NOTES

ALPHA-BETA SEARCH ENHANCEMENTS WITH A REAL-VALUE GAME-STATE EVALUATION FUNCTION

Jacek Mańdziuk[1] *and Daniel* $Osman$[1]

Warsaw, Poland

ABSTRACT

The note presents results of applying several Alpha-Beta search enhancements in the game of give-away checkers with a real-value state evaluation function. In particular, the MTD-bi (bisection) algorithm is tested and compared (1) with MTD(f) and (2) with Alpha-Beta search enhanced with transposition tables and the history heuristic. The results show that in the real-value domain the MTD(f) algorithm becomes impractical and loses its superiority over MTD-bi. Several remarks concerning possible implementations of different replacement schemes in transposition tables are presented and discussed.

1. INTRODUCTION

Integer evaluation functions are used in many game-playing programs. There are two reasons: (1) the speed (floating point operations are slower than integer ones) and (2) there is often no need for a real-value state evaluation. The elements of a game-state feature vector are usually integers or can be made integers by multiplying all values by a factor of 10. Moreover, an integer representation seems natural. However, this can change due to the growing popularity of computer learning algorithms. For instance, in Temporal Difference learning and in other reinforcement learning algorithms (Kaelbling, Littman, and Moore, 1996) the weights of the evaluation function are real values. This in itself forces us to abandon the integer representation. However, in minimax algorithms, the format of the state value is unimportant. Only the relative differences between values matter. Therefore one could convert the real-value state evaluation function into an integer one, e.g., in the following way: $V_{int} = round(\frac{1}{\varepsilon} \cdot V_{float})$, where ε is the acceptable error value, e.g., $\varepsilon = 0.01$. Yet, this type of conversion implies another problem: the information about small (less than ε) differences between state values is lost. It can be acceptable once an optimal set of weights is developed. However, during learning we usually cannot afford to ignore even the smallest differences between state values. For instance, if the evaluation function is a sigmoidal neural net (Tesauro, 1992), a difference of 0.001 does not necessarily imply that the compared states are similar. First, it is hard to estimate which level of granularity is satisfactory. Furthermore, ignoring even a small difference may result in losing some important line of strategy that has just been learned. In that case the chance of reinforcing the weights that were responsible for choosing this strategy is lost. Second, it is not the real-value representation by itself that introduces anything new in game-tree search algorithms. It is rather the increase of distinct values of the evaluation function or the increase of its granularity. The conversion presented above reduces the granularity only to some extent, depending on ε. In Plaat (1996a) the author suggested that a fine-grained evaluation function may be a problem for the MTD(f) algorithm (Plaat *et al.*, 1996b) which is considered the state-of-the-art game-tree search method in the integer domain. Up to our knowledge, no one has yet conducted experiments involving this problem.

Fixed-depth game-tree search algorithms have been widely analysed by several authors. Most practical tests were done using an integer evaluation function with only some exceptions - for instance, the CRAY BLITZ chess program (Hyatt, Gower, and Nelson, 1990) used real values. In this note we analyse a few common Alpha-Beta search enhancements in the real-value domain. The results presented in Section 5 show that a domain change does not effect the standard Alpha-Beta enhancements such as the history heuristic (Schaeffer, 1989) and the transposition tables (Lazar, 1995). The observable effect was that the MTD(f) algorithm lost

[1]Faculty of Mathematics and Information Science, Warsaw University of Technology, Plac Politechniki 1, 00-661, Warsaw, POLAND, email: mandziuk@mini.pw.edu.pl, dosman@prioris.mini.pw.edu.pl

its superiority. Throughout the note we try to explain why this happened and propose replacing MTD(f) by MTD-bi when performing game-tree search in the real-value domain. This note describes the first stage of a larger experiment tailored to the application of various temporal-difference approaches to playing give-away checkers (Mańdziuk and Osman, 2004).

The note is organized as follows. In Section 2 we briefly describe the game of give-away checkers and introduce the domain of the evaluation function. In Section 3 the MTD(f), realMTD(f), MTD-bi, and MTD-step(f) algorithms are discussed. Section 4 describes the experimental design and Section 5 presents the results. Conclusions and directions for future research are given in Section 6.

2. GIVE-AWAY CHECKERS

The rules of give-away checkers (Alemanni, 1993) are exactly the same as those of checkers (ACF, 1990). The only difference is the goal of the game. In give-away checkers the aim of the player is to lose all his[2] pieces. Formally a player is considered a winner when no legal move can be made in his turn.

It should be noted that there exist several variants of the game of checkers (and consequently also of give-away checkers). In this work we have adopted one of the most popular variants, with the so-called US rules (8 x 8 game board, capturing a piece by another piece is allowed only in a forward move, capturing a piece by a king is allowed in either direction, kings in non-capturing moves are allowed to move in any direction but only by one square). The game at first glance may seem trivial or at least not interesting, but actually its complexity goes far beyond this first impression. For instance, the naive approach based on losing stones as fast as possible is completely unsuccessful. A much more "sophisticated" approach is required to achieve a good level of play.

Due to sharing the same rules of playing, the games of checkers and give-away checkers share the same key characteristics important in analysing game-tree search algorithms (Schaeffer *et al.*, 1996). For instance, the branching factor for both games is located between 2 and 3.

In our experiment a real-value state evaluation function denoted by $V(s)$ is used. In brief, it is a straightforward weighted sum of game-state features limited by MIN and MAX, i.e., $V(s) \in (MIN; MAX)$, where $MIN = -100$ and $MAX = +100$.

3. ALGORITHMS TESTED

A transposition table with 2^{20} records was used in our experiments. However, the state space for checkers (and give-away checkers) is about 10^{20}, that is approximately 14 orders of magnitude greater than what can fit in a transposition table. Sooner or later transposition-table conflicts will occur while saving a new state in place of an old one. This problem must be handled by an appropriate replacement scheme (Breuker, Van den Herik, and Uiterwijk, (1994); Lazar, 1995). Three replacement schemes were tested in this work. "If newer" - the new record is always saved in place of an older one. "If deeper" - the results of a new search replace the old ones only if the new search was performed to a greater or equal depth. "If deeper + timestamp" - all records while being saved are timestamped with the current move number (incremented after each saving). The new record is saved only if $new.depth + new.timestamp \geq old.depth + old.timestamp$. Quite surprisingly, the simplest of the above schemes (i.e., "if newer") was the most effective in both the execution time and the evaluated nodes count. The results presented in Section 5 show that the overall performance varied up to 17 per cent depending on which scheme was selected.

The MTD(f) algorithm

In Plaat *et al.*, (1996b,c) the MTD class of algorithms was introduced, among which the MTD(f) algorithm is currently the state-of-the-art in fixed-depth game-tree search, at least for integer evaluation functions. The code of MTD(f) for integer values is presented in Figure 1(a). Function MT($s, b-1, b$) in line 4 is equivalent to a fail-safe, null-window Alpha-Beta($s, b-1, b$) algorithm enhanced with a transposition table and the history heuristic (Schaeffer, 1989). The search window is limited to $\alpha = b-1$ and $\beta = b$ and s is the state to be expanded. MT returns the lower (f^-) or the upper (f^+) bound of the final outcome in every repeat-until iteration. The idea of MTD(f) is to call MT($s, b-1, b$) or indeed any other fail-safe, null-window variant of Alpha-Beta repeatedly until f^- equals or exceeds f^+. The value f is the first rough prediction of the final outcome. Tests in Plaat *et al.* (1996b) showed that more accurate predictions tend to cause MTD(f)

[2]In this note we use 'his' and 'he' when 'her/his' and 'she/he' are possible

```
function MTD($f$)
1: $f^+ := MAX$; $f^- := MIN$;
2: if ( $f = MIN$ ) then $b := f + 1$ else $b := f$;
3: repeat
4:     $g := MT(s, b - 1, b)$
5:     if ( $g < b$ ) then $f^+ := g$ else $f^- := g$;
6:     if ( $g = f^-$ ) then $b := g + 1$ else $b := g$;
7: until $f^- \geq f^+$
8: return $g$;
```

(a) MTD(f) algorithm working on integer values.

```
function MTD-bi
1: $f^+ := MAX$; $f^- := MIN$;
2: repeat
3:     $b := (f^+ + f^-)/2 + \frac{\varepsilon}{2}$
4:     $g := MT(s, b - \varepsilon, b)$
5:     if ( $b - \varepsilon < g < b$ ) then return $g$;
6:     if ( $g < b$ ) then $f^+ := g$ else $f^- := g$;
7: until forever
```

(b) MTD-bi algorithm.

Figure 1: MTD(f) and MTD-bi algorithms.

to work faster because fewer repeat-until iterations need to be performed. Null-window Alpha-Beta search terminates faster than wide-window Alpha-Beta(s, MIN, MAX) because much more cutoffs occur during the search. Additionally, repeated executions of MT use results from previous calls saved in the transposition table. Empirical results show that MTD(f) should be faster than wide-window Alpha-Beta(s, MIN, MAX) search as long as the number of repeat-until iterations is kept low (below 20).

The realMTD(f) algorithm

A naive modification of MTD(f) making it useful in the real-value domain involves replacing every appearance of "$+1$" in Figure 1(a) by "$+\varepsilon$" which is the greatest acceptable error value. Additionally, in a continuous domain, instead of a null window an ε-window has to be used. Thus after line 4, a check whether g falls between ($b - \varepsilon$; b) needs to be performed. If it does, then g is not a bound but an exact value and the algorithm terminates. Finally, the termination condition (line 7) also needs to be changed into: **until** $f^+ - f^- < \varepsilon$. The algorithm modified in the way described above is denoted by realMTD(f).

Unfortunately the above straightforward modification of the discrete MTD(f) encounters serious practical problems. Let us assume that the repeatable execution of MT($s, b - \varepsilon, b$) in line 4 returns g, $g - 0.01, g - 0.02, \ldots, g - 0.99$ in the consecutive repeat-until iterations. Thus in each iteration a very small step towards the exact outcome is made. If for instance $f = 0$ and the actual backed-up score of state s is -50 then reaching this value could take a very long time. In fact, such situations occurred many times in our tests. For instance, returning the value -56.81 from $f = 0$ and $\varepsilon = 0.01$ required 752 repeat-until iterations and lasted 149.62 seconds. Computing the same move with a wide window MT(s, MIN, MAX) required only 15.01 seconds.

In integer domains the number of repeat-until iterations in MTD(f) is limited by $L = MAX - MIN$. This is because in the worst case in every iteration either f^+ is decreased by 1 or f^- is increased by 1. Hence, the difference ($f^+ - f^-$) has to decrease by at least 1 in each iteration. Using realMTD(f) in real-value domains can result in the number of iterations reaching $L = \frac{MAX - MIN}{\varepsilon}$. This can be a large number especially if ε is small. One can use greater values of ε in order to lower this theoretical bound. However, there are two reasons against doing this. First, greater ε means greater error. This problem can be solved by inserting the line $g := MT(s, f^-, f^+)$ just before returning g in the final line. Consequently one more execution of MT is performed, but an exact value is returned. Second, greater ε means wider search windows in MT($s, b - \varepsilon, b$). This means that the advantage of a narrow window search is partly lost. Using realMTD(f) would require to start always with a very accurate initial prediction f, which makes the algorithm impractical.

The MTD-bi algorithm

Instead of taking small ε-steps in every repeat-until iteration, an algorithm that takes greater steps towards the final outcome is needed in practice. That is why an algorithm based on MTD-bi (bisection) (Plaat *et al.*, 1996b) was implemented and used instead of realMTD(f). The code of MTD-bi is presented in Figure 1(b). Like before $\varepsilon = 0.01$ and s is the state being expanded. In every repeat-until iteration the algorithm performs an ε-window search in the middle of the interval defined by f^- and f^+. It is important to note that in this case ε does not stand for an acceptable error value. It only represents the size of a search window. MTD-bi for any $\varepsilon > 0$, always returns exactly the same value as would be returned by plain Alpha-Beta(s, MIN, MAX). This is guaranteed by line 5 and the lack of a termination condition in line 7. The algorithm terminates only when g is found to be in the current search window. In MTD-bi the number of repeat-until iterations is limited by $L = \log_2(\frac{MAX - MIN}{\varepsilon})$ which in the case of $MIN = -100$, $MAX = +100$ and $\varepsilon = 0.01$ gives $\lceil L \rceil = 15$. Actually, in practical tests described in Section 5 this theoretical bound was never reached.

The MTD-step(f) algorithm

Another variant of the MTD family that seems to be suitable in real-value domains is MTD-step(f). In this algorithm after the initial guess of f, a step is made towards the final outcome value. Step sizes can vary between iterations. With a good initial guess and smart step sizes MTD-step(f) outperforms MTD-bi. The latter algorithm however is less problem dependent and is recommended when the initial guess and the step sizes are hard to predict. One can also think of MTD-step(f) as realMTD(f) with greater steps.

4. EXPERIMENTAL DESIGN

In the experiment 25 random games were played, each terminated after the 40^{th} move (counting both sides, i.e., the 40^{th} ply). This gave a total of 1000 test states, although not pairwise different (e.g., the initial state in all games was the same). For each algorithm and for every game state, the best move was computed just as if it were a regular play. The number of leaf nodes evaluated and the time spent on computing that move was registered. Instead of executing the best move, a pre-defined move for that game was made. Every game had a random pre-defined move sequence associated with it. This ensured that all algorithms worked on exactly the same states. The values presented in the next section are the average ones counted for all 1000 states. The size of the transposition table was fixed throughout the experiment and equal to 2^{20}. Larger transposition tables did not seem to offer any speed-up. By default, in all algorithms the “if newer” replacement scheme was used. TTHH is defined as a wide-window Alpha-Beta(s, MIN, MAX) search algorithm enhanced with a transposition table and history heuristic. MTD-bi corresponds to the algorithm presented in Fig. 1(b).

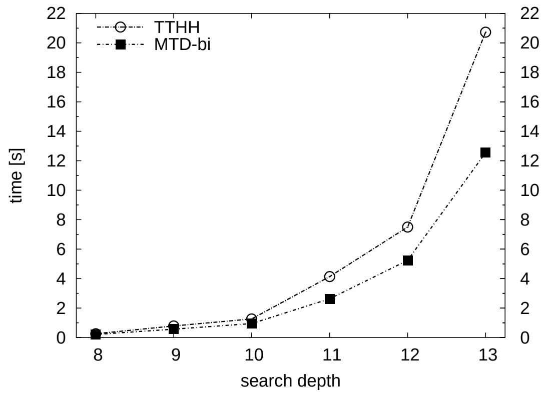
5. RESULTS

Initial experiments involved testing two standard Alpha-Beta search enhancements which are history heuristic HH (Schaeffer, 1989) and transposition table TT (Lazar, 1995). The results were quite similar to the ones presented in Schaeffer (1989). The relative usefulness of these enhancements in the real-value domain was the same as in the integer domain, with the combination of both, denoted by TTHH, being the most effective. The results confirmed our intuition that a domain change should not effect the standard Alpha-Beta enhancements (i.e., HH and TT). Moreover, there should be little or no difference between using integers or real numbers as state values. This reasoning can be extended to other Alpha-Beta enhancements, i.e., killer moves, iterative deepening (Schaeffer, 1989), ETC or EIB (Plaat *et al.*, 1996c), which should also prove to be useful in the real-value domain since they do not utilise any properties of the discrete value space on the contrary to MTD(f). The algorithm assumes that there are no values between ($b-1$; b) and that the minimal change in state values equals 1. These assumptions are clearly incorrect in the real-value domain.

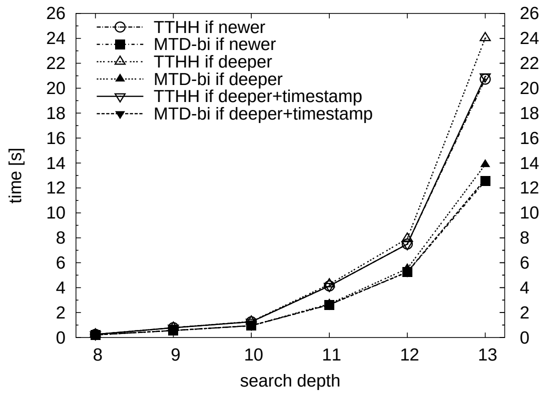
A comparison between TTHH and MTD-bi is presented in Figure 2(a). The MTD-bi algorithm outperformed TTHH by 39 per cent in execution time and 42 per cent in leaf nodes count (not presented in the figure) for depth 13. Moreover, the average interior nodes count (not presented) is about 37 per cent less for MTD-bi than for TTHH. Please note that since MTD-bi often revisits nodes, the actual reduction in *distinct* interior nodes count is even greater. That is the reason why MTD algorithms seem to work well, even with relatively small transposition tables. Visiting fewer nodes means that fewer states have to be saved in the transposition table. The size chosen in our experiments (i.e., 2^{20}) is more than sufficient for a game of checkers or give-away checkers (Plaat *et al.*, 1996b). The results for realMTD(f) are not shown because they were far worse than the ones for TTHH. This confirms our discussion of Section 3 that the MTD(f) algorithm is impractical in the real-value domain. However, the superiority of MTD-bi over TTHH confirms that ε-window search can still be successfully used in the real-value domain. Hence, the main idea behind MTD algorithms still proves to be useful.

The increase of performance between MTD-bi and TTHH reported in this note (about 40 per cent in favour of MTD-bi) should not be directly compared with the increase of performance that the MTD(f) algorithm introduced in the integer domain (i.e., about 20 per cent) as reported in Plaat *et al.*, (1996b) since in the cited paper the tests were performed on tournament class programs where achieving any performance increase is very hard. However, the thing worth noting is that in the real-value domain an increase of performance is possible even though ε-window instead of null-window search is applied.

Figure 2(b) shows the results of applying different replacement schemes in transposition tables. The “if deeper” scheme was in average 13 per cent worse than the “if newer” one for TTHH and depth 13 considering execution time and 17 per cent worse considering leaf nodes count (not shown). For MTD-bi the inferior replacement



(a) Wide-window TTHH($MIN; MAX$) algorithm compared with the MTD-bi algorithm.



(b) Results of applying three transposition -table replacement schemes in TTHH and MTD-bi.

Figure 2: Average time per move for TTHH($MIN; MAX$) and MTD-bi.

scheme ("if deeper") had less significant impact on the results (9 per cent worse compared to "if newer" in execution time and 12 per cent worse for leaf nodes count). This is surprising considering that the MTD algorithms depend highly on transposition tables. The "if deeper+timestamp" replacement scheme was in practice equally efficient to the "if newer" one. The latter one however has a clear advantage of being easier to implement and needing less storage space (no timestamp information has to be saved). One more replacement scheme called "if sharply deeper" (not shown in the figures) was also tested. In this scheme, the new state replaced the old one only if it was searched to a greater (not greater or equal) depth. With this scheme the average execution time per move for TTHH and depth 13 raised to approximately 28 seconds. This poor result was most probably caused by the inability to replace old records with the new ones. As the transposition table got filled up with states searched to great depths there was a decreasing possibility of replacing them by new records. Using the "if sharply deeper" replacement scheme with the MTD-bi algorithm caused its average results to degrade to the level of TTHH. The MTD-bi algorithm often needs to correct its initial guess (upper or lower bound) of the backed-up score of some state in the transposition table. If the state was searched to the same depth as before, it did not pass the "if sharply deeper" rule. Failing to do this caused the algorithm's performance to decrease.

This result together with the superiority of the "if newer" scheme shows the importance of storing the latest information in the transposition table even for the cost of losing search results performed to greater depths that might have been computationally more expensive.

The number of repeat-until iterations for MTD-bi ranged from 3 to 14. For depth 8 the average was 7.5 iterations and gradually increased with depth to 10.7 for depth 13. These are reasonable numbers. Similar results were reported by other authors (Plaat *et al.*, 1996b) using MTD(f) with integer evaluation functions.

One could suspect that the superiority of MTD-bi over TTHH is caused by the greater number of transpositions (successful uses of transposition table) in case of MTD-bi. This however is not the case. In fact the number of transpositions in MTD-bi was 40 per cent less than in TTHH. It seems to correspond well with the 42 per cent reduction in leaf nodes count and 37 per cent reduction in interior nodes count. The transpositions per interior node ratio was approximately the same for MTD-bi (0.180) and TTHH (0.187). The result confirms that MTD-bi makes smarter use of a transposition table and visits a smaller number of nodes which are irrelevant for computing the final score.

6. CONCLUSIONS

MTD(f) is currently a state-of-the-art game-tree search method in the integer domain. Unfortunately, as can be observed in our experiment a straightforward modification of this method denoted by realMTD(f), suitable for the real domain, becomes inefficient and MTD-bi gives far better results.

MTD-bi was also superior to Alpha-Beta combined with transposition tables and the history heuristic (denoted by TTHH). In this group of tests three replacement schemes in transposition tables were tested, denoted by "if newer", "if deeper" and "if deeper + timestamp". Surprisingly, the scheme "if newer" in which always the new record is saved in the transposition table regardless of the search depth of a previously stored record was the most effective. Another interesting observation concerns the frequency of using transposition tables in MTD-bi and TTHH methods. The ratio of transpositions per interior node was approximately the same for both methods, which implies that the transpositions in MTD-bi are more effective.

The main conclusion drawn from this work is that in the *real domain* narrow window (ε-window) search methods, such as the MTD-bi algorithm, are very efficient and outperform standard wide-window search methods. Although in the real domain the realMTD(f) algorithm becomes inefficient, other MTD algorithms such as MTD-bi or MTD-step(f) can be successfully used.

One of our current research goals is a closer examination of the three tested replacement schemes in transposition tables in order to explain in more detail the phenomenon of the superiority of the "if newer" scheme over the two other ones. Moreover we continue to work on TD approaches to playing give-away checkers (Mańdziuk and Osman, 2004).

7. REFERENCES

ACF (1990), American Checkers Federation. http://www.acfcheckers.com/.

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

Breuker, D. M., Uiterwijk, J., and Herik, H. J. (1994). Replacement Schemes for Transposition Tables. *ICCA Journal*, Vol. 17, No. 4, pp. 183–193.

Hyatt, R., Gower, A., and Nelson, H. (1990). Cray Blitz. *Computers, Chess, and Cognition* (eds. T. Marsland and J. Schaeffer), pp. 111–130, Springer-Verlag, New York, N.Y. ISBN 0–387–97415–6.

Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237–285. ISSN 1076–9757.

Lazar, S. (1995). *Analysis of Transposition Tables and Replacement Schemes*. Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County.

Mańdziuk, J. and Osman, D. (2004). *Temporal Difference approach to playing Give-Away Checkers*. Working paper.

Plaat, A. (1996a), MTD(f). A Minimax Algorithm faster than NegaScout. http://www.cs.vu.nl/ aske/mtdf.html.

Plaat, A., Schaeffer, J., Pijls, W., and de Bruin, A. (1996b). Best-First Fixed-Depth Minimax Algorithms. *Artificial Intelligence*, Vol. 87, Nos. 1–2, pp. 255–293. ISSN 0004–3702.

Plaat, A., Schaeffer, J., Pijls, W., and de Bruin, A. (1996c). Exploiting Graph Properties of Game Trees. *13th National Conference on Artificial Intelligence (AAAI-96)*, Vol. 1, pp. 234–239, Menlo Park, CA. ISBN 0–262–51091–X.

Schaeffer, J. (1989). The History Heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 11, pp. 1203–1212. ISSN 0162–8828.

Schaeffer, J., Lake, R., Lu, P., and Bryant, M. (1996). Chinook: The World Man-Machine Checkers Champion. *AI Magazine*, Vol. 17, No. 1, pp. 21–29. ISSN 0738–4602.

Tesauro, G. (1992). Practical Issues in Temporal Difference Learning. *Australian Journal of Intelligent Information Processing Systems*, Vol. 8, pp. 279–292.