Improvement Obtained by Ordering the Tree-Search of an Agent for Draughts which Learns by Reinforcement

Valquíria Aparecida Rosa Duarte, Alana Bueno Otsuka, and Rita Maria Silva Julia

Federal University of Uberlandia - UFU, Uberlandia, Brazil
{valquiriaduarte@gmail.com,alana.otsuka@gmail.com,rita@facom.ufu.br}

Abstract. VisionDraughts is a good automatic draughts player which uses Temporal Difference learning to adjust the weights of an artificial Neural Network whose role is to help an alpha-beta algorithm combined with transposition table and iterative deepening to choose the best move corresponding to the current board states represented in the input of the network. The output of the network (prediction) is a value that indicate to witch extent the input state is favorable to the agent and it will be used in the process of updating the weights. This work improves the general performance of VisionDraughts by using the data stored in the transposition table to order the tree-search generated by the alpha-beta algorithm. As shown in this paper, this strategy will speed up the search process and will substantially reduce the occurrence of endgame loops.

Key words: Draughts, Temporal Difference, Alpha-Beta Pruning, Transposition Table, Iterative Deepening, Zobrist Key, Ordering Tree-Search.

1 Introduction

The choice of Draughts as an application domain is due to the fact that it presents significant similarities with several practical problems, such as interaction problems issue of the dialogue human/machine and Urban vehicle traffic control. To solve this kind of problem, an agent must be able to learn how to behave in an environment where the acquired knowledge is stored in an evaluation function, it must choose a concise set of possible attributes that best characterize the domain and to select the best action corresponding to a determined state [6].

In this context, Caixeta and Julia implemented the draughts playing program, VisionDraughts [7]. It uses an artificial neural network trained by the method of temporal difference (TD) learning and self-play with cloning. To choose a good move from the current state, it employs the alpha-beta pruning algorithm with transposition table and iterative deepening [8].

In order to improve VisionDraughts, the goal, here, is to exhibit the impact of partial ordering in tree-search. The thee-search is created during the search

process for the best move. This tree-search ordering is only possible because the player has a transposition table which keeps previously explored board states in the memory. These boards states can be used in future board evaluations [7].

Despite its good performance, VisionDraughts frequently do not succeed in final phases of a game, even being in advantageous situation compared to its opponent (endgames loops) [8], [6], [2]. With partial ordering inserted in this player, it was possible to notice that the incidence of end game loops was considerably reduced.

2 State of Art

The first great experiment in automatic learning for draughts is devoted to Samuel [1]. He uses features (NETFEATUREMAP mapping) to provide qualitative measures to better represent the properties of pieces on a board [1], [2].

Chinook is the world man-machine draughts champion and uses linear handcrafted evaluation functions (whose role is to estimate how much a board state is favorable to it) [3], [4], [5]. It has access to a library of opening moves from games played by grand masters and to an endgame database. To choose the best action to be executed, Chinook uses a parallel iterative alpha-beta search with transposition tables and the history heuristic [4]. The iterative deepening [13] is used to acquire move-ordering information in the transposition table. The human interference has been strongly significative for the high level of efficiency of Chinook.

Mark Lynch's Draughts player, NeuroDraughts, consists of a MLP (MultiLayer-Perceptron) neural network whose weights are updated by TD(λ) Reinforcement Learning methods. The agent is trained by self-play. The Minimax algorithm is used to choose the best action to be executed considering the current game board-state [2]. NeuroDraughts adopts the NETFEATUREMAP mapping techniques to represent the game board-states. The features are manually selected. Mark Lynch developed his draughts player trying to use the minimum of human intervention as possible.

In order to improve the NeuroDraughts abilities, Neto and Julia proposed LS-Draughts [6]: an automatic draughts player which extends the architecture of NeuroDraughts with a genetic algorithm that automatically generates, by means a Genetic Algorithm, a concise set of feature which are essencial for representing the game board states and to optimize the training of the neural network.

Still in order to improve NeuroDraughts, Caixeta and Julia proposed VisionDraughts [7], [8]. It is an automatic draughts player which replaces a very efficient tree-search routine which uses alpha-beta pruning, transposition table and iterative deepening for the minimax search module in NeuroDraughts. As the present work extends this player, the next section summarizes the techniques used in it.

3 Theoretical Foundations

This section shows the principal techniques used in the implementation of VisionDraughts. Considering that the main contribution of this work is focused on the ordering of the tree-search, the alpha-beta algorithm will be explained in the next section together with the general architecture of the VisionDraughts.

3.1 MLP Neural Network

A Neural Network is a computational model based on biological neural networks which consists of a network of basic units called neurons. These artificial neurons, originally intended to simulate the operation of a single brain cell. The same algorithm runs on each neuron, witch communicates with a subset of other neurons of the same network [9], [10].

As said before, the agent correspond to MPL networks. A multilayer-perceptron is a feedforward artificial neural network model that maps sets of input data onto an appropriate output set. It has at least one hidden layer of neurons.

3.2 Reinforcement Learning and Temporal Difference Learning

The Reinforcement Learning paradigm has been of great interest for the learning machine due to the fact that it does not require a "coach" during the learning process. This fact is particularly appropriate in complex areas where the collection of examples for learning is difficult or impossible [10].

Among the Reinforcement Learning methods, one can spot the learning methods of the temporal differences, TD(λ). This technique [11], has emerged as a powerful reinforcement learning technique for incrementally tuning parameters. The basic idea is: a learning agent receives an input state that is continuously modified by means of the actions performed by the agent. Each current state is evaluated based on the previous one. At the end of the process, it outputs a signal and then receives a scalar *reward* from the environment indicating how good or bad the output is (reinforcement). That is, the learner is rewarded for performing well and, otherwise, it is punished.

4 VisionDraughts Player

VisionDraughts is the benchmark environment used to estimate the significant contribution of partially ordering the nodes of the tree search during the process of choosing the best move corresponding to the current board states. As said before, VisionDraughts taking advantage of search space properties with no explicit domain-specific knowledge, substitute a very efficient tree search module for the non-optimized search routine of NeuroDraughts [2]. The learning process and the efficient tree-search module of VisionDraughts are described in subsection below.

4.1 Learning Process of VisionDraughts

Figure 1 shows the learning process of VisionDraughts, that is: the tree-Search Routine selects, using an efficient search method (alpha-beta pruning and transposition table), the best move corresponding to the current board state. The TD Learning module is responsible for training the player neural network. The Artificial Neural Network module corresponds to a three layer feedforward network whose output layer is composed of a single neuron. The role of this network is to evaluate to what extent the board state,represented by NET-FEATUREMAP in the input layer, is favorable to the agent (it is quantified by a real number comprised between 0 and 1, called prediction, which is available in the output of the third layer neuron).

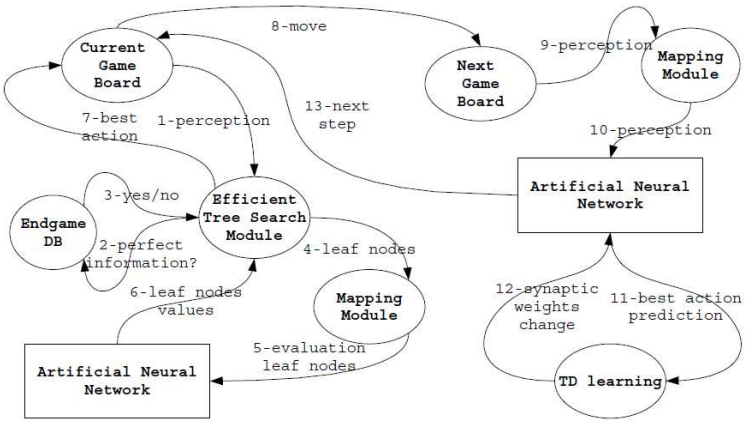

Fig. 1. Learning process VisionDraughts

4.2 Efficient Tree-search module of VisionDraughts

The search process of VisionDraughts is much more efficient then the one of NeuroDraugths: the alpha-beta algorithm combined with transposition table and iterative deepening reduced from more then 95% the search time required by NeuroDraughts. Next subsections resume the search process of VisionDraughts.

Adapting Alpha-Beta Pruning to the VisionDraughts: The first attempt to integrate the alpha-beta algorithm and transposition table in VisionDraughts failed.

Such attempt was based on classical alpha-beta version (hard-soft). This version, the prediction associated to board states is connected to the search range limit and not the real value of prediction associated with the state. When this value is stored in a transposition table, it is stored the inconsistent value. This fact is harmful when this state is re-visited and the recuperated value does not correspond to the real value of prediction.

To avoid this problem, a variant version of alpha-beta was adopted, the fail-soft alpha-beta. In the fail-soft alpha-beta approach, if n is a *maximizer* node, whenever **besteval** $\geq$ **beta**, alpha-beta returns **besteval**. Similarly, if n is a *minimizer* node, whenever **besteval** $\leq$ **alpha**, alpha-beta returns **besteval**. This version returning a bound on true minimax value is called fail-soft alpha-beta because a fail high or fail low still returns useful information [10]. Then, the returned value always represents one limit on the minimax value.

In spite of this, the fail-soft version is essential in the proposed learning system in order to integrate with the transposition table. See the fail-soft version in figure 2.

Iterative Deepening: The iterative Deepening combines the benefits of breadth-first and Best-first search. It is complete when the branching factor is finite and optimal if the cost of path is is proportional to the depth [13], [8].

VisionDraughts combines alpha-beta with iterative deepening to delimit the depth of search in each iteration. Considering that the alpha-beta algorithm does not keep a history of solutions previously obtained and that the iterative deepening can be extremely repetitive during the search process [10], the use of transposition table associated with them improves a lot the search method [7]. It allows to count on a deeper look-ahead during the process of choosing the best move.

The utilization of Transposition Table, Alpha-Beta Pruning and Iterative Deepening in the Search Module: The dynamics of Draughts allows that a certain board state occurs several times during a game (transposition). Whenever a transposition occurs for a board state S, it is not worthwhile to recalculate the prediction for S each time it appears in the game. That is why VisionDraughts uses a transposition table to store predictions that have already been calculated (see routine module in figure 1). In order to check a board state against stored information in memory, a hash value is assigned to each board state [9].

There is a common technique for creating hash codes corresponding to board games that uses a set of fixed-length random bit (number) patterns stored for each possible state of each possible board square [17], [9]. The technique is called **zobrist hashing** and the bit (number) pattern is called **zobrist key**.

Draughts has 32 squares and each square can be empty or have 1 of 2 different pieces on it, each of 2 possible colors. Then, the **zobrist key** for draughts needs to have 32 x 2 x 2 = 128 entries. In this case, the **Tree-Search Routine** module, showed in the figure 1, makes use of the **zobrist key**.

The hash code assigned to a board state is the first field stored in a transposition table entry and is represented as following:

Example of a transposition table entry

```
struct TranspTable{
  int64   hash_value;
  int     score_type;
```

```
    int     score_value;
    int     depth;
    MOVE    best_move;
}
```

Moreover, when a tree is expanded by the alpha-beta routine, the algorithm output value, the search depth and the best move to be executed are stored in the following fields, respectively: ***score_value***, ***depth*** and ***best_move***. The **depth** field is responsible for assuring that the prediction for a board state is sufficiently accurate.

Furthermore, the alpha-beta routine rarely outputs the exact minimax value of a node (the board state represented by ***hash_value*** field), but the fail-soft variant always outputs the lower bound or the upper bound on the true minimax value (prediction). Then, it is important to store a flag that indicates what the *score-value* field means. For example, if the ***score_value*** field contains the value 0.3 and the ***score_type*** field contains the *Exact* flag, this means that the value of the node is exactly 0.3. If the score type field contains the fail low flag, this means that the value of the node is at most 0.3. Similarly, if the score type field contains the fail high flag, this means that the value of the node is at least 0.3. So, the score type, or ag, is also recorded in the score type field.

Iterative deepening is used in conjunction with TranspTable. The alpha-beta routine is called repeatedly with increasing depth until either the established time is over or the search reaches the maximum look-ahead previewed. In spite of the fact that this method apparently waste time performing shallow searches instead of just one deep search, actually it allows to improve the search efficiency, once the former iterations are used to obtain a higher quality of move ordering what allows more cutoffs [16].

5 Partial Ordering in all Tree-Search nodes

The transposition table is used in association with iterative deepening to obtain partially ordered tree-searches. When the iterative deepening searches a deeper level and revisits a state, the information of the transposition table referent to an already evaluated state, can be used to order the tree-search. The data of the table can be used provided that the depth of the current state is compatible with the depth of the state stored in the table (equal or greater than depth).

The best move (*bestmove*) is the child of the state (found in the transposition table) which got the best prediction during previously made evaluations. The partial ordering of nodes consists in making the best move node occupy the first position from left to right in the tree-search. Therefore, the best move from a state at depth *d*, will be the first to be explored in depth *(d +1)* and in accordance with Plaat [16], the best move for depth d is possibly the best movement to the depth *(d + 1)*.

Following is an explanation of the main lines of the algorithm of figure 2:

– **Line 1**: The alpha-beta algorithm receives a Draughts board *n*, a search depth *d* and an output parameter, *bestmove*, to store the best action to be

executed. From the board, a search range is delimited by the parameters *min* (lower limit of the search range) and *max* (upper limit of the search range). The result returned by the algorithm is the prediction associated to state *n*, which corresponds to the evaluation of this state from the point of view of the agent player;

- **Line 2**: The alpha-beta algorithm is a recursive procedure and makes a left to right depth-first search in the game tree. As a recursive procedure, this requires a stop condition - the condition that interrupts the algorithm and checks if the board *n* is a leaf;
- **Line 3**:The output of the function that indicates the prediction given by the neural network for the board state *n* is stored in *besteval*;
- **Line 4**: When the the search algorithm calls the neural network on line 3 (*evaluate(n)*), to estimate the prediction associated to the leaf state *n*, this recent calculated prediction is stored in a transposition table through the *store* procedure. Since *n* is a leaf, the value of the prediction has *scoretype* = *hashExact*, so the prediction is equal to *besteval*;
- **Line 6**:This line checks if the state of the board present at the root of the game tree constructed by algorithm alpha-beta is a *max* node;
- **Line 9**: This line retrieves the *bestmove* stored for the depth *d-2* from the transposition table and stores it in the variable found, if the board *n* is not a leaf or root;
- **Line 10**: This line checks if the value of the variable is different from zero, that is, there was a *bestmove* associated with the board n for depth *d-2* - and if this value is valid (*scoreType* different from *hINVALID*);
- **Line 11**: The procedure *setChildrenOrden(n, bestmove)* is called . After its execution, the left branch of the search-tree will contain the best move obtained for depth *d-2*;
- **Line 12**: For each child of a *max* node *n*, the transposition table should be checked before calling the search routine recursively. Then the algorithm calls the procedure retrieve with *NodeType* = *parentIsMaxNode*. Another very important detail in this line is the fact that the depth of search must be equal to the *depth -1*, which corresponds to the depth of *n*;
- **Line 13**:If the child of state *n* is stored in the transposition table and if the method *retrieve* obtains success in the treatment of information, the value of the prediction returned by retrieve, must be used instead of calling the alpha-beta algorithm recursively;
- **Line 19**: After detecting the occurrence of a beta pruning, the variable *besteval* will contain the minimum acceptable value for the board represented by a node *n*. Thus, the prediction in variable *besteval* should be stored in the transposition table by the store method with *hashAtLeast* flag indicating that the prediction associated with the state board *n* is at least equal *besteval*;
- **Line 22-37**: These lines execute the same procedures of lines 6 to 21, but considering the root node a *min* node;
- **Line 38**: After the algorithm examines all children of the node, the variable *besteval* will contain the exact value of the prediction for the board represented by node *n*. Thus, the prediction on *besteval* must be stored in

the transposition table with a *hashExact* flag indicating that the prediction associated with n is equal to *besteval* ;

```
1. fun alfaBeta(n:node,depth:int,min:int,max:int,bestmove:move):float =
2.   if leaf(n) or depth=0 then
3.        besteval := evaluate(n)
4.        store(n, besteval,bestmove,depth,hashExact)
5.        return besteval
6.    if n is a max node
7.        besteval := min
8.        if (!leaf(n) and !depth=0) then
9.            found= retrieve(child,besteval,bestmove,depth-2,parentIsMaxNode)
10.           if (found != 0 AND scoreType!= hINVALID)
11.               setChildrenOrder(n,bestmove)
12.           for each child of n
13.               if retrieve(child,besteval,bestmove,depth-1,parentIsMaxNode)
14.                   then v := besteval
15.               else v := alfabeta(child,depth-1,besteval,max,bestmove)
16.               if v > besteval
17.                   besteval:= v
18.                   thebest = bestmove
19.               if besteval >= max then
20.                   store(child,besteval,bestmove,depth,hashAtLeast)
21.                   return besteval
22.   if n is a min node
23.       besteval := max
24.        if (!leaf(n) and !depth=0) then
25.           found= retrieve(child,besteval,bestmove,depth-2,parentIsMinNode)
26.           if (found != 0 AND scoreType!= hINVALID)
27.               setChildrenOrder(n,bestmove)
28.           for each child of n
29.               if retrieve(child,besteval,bestmove,depth-1,parentIsMinNode)
30.                   then v := besteval
31.               else v := alfabeta(child,depth-1,besteval,max,bestmove)
32.               if v < besteval
33.                   besteval:= v
34.                   thebest = bestmove
35.               if besteval <= max then
36.                   store(child,besteval,bestmove,depth,hashAtMost)
37.                   return besteval
38.   bestmove = thebest
39.   store(n,besteval,bestmove,depth,hashExact)
40.   return besteval
```

Fig. 2. Pseudo-code of algorithm fail-soft alpha-beta with transposition table, databases of the end of game and ordering

Therefore, the search process tends to be faster since in each level of the search-tree the best move occupies the least position. As a consequence, the chances of pruning increases. This increase in the number of pruning means that the best moves are really found in the left side of the search-tree, confirming what was said in Plaat [16].

6 Experimental Results

This section presents the improvement obtained in VisionDraughts by the insertion of ordering in the tree-search. To illustrate the improvement, the ordering VisionDraughts was compared to VisionDraughts without ordering and to LS-Draughts[6]. The parameter taken into account for the comparison is the number of wins, draws and losses obtained by each one in a tournament of 14 games. All players were trained in a similar way, that is, by self-play with cloning, during 10 tournaments of 200 games each one. Figure 1 shows the results of the competition between them.

Result in 14 games	Improve VisionDraughts x VisionDraughts				Improve VisionDraughts x LS-Draughts				VisionDraughts x LS-Draughts			
	Winning	**Loss**	**Draw**	**Loops**	**Winning**	**Loss**	**Draw**	**Loops**	**Winning**	**Loss**	**Draw**	**Loops**
Ordinary	8	1	5	3	8	0	6	2	3	1	10	7
Percentage	57%	7%	36%	21%	57%	0%	43%	15%	21%	7%	71%	50%

Table 1. Results of tournament: Improve VisionDraughts x VisionDraughts, Improve VisionDraughts x LS-Draughts and VisionDraughts x LS-Draughts

In order to check the loops that occur in VisionDraughts, the improve VisionDraughts decreased the occurrence of this. Loop's column in table 1 shows that cases of loops between the proposed system and the VisionDraughts corresponds to 21% at all the games. In the tournament between the proposed system and the LS-Draughts, loops correspond to 15%. Already in tournament between the VisionDraughts and LS-Draughts loops correspond to 50% at all the games.

All these results confirm the significant contribution of inserting ordering in the tree-Search of automatic players.

7 Conclusions and Future Works

This paper presented how much an efficient search module based on alpha-beta pruning, transposition table, iterative deepening and ordered tree-search can improve the learning performance of intelligent automatic players.

The strategy of ordering the tree-search aimed to speed up the search process for the best move, witch allows a deeper look ahead that optimizes the choice of the move.

A competition was executed, where the proposed system played several games against the original VisionDraughts and against LS-Draughts [6]. The results of this competition (figure 1) confirm the high improvement obtained in the learning process as well as in the performance of the player agent. As future works, the authors intend to introduce the same efficient search module with ordering tree-search in the LS-Draughts.

References

1. Samuel, A. L.: Some Studies in Machine Learning Using the Game of Checkers. IBM Journal on Research and Development,(1959).
2. Lynch, M.: Neurodraughts - An application of temporal difference learning to draughts. Master's thesis, Department of Computer Science and Information Systems, University of Limerick, Ireland, (1997)
3. Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P.,Szafron, D.: A world championship caliber checkers program. In: Artificial Intelligence, vol. 53, (1992).
4. Schaeffer, J., Lake, R., Bryant, M., Lu, P.: Chinook: The world manmachine checkers champion. In: AI Magazine, (1996).
5. Schaeffer, J., Burch, N., Bjrnsson, Y., Kishimoto, A., Muller, M., Lake, R., Lu, P., Sutphen S.: Checkers is solved. In: Science Express, vol. 317, (2007)
6. Neto, H. C., Julia, R. M. S.: LS-Draughts - a draughts learning system based on genetic algorithms, neural network and temporal diferences. IN: IEEE Congress on Evolutionary Computation, (2007)
7. Caixeta, G. S.,Julia, R. M. S.: A draughts learning system based on neural networks and temporal differences: The impact of an efficient tree-search algorithm. In: The 19th Brazilian Symposium on Artificial Intelligence, SBIA, (2008)
8. Caixeta, G. S.: Visiondraughts -um sistema de aprendizagem de jogos de damas baseado em redes neurais, diferencas temporais, algoritmos eficientes de busca em arvores e informacoes perfeitas contidas em bases de dados. Master's thesis, Federal University of Uberlandia, (2008)
9. Millington, I.: Artificial Intelligence for Game. Morgan Kaufmann,(2006)
10. Russel, S., Norving, P.: Inteligência Artificial - Uma abordagem Moderna, 2nd ed. Editora Campus, (2004).
11. Tesauro, G. J.: Temporal difference learning and td-gammon. In: Communications of the ACM, vol. 38, no. 3, (1995).
12. Schaeffer, J., Plaat, A.: New Advances in Alpha-Beta Searching. In: Proceedings of the 1996 ACM 24th Annual Conference on Computer Science. Philadel- phia, Pennsylvania, United States (1996)
13. Schaeffer, J., Paul, L., Lake, D. S. R.: A Re-Examination of Brute-Force Search. In: Games - Planning and Learning, (1993)
14. Schaeffer, J. et al. Chinook: World Man-Machine Checkers Champion. 2008. Perfect Play: Draw!, `http://www.cs.ualberta.ca/~chinook/index.php`
15. Schaeffer, J., Hlynka, M., Jussila, V.: Temporal Difference Learning Applied to a Highperformance Game-Playing Program. In: International Joint Conference on Artificial Intelligence, pp. 529–534, (2001).
16. Plaat, A.: Research Re: Search & Re-Search. PhD thesis, Erasmus University, Amsterdam (1996)
17. Zobrist, A. L.: A Hashing Method with Applications for Game Playing. Tech. Rep. 88, Computer Sciences Department, University of Wisconsin, Wisconsin (1969)