TD-GAC: Machine Learning Experiment with Give-Away Checkers

Daniel Osman and Jacek Mańdziuk

Faculty of Mathematics and Information Science,
Warsaw University of Technology,
Plac Politechniki 1, 00-661 Warsaw, Poland
E-mail: dosman@prioris.mini.pw.edu.pl,
mandziuk@mini.pw.edu.pl

Abstract

This paper presents results of applying the temporal difference learning methods in the game of give-away checkers. Each of the following: the TD(λ), TDLeaf(λ) and a pseudo evolutionary algorithm was used in order to modify weights of a linear state value function in a self-play mode. Empirical results show the success of Temporal Difference methods in improving the quality of computer player's policy. The most promising results were achieved while learning on non-positive game outcomes and on strong opponents using the TD(λ) algorithm.

1 Introduction

The temporal difference algorithm TD(λ) [1] was successfully used in learning computer playing policy in games like backgammon [2, 3], checkers [4, 5], chess [6], go [7], othello [8] and other [9, 10]. The algorithm was first used by A.L. Samuel in 1959 [4] but received its name after the work of R. Sutton [1].

In this paper we present the results of applying the TD(λ) [1], TDLeaf(λ) [11] and a pseudo evolutionary algorithm EVO [10] in the US variant of give-away checkers (GAC). Our goal was to compare the efficacy of several implementations of TD and TDLeaf, i.e. learning on random opponents vs. learning on strong ones and learning on all games' outcomes vs. learning on non positive ones only. The best results were recorded after training the learning players on non positive game outcomes and strong opponents. Next we compared these results with the ones achieved by the EVO algorithm. The pseudo evolutionary algorithm was significantly inferior to TD and TDLeaf.

The rules of making moves in GAC [12] are exactly the same as in checkers [13]. What is different between these two games is the goal: in GAC the aim is to either loose all pieces or to be unable to make a legal move on one's turn.

Although computer checkers have achieved world class game play [14], the GAC program able to compete with the best human players hasn't been developed. The game at first glance may seem trivial or at least not interesting. A closer look however reveals that it is not the case. For example a simple piece disadvantage isn't a good estimation of in-game player's performance. A player left with one king can easily be forced to eliminate all of the opponent's pieces. The main idea of this experiment was to check if the computer player could be trained to some reasonable level of GAC play using a simple value function and the TD algorithm for modifying weights.

This article is an extended version of the work presented in [15]. Several new results and observations are also presented - in particular the ones concerned with the third training/testing stage of the experiment.

2 Value Function

Following [16] we used the game state evaluation function of the form

$$P(s,w) = \begin{cases} MAX & \text{if } s \text{ is a win} \\ MIN & \text{if } s \text{ is a loss} \\ 0 & \text{if } s \text{ is a tie} \\ V(s,w) & \text{for all other } s \in S, \end{cases} \qquad (1)$$

where $MAX = +100$, $MIN = -100$ and

$$V(s,w) = a \cdot \tanh\left(b \cdot \sum_{i=1}^{n} w_i x_i\right), \qquad a = 99,\ b = 0.027 \qquad (2)$$

$V(s,w)$ is a state value function where $x = [x_1, \ldots, x_n]$ is a game state feature vector consisting of state to integer mappings, $w_1, \ldots, w_n$ are real coefficients. The parameter a equals 99 to guarantee that $V(s,w) \in (-99;+99)$ and b equals 0.027 to prevent $V(s,w)$ from growing too fast and to cause $|V(s,w)|$ to approach 99 only when the sum in eq. (2) is greater than +99 or less than -99. $V(s,w)$ can be seen as a simple weighted sum of game state features. $\tanh(\cdot)$ in eq. (2) was only used for technical reasons in order to limit the possible values of the evaluation function by $P(s,w) \in (MIN;MAX)$. The value of $P(s,w)$ for state s depends on the weight vector w. The TD(λ) algorithm was responsible for changing these weights in order to achieve a good playing policy.

The 22 feature detectors or terms implemented in GAC were based on the ones introduced by Samuel [4]. We have chosen to use the simplest ones (without complex terms) and we did not use fixed weights for the piece advantage term. Every feature represents a numerical property of a game state. For example the number of pieces on a board, the number of pieces on a diagonal, the number of some predefined formations, etc. Every feature for the active side is computed relatively to the same feature for the passive side. So in fact we have a piece advantage term rather than a feature that describes the number of active pieces on the board.

The value function is used to assign values to leaf nodes of a fixed-depth 4-ply min-max game search tree. After that a move is executed following the best line of play found. In this case it is said that a player follows a greedy policy because at every state always the best move according to the program is made.

3 Temporal Difference Algorithm

The TD(λ) algorithm [1] was chosen to modify the weights of the value function presented in section 2. At time t the weight update vector Δw_t is computed from the following equation:

$$\Delta w_t = \alpha \cdot \delta_t \cdot e_t \tag{3}$$

where $\alpha \in (0, 1)$ is the learning step-size parameter. The second component $\delta_t = r_{t+1} + \gamma V(s_{t+1}^{(l)}, w) - V(s_t^{(l)}, w)$ represents the temporal difference in state values. r_{t+1} is the scalar reward obtained after a transition from state s_t to s_{t+1}. $\gamma \in (0, 1)$ is the discount parameter. In our experiments $\gamma = 1$ and $r_t = 0$ for all t, although a small negative value of r_t could have been used in order to promote early wins. $s_t^{(l)}$ is the principal variation leaf node obtained after performing a d-ply min-max search starting from state s_t (the state observed by the learning player at time t). In other words $V(s_t^{(l)}, w)$ is the min-max value of state s_t or a d-step look-ahead value of s_t. The last component of equation (3) is the eligibility vector e_t updated in the following recursive equation:

$$e_0 = 0, \qquad e_{t+1} = \nabla_w V_{t+1} + (\gamma\lambda) e_t, \tag{4}$$

where $\lambda \in (0, 1)$ is the decay parameter. $\nabla_w V_t$ is the gradient of $V(s_t, w)$ relative to weights w. Formally, if we treat the value function $V(s, w)$ as a weighted sum of state features, the i-th element of this gradient equals:

$$(\nabla_w V_t)_i = \frac{\partial V(s_t, w)}{\partial w_i} = \phi_i(s_t) \quad i = 1, \ldots, n, \;\; t = 1, 2, ... \tag{5}$$

where n is the size of the weight vector and s_t is the state observed at time t. Combining eqs. (4) and (5) gives the straightforward formula for updating e_t:

$$e_0 = 0, \qquad e_{t+1} = \phi(s_{t+1}) + (\gamma\lambda) e_t,$$

where $[\phi(s_{t+1}) = \phi_1(s_{t+1}), \ldots, \phi_n(s_{t+1})]$ is the vector of game state features of state s_{t+1}. The eligibility vector holds the history of state features encountered. The elements of this vector (unless they are encountered again) decay exponentially according to the decay parameter λ. The features in e_t are thought to be significant while performing weight updates at time t. The eligibility vector is the main tool used by delayed reinforcement learning for assigning credit to past actions [17].

In TD(λ) weight updates can be computed on-line after every learning player's move. However in this paper the learning player's weights are modified only after the game ends (off-line TD(λ)). This enables to condition weight replacement with the final result of the game (win, loss or tie).

The weight vector determines the value function which in turn determines the player's policy $\pi : (S, R^K) \rightarrow A$. Policy $\pi(s, w)$ is a function that for every state $s \in S$ and some weight vector $w \in R^K$ maps an action $a \in A$. Thus $\pi(s, w)$ tells the computer player which move to perform at state s. In the experiments presented in this paper, the computer player always performed the move that followed the best line of play found. The best line of play was determined by evaluating the leaf nodes of a game search tree. Our goal is to learn an optimal policy that maximizes the chance of a win (learning control). This is achieved indirectly by learning an optimal value function.

Following the greedy policy in each move can lead to insufficient state space exploration. This is why some authors chose to introduce random instabilities in the process of optimal move selection. This is known as following an ε-greedy policy. We have decided to use a different approach. The learning player always followed the greedy policy defined by his current weight vector and state space exploration was forced by playing in turn with 25 different opponents. This approach made the learning process fully deterministic which helped in comparing learning strategies described in section 4.1.

4 Experiment Design

The computer player that has his weights changed (trained) after every game by the $TD(\lambda)$ algorithm is called the learning player. One training stage consisted of 10,000 training games. We present the results of 3 training stages. In each stage there were 10 learning players each of which trained against a set of 25 opponents. In the first stage all weights for players 1 and 6 were initialized with 0. The rest of the learning players had their weight vectors initialized with random numbers $r \in (-10.0, +10.0)$. The first stage was the only one that had different training opponents for each learning player. Each of the 250 opponents had its weight vector initialized with random numbers. All the opponent's weight vectors were pairwise different and were never changed during learning. In training game number i the learning player played against opponent $(i \bmod 25) + 1$. Players 1 to 5 always played using white pieces (they performed the first move). Players 6 to 10 played using red pieces. For every win the learning player received 1 point, for a tie 0.5 and for a loss 0 points.

The second stage was similar. The only difference was that the opponents were no longer random players. Twenty weight vectors used by the opponents were taken from the learning players achieved in the first stage. The remaining 5 vectors were initialized with random numbers. These 25 opponents were the same for all learning players. In the third stage the same opponent enhancement process was performed. However this time all the 25 opponents used weights achieved by the learning players in the second stage.

The game tree search depth was fixed to $d = 4$ mainly due to time limitations. The size of the weight vector was $n = 22$.

In order to check if the performance in the training phase really reflects policy improvement, a testing phase was introduced for each training stage. One test phase was performed after every 250 training games. In the test phase the current weight vectors of the 10 learning players were used to play against 100 opponents. These test opponents were not encountered earlier by the learning players. Two groups of test opponents were used. One consisted of random opponents. The second one represented a group of 100 strong opponents trained previously in another experiment. There was no learning during the test phase.

4.1 Learning Strategies

In order to perform training, some form of training strategy needs to be chosen. We have applied three different learning strategies denoted by LB, LL and L3. The first one - LB - was the simplest. The player's weights were modified after each game no matter the final outcome. In LL the weights were modified only after non-positive game outcomes (ie. tie or lost). The L3 learning strategy was similar to LL in that learning took place only after non-positive game outcomes. Additionally however the learning player could play up to three times in a row with the same opponent if the opponent kept winning with the learning player. In short, the L3 strategy can be seen as a way of forcing the learning player to play more games against stronger opponents. According to our knowledge, the L3 method was used for the first time in our experiments.

4.2 TDLeaf and EVO

TDLeaf(λ) [11] is a modification of TD(λ), enhanced for use in domains where a d-step look ahead state search is performed in order to choose an action to be executed at a given time step. Compared to TD, the TDLeaf algorithm differs only in that in eq. (4) the gradient $\nabla_w V_{t+1}$ is computed for the leaf state s^l_{t+1} and not the root state s_{t+1}.

In the pseudo evolutionary learning method (EVO) [10] only one opponent was used during training. This opponent was modified once every two games by adding Gaussian noise to all of its weights. If the learning player lost two games in a row or lost one and drawn one then its weights were shifted 5% in the direction of the opponent's weight vector. Otherwise no modification of the learning player's weights was performed. The two players changed sides after each game.

5 Results

5.1 Tuning α and λ

Over 300,000 initial games were played in order to observe performance for different choices of parameters α and λ. The results in this phase showed that

games	α	λ
1 - 2,500	0.000100	0.95
2,501 - 5,000	0.000020	0.70
5,001 - 7,500	0.000010	0.50
7,501 - 10,000	0.000005	0.20

Table 1: The scheme for decreasing the values of α and λ.

	TD		EVO
games	LL	LB	
1 - 2,500	64.9%	64.7%	64.5%
2,501 - 5,000	70.5%	70.2%	63.5%
5,001 - 7,500	72.0%	72.2%	64.8%
7,501 - 10,000	70.8%	69.1%	63.1%

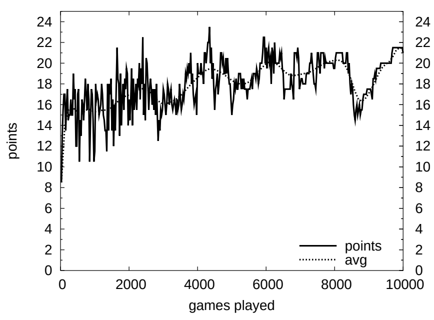
Table 2: Training phase stage 1.

applying appropriate scheme for changing these parameters can substantially shorten the learning time and enhance the results. Decreasing the step-size parameter α was necessary in order to achieve good performance. Tuning the decay parameter λ had less impact. It only helped in reaching the maximum performance in shorter time, i.e. after 7,500 games rather than 10,000 ones or more. Finally, the scheme described in Table 1 for decreasing α and λ was chosen. All the following experiments were based on games played using this scheme.

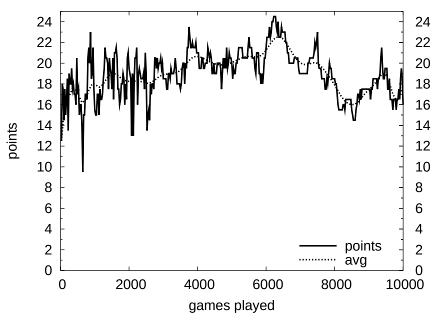
5.2 Stage 1. Learning on 25 Random Opponents

Table 2 presents the results of training the learning players on 25 random opponents. It is worth noting that the learning players' results exceeded 50% which means that policy learning really took place in the training phase. In this experiment TD-LL and TD-LB methods achieved comparable results. The highest average result (72.2%) was obtained using the LB method during games 5,001-7,500. In subsequent games the performance decreased.

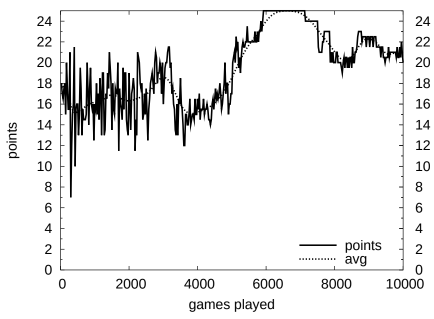
Results of the learning players during training were observed after every 25 games. The history of performance changes for the best players trained with TD-LL and TD-LB methods on 25 random opponents are presented in Figs. 1(a) and 1(b), resp. The best overall result was achieved by player 9-LB some time after the 6000-th game (Fig. 1(b)) and reached 24.5 out of 25 points. The worst overall result was a fall to 13 points (out of 25) for player 7-LL after $10,000$ games (not presented). The TD(λ) algorithm caused a quick improvement of the learning player's performance against random opponents. This improvement (from 50% to 64%) took place in less than 200 games. In subsequent games the performance increase was much slower. For the best player using the LL method there was a raise from 64% to 80% after 6000



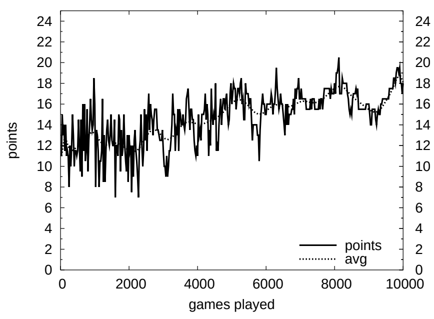
(a) Player 8-LL. Random opponents.



(b) Player 9-LB. Random opponents.



(c) Player 1-LL. Strong opponents.



(d) Player 10-LB. Strong opponents.

Figure 1: History of performance changes for the best players using LL and LB training strategies against random and strong opponents.

games (see Fig. 1(a)).

The test phase results are presented in Table 3. The first test phase with random opponents does not show significant differences between TD-LL and TD-LB. The second test phase, against strong opponents, is different and 1 to 8 percent point superiority of TD-LL over TD-LB can be observed. This result will be confirmed during the second training stage presented in section 5.3.

In both test phases the EVO algorithm was significantly inferior to TD. In Table 3 the EVO results exceeded 50% which means that this algorithm was able to play better than random players, but its performance is not satisfactory when compared with strong test opponents.

5.3 Stage 2. Learning on 25 Strong Opponents

After playing 10,000 games against 25 random opponents, in the second stage the learning players were trained against 25 strong opponents during a period of another 10,000 games. The results are presented in Table 4. This time the TD-LL method was superior to TD-LB one. The difference in performance was

	Random opponents			Strong opponents		
	TD		EVO	TD		EVO
games	LL	LB		LL	LB	
1 - 2,500	66.8%	69.9%	51.7%	44.2%	43.9%	30.1%
2,501 - 5,000	70.1%	70.9%	55.7%	51.6%	43.2%	32.9%
5,001 - 7,5000	71.7%	70.0%	57.2%	49.2%	43.5%	35.3%
7,501 - 10,000	70.7%	70.7%	59.4%	50.0%	44.5%	36.8%

Table 3: Test phase. Stage 1.

	TDLeaf		TD			EVO
games	LL	LB	LL	LB	L3	
1 - 2,500	61.3%	59.0%	61.8%	52.9%	56.7%	65.3%
2,501 - 5,000	66.3%	65.6%	64.3%	54.9%	45.5%	65.7%
5,001 - 7,500	60.1%	63.3%	71.9%	51.8%	40.8%	62.2%
7,501 - 10,000	60.1%	64.6%	62.8%	52.3%	37.5%	70.1%

Table 4: Training phase stage 2.

about 10 to 20 percent points in favor of TD-LL during all the games played. The history of performance changes for the best TD-LL and TD-LB players is presented in Figs. 1(c) and 1(d), resp. While playing against strong test opponents, the main performance increase was observed only with the TD-LL method. Using the TD-LB method caused the results to be only slightly better than 50%. Player 10-LB (Fig. 1(d)) was the only learning player using the TD-LB method that recorder a steady performance increase.

The TD-L3 results were the worst during this training phase. Over time the learning players started to loose more and more games with the training opponents. This was caused by the decreasing of α in subsequent games. While playing against strong opponents it appeared that one or two weight modifications (after the first and the second subsequently lost games) were usually not enough to win in the third game. This was especially true when a small learning step size was used. Therefore in most cases the learning player recorded 3 loses in a row with the same opponent. The results when α was not decreased were as follows: 56.7%, 45.5%, 44.5%, 44.0%. This is 4-7 percent points better compared to the case of decreasing α. Please note that the overall training results of TD-L3 are expected to be inferior when compared to other methods. This is caused by the fact that we deliberately play more games against opponents with which we lost earlier.

The TDLeaf-LL results were similar to TD-LL except for the 71.9% result for TD-LL during games 5001-7500. The LB learning strategy for TDLeaf was enhanced by preventing the learning player from learning on the opponents' blunders (state transitions that were not predicted by the learning player). This limitation on the number of states that the learning player could learn on was applied only when the learning player won with the opponent. In case of a

	TDLeaf		TD			EVO
games	LL	LB	LL	LB	L3	
1 - 2,500	71.8%	71.9%	72.0%	69.5%	73.5%	58.7%
2,501 - 5,000	70.6%	72.5%	74.4%	71.2%	74.9%	56.2%
5,001 - 7,500	70.4%	72.4%	72.5%	72.1%	72.8%	56.8%
7,501 - 10,000	70.2%	71.4%	73.1%	73.2%	74.4%	54.9%

Table 5: Test phase - random opponents. Stage 2.

	TDLeaf		TD			EVO
games	LL	LB	LL	LB	L3	
1 - 2,500	51.7%	50.4%	53.3%	43.3%	53.3%	35.2%
2,501 - 5,000	52.2%	55.0%	56.7%	44.8%	57.4%	33.4%
5,001 - 7,500	50.6%	56.1%	54.0%	47.9%	56.7%	32.8%
7,501 - 10,000	50.8%	56.0%	54.8%	48.2%	56.1%	31.6%

Table 6: Test phase - strong opponents. Stage 2.

loss or tie, the learning player learned on all encountered states. A similar LB enhancement was used earlier in [6]. This enhancement caused that for TDLeaf the LB method was not so inferior (compared to LL) as it was for TD. Using the TDLeaf-LB method resulted in stable performance, without the sudden decrease observed in TDLeaf-LL. Moreover TDLeaf-LB was visibly superior over TD-LB.

The EVO results presented in Table 4 were obtained as a direct continuation of stage 1 learning. No opponent change was performed at the beginning of this stage due to the different learning strategy involved with EVO. The results achieved in stage 2 are slightly better than these from stage 1.

Results from the test phase of stage 2 are presented in Tables 5 and 6. They confirm the superiority of the TD-LL method used for training with strong opponents. The results for the TD-LL method in the test phase were a little better than for the TD-LB method after playing against random opponents. However while playing against strong test opponents, the difference in performance between the TD-LL and TD-LB methods reached 6-12 percent points in favor of TD-LL. The L3 method added a few more percent points to the TD-LL results which made this method superior among the ones tested.

Once again the EVO algorithm turned out to be the weakest. In Table 6 a steady performance increase can be observed but the results are still significantly inferior to TD and TDLeaf.

The TDLeaf results were similar to TD with TDLeaf-LB even slightly outperforming TD-LL during a few games. However these results never exceeded the ones obtained for TD-L3.

	TDLeaf	TD	
games	LB	LL	L3
1 - 2,500	47.0%	53.6%	34.8%
2,501 - 5,000	51.6%	52.2%	33.0%
5,001 - 7,500	53.7%	55.9%	31.7%
7,501 - 10,000	53.9%	54.9%	32.0%

Table 7: Training phase stage 3

	Random opponents			Strong opponents		
	TDLeaf	TD		TDLeaf	TD	
games	LB	LL	L3	LB	LL	L3
1 - 2,500	71.0%	71.8%	71.9%	49.9%	55.8%	51.4%
2,501 - 5,000	71.0%	70.5%	70.9%	48.1%	55.3%	56.5%
5,001 - 7,500	70.6%	70.8%	71.6%	49.8%	55.3%	55.1%
7,501 - 10,000	70.5%	70.8%	70.4%	50.3%	54.6%	55.3%

Table 8: Stage 3. The most promising learning methods. Test phase - random and strong opponents.

5.4 Stage 3. Training on 25 yet Stronger Opponents

In the third stage we continued training for a further 10,000 games. We picked only the most promising algorithms and methods in order to check whether subsequent training would increase the results any further. The results of the training phase are presented in Table 7. Note that these results should not be directly compared with the ones presented in Table 4 because in stage 3 the training opponents were stronger than those used in stage 2. The thing worth noting however is the steady increase of performance of TDLeaf-LB. The learning players over time learned to defeat their training opponents.

The test phase results for stage 3 are presented in Table 8. The results are between 1 and 7 percent points inferior to the ones presented for stage 2. The greatest decrease in performance can be seen in the TDLeaf-LB method and strong opponents. This is probably caused either by overtraining or by inappropriate use of the LB method when learning on strong opponents. The other reason could be that in the third stage all the training opponents were very strong. Training only on strong and perhaps similar opponents may not be an optimal learning strategy. On the other hand, it is worth to note that the results of TD-L3 did not degrade as much as in stage 2.

The results from stage 3 show that further improvement is not possible using the same techniques as before. Perhaps we have reached the maximum performance that could be achieved with a simple value function and shallow tree search used.

6 Conclusions

Results presented in sections 5.2 and 5.3 show that the TD(λ) algorithm can be successfully applied in the game of give-away checkers for learning the computer player's policy. The most important results are the ones presented in the test phases because they show the general policy quality of the trained computer players. The best test phase results (75% against random and 57% against strong opponents) were obtained when the computer player was trained using the L3 method. In this method the weights of the value function were updated only after non-positive game outcomes (i.e. loss or tie) and additionally the learning player concentrated on stronger opponents.

Another training method worth noting was LL. In this method learning took place only after non-positive game outcomes. An approach similar to the LL method presented in this article was used earlier by Baxter [6]. In his chess learning program (KnightCap), there was no weight update associated with moves not predicted by the learning player (blunders) in the case when the learning player won with a lower ranked opponent. However this method was not directly compared to LB in the cited paper.

The superiority of the L3 and LL methods can be explained in the following way. A loss or a tie of the learning player means that the weight vector is not optimal and a weight update is recommended. A win however probably means only that the opponent was not strong enough and another one could have performed better. Learning on such outcomes can be misleading. As can be seen from our results, ignoring these outcomes caused the training performance to increase significantly.

There was little or no difference between the LB and LL methods' performance in the first stage of the experiment. The reason behind this could be that while learning on random (weak) opponents no special care has to be taken in order to achieve good results and TD-LB is sufficient. Another explanation is that frequent weight updates, related to the LB method, may be desirable when playing against random opponents that share no common strategy. Frequent weight changing in this case could compensate for the superiority of L3 and LL learning.

References

[1] Sutton, R.: Learning to predict by the method of temporal differences. Machine Learning **3** (1988) 9–44

[2] Tesauro, G.: Practical issues in temporal difference learning. Australian Journal of Intelligent Information Processing Systems **8** (1992) 279–292

[3] Tesauro, G.: Temporal difference learning and td-gammon. Communications of the ACM **38** (1995) 58–68

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

[5] Schaeffer, J., Hlynka, M., Jussila, V.: Temporal difference learning applied to a high-performance game-playing program. In: International Joint Conference on Artificial Intelligence (IJCAI). (2001) 529–534

[6] Baxter, J., Tridgell, A., Weaver, L.: Knightcap: A chess program that learns by combining td(lambda) with game-tree search. In: Machine Learning, Proceedings of the Fifteenth International Conference (ICML '98), Madison Wisconsin (1998) 28–36

[7] Schraudolph, N.N., Dayan, P., Sejnowski, T.J.: Learning to evaluate go positions via temporal difference methods. In Baba, N., Jain, L., eds.: Computational Intelligence in Games. Volume 62. Springer Verlag, Berlin (2001)

[8] Walker, S., Lister, R., Downs, T.: On self-learning patterns in the othello board game by the method of temporal differences. In: Proceedings of the 6th Australian Joint Conference on Artificial Intelligence, Melbourne, World Scientific (1993) 328–333

[9] Kalles, D., Kanellopoulos, P.: On verifying game designs and playing strategies using reinforcement learning. In: Proceedings of the 2001 ACM Symposium on Applied Computing (SAC 2001), Las Vegas, NV, USA, ACM (2001)

[10] Kotnik, C., Kalita, J.K.: The significance of temporal-difference learning in self-play training td-rummy versus evo-rummy. In Fawcett, T., Mishra, N., eds.: Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), Washington, DC, USA, AAAI Press (2003) 369–375

[11] Baxter, J., Tridgell, A., Weaver, L.: Tdleaf(lambda): Combining temporal difference learning with game-tree search. Australian Journal of Intelligent Information Processing Systems **5** (1998) 168–172

[12] Alemanni, J.B.: Give-away checkers. http://perso.wanadoo.fr/alemanni/give_away.html (1993)

[13] ACF: American checkers federation. http://www.acfcheckers.com/ (1990)

[14] Schaeffer, J., Lake, R., Lu, P., Bryant, M.: Chinook: The world man-machine checkers champion. AI Magazine **17** (1996) 21–29

[15] Mańdziuk, J., Osman, D.: Temporal difference approach to playing give-away checkers. In Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A., eds.: Artificial Intelligence and Soft Computing - ICAISC 2004, 7th

International Conference, Zakopane, Poland, June 7-11, 2004, Proceedings. Volume 3070 of Lecture Notes in Computer Science., Springer (2004) 909–914

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

[17] Singh, S.P., Sutton, R.S.: Reinforcement learning with replacing eligibility traces. Machine Learning **22** (1996) 123–158