Temporal Difference Learning Applied to a High-Performance Game-Playing Program

Jonathan Schaeffer, Markian Hlynka
{jonathan, markian}@cs.ualberta.ca
Department of Computing Science
University of Alberta
Edmonton, Canada T6G 2H1

Vili Jussila
vili@cc.hut.fi
Laboratory of Computational Engineering
Helsinki University of Technology
Helsinki, Finland

Abstract

The temporal difference (TD) learning algorithm offers the hope that the arduous task of manually tuning the evaluation function weights of game-playing programs can be automated. With one exception (*TD-Gammon*), TD learning has not been demonstrated to be effective in a high-performance, world class game-playing program. Further, there has been doubt expressed by game-program developers that learned weights could compete with the best hand-tuned weights. *Chinook* is the World Man-Machine Checkers Champion. Its weights were manually tuned over 5 years. This paper shows that TD learning is capable of competing with the best human effort.

1 Introduction

The most time-consuming aspect of building a high-performance game-playing program is the design, implementation and tuning of the evaluation function. Designing the knowledge-based features in the evaluation function and implementing them in a fast, efficient manner remains a difficult task for humans, although there have been some limited successes at automating this task [Buro, 1995; van Rijswijck, 2001; Fawcett and Utgoff, 1992]. Historically, tuning the evaluation function—adjusting the weight (importance) of each feature contributing to the evaluation—has been a tedious, manual task. There have been numerous attempts to automate this (for example, [van der Muelen, 1989; Anantharaman, 1991]), but none of these techniques achieved the requisite high performance. Buro has achieved impressive results using linear regression in his Othello program [Buro, 2001], but it is not clear that similar techniques will work for a broader class of games.

Temporal difference (TD) learning has emerged as a powerful reinforcement learning technique for incrementally tuning parameters [Sutton and Barto, 1998]. Tesauro applied TD learning to tune the weights of a neural net, in the process building a world class backgammon program (*TD-Gammon*) [Tesauro, 1995]. For several years, this remained an isolated success story in the games literature, as the conditions in backgammon that appeared to favor TD learning did not exist in other high profile games, such as chess. In 1997, the TDLeaf algorithm was introduced (TD learning applied to minimax search) [Beal, 1997] and it achieved some success with chess (*KnightCap* [Baxter *et al.*, 1998a; 1998b; 2000]).

In none of the above cases has it been possible to compare the performance of TD learning to that of the best-tuned human weights. *TD-Gammon* learned through self-play; a human-tuned version of the program does not exist. *KnightCap* learned through playing speed chess against humans on the Internet. A human-tuned version of the program does exist, but both it and the TD version of the program are far below grandmaster level in strength. Also, tuning for speed chess is not necessarily representative of what needs to be learned for tournament chess (where the search depths are greater). In all the examples of TD learning applied to games, there has been a nagging question: Can TD-learned weights be successful in strong (world-championship-calibre) game-playing programs? For challenging games, such as chess, game developers have expressed doubt that tuned weights would be sufficient to achieve the highest levels of performance.

Chinook is the World Man-Machine Checkers Champion [Schaeffer, 1997]. Its evaluation function weights were tuned manually over a period of 5 years. They were extensively tested both in self-play games and in hundreds of games against top human players (including playing 96 games for the World Checkers Championship). This paper investigates whether the tuning of evaluation function weights in *Chinook* can be replaced by TDLeaf learning. The experimental data indicates that the answer is "yes", as well as giving new insights into TD learning in game-playing programs. This is the first known attempt to conduct a detailed study that compares hand-tuned and TD-trained weights in an established high-performance game program.

2 Temporal Difference Learning

Temporal difference learning is an unsupervised reinforcement learning algorithm [Sutton and Barto, 1998]. It learns from experience without a model of the environment's dynamics, and updates its estimates based on other, as yet unconfirmed, estimates. Thus, TD can learn without waiting for a final outcome on a given task; it evaluates the sub-steps between evaluations.

The TD(λ) algorithm can be succinctly expressed as follows [Sutton and Barto, 1998]. Given a series of predictions, $P_0 ... P_{t+1}$ (search results from a game in this context), then the weights in the evaluation function can be modified as follows:

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \qquad (1)$$

The change in weights (Δw_t) depends upon the predictions (P_k) and the gradient of the predicted value of the k^{th} state with respect to the weights (∇P_k).

The λ term is a decay-rate parameter. It determines the extent to which learning is affected by subsequent states. A λ of zero is equivalent to learning only from the next state. A λ of 1 indicates learning only from the final reinforcement signal; in the case of a game, the final won/lost assessment. α is a step size parameter: the proportion of adjustment to allow on each iteration. Thus, the λ parameter determines whether the algorithm is applying short or long range prediction, while α determines how quickly this learning takes place.

TD(λ) is a proven algorithm for reinforcement leaning. One of its important advantages is that it can be computed incrementally. However, to apply it to problems utilizing search, some refinements are required. The TDLeaf algorithm is essentially TD learning applied to minimax search. TDLeaf was originally implemented by Beal and Smith [Beal, 1997], though not under the name TDLeaf (which is attributed to [Baxter *et al.*, 1998a; 1998b]). The crux of the algorithm is *not* to use the position at the root of the search tree to tune the search. Instead, tuning takes place using the position of the leaf node at the end of the principal variation of the search. The principal variation is the line of best play; the position at the end of this line of play has had its value backed-up to the root of the search.

TDLeaf was implemented in the chess program *KnightCap* [Baxter *et al.*, 1998a; 1998b]. Baxter *et. al* report that the program's chess rating rose from 1650 to 2150 in three days (308 games). While this sounds impressive, there are a few caveats that need to be mentioned. First, the results were achieved at speed chess; there is no indication that these results will apply to (slower) over-the-board chess. Second, the learning plateaued well before achieving a high level of play. Finally, despite the early promise of TDLeaf, no one has demonstrated that it can out-perform the best set of human-tuned weights. Many researchers active in the computer games community (including the first author) have publicly doubted that TD learning is capable of achieving the high level of performance required in a game-playing program.

3 Training

Chinook's evaluation function is the linear combination of 23 knowledge-based features for each of 4 game phases. Two features cannot be modified (the value of a checker and the value of a king) because of some search code dependencies. Hence, a total of 84 parameters need to be tuned.[1]

Chinook supports an integer evaluation function and integer weights. TD learning is inherently a real-number task. Thus, *Chinook* was modified to accept floating point values. However, the final position evaluation would be converted to an integer, allowing these changes to be restricted to the evaluation function.

Chinook and the TD learning (TDL) were kept as separate programs which communicated with each other using text files. The file *Chinook* reads in includes information about the opening sequence, search depth, number of turns to play, and the weights for both sides. After a game finishes, *Chinook* outputs a file containing the result of the game and evaluations for each weight component. TDL uses this file to adjust the weights and then starts a new game with the revised weights. During this process TDL also saves information about the progress of the learning, such as the results of each played game and the value of the weights after each game. Because not all weights are updated every turn, we also record the frequency at which weights are modified to discover if some game situations happen so seldomly that the corresponding weight does not get much training.

The TDLeaf algorithm in TDL operates on pairs of moves. The weights of move i are updated based in part upon the evaluation at move $i+1$. Not all pairs of moves were candidates for TDLeaf updating. Capture moves are forced in checkers, so if only one move is legal in a position, no updating would occur. Also, occasionally *Chinook*'s search algorithm was incapable of recovering the principal variation as far as the leaf node. In this case, TDLeaf could not be applied.

The training routine was as follows:

1. *Chinook* is used to play two weight files against each other.
2. TDL modifies one or both of the weight files based on the game played.
3. This procedure is iterated until learning is seen to plateau (typically before 10,000 iterations).

To prevent the programs from playing the same moves in every game, an opening book was used which included the standard 144 checkers openings, each 3 ply long. The learning rate α was chosen to be 0.01 and the TD parameter λ was set to 0.95. These values were chosen based on the *KnightCap* experience, but finding the best settings remains an open question.

Several different approaches were attempted for learning. Each experiment involved starting with all weights set to zero, train using TD learning, and then evaluate

[1]Note that the 21 tunable features are each the result of a function that itself may contain many parameters. These lower-level parameters are not addressed in this paper.

the learned weights by using them in a match against the tournament version of *Chinook*.

The first approach involved training the weights by playing against tournament *Chinook* (*teacher learning*). The goal was to determine how effective the learning was given the benefit of a high-performance teacher. The second set of tests involved self-play (*self-play learning*). Here the goal was to see if the learning could boot-strap itself to achieve high performance. In both cases, separate experiments were performed using 5, 9, and 13-ply searches, generating separate weights for the black and white sides.

Examining the output of a training session shows that the performance of the learned weights against tournament *Chinook* rapidly improves at the beginning of the session due to the poor starting values. After this initial period, the rate of improvement slows until at roughly 4,000 games a stable state is reached.[2] In the experiments, only 84 weights had to be learned. In contrast, *KnightCap* had to learn 1,500 parameters in its first set of experiments. This was later expanded to 6,000 parameters [Baxter *et al.*, 1998a]. The small number of parameters used in *Chinook* accounts for the relatively fast learning phase.

4 Results

Trained weight sets were tested against the tournament version of *Chinook*. Evaluation consisted of a 288-game match (each program playing both sides of the 144 openings). All versions of the program used *Chinook's* 6-piece endgame databases. Tournament *Chinook* has no knowledge of how to play simplified endgame positions because it assumes that the database will always be used. Using the databases had the benefit of speeding up the experiments since, once a position with 6 or fewer pieces was reached, the databases would give the final result of the game, thereby ending the game.

4.1 Baseline

How important are the evaluation function weights? The obvious way to answer this question is to set all the weights to zero and see how the program performs. In effect, this "zero knowledge" program uses only material for its evaluation. The result of the match is not surprising: a 34–254 game loss to tournament *Chinook* with 15-ply searches (the endgame database knowledge salvaged many draws). Since both programs used the same search depth, the quality of the knowledge is solely responsible for the match score. As an additional data point, all the weights were set to one. Now the program "knows" how to evaluate a position, but it does not understand the relative importance of each feature. Having some knowledge is obviously beneficial, as this program loses by a smaller margin (an average score of 94.5–193.5).

[2] *KnightCap* required fewer training games, but its performance levels off at a playing strength that is considerably below world-championship caliber.

4.2 Teacher Learning

Figures 1a, 1b, and 1c shows the performance of white and black weights that were trained using 5, 9, and 13-ply searches, respectively. The x-axis shows the search depth used for the evaluation, and the y-axis shows the number of wins minus losses from the learning program's point of view.

The 5-ply-trained weight set does well against *Chinook* when playing games with a search depth of 5 ply, but performance quickly tapers off as the programs play games using larger search depths (Figure 1a). A similar pattern is seen with the 9-ply-trained weights (Figure 1b). The experiment shows the learned weights defeating *Chinook* in matches up to 9-ply, but tapering off with deeper searches. Whereas with 5-ply searches, the results of training using the white positions dominates those for the black positions, with 9-ply searches the difference between the two sets of weights essentially disappears.

For the 13-ply results (Figure 1c), the data is not as clear. As before performance seems strong around the training search depth (13-ply) and there is the suggestion that it is beginning to taper off for deeper searches (it would take several weeks to get the 17-ply data). Unlike the previous graphs, the performance of the weights using search depths shallower than the training depth are poorer. However, the difference between the 7-ply and 13-ply results in Figure 1c represents only a 7% improvement, well within the statistical variability expected.

The graphs reveal an important insight for anyone using TD learning in game-playing programs: the weights must be trained using the depths of search expected to be seen in practice. Deeper searches provide a more accurate approximation of the root position's true value for the TD algorithm to learn. This suggests that the *KnightCap* weights that were obtained using speed chess will not perform well in slower tournament chess (similarly, [Anantharaman, 1991] needs deeper searches to be effective). In effect, there is no free lunch; you can't use shallow search results to approximate deep results.

We experimented with creating separate weight sets for playing white and black. The purpose was to see if the specialization of the weight sets could lead to better play, given that white generally has an opening advantage. Surprisingly, using the black weights only when playing black, and the white weights only when playing white, does not seem to be statistically significantly better in our experiments. After the opening phase of the game, the resulting types of positions seen are similar for white and black, resulting in a similar learning experience. This is more pronounced with deeper searches (since the search can see "beyond" the opening) than it is with shallower searches. This would account for the large difference between the white and black performance in Figure 1a.

The previous experiments have not been entirely fair. Both the training and evaluation was done using the same 144 starting positions. The good results for the learned weights might be a consequence of the program

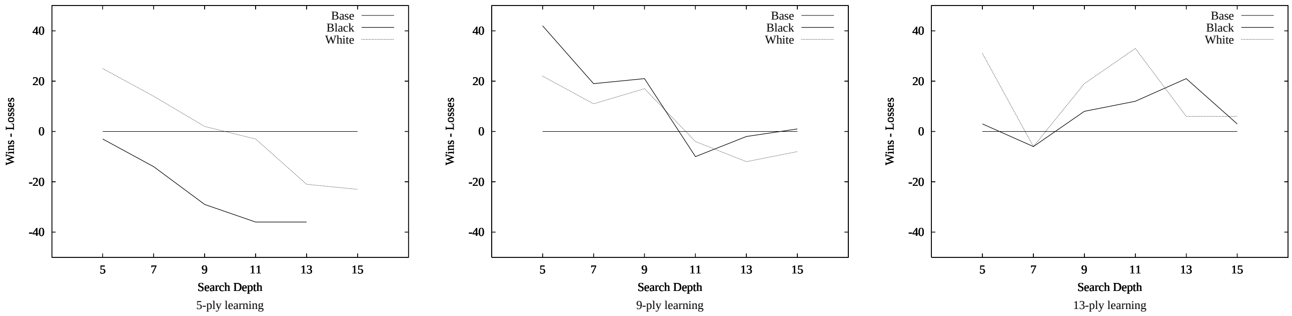

Figure 1: Teacher learning: a) 5-ply, b) 9-ply and c) 13-ply.

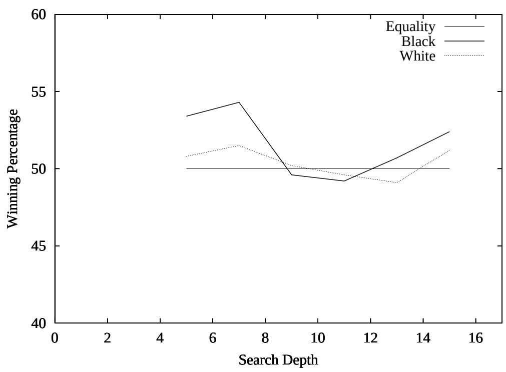

Figure 2: 786 game matches using 13-ply learning.

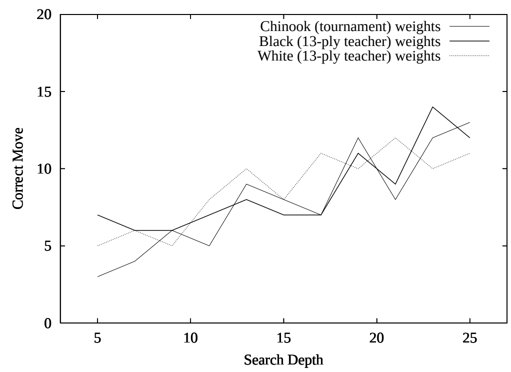

Figure 3: *Chinook* test set data.

being trained to play the same opening positions that are used in the evaluation. To get another indicator of the performance of the trained data, a second experiment was performed. From a collection of games played by former world champion Marion Tinsley, all positions 8-ply into the games were extracted (393 positions). These positions were used as the openings for a 786-game match between the learned weights and tournament *Chinook*.

Figure 2 shows the results for both the 13-ply white and black trained weight sets, expressed as the percentage of total points scored (over 786 games). At deeper search depths, the tuned weights perform slightly better than the hand-tuned weights, although the difference is not statistically significant. Given the match length, it is safe to say that the performance of the TD weights is comparable to that of the best hand-tuned effort.

4.3 Self-Play Learning

In this set of experiments, the program learned through self-play without the benefit of having a strong opponent to train against. All the self-play results analyzed to date are consistent with that seen in the previous section, with the exception that the training takes longer to plateau. The 13-ply-trained black weight sets scored 50.2% of the points in a 786-game match (using 15-ply searches) with tournament *Chinook*, while the white weights scored 48.3%.

The self-play data strongly indicates that a good teacher is not needed for the program to learn a set of evaluation function weights that achieves world-championship-calibre performance. This is wonderful news for game-program developers, as it suggests that manual weight tuning may be a thing of the past.

The *KnightCap* self-play results are not as good as those reported here. This is likely a consequence of the number of parameters being tuned; fewer parameters are easier to fit.

4.4 Additional Data

There is a test set of 19 positions (taken from *Chinook* games) that have proven to be particularly difficult for the program to solve. In these positions, the opponents (mostly humans players) demonstrated profound insights into the game that *Chinook*, at the time the game was played, could not match. None of the positions is easily resolved by search; the quality of the knowledge is the critical factor. Most of these positions were the motivation for adding additional features to the evaluation function and/or making major changes to the feature weights. During the development of the program, these positions were often used to benchmark the program.

Chinook has been tested on these positions using three weight sets: original, white teacher training at 13-ply, and black teacher training at 13-ply. The results are shown in Figure 3. For each of the positions, the program versions searched 5-ply to 25-ply deep (in increments of 2 ply). The figure records which search depths produced the correct solution to the positions. Note the general trend that increased search depth results in more frequent correct solutions. However, in most of the positions, the programs get the correct answer at the end of an iteration, only to switch to a different move on the next iteration. All versions tested were indecisive in their move choice for most of the positions (a further indication that the positions are indeed still very hard for *Chinook*). Both TD weight sets perform comparably to the original weight set in *Chinook*. There is nothing to suggest that one weight set is significantly better than the others.

4.5 Comments

One must caution that most of the experimental results have been obtained from machine-versus-machine games.[3] The results may be different in machine-versus-human play. Unfortunately, with *Chinook* retired and the program significantly stronger than all human players, there are no opportunities to evaluate just how good the weights are in play against humans.

Although TD learning promises to reduce the effort to build a high-performance game-playing program, deciding on the evaluation function features still remains largely a manual chore. Some of the features in *Chinook*'s evaluation function came as the result of extensive human analysis of the program's play to identify deficiencies in the program's knowledge. Once a new feature was added to the program, then the manual tuning would begin again. TD learning makes this a less painful process. The human identifies and adds the new knowledge; the program learns the new weight set.

5 Examining the Weights

Table 1 shows *Chinook*'s original weights and those learned from the white positions with 13-ply searches. Not unexpectedly, there are some major differences:

1. Several of the features occur rarely in certain phases of the game and, hence, the computer-generated weights may be off (or irrelevant) because of insufficient training. For example, "free king", "king center" and "loose checker" are mainly endgame features. Over 8,533 games (a total of 238,0403 learning updates), in phase 1 these features occurred only 177, 16, and 184 times, respectively.

2. "Value of move" is a small bonus given to the side whose turn it is to move. In most positions, from a human's point of view, having the right to move is a small advantage. From the computer's point of view, value of move is just a constant added to the evaluation function. The negative value for this weight suggests that, in general, the evaluation scores obtained using trained weights are a bit high, and this feature is being used to make a small linear adjustment to the value to get a better fit.

3. The mobility terms are the most important part of *Chinook's* evaluation function, after material balance. The computer-generated weights are comparable to the human weights in that they generally have the same sign and similar magnitudes.

[3][Berliner *et al.*, 1990] mentions the pitfalls that can arise from basing conclusions solely on self-play games.

4. The terms "frozen", "dog hole", "loose men", "d2e7", and "free king" were late additions to the evaluation function. These terms were added to address problems that arose in play against human players. Both human and machine weights are affected by the infrequency with which these features occur. It is also likely that these features are not as common in machine-versus-machine play as they are in human-versus-machine play.

5. The biggest surprise is the difference in value for "trapped kings" (kings that are immobile in corners and cannot be freed). This is a symptom of the above problem. Against computers, some humans play for a trapped king since, historically, that was a major weakness in computer play (and, indeed, was a problem with early versions of *Chinook*). The evaluation function detects this situation and penalizes it heavily. However, since the training is from self-play, *Chinook* never plays to "dupe" *Chinook* into trapping its king. Consequently, the TD-learning infrequently sees this feature arising and, when it does, it is usually not a position where this is the decisive factor.

The lesson here is that play against human players is necessary to complete the training. Humans have their own set of biases, predilections, and notion of "good" and "bad". The additional training will be most pronounced in the weights of the features that infrequently occur in machine-versus-machine play.

Despite the radical differences between the TD-learned and the human-tuned set of weights, one cannot dispute the success of each version. On the one hand, it is remarkable that TD learning is as successful as it is given that the learning is based solely on game-play feedback with no human intervention. On the other hand, it is a triumph of human cognitive abilities that the human solution to a complicated optimization problem can indeed be competitive with a computer solution. The final result, that the human-tuned solution and the TD-tuned solution are roughly equivalent in performance, reflects well on both man and machine.

6 Conclusions

There are two parts to an evaluation function: the function terms and the weighting of these terms. This paper strengthens the case that TD learning provides an effective solution to the latter problem. Learned weights can compete with (and perhaps exceed) the performance of the best hand-tuned weights in a high-performance game-playing program.

TD learning opens up new opportunities for improving a program's abilities. For example, the program could have a different set of weights for each opening, or for different classes of positions. Different weights could be used based on the expected depth of search. In addition, the program developer can experiment with new features, and let the learning algorithm decide what is relevant. None of this would be practical if these weights had to be tuned manually.

	Original Weights				Learned Weights			
Name	1	2	3	4	1	2	3	4
Value of move	4	3	3	2	-2.30	-6.94	-2.48	0.46
Free mobility	1	2	3	4	3.40	6.50	2.77	6.47
Some mobility	-4	-6	-8	-10	0.89	-4.62	-8.97	-6.08
Recapture mobility	3	3	3	3	-1.72	2.33	5.77	3.25
No-move mobility	-1	-1	-2	-4	-2.13	-4.45	-4.17	-1.30
Exception mobility	0	0	0	0	-0.47	0.89	5.56	2.47
Double-cap mobility	-6	-6	-6	-6	-0.15	-1.45	-2.34	-1.08
Balance	5	4	3	2	1.14	4.53	1.35	-0.86
Advancement	-1	0	0	0	3.59	-3.54	-0.39	-0.68
Centrality	2	2	1	0	-1.91	8.44	1.25	-1.86
Angle	1	1	0	0	0.79	3.26	3.24	3.23
Back row	4	3	3	2	1.93	8.75	11.77	6.28
Shadow	3	2	1	0	0.74	4.05	1.23	-0.19
Trapped king	32	32	32	32	-0.01	-0.02	0.90	0.73
Loose checker	5	5	5	5	0.03	1.38	4.76	3.72
King center	3	3	3	3	-0.01	0.89	5.65	5.94
D2E7	3	2	1	0	0.09	-0.06	0.22	0.57
Free king	20	20	20	20	0.38	1.66	5.37	3.69
Dog-hole	5	5	5	5	-0.08	0.16	1.66	0.89
Loose men	15	15	15	15	0.25	1.88	5.33	5.56
Frozen	10	10	10	10	0.00	0.05	-0.09	-0.12

Table 1: Comparing weights.

The dream of creating a games engine that can achieve high performance for any board game is one step closer to reality. High-performance black box search engines now exist (e.g. [Brungger *et al.*, 1999; Romein, 2000]), as well as generic (mediocre performance) games engines (see www.zillionsofgames.com). The last piece of the puzzle, automatically discovering the features needed for the evaluation function, remains elusive.

7 Acknowledgments

Financial support was provided by NSERC and iCORE.

References

[Anantharaman, 1991] T. Anantharaman. *A Statistical Study of Selective Min-Max Search.* PhD thesis, Carnegie Mellon University, 1991.

[Baxter *et al.*, 1998a] J. Baxter, A. Tridgell, and L. Weaver. Experiments in parameter learning using temporal differences. *ICCA Journal*, 21(2):84–99, 1998.

[Baxter *et al.*, 1998b] J. Baxter, A. Tridgell, and L. Weaver. KnightCap: A chess program that learns by combining TD(λ) with game-tree search. *ICML*, pages 28–36, 1998.

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

[Beal, 1997] D. Beal. Learning piece values using temporal differences. *ICCA Journal*, 20(3):147–151, 1997.

[Berliner *et al.*, 1990] H. Berliner, G. Goetsch, M. Campbell, and C. Ebeling. Measuring the performance potential of chess programs. *Artificial Intelligence*, 43(1):7–21, 1990.

[Brungger *et al.*, 1999] A. Brungger, A. Marzetta, K. Fukuda, and J. Nievergelt. The parallel search bench ZRAM and its applications. *Annals of Operations Research*, 90:45–63, 1999.

[Buro, 1995] M. Buro. Statistical feature combination for the evaluation of game positions. *JAIR*, 3:373–382, 1995.

[Buro, 2001] M. Buro. Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 2001. To appear.

[Fawcett and Utgoff, 1992] T. Fawcett and P. Utgoff. Automatic feature generation for problem solving systems. *ICML*, pages 144–153, 1992.

[Romein, 2000] J. Romein. *Multigame – An Environment for Distributed Game-Tree Search.* PhD thesis, Vrije Universiteit, 2000.

[Schaeffer, 1997] J. Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers.* Springer Verlag, 1997.

[Sutton and Barto, 1998] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[Tesauro, 1995] G. Tesauro. Temporal difference learning and TD-Gammon. *CACM*, 38(3):58–68, 1995.

[van der Muelen, 1989] M. van der Muelen. Weight assessment in evaluation functions. *Advances in Computer Chess 5*, pages 81–89, 1989.

[van Rijswijck, 2001] J. van Rijswijck. Learning from perfection (a data mining approach to evaluation function learning in awari). *2nd International Conference on Computers and Games*, 2001. To appear.