MCTS with Dynamic Depth Minimax

James Ji and Michael Thielscher

University of New South Wales, Australia
jamesx.ji@gmail.com, mit@unsw.edu.au

Abstract. Hybrid models combining Monte-Carlo Tree Search (MCTS) with fixed depth minimax searches have shown great success as the brute force search allow the model to navigate highly tactical domains. However, minimax is computationally expensive and unnecessary in positions that do not require precise calculations. Ideally, we can adjust the depth to efficiently rely on minimax only when needed. In this paper, we build up the motivation for augmenting MCTS with *dynamic* depth minimax searches. We analyse the nature of different domains to create some simple dynamic depth adjustment functions which we then benchmark to reinforce our hypothesis that dynamic adjustments of the search depth in MCTS-Minimax hybrids result in stronger play. For this paper we assume that heuristics or evaluator functions are not available to the player, e.g. as in the context of General Game Playing.

1 Introduction

MCTS describes an application of random-sampling in guiding the growth of a decision tree [1]. It evaluates positions by back propagating the results of simulations. Its evaluations become more accurate over time, eventually converging to optimal play [2]. It came to special prominence in 2016 after Google Deepmind's AlphaGo used a combination of MCTS and neural networks [3] to defeat the 9-dan player Lee Sedol in a five-game match [4]. The inherent randomness of MCTS makes it weak in tactical domains where it takes time to discover narrow, decisive lines. A similar concept is referred to as *traps* [5], where a *level-k* search trap is when the opponent has a winning refutation at most k plies deep. MCTS has also been characterised as making *optimistic moves* [6], as some moves initially seem strong due to a wide range of winning lines but can be refuted with precise play.

Baier and Winands [7, 8] discovered an effective approach to this dilemma by embedding shallow minimax searches throughout *MCTS-Minimax Hybrids*. Their models could immediately minimax the surrounding search space to detect decisive lines of play. However, their original paper experimented only with fixed-depth minimax searches. This meant that costly minimax searches were run regardless of the game state, even if no decisive lines would be detected within the search depth. If there are no decisive lines, computation is wasted.

This paper explores the motivation behind augmenting MCTS with dynamically depth-adjusted minimax searches to efficiently take advantage of the tactical strengths of minimax only when the position requires so. We show how the

frequency of decisive lines can be analysed and suggest some simple functions that adjust the minimax depth accordingly. We then benchmark our dynamic models against the strongest fixed depth models on the same domains as the original paper. Despite the simplicity of our adjustment methods and the strength of our opponent, we achieve strong results in many settings, demonstrating the promise of the dynamic approach alongside the potential to further improve with less crude adjustment methods.

2 Background

MCTS-Minimax Hybrids as first defined by Baier and Winands [7] describes hybrid models which combine standard MCTS with UCT and minimax to significantly outperform MCTS-Solver, a UCT variant that backpropagates proven results [9]; we will revisit their results in section 4.1. In this paper, we will focus on minimax embedded at the selection and expansion phase (MCTS-MS). Before outlining the prior research in this area, we will first motivate the need for such hybrid models. We assume the reader is familiar with MCTS and Minimax.

Tactical weakness of MCTS. Below displays a position from Chess with White to move.

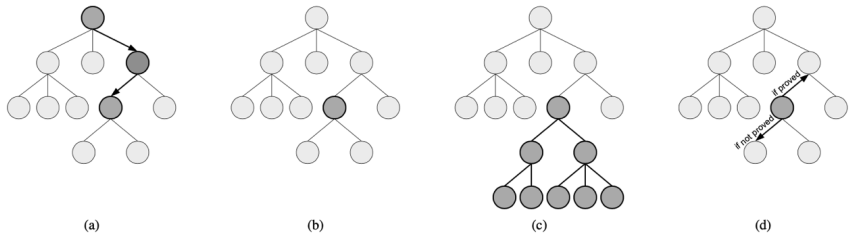
White is at a piece disadvantage but can play Re8# (Rook to e8, checkmate) to win. However, since MCTS randomly selects children, it might initially select Re2 (Rook to e2), say, giving Black time to defend. To add to the confusion, the randomness of MCTS might lead Black to reply with Nc3 (Knight to c3). If White now follows with Re8#, MCTS will backpropagate this as a win, resulting in an incorrect evaluation of the child node, Re2.

MCTS will eventually evaluate Re8# as the clear winning move; however, the time frame required might be beyond reasonable for the game settings.

Minimax in the selection and expansion phases. When a node satisfying some criteria is encountered during the selection and expansion stages, a fixed-depth minimax search is run and if possible, decisively evaluates the state. Baier

and Winands used a visit count threshold as the criterion, but they note that any criterion can be used. This theoretically improves MCTS by guiding the tree growth to avoid shallow losses and detect shallow wins. Such hybrids would easily detect decisive lines such as Re8# in the Chess example. They refer to this as MCTS-MS-d-Visit-v, where d refers to the depth of the minimax and v refers to the visit count required for a node to trigger a minimax search.

The below figure, taken from [7], illustrates the MCTS-MS hybrid. (a) Promising nodes are selected. (b) A node satisfying the minimax criterion is selected. (c) Minimax is run to detect proven game results. (d) If the node's value can be proven, the result is backpropagated. Otherwise, selection continues as normal.



In this paper, we use MCTS-MS-d-Visit-2 as the baseline hybrid model for benchmarking. The original paper used an augmented minimax enhanced with $\alpha\beta$ pruning [10], and in our paper we will also use the enhanced minimax.

3 Dynamic Depth Minimax

Without an external evaluator function, as is assumed in this paper, minimax can only back propagate useful information if it reaches terminal states. In circumstances where terminal states lie outside the search depth, computation resources are simply wasted, leading to poorer performance. We hypothesise that models combining MCTS and minimax can be improved by intelligently allocating computation to minimax, relying on it more in tactical scenarios as in the Chess example in Section 2, where it is likely to discover terminal states. We will refer to moves that lead to a solvable position within a given maximum depth as a *terminal line*. Intuitively, we should increase the minimax depth for positions containing many terminal lines and decrease it otherwise.

3.1 Analysis of terminal lines

The problem lies in determining whether or not we are in a region with many terminal lines. There are many ways to approach this; here, we analyse the frequency of terminal lines in correlation to three independent variables: turn number, branching factor and rollout length.

Test Conditions. For a given domain, MCTS-Solver played itself for 400 games with 1 second per move. Before starting the timer for each turn, we recorded the number of terminal lines and the aforementioned independent variables.

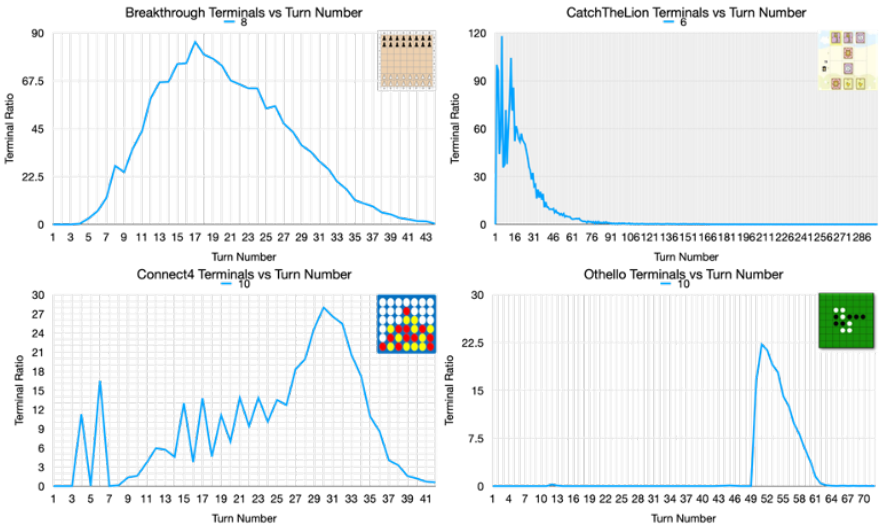


Fig. 1. Terminal Ratio vs Turn Number for the four games

Measurements. The terminal lines were measured by counting the number of children which could be solved by minimax within a specified maximum depth.

Terminal Ratio. For each turn, we summed the total number of terminal lines across all ongoing games and divided it by the number of ongoing games to produce the *terminal ratio*. Note that some games will end early while others have not reached a conclusion at a given turn number; we refer to the latter as "ongoing games." The current turn number was recorded. Branching factor was recorded as the number of children of the root node. Rollout length was recorded as the average length of 1000 simulations from the root node.

We experimented on the four domains used in the original paper [7]: Breakthrough 8×8 (the original paper used a 6×6 board), Catch the Lion, Connect-4 and Othello 8×8. These domains also happen to span a wide range of characteristics making them suitable for our purpose. We note that our experiments extend beyond these four domains and can work in any 2-player turn-based setting.

3.2 Analysis and Discussion

Fig. 1–3 show the Terminal Ratio measured against turn number, average branching factor and average rollout length. Note that the x-axis does not always span the entire possible domain because not all values appeared in the test. We evaluate the Terminal Ratio as 0 for such cases.

In Fig. 1, the terminal ratio of Connect4 and Breakthrough increases throughout the game because every turn progresses the game towards the end where terminal states are found. They drop off later on where we might expect them

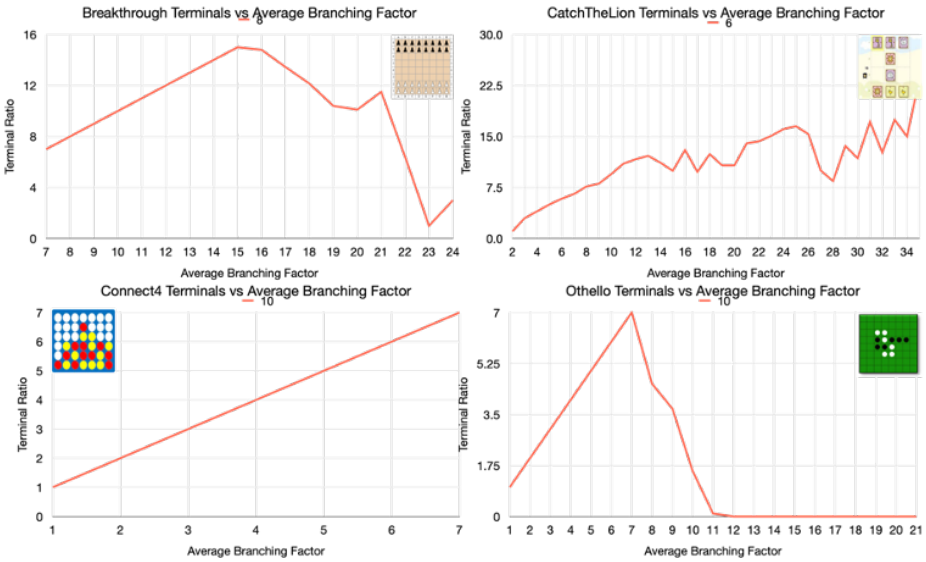


Fig. 2. Terminal Ratio vs Average Branching Factor for the four games

to increase due to more terminal states being encountered. However, this is explained by the terminal ratio being an absolute measure, decreasing because there are fewer moves to make.

Fig. 2 shows a very strong correlation between the branching factor and Connect4 and Catch the Lion. In these games, more possible lines of play might correlate to more tactical scenarios, with many decisive moves, which would benefit from minimax searches.

Fig. 3 is arguably more accurate as rollout length better indicates when we are close to terminal states. For example, turn 30 might be at the end of one Breakthrough game and only the middle of another. With rollout length, this variance is mitigated and we see a much stronger correlation. This is especially true in Catch the Lion, where turn number is almost meaningless as pieces can move backwards to negate their progress.

4 Dynamic Depth Models and Benchmarks

In this section, we will perform some benchmarks to determine the strongest fixed depth MCTS-MS-d-Visit-2 model for each domain. Then we will introduce some simple dynamic depth models based on our analysis of terminal lines. Finally, we will benchmark these dynamic depth models against the strongest fixed depth model and analyse the results.[1]

[1] We used a 2013 iMac, 2.7Ghz Quad-Core Intel Core i5, 16GB RAM, 1600MHz DDR3. We played 200 games per side for each match with 1 second per move. We

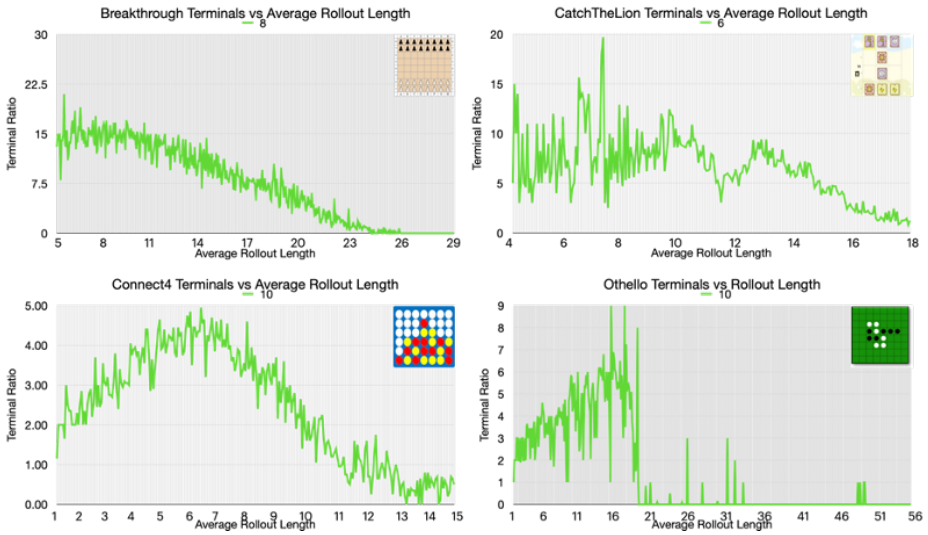


Fig. 3. Terminal Ratio vs Average Rollout Length for the four games

4.1 Strongest fixed depth model

The below tables show the win-rates of MCTS-MS-d-Visit-2 against MCTS-Solver in each domain for depths 1 to 8 to demonstrate the strength of embedding minimax in MCTS. Results for depths above 8 were omitted for brevity due to their low win-rates. All tables give 95% confidence bounds (cf. Baier & Winands).

	1	2	3	4
Breakthrough	**60.83**(±4.8)	**72.50**(±4.4)	**71.00**(±4.4)	**68.67**(±4.5)
Catch the Lion	**87.91**(±3.2)	**93.69**(±2.4)	**94.22**(±2.3)	**93.42**(±2.4)
Connect-4	**49.45**(±4.9)	**51.28**(±4.9)	**56.16**(±4.9)	**52.95**(±4.9)
Othello	**55.59**(±4.9)	43.01(±4.9)	44.77(±4.9)	37.99(±4.8)

	5	6	7	8
Breakthrough	28.00(±4.4)	14.00(±3.4)	18.50(±3.8)	13.00(±3.3)
Catch the Lion	**92.95**(±2.5)	**87.59**(±3.2)	**58.63**(±4.8)	31.36(±4.5)
Connect-4	38.97(±4.8)	31.41(±4.5)	15.35(±3.5)	17.28(±3.7)
Othello	31.55(±4.6)	20.21(±3.9)	19.90(±3.9)	9.28(±2.8)

We then determined the strongest depth for MCTS-MS-d-Visit-2 in each domain via round robin. The below tables show the win-rates of the strongest model for each domain against all other depths 1 to 8 to provide an idea of their relative strength. The strongest fixed depth model in each domain is as follows: Breakthrough: 3, Catch the Lion: 4, Connect-4: 3, Othello: 1.

played more games to differentiate close results as needed. Drawn games (which are infrequent) are discarded. Win rates are given with 95% confidence intervals.

	1	2	3	4
Breakthrough	**67.25**(±4.6)	**63.50**(±4.7)	–	**50.75**(±4.9)
Catch the Lion	**80.05**(±3.9)	**71.21**(±4.4)	**57.14**(±4.8)	–
Connect-4	**54.96**(±4.9)	48.21(±4.9)	–	**52.67**(±4.9)
Othello	–	**53.28**(±4.9)	**50.65**(±4.9)	**59.63**(±4.8)

	5	6	7	8
Breakthrough	**77.50**(±4.1)	**90.00**(±2.9)	**88.50**(±3.1)	**91.50**(±2.7)
Catch the Lion	**68.01**(±4.6)	**73.74**(±4.3)	**85.00**(±3.5)	**89.22**(±3.0)
Connect-4	**65.31**(±4.7)	**73.71**(±4.3)	**85.57**(±3.4)	**89.19**(±3.0)
Othello	**65.42**(±4.7)	**73.02**(±4.4)	**83.55**(±3.6)	**89.97**(±2.9)

Note that occasionally the strongest model might perform slightly worse against a model of another depth. However, they performed the best overall in round robin. This satisfies our purpose of picking a strong fixed depth model to benchmark against.

4.2 Adjusting dynamic depth models

Intuitively, we should adjust our depth according to our analysis, increasing and decreasing the depth when the terminal ratio is high and low. A simple approach involves directly correlating the minimax depth with the terminal ratio throughout the game. The minimax depth then needs to be linearly scaled to an appropriate range to ensure the experiment can run within the time constraints.

We created four simple depth adjustment functions which closely follow the graphs from our analysis. We will use LS to represent a function linearly normalised to the range $[0, 1]$. Let the graphs in Fig. 1, Fig. 2 and Fig. 3 be denoted as functions, f_{turn}, f_{branch} and f_{roll} respectively.

Turn-A $LS(f_{turn})$ Simple linearly normalised function on turn number.

Turn-B $LS(LS(f_{turn}) \cdot LS(t))$ where t denotes the current turn number. This essentially puts more weighting on later turn numbers, thus increasing minimax depth more during later turns.

Branch-A $LS(f_{branch})$. Simple linearly normalised function on average branching factor.

Roll-A $LS(f_{roll})$. Simple linearly normalised function on average rollout length.

For each depth adjustment function we will create a dynamic depth model by replacing the d in MCTS-MS-d-Visit-2 with our own depth adjustment function. We also benchmarked multiple weightings for each depth adjustment function like so: $f_{dynamic} \cdot k$ for $k \in \mathbf{Z}, 1 \leq k \leq d_{max}$ where $f_{dynamic}$ denotes any depth adjustment function and d_{max} denotes the maximum minimax depth used for that domain. Since we tested multiple depths to determine the strongest fixed

depth model, it is fair to test multiple weights for our depth adjustment function. For convenience, we named our models following the convention **FUNCTION-k**. For example, the model equipped with depth adjustment function **TURN-A** with weighting $k = 5$ is referred to as **TURN-A-5**.

4.3 Analysis of results

Turn-A In Connect-4 the dynamic depth model outperforms when $k = 4$. This suggests that by varying the depth of the minimax searches to be mostly shallow but occasionally going deeper than the fixed depth model ($d = 3$), we can achieve stronger results, reinforcing our hypothesis of efficient minimax usage. Our model also shows strong performance in Othello for many k. This matches our intuition that fixed depth minimaxes would be useless early in Othello as the game only nears completion when the board is filled. There are some promising results in Breakthrough ($k = 2, 4$). Performance in Catch the Lion is poor across the board, however. This could be due to the small board space of Catch the Lion, requiring precise minimaxes early on to decisively maneuver into a winning position. The tables below shows the win-rate of TURN-A-k vs the strongest fixed depth model in each domain for $1 \leq k \leq d_{max}$.

	1	2	3	4	5
Breakthrough (3)	37.25(±4.7)	**49.25**(±4.9)	48.25(±4.9)	**49.25**(±4.9)	44.5(±4.9)
Catch the Lion (4)	40.83(±4.8)	43.18(±4.9)	42.82(±4.8)	39.67(±4.8)	42.79(±4.8)
Connect-4 (3)	46.24(±4.9)	41.37(±4.8)	42.66(±4.8)	**51.37**(±4.9)	**49.87**(±4.9)
Othello (1)	**52.00**(±4.9)	**49.13**(±4.9)	48.38(±4.9)	**49.50**(±4.9)	48.13(±4.9)

	6	7	8	9	10
Breakthrough (3)	27.5(±4.4)	26.5(±4.3)	19.25(±3.9)	–	–
Catch the Lion (4)	33.45(±4.6)	–	–	–	–
Connect-4 (3)	45.08(±4.9)	46.87(±4.9)	42.51(±4.8)	46.61(±4.9)	32.47(±4.6)
Othello (1)	46.63(±4.9)	**53.25**(±4.9)	**49.88**(±4.9)	**51.63**(±4.9)	45.75(±4.9)

Turn-B By weighing the depth more so that later turns trigger deeper minimaxes, we achieve even stronger results in Othello, notably at $k = 10$. This is due to turn number being a very strong signal in this domain since terminal lines are only found in the end game. This reinforces our hypothesis that intelligent usage of minimax is important. This also achieves comparable performances to the fixed depth model in Breakthrough $k = 3$ and Connect-4 $k = 8, 9$. The tables below show the win-rate of TURN-B-k vs the strongest fixed depth model in each domain for $1 \leq k \leq d_{max}.1 \leq k \leq d_{max}$.

	1	2	3	4	5
Breakthrough (3)	38.50(±4.8)	45.83(±4.9)	**49.67**(±4.9)	48.92(±4.9)	40.00(±4.8)
Catch the Lion (4)	11.50(±3.1)	31.82(±4.6)	23.06(±4.1)	18.50(±3.8)	13.57(±3.4)
Connect-4 (3)	46.70(±4.9)	46.13(±4.9)	46.17(±4.9)	46.22(±4.9)	40.32(±4.8)
Othello (1)	**52.05**(±4.9)	**50.81**(±4.9)	**51.24**(±4.9)	**52.12**(±4.9)	48.32(±4.9)

	6	7	8	9	10
Breakthrough (3)	26.75(±4.3)	18.50(±3.8)	15.5(±3.5)	–	–
Catch the Lion (4)	11.50(±3.1)	–	–	–	–
Connect-4 (3)	44.47(±4.9)	48.33(±4.9)	**49.07**(±4.9)	**50.68**(±4.9)	42.01(±4.8)
Othello (1)	**49.18**(±4.9)	**49.87**(±4.9)	**53.39**(±4.9)	**51.09**(±4.9)	**54.57**(±4.9)

Branch-A Connect-4 achieves an incredibly strong result at $k = 3$. The reason could be due to the linear relationship between branching factor and terminal ratio shown in Fig. 2, suggesting the former is a strong predictor of the latter. This suggests that the fixed-depth model $d = 3$ was wasting unnecessary minimax searches at the cost of performance. Results in Othello remain strong across the board for reasons outlined earlier. The tables below show the win-rate of BRANCH-A-k vs the strongest fixed depth model in each domain for $1 \leq k \leq d_{max}.1 \leq k \leq d_{max}$.

	1	2	3	4	5
Breakthrough (3)	40.00(±4.8)	41.50(±4.8)	47.00(±4.9)	48.25(±4.9)	24.50(±4.2)
Catch the Lion (4)	8.88(±3.0)	10.25(±3.0)	18.14(±3.7)	21.75(±4.0)	27.85(±4.4)
Connect-4 (3)	45.00(±4.9)	39.11(±4.8)	**59.56**(±4.8)	**50.79**(±4.9)	45.16(±4.9)
Othello (1)	**54.40**(±4.9)	**50.55**(±4.9)	**50.41**(±4.9)	48.53(±4.9)	48.91(±4.9)

	6	7	8	9	10
Breakthrough (3)	8.25(±3.7)	10.25(±3.0)	10.50(±3.0)	–	–
Catch the Lion (4)	42.96(±4.9)	–	–	–	–
Connect-4 (3)	34.57(±4.7)	31.15(±4.5)	13.37(±3.3)	12.37(±3.2)	5.00(±2.1)
Othello (1)	**51.11**(±4.9)	48.61(±4.9)	**53.51**(±4.89)	**53.33**(±4.9)	47.83(±4.9)

Roll-A The results are disappointing for all domains apart from Othello. This could be explained by the high variance in our average rollout length measurements. Since our model minimaxed on the second visit, our average rollout length was sampled from just two random rollouts. For fairness, sampling more rollouts to obtain a more reliable average was not an option as it would increase computation costs, making it impossible to isolate the impact of rollout lengths on the dynamic depth model when benchmarking against the fixed depth model. Alternatively, we could increase the visit threshold (e.g. from 2 to 10) for all models to even the playing field. We leave this to future experimentation. The tables below show the win-rate of ROLL-A-k vs the strongest fixed depth model in each domain for $1 \leq k \leq d_{max}.1 \leq k \leq d_{max}$.

	1	2	3	4	5
Breakthrough (3)	41.25(±4.8)	40.50(±4.8)	42.00(±4.8)	35.25(±4.7)	26.00(±4.3)
Catch the Lion (4)	13.00(±3.3)	17.34(±3.7)	18.80(±3.8)	28.82(±4.4)	29.29(±4.5)
Connect-4 (3)	43.29(±4.9)	44.96(±4.9)	46.92(±4.9)	34.54(±4.6)	39.10(±4.8)
Othello (1)	**52.96**(±4.9)	48.64(±4.9)	**49.31**(±4.9)	**51.60**(±4.9)	**50.00**(±4.9)

	6	7	8	9	10
Breakthrough (3)	7.75(±2.6)	3.50(±1.8)	4.75(±2.1)	–	–
Catch the Lion (4)	20.40(±3.9)	–	–	–	–
Connect-4 (3)	31.56(±4.6)	26.87(±4.3)	12.76(±3.3)	11.96(±3.2)	8.51(±2.7)
Othello (1)	**59.40**(±4.8)	41.42(±4.8)	35.22(±4.7)	34.51(±4.7)	**51.08**(±4.9)

Strong performance in Othello Our models achieve strong results in Othello across the board. Othello is the perfect example of a domain where minimax searches are wasted early game and as a result, is almost guaranteed to benefit from dynamic depth adjustments, even with our crude adjustment functions.

Weak performance in Catch the Lion Our models achieve the worst results in Catch the Lion. A likely explanation is that our strongest fixed depth model in Catch the Lion is already very strong as shown in the tables at the end of Section 4.1, where its win-rate against other fixed depth models range from a minimum (but still very high) 57.14% to as high as 80.05%. More precise dynamic depth functions might be needed to outperform the strongest fixed depth model.

Results vs MCTS-Solver The purpose of Fig. 4 is to show that our model is strong generally and not only against the fixed depth model. Indeed we see strong performances across the board from our dynamic depth models. Note that whilst the results for Othello might seem unimpressive, the fixed depth model also sees the same issue. As mentioned previously in this paper, this domain is one in which minimax searches must be used very carefully as they are strictly useless for most of the game.

5 Conclusion

Dynamic depth minimax models are promising, occasionally matching or surpassing the strongest fixed depth model. Their strong performance in domains such as Othello show that intelligent minimax usage is important. However, the key takeaway is while the strength of the fixed depth models are capped as there is no room for change, the dynamic depth models have potential for improvement. Note that we used extremely rudimentary adjusment functions and we also benchmarked against the strongest fixed depth models. It is not a stretch to assume that with better adjustment functions, we can achieve even stronger performances.

We also leave to future research other novel approaches for analysing the nature of terminal lines which could lead to new ideas on dynamic depth models. In this paper, we used turn number, branching factor and rollout length. However, other heuristics such as number of pieces on the board, or even implicit heuristics are conceivable.

References

1. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: 5th International Conference on Computer and Games, pp. 72–83 (2006)
2. Koscis, L., Szpesvári, C.: Bandit based Monte-Carlo planning. In: 17th European Conference on Machine Learning (ECML), pp. 282–293 (2006)
3. Silver, D., Huang, A., Maddison, C., et al: Mastering the game of Go with deep neural networks and tree search. Nature 529:484–489 (2016)

TURN-A	1	2	3	4	5
Breakthrough (3)	**61.00**(±4.8)	**62.75**(±4.7)	**67.5**(±4.6)	**69.00**(±4.5)	**50.50**(±4.9)
Catch the Lion (4)	**84.89**(±3.5)	**86.87**(±3.3)	**92.17**(±2.6)	**92.17**(±2.6)	**90.93**(±2.8)
Connect-4 (3)	**52.87**(±4.9)	45.19(±4.9)	**52.81**(±4.9)	**49.46**(±4.9)	**56.25**(±4.9)
Othello (1)	**49.59**(±4.9)	**49.59**(±4.9)	47.43(±4.9)	**49.46**(±4.9)	**49.46**(±4.9)

TURN-A	6	7	8	9	10
Breakthrough (3)	**51.75**(±4.9)	**39.25**(±4.8)	38.75(±4.8)	–	–
Catch the Lion (4)	**92.19**(±2.6)	–	–	–	–
Connect-4 (3)	46.51(±4.9)	**53.68**(±4.9)	46.74(±4.9)	**52.69**(±4.9)	48.91(±4.9)
Othello (1)	45.71(±4.9)	**49.28**(±4.9)	**53.62**(±4.9)	47.83(±4.9)	46.43(±4.9)

TURN-B	1	2	3	4	5
Breakthrough (3)	**61.00**(±4.8)	**62.75**(±4.7)	**67.50**(±4.6)	**69.00**(±4.5)	**50.50**(±4.9)
Catch the Lion (4)	**84.00**(±3.6)	**93.39**(±2.4)	**84.00**(±3.6)	**95.96**(±1.9)	**95.88**(±1.9)
Connect-4 (3)	**53.41**(±4.9)	47.73(±4.9)	**51.14**(±4.9)	**60.00**(±4.8)	**56.82**(±4.9)
Othello (1)	**54.57**(±4.9)	**51.09**(±4.9)	**53.59**(±4.9)	**49.87**(±4.9)	**49.18**(±4.9)

TURN-B	6	7	8	9	10
Breakthrough (3)	**51.75**(±4.9)	39.25(±4.8)	34.25(±4.7)	–	–
Catch the Lion (4)	**93.00**(±2.5)	–	–	–	–
Connect-4 (3)	**54.65**(±4.9)	**52.81**(±4.9)	**52.75**(±4.9)	**55.56**(±4.9)	**53.41**(±4.9)
Othello (1)	48.38(±4.9)	**52.11**(±4.9)	**51.24**(±4.9)	**50.81**(±4.9)	**52.05**(±4.9)

BRANCH-A	1	2	3	4	5
Breakthrough (3)	**58.67**(±4.8)	**68.67**(±4.5)	**75.33**(±4.2)	**68.67**(±4.5)	**50.67**(±4.9)
Catch the Lion (4)	**84.5**(±3.5)	**83.76**(±3.6)	**87.94**(±3.2)	**90.95**(±2.8)	**93.91**(±2.3)
Connect-4 (3)	**53.41**(±4.9)	47.73(±4.9)	**51.14**(±4.9)	**60.00**(±4.8)	**56.82**(±4.9)
Othello (1)	**50.38**(±4.9)	**51.43**(±4.9)	48.18(±4.9)	43.51(±4.9)	42.86(±4.8)

BRANCH-A	6	7	8	9	10
Breakthrough (3)	20.00(±3.9)	14.67(±3.5)	13.33(±3.3)	–	–
Catch the Lion (4)	**92.96**(±2.5)	–	–	–	–
Connect-4 (3)	**54.65**(±4.9)	**52.81**(±4.9)	**52.75**(±4.9)	**55.56**(±4.9)	**53.41**(±4.9)
Othello (1)	39.01(±4.8)	35.21(±4.7)	21.92(±4.05)	19.73(±3.9)	16.08(±3.6)

ROLL-A	1	2	3	4	5
Breakthrough (3)	10.25(±3.0)	7.82(±4.0)	17.25(±3.7)	43.25(±4.9)	**61.75**(±4.8)
Catch the Lion (4)	**86.50**(±3.3)	**91.90**(±2.7)	**92.73**(±2.5)	**92.15**(±2.6)	**90.48**(±2.9)
Connect-4 (3)	9.95(±2.9)	16.36(±3.6)	31.49(±4.5)	34.29(±4.7)	38.11(±4.8)
Othello (1)	**49.35**(±4.9)	44.12(±4.9)	38.92(±4.8)	44.05(±4.9)	48.36(±4.9)

ROLL-A	6	7	8	9	10
Breakthrough (3)	**68.00**(±4.6)	**57.00**(±4.9)	**65.25**(±4.7)	–	–
Catch the Lion (4)	**88.13**(±3.2)	–	–	–	–
Connect-4 (3)	44.17(±4.9)	**52.66**(±4.9)	**55.83**(±4.9)	**52.04**(±4.9)	**52.96**(±4.9)
Othello (1)	**50.42**(±4.9)	46.59(±4.9)	**49.03**(±4.9)	**50.28**(±4.9)	**49.02**(±4.9)

Fig. 4. Win-rate of TURN-A, TURN-B, BRANCH-A, ROLL-A vs MCTS-Solver in each domain for $1 \leq k \leq d_{max}$

4. The Guardian, https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol. Last accessed 9 Nov 2023
5. Ramanujan, R., Sabharwal, A., Selman, B.: On adversarial search spaces and sampling-based planning. In: 20th International Conference on Automated Planning and Scheduling (ICAPS), pp. 242–245 (2010)
6. Finnson, H., Björnsson, Y.: Game-tree properties and MCTS performance. In: IJCAI Workshop on General Intelligence in Game Playing Agents, pp. 23–30 (2011)
7. Baier, H., Winands, M.: Monte-Carlo tree search and minimax hybrids. In: IEEE Conference on Computational Intelligence in Games (CIG), pp. 1–8 (2013)
8. Baier, H., Winands, M.: MCTS-Minimax hybrids. In: IEEE Transactions on Computational Intelligence and AI in Games 7(2):167–179 (2015)
9. Winands, M., Björnsson, Y., and Saito, J.: Monte-Carlo tree search solver. In: 6th International Conference on Computers and Games, pp. 25–36 (2008)
10. Knuth, D.E., Moore, R.W.: An Analysis of Alpha-Beta Pruning. Artificial Intelligence 6(4):293–326 (1975)