would be more likely to reach positions where the two evaluation functions differ most, providing improved winning chances for the program whose evaluation function is more accurate, and object lessons for the subsequent improvement of the other.

Identifying the most realistic mapping of CRAFTY's machine assessments to the seven human positional assessments is also of interest. This information would allow CRAFTY (or a graphical user interface connected to CRAFTY) to present scoring information in a human-friendly format alongside the machine score.

Acknowledgements

We would like to thank: Yngvi Björnsson, for the use of his automated game-playing software, and for fruitful discussions; Don Dailey, for access to his suite of 200 test positions; Robert Hyatt, for making CRAFTY available, and also answering questions about its implementation; Peter McKenzie, for providing PGN to EPD conversion software; NSERC, for partial financial support [Grant OPG 7902 (Marsland)].

References

Baxter, J., Tridgell, A., and Weaver, L. (1998). KnightCap: A Chess Program that Learns by Combining TD(λ) with Game-tree Search. *Proceedings of the Fifteenth International Conference in Machine Learning* (IMCL) pp. 28-36, Madison, WI.

Beal, D. F. and Smith, M. C. (1997). Learning Piece Values Using Temporal Differences. *ICCA Journal*, Vol. 20, No. 3, pp. 147-151.

Beal, D. F. and Smith, M. C. (1999a). Learning Piece-Square Values using Temporal Differences. *ICCA Journal*, Vol. 22, No. 4, pp. 223-235.

Beal, D. F. and Smith, M. C. (1999b). First Results from Using Temporal Difference Learning in Shogi. *Computers and Games* (eds. H.J. van den Herik and H. Iida), pp. 113-125. Lecture Notes in Computer Science 1558, Springer-Verlag, Berlin, Germany.

Buro, M. (1995). Statistical Feature Combination for the Evaluation of Game Positions. *Journal of Artificial Intelligence Research* 3, pp. 373-382, Morgan Kaufmann, San Fransisco, CA.

Buro, M. (1999). From Simple Features to Sophisticated Evaluation Functions. *Computers and Games* (eds. H.J. van den Herik and H. Iida), pp. 126-145. Lecture Notes in Computer Science 1558, Springer-Verlag, Berlin, Germany.

Cliff, N. (1996). *Ordinal Methods for Behavioral Data Analysis*. Lawrence Erlbaum Associates.

Elidan, G., Ninio, M., Friedman, N., and Schuurmans, D. (2002). Data Perturbation for Escaping Local Maxima in Learning. *AAAI 2002*, pp. 132-139.

Hartmann, D. (1989). Notions of Evaluation Functions tested against Grandmaster Games. *Advances in Computer Chess 5* (ed. D.F. Beal), pp. 91-141, Elsevier Science Publishers, Amsterdam, The Netherlands.

Hyatt, R.M. (1996). CRAFTY – Chess Program. ftp://ftp.cis.uab.edu/pub/hyatt/v19/crafty-19.1.tar.gz.

Kendall, G. and Whitwell, G. (2001). An Evolutionary Approach for the Tuning of a Chess Evaluation Function. *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*. http://www.cs.nott.ac.uk/~gxk/papers/cec2001chess.pdf.

Marsland, T. A. (1983). Relative Efficiency of Alpha-beta Implementations. *IJCAI 1983*, pp. 763-766.

Nunn, J. (1999). http://www.computerschach.de/test/nunn2.html.

Nowatzyk, A. (2000). http://www.tim-mann.org/deepthought.html. Also, see publications by Anantharaman et al. (1987) and Hsu et al. (1988).

Pinchak, C., Lu, P., and Goldenberg, M. (2002). Practical Heterogeneous Placeholder Scheduling in Overlay Metacomputers: Early Experiences. *8th Workshop on Job Scheduling Strategies for Parallel Processing*, Edinburgh, Scotland, U.K., pp. 85-105, also to appear in LNCS 2537 (2003), pp. 205-228, also at http://www.cs.ualberta.ca/~paullu/Trellis/Papers/placeholders.jsspp.2002.ps.gz.

Plaat, A., Schaeffer, J., Pijls, W., and Bruin, A. de (1996). Best-First Fixed-Depth Game-Tree Search in Practice. *Artificial Intelligence*, Vol. 87, Nos. 1-2, pp. 255-293.

Shannon, C. E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, Vol. 41, pp. 256-275.

Sahovski Informator (1966). *Chess Informant*: http://www.sahovski.com/.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, No. 3, pp. 211-229.

Samuel, A. L. (1967). Some Studies in Machine Learning Using the Game of Checkers. II – Recent Progress. *IBM Journal of Research and Development*, Vol. 2, No. 6, pp. 601-617.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, Vol. 3, pp. 9-44.

Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, Vol. 38, No. 3, pp. 55-68. http://www.research.ibm.com/massive/tdl.html.

Thompson, K. (1982). Computer Chess Strength. *Advances in Computer Chess 3*, (ed. M.R.B. Clarke), pp. 55-56. Pergamon Press, Oxford, UK.

Thompson, K. (1986). Retrograde Analysis of Certain Endgames. *ICCA Journal*, Vol. 9, No. 3, pp. 131-139.