IOP Conference Series: Materials Science and Engineering

PAPER • **OPEN ACCESS**

The Analysis of Alpha Beta Pruning and MTD(f) Algorithm to Determine the Best Algorithm to be Implemented at Connect Four Prototype

To cite this article: Lukas Tommy *et al* 2017 *IOP Conf. Ser.: Mater. Sci. Eng.* **190** 012044

IOP Conf. Series: Materials Science and Engineering **190** (2017) 012044 doi:10.1088/1757-899X/190/1/012044

The Analysis of Alpha Beta Pruning and MTD(f) Algorithm to Determine the Best Algorithm to be Implemented at Connect Four Prototype

Lukas Tommy[1)] Mardi Hardjianto[2)] Nazori Agani[3)]

[1)2)3)]Graduate Program of Master of Science in Computer Science, Budi Luhur University
Jl. Ciledug Raya, Petukangan Utara, Kebayoran Lama, South Jakarta 12260

lukastommy92@gmail.com[1)] mardi.hardjianto@budiluhur.ac.id[2)]
nazori.agani@gmail.com[3)]

Abstract. Connect Four is a two-player game which the players take turns dropping discs into a grid to connect 4 of one's own discs next to each other vertically, horizontally, or diagonally. At Connect Four, Computer requires artificial intelligence (AI) in order to play properly like human. There are many AI algorithms that can be implemented to Connect Four, but the suitable algorithms are unknown. The suitable algorithm means optimal in choosing move and its execution time is not slow at search depth which is deep enough. In this research, analysis and comparison between standard alpha beta (AB) Pruning and MTD(f) will be carried out at the prototype of Connect Four in terms of optimality (win percentage) and speed (execution time and the number of leaf nodes). Experiments are carried out by running computer versus computer mode with 12 different conditions, i.e. varied search depth (5 through 10) and who moves first. The percentage achieved by MTD(f) based on experiments is win 45,83%, lose 37,5% and draw 16,67%. In the experiments with search depth 8, MTD(f) execution time is 35, 19% faster and evaluate 56,27% fewer leaf nodes than AB Pruning. The results of this research are MTD(f) is as optimal as AB Pruning at Connect Four prototype, but MTD(f) on average is faster and evaluates fewer leaf nodes than AB Pruning. The execution time of MTD(f) is not slow and much faster than AB Pruning at search depth which is deep enough.

Keywords : Board Games, Connect Four, Artificial Intelligence, Alpha Beta Pruning, MTD(*f*)

1. Introduction

Connect Four is a two-player game which the first player chooses disc colour. Players then take turns dropping coloured discs into a vertically suspended grid. The objective of Connect Four game is to connect four of one's own discs next to each other (vertically, horizontally, or diagonally)[1].

In its implementation, usually Connect Four is played by humans against other humans or computer. If against computer, an artificial intelligence (AI) is required by the computer in order to play properly like human.

There are many AI algorithms that can be implemented to board games like Connect Four, i.e. random, brute force, greedy and minimax as well as its variants such as negamax, alpha beta (AB) Pruning and negamax AB Pruning.

Research on the implementation and comparison of AI algorithms at Connect Four and other board games had been done before. Research[2] implements minimax and AB Pruning algorithm, while research [3] implements random and greedy algorithm at Connect Four. Research[4] implements

negamax and AB Pruning, while research[5] implements brute force and negamax AB Pruning algorithm at Othello.

There are problems in the existing AI algorithms, random, brute force and greedy algorithm execution time are fast, but return less optimal move than minimax variants[3][5]. Minimax variants, return optimal move, but their execution time are long if the search depth (d) is deep enough[5].

To solve above problems, it is necessary to perform algorithms analysis and comparison to determine the suitable ones to be implemented at Connect Four. The suitable algorithms means optimal in choosing move and its execution time is not slow at search depth which is deep enough. The selected algorithms are AB Pruning and MTD, which later analysed and compared in terms of win percentage, execution time and the number of evaluated leaf nodes (LN). AB Pruning is selected because optimal like minimax but faster[4][2]. Negamax AB Pruning is not selected because it evaluate same LN as AB Pruning even though its implementation is simpler [6]. MTD(f) is selected because it is claimed to be simpler, more efficient and on average is better than the previous algorithms [7][8][9].

The research on AB Pruning and MTD (f) algorithm analysis and comparison has some limitations. The used AB Pruning is standard AB Pruning. The used Connect Four board is standard size, i.e. 7 columns and 6 rows. The tested parameters are evaluation value of each move, win percentage, the number of evaluated LN and the execution time of the algorithm. The used depth in the experiments are 5 through 10. The research only includes computer versus computer mode, where both computers use same depth.

With the analysis and comparison of both algorithms it is expected to determine the best algorithm to be implemented at Connect Four with deep enough depth.

2. Theoretical basis

2.1. Connect Four

There are some rules in playing Connect Four according to[10], namely:

a. Played on a grid of 7 columns 6 rows size.
b. Players take turns occupying a hole that only can be occupied once.
c. Only bottom of the empty hole in a column that can be occupied.
d. The first player who occupies four holes consecutively (orthogonally or diagonally) is the winner.

Figure 1. Connect four board [10]

2.2. Alpha Beta (AB) Pruning

AB Pruning computes the same optimal move as minimax, but more efficient by eliminates sub trees which can be proved irrelevant. AB pruning can be applied to trees of any depth and it is often possible to prune not only leaves, but the entire sub trees[11]. Example of AB Pruning searching process can be seen in Figure 2.

There are two parameters in AB Pruning that describe bounds on the backed-up values that appear anywhere along the path. Alpha (α) is the value of the best choice which is found so far at any choice point along the path for MAX. Beta (β) is the value of the best choice which is found so far at any choice point along the path for MIN.

AB Pruning updates the values of α and β as it goes along and prunes the remaining branches at a node. Pruning is done as soon as the value of the current node is known to be worse than the current α value for MAX or the current β value for MIN. The pseudocode of AB Pruning is shown Figure 2.

```
function ab(node, depth, α, β,MAXPlayer)
  if depth = 0 or node is a terminal node
      return the heuristic value of node
  if MAXPlayer
      v ← -∞
      for each child of node
          v ← max(v, ab(child, depth-1,
α, β, FALSE))
          α ← max(α, v)
          if β ≤ α
              break (* β cut-off *)
      return v
  else
      v ← ∞
      for each child of node
          v ← min(v, ab(child, depth-1,
α, β, TRUE))
          β ← min(β, v)
          if β ≤ α
              break (* α cut-off *)
      return v
```

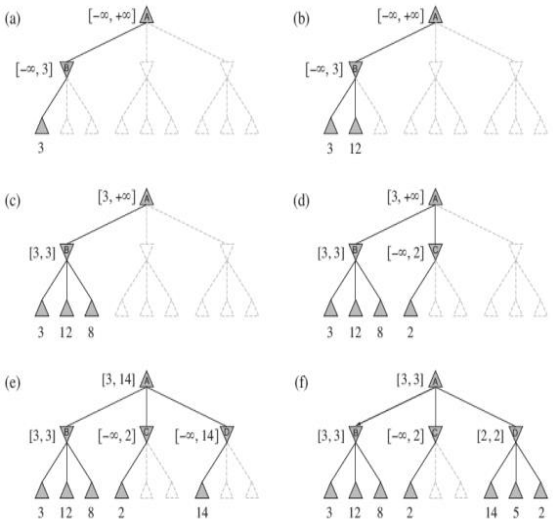

Figure 2. Searching process of AB pruning [11]

2.3. MTD(f)

MTD(f) or Memory-enhanced Test Driver with node n and value f works by calling AlphaBeta (AB) with memory a number of times with a search window of zero size. Each AB call returns a bound on the minimax value which is stored in upperbound and lowerbound, forming an interval around the true minimax value for that search depth[8]. The pseudocode of MTD(f) algorithm is as following:

```
function MTDF(root,f,d)
  g ← f
  upperbound ← +INFINITY
  lowerbound ← -INFINITY
  repeat
      if g = lowerbound then beta ← g +
1 else beta ← g;
      g ← ABMemory(root,beta-1, beta, d)
      if g < beta then upperbound ← g
else lowerbound ← g;
  until lowerbound >= upperbound;
  return g
```

MTD(f) uses a window of zero size, so that on each call AB, will either fail high or fail low. Fail high returning lowerbound, while fail low returning upper bound on the minimax value. Zero window calls cause more cutoffs, but return less information (i.e. a bound on the minimax value). MTD(f) has to call AB a number of times to find that information, nevertheless. The overhead of re-exploring parts of the search tree in repeated calls to AB disappeared. This is because MTD(f) uses AB that stores and retrieves the nodes it sees in memory. An entire sub tree in MTD(f) is aborted when a cutoff happens, furthermore, the recursive search code does not spawn re-searches anymore. The AB in MTD(f) can be simplified to use a single input bound, instead of both alpha and beta, since alpha is always one less than beta($\alpha = \beta - 1$) [8].

MTD(f) calls an AB that stores nodes in memory as it has determined their value. That nodes value then retrieved when a re-search happens, making MTD(f) become efficient. MTD(f) would re-explore most of those nodes each time a pass happens, if nodes storing are not done[8]. The Pseudocode of AB with memory is as following:

```
function ABMemory(n,alpha,beta,d)
  if retrieve(n) = OK then
    if n.lowerbound >= beta then return
n.lowerbound;
    if n.upperbound <= alpha then return
n.upperbound;
    alpha ← max(alpha, n.lowerbound)
    beta ← min(beta, n.upperbound)
  if d = 0 then g ← evaluate(n)
  else if n = MAXNODE then
    g ← -INFINITY; a ← alpha;
    c ← firstchild(n);
    while (g < beta) and (c != NOCHILD)
do
        g ←max(g,ABMemory(c,a,beta,d-1))
        a ← max(a, g);c ←
nextbrother(c);
  else
    g ← +INFINITY; b ← beta;
    c ← firstchild(n);
    while (g > alpha) and (c !=
NOCHILD)do
        g←min(g,ABMemory(c,alpha,b,d-1))
        b ← min(b, g);c ←
nextbrother(c);
  if g <= alpha then n.upperbound ← g;
store n.upperbound;
  if g >= beta then n.lowerbound ← g;
store n.lowerbound;
  return g
```

The access of transposition table takes place in the retrieve and store calls. The lines around retrieve make sure that if a value is present in the table, it is used instead of continuing the search. The store function is needed to make sure that the table is filled with values as they become available. In a real program, the best move can also be stored in the transposition table, and upon retrieving, that best move is searched first [8].

In MTD(f) searching process, if the value of the first evaluated leafnode (LN) is less than beta, upperbound(f +) will be returned and the minimax value is determined by MAX solution tree. Meanwhile, if the value of the first evaluated LN is greater than or equal to beta, lowerbound (f -) will be returned and the minimax value is determined by MIN solution tree [12].

At MAX solution tree, all child of MAX node are expanded, while only the first child of MIN node is expanded. At MIN solution tree, all child of MIN node are expanded, while only the first child of MAX node is expanded[13].

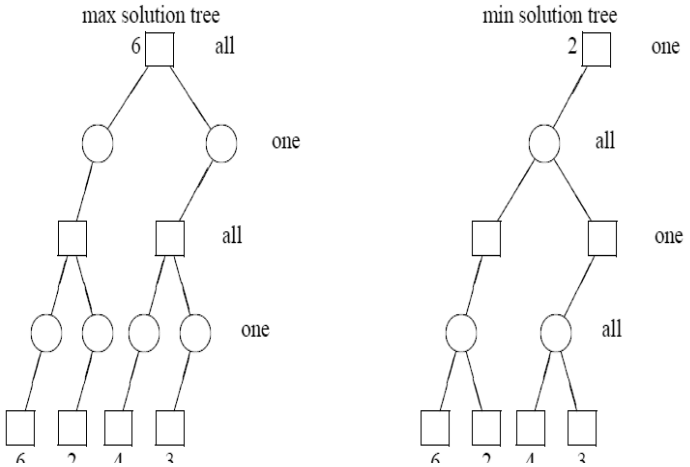

Figure 3. Solution tree [12]

2.4. Related Work

Lestari and Winata[2] implement minimax and AB Pruning algorithm at Connect Four. Experiments are performed on the time difference in taking the best move between minimax and AB Pruning computer. At depth 2 and 4, the time difference between the minimax and AB Pruning computer is not different enough. Minimax computer at depth 6 requires a long time to take the best move, unlike AB Pruning. The time required to take the best move by AB Pruning computer is less than minimax.

Sarhan et al. [3] implement random and greedy algorithm at Connect Four. The implemented algorithms are good enough at Connect Four which is played in real-time. Greedy algorithm may not returns optimal move, but it has met the system requirements without too much overhead.

Handayani et al. [4] implement negamax and AB pruning at Othello. Negamax algorithm is not efficient when used alone, because the search space is way too big. AB Pruning cutoff and reduce the search space, so that the search and evaluation process can be done faster.

Siswanto et al. [5] implement brute force and negamax AB Pruning algorithm at Othello. Brute force computer as fast as possible puts discs in any boxs as long as the move is valid, regardless of the box value of the taken position. Meanwhile, negamax AB Pruning computer apply the principle of zero-sum and heuristic value in the search tree in choosing box. This causes the taken box will be more optimal than brute force, but its execution time is long enough if the search depth is deep enough.

Gunawan et al. [7] implement negascout and MTD(f) algorithm at Othello. Based on 12 experiments versus downloaded Othello cyclog computer, negascout won 10 times, while the MTD (f) won 11 times. MTD(f) algorithm on average is more efficient than negascout, in terms of the number of searched nodes and average execution time at depth 6.

Shibahara et al. [9] compare MTD(f), negascout and AB Pruning algorithm to 500 randomly generated SHOGI game trees with fixed depth. The game trees are made as close as possible to the actual SHOGI game. MTD(f) has higher performance than the other two algorithms, even at depth 6, MTD(f) execution time is 1,8 times faster than negascout. MTD(f) performance will be more superior if the costs for evaluating the position can not be ignored like the actual game.

Plaat et al. [14] compare aspiration negascout (AspNS), AB Pruning, MT-Dual*, MT-SSS* and MTD(f) algorithm at Checkers, Othello and Chess. The compared factors are the number of evaluated leaf nodes and the number of nodes. Based on the experiments of 3 games, MTD (f) is the best, followed by AspNS, AB Pruning, MT-Dual* and lastly MT-SSS*. The number of MTD(f) leaf nodes on average is 5-10% fewer than AspNS (depend on the game), even MTD(f) also outperformed AspNS in terms of the number of nodes. The execution time of MTD(f) at Checkers and Othello is 5% faster than AspNS, while in Chess MTD(f) is 9-16% faster.

2.5. Research Object Review

Research objects which become the focus of this research are the optimality and the speed of AB Pruning and MTD(f) algorithm. Algorithm optimality is the optimality (evaluation value) of holes chosen by computer to achieves win. Algorithm speed is the execution time required by computer and the number of evaluated leaf nodes in choosing those holes.

2.6. Conceptual Framework

The conceptual framework that will be used in solving the problem formulations in this research is as shown in Figure 4

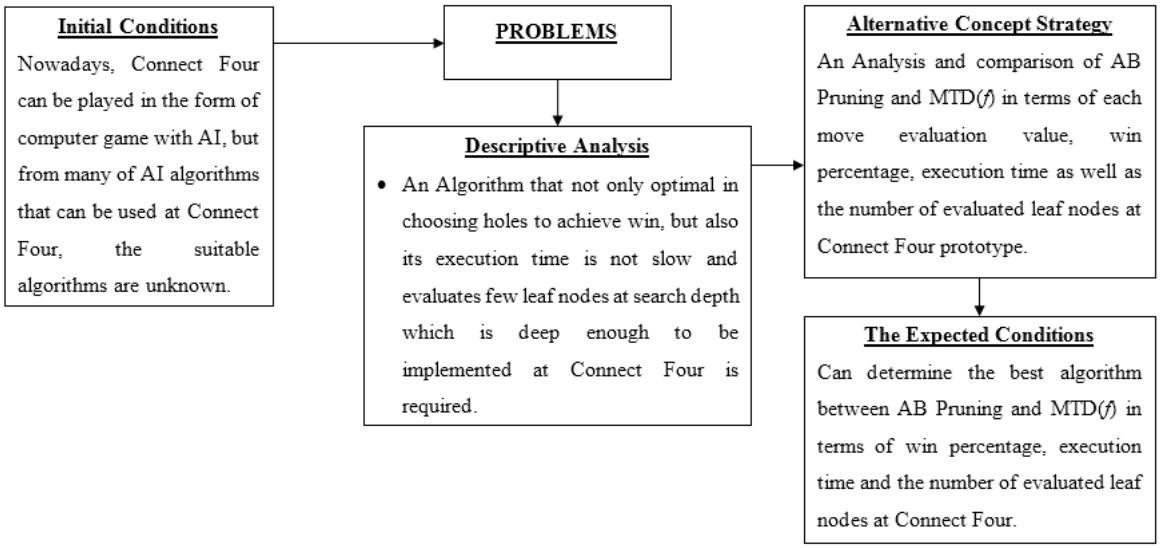

Figure 4. Conceptual framework of research

2.7. Hyphothesis

Based on the conceptual framework that has been described, the provisional guess to the research results that will be done are MTD (f) algorithm is guessed to be as optimal as AB Pruning with equal win percentage, however its execution time is faster and evaluates fewer leaf nodes at Connect Four prototype. The execution time of MTD(f) is guessed to be not slow at depth which is deep enough.

3. System and application design

Literature study method is the first step in this research, where the theoretical basis which related to the analysis of AB Pruning and MTD(f) algorithm at Connect Four on some literature and other references are studied.

The second step in this research is observation. Observation method is carried out by playing Connect Four game which is available online and offline, then analyse the used algorithms.

The analysis and comparison of AB pruning and MTD(f) algorithm will be conducted at Connect Four prototype. Both algorithms are used by the computer in choosing holes. The analysed and compared parameters are the optimality (evaluation value) of chosen holes and the number of wins, as well as the execution time and the number of evaluated leaf nodes.

The environment used in the experiments is a laptop with a dual-core processor @ 2.4GHz, 4GB memory and Windows 7 64-bit operating system.

Experiments are carried out by running Connect Four prototype with computer X (AB pruning) versus computer O (MTD(f)) mode with a same depth for both algorithms. The tested depths are 5 through 10 and at each depth, experiments are carried out with the first turn is computer X and the first turn is computer O.

Each condition is tested 10 times and the selected execution time for each move to be compared is the average of mentioned 10 test. Based on experiments which have been carried out, information such as the average execution time, the number of evaluated leaf nodes and the win percentage in each

condition are acquired. These information then analysed to determine which is the best between both algorithms.

The results of this research are an algorithm which is the fastest, most optimal and evaluate the least leaf nodes at Connect Four can be determined. The suitable algorithm may also be determined to be applied at Connect Four with deep enough depth

3.1. Algorithm Design

The pseudocode of algorithm which is designed for computer versus computer mode is as following:

```
Select who moves first
Select search depth
turns ← 1
LN_X ← 0; LN_O ← 0; Pass_O←0;
Order ← [3,4,2,5,1,6,0]
Show empty Connect Four board
WHILE gameover=False DO
  IF turn=”X” THEN
    Count start time
    Do AB Pruning function with depth n
    Update LN_X
    Choose hole and print evaluation value
    execution_time← current time-start time
    turn←”O”
  ELSE
    Count start time
    Do MTD(f) function with depth n
    Update LN_O; Update Pass_O
    Choose hole and print evaluation value
    execution_time← current time-start time
    turn←”X”
  END IF
  Update Connect Four board
  Print execution_time
  Check H,V,D segments at board
    IF connected X coin >=4 THEN
      winner←”X”
      gameover←True
    ELSE IF connected O coin >=4 THEN
      winner←”O”
      gameover←True
    END IF

  turns+=1
  IF turns > 42 THEN
    winner←”Draw”
    gameover←True
  END IF

END WHILE

Print turns, winner, LN_X,LN_O,Pass_O;
```

3.2. Evaluation Function Design

An AI algorithm requires evaluation function so that the computer can gives value to the current state of the game board. This is done so the computer can chooses the best hole from the existing holes to win the game.

At standard Connect Four board, there are 69 win segments at board, which is 24 horizontal segment (H), 21 vertical segments (V) and 24 diagonal segments (D). Each segment consists of four holes and each hole is weighted by the number of segments that can be created from that hole[15], as shown in Figure 5. The computer will consider to search the hole with highest weight first.

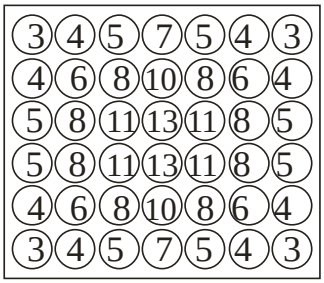

Figure 5. Weighted holes of connect four

There are some rules in evaluating segment, i.e. [16]:

a. 0 for empty holes or there are different kind of coins
b. 1 for a coin of one kind
c. 10 for two coins of one kind
d. 100 for three coins of one kind

e. Infinity (∞) for four coins of one kind (win)

The value of the segment will become positive if the evaluated segment is the coin of MAX player(X) or negative if the evaluated segment is MIN player (O).

The value returned from the designed evaluation function is the sum of evaluation value of all player's segments minus the sum of evaluation value of all opponent's segments.

3.3. Experimental Design

Experiments are carried out by running computer versus computer mode at Connect Four prototype with various different conditions. The number of game conditions that will be tested are 12 with details as following:

a. X moves first (X1st) with depth (d) 5 through 10.
b. O moves first (O1st) with depth (d) 5 through 10.

Each game conditions will be tested 10 times to find the average execution time for each move. In the experiments that will be carried out, the obtained information during experiment are recorded. Those information are in the form of the chosen holes of each turn and their evaluation value, the execution time of each turn, the average execution time, the number of LN and the total of execution time. The information of these experiments then summarized and grouped by who moves first and by depth. These information are in the form of depths, the average execution time, the number of LN, the percentage of O speed compared to X, the percentage of O LN compared to X and the end result of the game. These information then analysed and compared to determine which is the fastest and the most optimal between both algorithms, to be implemented at Connect Four.

4. Results and discussion

4.1. Experimental Setup

The experiments of AB Pruning and MTD(f) algorithm at Connect Four prototype are carried out with 12 different conditions. The experiments conditions such as the computer with which algorithm which moves first (1st) and varied depth (d). The AB Pruning algorithm is implemented on X computer, while MTD(f) algorithm is implemented on O computer. In each experiment, the same depth applied to both algorithms. The tested parameters are the execution time, the number of leaf nodes, the evaluation value of chosen holes and the number of achieved wins. Each condition respectively is tested 10 times and the chosen execution time is the average. From the results of carried out experiment, the fastest and the most optimal from both tested algorithms can be determined to be implemented at Connect with deep enough depth.

4.2. Experimental Results

The summary of experimental results of each depth is shown in Table 1. At depth 8, O(MTD(f)) as well as X(AB Pruning) won once. The average execution time of each move of X is 2.3185 seconds and O is 1,5027 seconds, while the number of leaf nodes (LN) of X is 218.777 and O is 73.957. The execution time of O is 35,19% faster and evaluate 66,2% fewer leaf node (LN) than X. Parenthesis will be provided at speed percentage if O is slower than X. The number of pass/ ABMemori function calls by O is 715 times. Similarly, the other depth.

Comparison of the number of wins of each depth are summarized in the chart as shown in Figure 6, where X won 5 times, O won 5 times and draw 2 times.

Comparison of the average execution time of each depth are summarized in the graph as shown in Figure 7. Comparison of the number of LN evaluated by both algorithms at each depth are summarized in the graph as shown in Figure 8. The difference in the execution time and the number of LN of both algorithms increases with deeper depth and is seen much different at depth 9 and 10.

Table 1. Summary of experimental results of each depth

Depth	Average Execution Time (seconds)		Speed O:X Percentage	Number of *Leaf Node* (LN)		Number of LN O:X Percentage	MTD(*f*) Number of Pass	End Results (Win)
	X	O		X	O			
5	0,1317	0,1089	17,28%	12242	5451	55,47%	201	X(1), O(1)
6	0,3083	0,3089	(0,19%)	17885	8705	51,33%	255	X(1), O(1)
7	1,6981	0,6767	60,15%	175759	36906	79,00%	456	X(1), O(1)
8	2,3185	1,5027	35,19%	218777	73957	66,20%	715	X(1), O(1)
9	14,7149	3,0543	79,24%	1311365	143557	89,05%	478	X(1), O(1)
10	27,1850	5,0731	81,34%	3352837	327609	90,23%	590	Draw(2)
Sum of End Results				**X win 5 times, O win 5 times, Draw 2 times**				

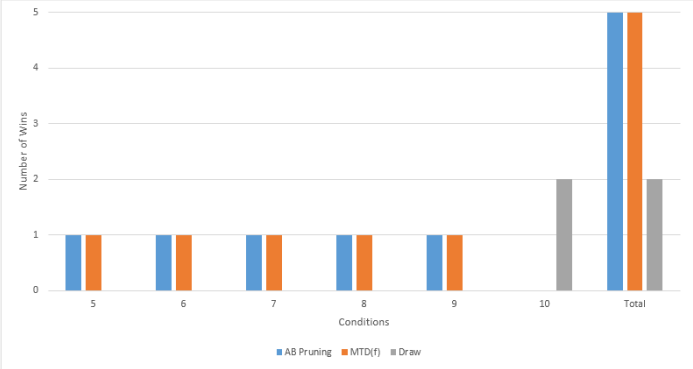

Figure 6. Number of wins comparison of each depth

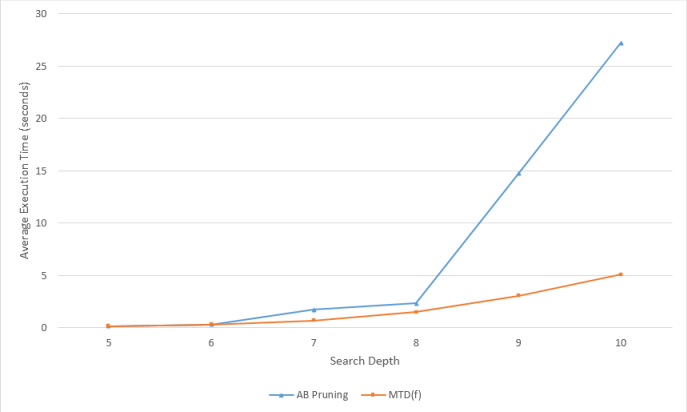

Figure 7. Execution time comparison of each depth

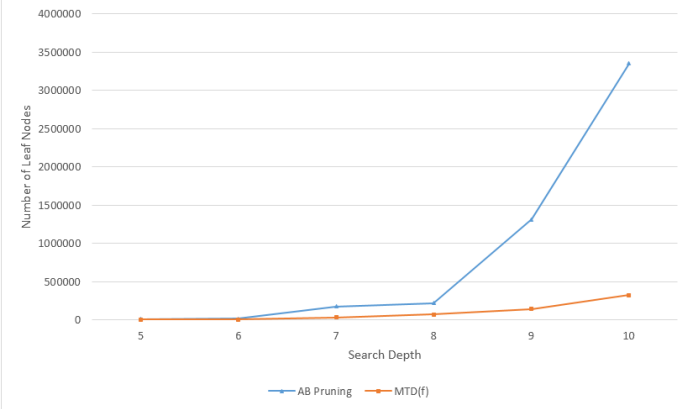

Figure 8: Number of ln comparison of each depth

4.3. *Analysis and Discussion*

MTD(f) chooses holes which are same as AB Pruning because both algorithms are minimax search variant and use same evaluation function.

At depth 6, the execution time of MTD(f) algorithm is a little slower (almost equal) than AB Pruning even though MTD(f) evaluates fewer LN. This is because the code complexity of algorithm MTD(f) is greater than standard AB Pruning code. As a result, the execution time of MTD(f) become slower, so that the difference in the number of LN is required even more.

The number of LN of O compared to X percentage at depth 7 is greater than depth 8. This anomaly occurs because of the number of passes MTD(f) at odd depth on average are fewer than at even depth. The number of evaluated LN will be reduced if the number of passes are few. These anomalies also occur as a result of reduction in the number of AB Pruning branching factor as square root at even depth. This leads to pruning occur more often causing the number of LN evaluated by AB Pruning are reduced.

Weighting holes as shown in Figure 5 makes computer explore holes with order from the largest to the smallest weight. The number of LN is reduced because the minimax value can be found earlier which lead to early pruning.

At Connect Four, the end result of game is not only affected by algorithm optimality but also affected by who moves first. First turn computer has greater chance to win the game. The execution time and the number of leaf nodes are also affected by the factor of who moves first.

MTD(f) on average is faster and evaluate fewer leaf nodes than AB Pruning. These are because MTD(f) using zero-window search and tranposition table. Zero-window search reduces the search space by pruning earlier than AB Pruning wide-window search. Transposition table is used to store and retrieve the evaluation value of board states so that it is not necessary to reevaluate the same board states. The addition of reflection function at transposition table make the computer does not need to evaluate board states which are as same as the reflection of previous board states. At deep enough depth (9 and 10), the execution time of MTD(f) is not slow and much faster than AB Pruning.

Based on experiments with 12 different conditions which have been carried out, the win percentage of X is 41,67%, the win percentage of O is 41,67%, while the percentage of the game ended in a draw is 16.66%.

The suitable algorithm for Connect Four from two tested algorithms are MTD(f). This is because MTD(f) is as optimal as AB Pruning, but its execution time is not slow and much faster if the depth is deep enough.

5. Closing

5.1. Conclusions

Based on the analysis results done on the problems, the problems formulation, research design and experiments in this research, the following conclusions are found:

a. MTD(f) algorithm is as optimal as AB Pruning at Connect Four. Based on experiments with 12 different conditions, the percentage achieved by MTD(f) is win 41,67%, lose 41,67% and draw 16,66%.
b. The execution time of MTD(f) algorithm on average is faster than AB Pruning at Connect Four. At depth 6, the execution time of MTD(f) is 0,19% slower than AB Pruning. At depth 5, 7, 8, 9 and 10, the execution time of MTD(f) is respectively 17,28%, 60,15%, 35,19%, 79,24% and 81,34% faster than AB Pruning.
c. MTD(f) algorithm on average evaluates fewer leaf nodes than AB Pruning at Connect Four. At depth 5, 6, 7, 8, 9 and 10, MTD(f) computer respectively evaluates 55,47%, 51,33%, 79%, 66,2%, 89,05% and 90,23% fewer leaf nodes than AB Pruning computer.
d. MTD(f) algorithm is suitable to be implemented for Connect Four. MTD(f) is as optimal as AB Pruning, but its execution time is not slow and much faster if the depth is deep enough.

5.2. Recommendations

Below are the recommendations which can be given as consideration for further research:

a. Comparing MTD(f) algorithm with newer and better algorithms than standard AB Pruning.

b. Improving the existing Connect Four evaluation function, so that AI can be more accurate and optimal in choosing holes.
c. Search algorithm optimization to improve execution time by using database. The evaluation value of each segment on board is pre-calculated and stored in form of hash table in database.

References

[1]. M. Sipper, *Evolved to Win*. Raleigh, USA: Lulu, 2011.
[2]. J. Lestari and A. Winata, "Implementasi Algoritma Minimax dengan Optimasi Alpha-Beta Pruning pada Aplikasi Permainan Connect Four," in *Prosiding Seminar Nasional Multidisiplin Ilmu (SeNMI)*, 2012, pp. C–50–C–58.
[3]. A. M. Sarhan, A. Shaout, and M. Shock, "Real - Time Connect 4 Game Using Artificial Intelligence," *J. Comput. Sci.*, vol. 5, no. 4, pp. 283–289, 2009.
[4]. M. S. Handayani, D. Arisandi, and O. S. Sitompul, "Rancangan Permainan Othello Berbasis Android Menggunakan Algoritma Depth - First Search," *J. Dunia Teknol. Inf.*, vol. 1, no. 1, pp. 28–34, 2012.
[5]. Siswanto, Laurentsia, and M. Anif, "Pengembangan Aplikasi Permainan Othello dengan Negamax Alpha Beta Pruning dan Brute Force," in *Seminar Nasional Sistem Informasi Indonesia (SESINDO)*, 2014, pp. 346–352.
[6]. I. Millington and J. Funge, *Artificial Intelligence for Games*, 2nd ed. Burlington, USA: Morgan Kaufmann, 2009.
[7]. Gunawan, Y. Kristian, and H. Andika, "Game Playing untuk Othello dengan Menggunakan Algoritma Negascout dan MTDF," in *Seminar Nasional Aplikasi Teknologi Informasi (SNATI)*, 2009, p. F–61–F–66.
[8]. A. Plaat, "Aske Plaat MTD(f), a new chess algorithm," 1997. [Online]. Available: people.csail.mit.edu/plaat/mtdf.html. [Accessed: 29-Feb-2016].
[9]. K. Shibahara, N. Inui, and Y. Kotani, "Adaptive Strategies of MTD-f for Actual Games," in *Proceedings of the 2005 {IEEE} Symposium on Computational Intelligence and Games (CIG05), Essex University, Colchester, Essex, UK, 4-6 April*, 2005.
[10]. E. Adams and J. Dormans, *Game Mechanics: Advanced Game Design*. Berkeley, USA: New Riders, 2012.
[11]. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, USA: Pearson Education, 2010.
[12]. A. Plaat, J. Schaeffer, W. Pijls, and A. De Bruin, "*An Algorithm Faster than NegaScout and SSS * in Practice*," unpublished, 1995.
[13]. A. Plaat, J. Schaeffer, W. Pijls, and A. De Bruin, "*A new paradigm for minimax search*," Dept. Comput. Sci., Univ. Alberta, Edmonton, Tech. Rep. TR-94-18, Dec. 1994.
[14]. A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin, "*Best-first fixed-depth minimax algorithms*," *Artif. Intell.*, vol. 87, pp. 255–293, 1996.
[15]. M. Stenmark, "*Synthesizing Board Evaluation Functions for Connect4 using Machine Learning Techniques*," M.C.S. thesis, Dept. Comput. Sci., Østfold University College, Halden, Norway, 2005.
[16]. R. L. Rivest, "*Game Tree Searching by Min / Max Approximation*," *Artif. Intell.*, vol. 34, pp. 77–96, 1988.