RL2: Monte Carlo Tree Search vs Alpha-Beta with ID & TT

Wilco de Boer, s1704362 Jasper Ouwerkerk, s2494876
Vincent van der Sluis, s2675358

March 27, 2020

1 General Approach

In this report we will explore the potential of Monte Carlo Tree Search (MCTS) [1] to tackle the game Hex. In the previous report the game hex and the alpha-beta[2] algorithm with iterative deepening [3] and transposition tables [4] (ID & TT) using Dijkstra shortest path algorithm [5] as an evaluation function were already explored and tested. In this report the MCTS algorithm will be compared to the Alpha-Beta algorithm with ID & TT. These algorithms are all implemented within Python. In this report the MCTS algorithm is first explained, next the MCTS algorithm is compared against the Alpha-Beta algorithm using the Elo rating[6]. The Elo rating can determine the "True" skill of a player using the Bayesian inference algorithm. In this report the rating is used to prove which program is statistically better than the other. Multiple parameters are tested for both the MCTS and the Alpha-Beta algorithm. Lastly, the MCTS parameters are tuned by testing multiple parameter sets and letting the different MCTS algorithms compete with each other. Here the Elo rating and execution time are used to determine which parameters are optimal.

2 The MCTS algorithm

Monte Carlo Tree Search algorithm is a simple algorithm that can be easily implemented. The principle of Monte Carlo Tree Search algorithm is to simulate N multiple games, where the AI and the opponent play random moves. These games offer very little information for the AI to learn a good strategy. However, when implementing on large numbers of random simulated games the strategy of the AI can increase drastically. In order to find good paths, future game states are simulated by the following mechanism, see Figure 1:

Selection: The selection starts with the initial state S_0. It starts at the root node, from there one of the bottom leafs are chosen using the UCT (Upper Confidence Bound) selection rule. This rule controls the exploration/exploitation behavior of MCTS. The formula is written below, where n_i is the parent of n_j:

$$UCT(j) = \frac{w_j}{n_j} + C_p\sqrt{\frac{ln(n_i)}{n_j}} \tag{1}$$

The parameter which can control the exploration and exploitation is C_p. When the C_p is high there is more exploration, and less exploitation, and vice versa. The $\frac{w_j}{n_j}$ is the win rate of the child node, and $\sqrt{\frac{ln(n_i)}{n_j}}$ implies for the "newness" (exploration) of the child node. Children with the highest UCT value will be selected.

Expansion: From the selected node, a new child will be generated. For each simulation the tree is expanded by one node.

Simulation: The (created) node will now randomly play out until the end of the game is reached. The wins

and loses of all games will be noted. The reward r will be based on the win/loses, +1 for a win, and -1 for a lose.

Backpropagation: The tree is updated in the backpropagation step. The backpropagating happens from the new node to the root node. For each node the visit count and the win count will be updated.

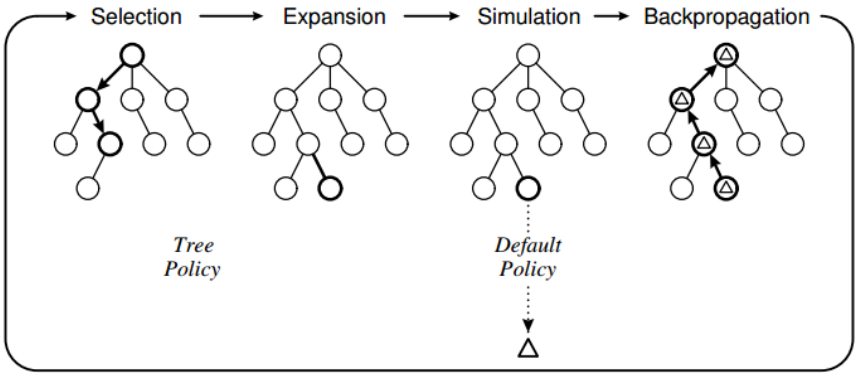


Figure 1: *An overview of a Monte Carlo Tree Search. Consisting of (1) selection, (2) expansion, (3) simulation and (4) backpropagation.*

2.1 Testing the MCTS

In order to verify that our MCTS implementation functions as intended, we use the same approach as in Assignment 1. Using the `igraph`[1] Python package, the MCTS tree which stores the wins and visits for each board state can be visualized. When we use a 2x2 board and limit MCTS to 5 iterations, it becomes easy to manually verify the algorithm's choices.

In Figure 2 the state tree after the first iteration of MCTS is visualized. As can be clearly seen, a series of random moves is simulated, which leads to a win for the MCTS player. Thus, both the win and visit count are incremented (from 0 to 1) for all involved board states. After four iterations of the algorithm, we have the state tree visualized in Figure 3. As each board state starts with zero visits, and thus has an UCT value of infinity, MCTS should move to a random unvisited board state first during each selection. As can be clearly seen in Figure 3, each of the four initial moves has been selected exactly once, in a random order, thus satisfying this property. Additionally, it should be noted that the sequence of moves (1b, 0b, 0a, 1a) leads to the same node as the sequence (0a, 1a, 1b, 0b), as this is indeed the same board state. Thus, this state is appropriately marked as visited twice. In bigger state trees, this deduplication will lead to more appropriate selection of moves according to the UCT formula. At this point, we can use the UCT formula (with $C_p = 1$) to determine which of the four initial moves MCTS should expand next:

$$UCT(1a) = \frac{1}{1} + \sqrt{\frac{ln(4)}{1}} = \sqrt{ln(4)} + 1 \tag{2}$$

$$UCT(0a) = UCT(0b) = UCT(1b) = \frac{0}{1} + \sqrt{\frac{ln(4)}{1}} = \sqrt{ln(4)} \tag{3}$$

It should now be obvious that according to UCT selection move 1a should be expanded first. As can be observed in Figure 4, this move is indeed expanded in the next iteration. Furthermore, when expanding the children for this move, a move leading to an unvisited board state is selected first.

[1] `https://igraph.org/python/`

Figure 2: *MCTS tree after one iteration on a 2x2 board. Edges are labelled with the move that was expanded, nodes are labelled according to [# of wins / # of visits]. This tree is for the first move for player one.*

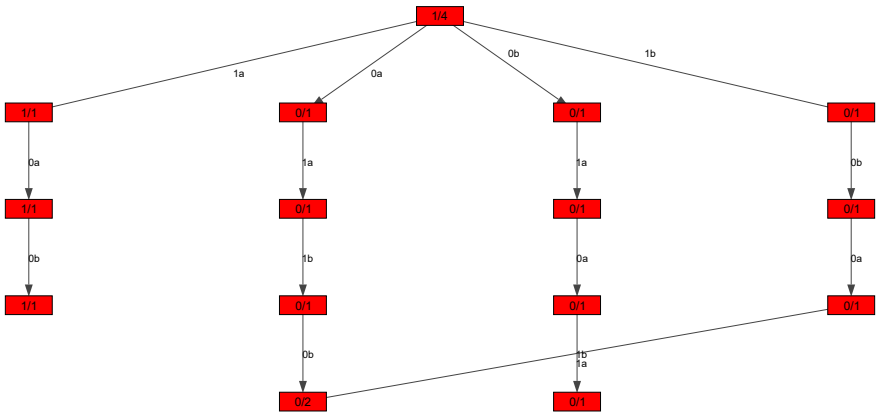


Figure 3: *MCTS tree after four iteration on a 2x2 board. Edges are labelled with the move that was expanded, nodes are labelled according to [# of wins / # of visits]. This tree is for the first move for player one.*

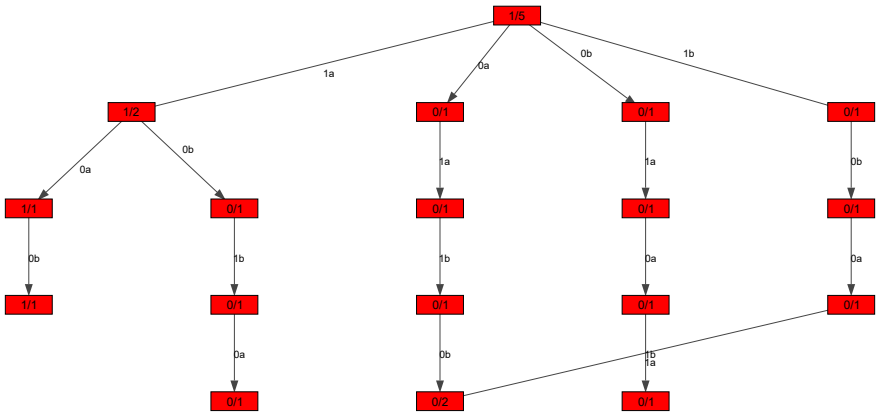


Figure 4: *MCTS tree after five iteration on a 2x2 board. Edges are labelled with the move that was expanded, nodes are labelled according to [# of wins / # of visits]. This tree is for the first move for player one.*

After these five iterations, the MCTS algorithm is terminated. At this point, all moves from the root node are examined and the move leading to the board state with the highest winrate is selected. When examining Figure 4, it should be obvious that move 1a leads to the highest winrate of 1/2, and should thus be selected. In Figure 5 it can be observed that this move is indeed performed by our implementation.

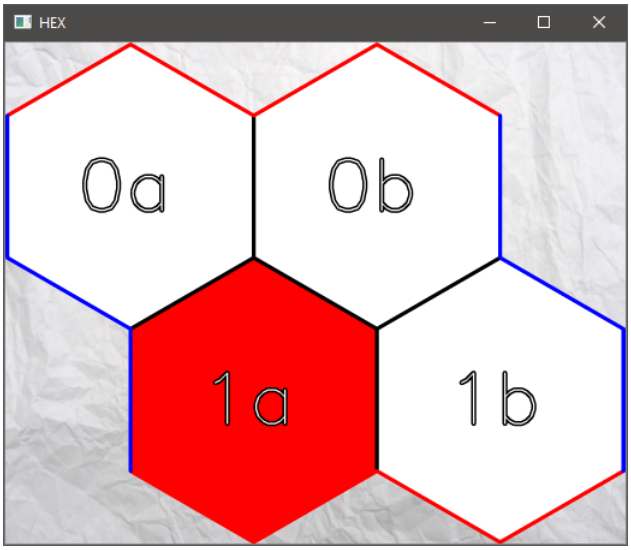


Figure 5: *2x2 Hex board after one move made by the MCTS algorithm.*

After the opponent has made their move, our MCTS algorithm needs to be run again. For the calculation of this second move, part of the tree calculated for the first move is still relevant, and should thus be kept. For this, the node representing the current board state is selected as the new root for the existing state tree. Then, every state not reachable from this new root is pruned. This subgraph is then used as a starting point for the calculation of the second move. Figure 6 shows the state tree for the second move after one iteration of the algorithm. However, two initial moves have already been expanded due to this persisting of information. When comparing Figure 6 to Figure 4, we can reason that our opponent has performed move 0a.

This is because the subgraph after move 0a is present in our new state tree, whilst the subgraph after move 0b is not. This also verifies that our implementation, mostly the part related to pruning the old tree, works as intended. Additionally, we can see that once again our MCTS implementation expands moves leading to unvisited board states first, as a new initial move is expanded whilst the remaining subgraph remains untouched.

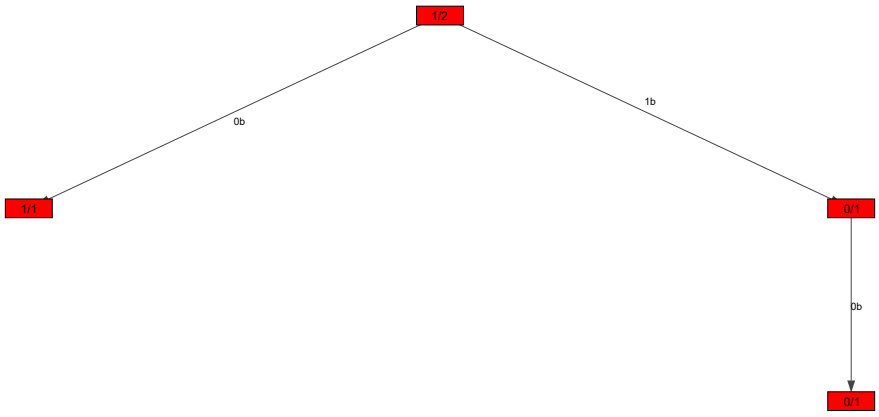


Figure 6: *MCTS tree after one iteration on a 2x2 board. Edges are labelled with the move that was expanded, nodes are labelled according to [# of wins / # of visits]. This tree is for the second move for player one.*

3 Experiments MCTS and IDTT

Previously we experimented with the ID-TT (Iterative Deepening and Transposition Tables). ID-TT showed to be better than Alpha-Beta and ID approach. In this section we want to experiment with the MCTS algorithm and compare it to ID-TT. TrueSkill is used to calculate the skill of MCTS and ID-TT. Parameter selection plays a big role in the performance. In this experiment a lot of parameter were tested, to get a good idea of the overall performances between MCTS and ID-TT. For the MCTS, 4 different values for C_p are tested, namely 0.5, 1.0, 1.5 and 2.0. This is done over a 10 seconds timeout. For the IDTT, 4 different time limits are set, namely 2, 5, 10 and 15 seconds. All the experiments were done on a 4 by 4 hexboard.

In Figure 7 the convergence of the different scores is shown. The random player in purple is added to show contrast between the algorithms and just random moves. **Note that, all games are ran in parallel on the (mithril) server.** From the figure it's clear that the random player performed the worst. This can be seen especially at the first matches, all players increase in score, except the random player. Probably due to that matches against random player are finished earlier, because these are less computationally intensive. After 40 games we see a big jump from the random player. The skill went from -1.26 to 5.64, meaning that the random player won, which is very plausible when so, many games are played. The final skill results can be seen in Table 1. The random player performed the worst with **5.75** TrueSkill value. Almost all IDTT players scored higher than MCTS, with **34.33** as the highest. Except, for MCTS(C_p=0.5) with **25.44**, it slightly outperformed IDTT(sec=2); 25.3. MCTS would probably beat IDTT if it had more time, however due to long testing time this could not be discovered in time.

Table 1: *The final TrueSkill values for each player ordered highest to lowest.*

Player	IDTT (sec 15)	IDTT (sec 10)	IDTT (sec 5)	MCTS (0.5, 10)	IDTT (sec 2)	MCTS (2.0, 10	MCTS (1.0, 10	MCTS (1.5, 10)	Random
TrueSkill Value	34.33	31.07	27.75	25.44	25.3	24.66	24.06	23.42	5.75

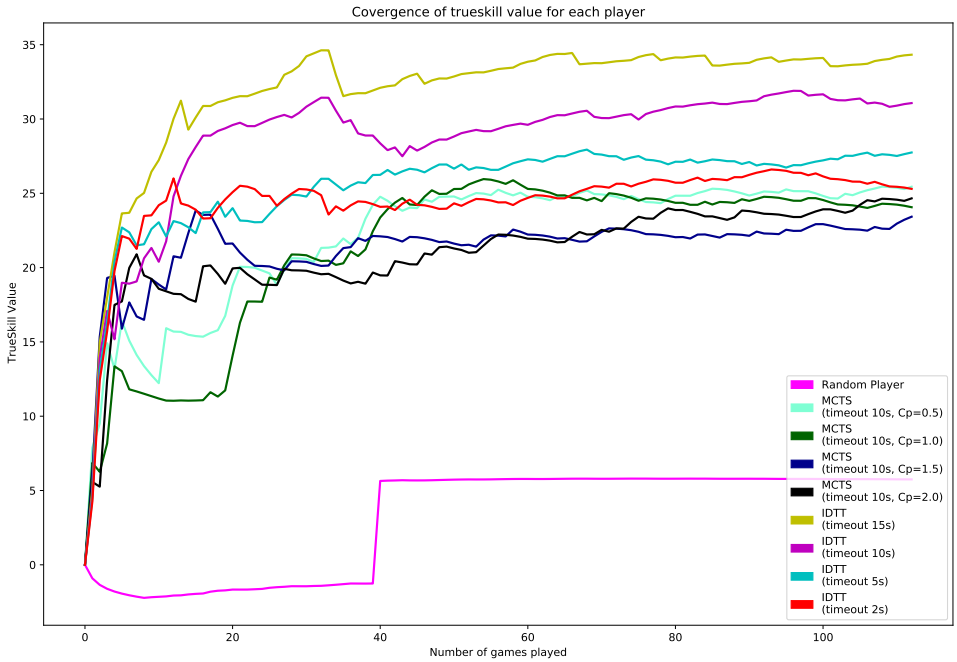


Figure 7: *The convergence of all players, when played against each other. The x-axis represents the number of games played for that moment.The x-axis represents the TrueSkill value. The IDTT show greater performance than the MCTS. The random player performed the worst(purple).*

The TrueSkills developers stated that: the average number of games needed, before a rank of a player convergence in a head-2-head game is $2 * \log_2(n)$ (n = number of users). This formula was also used in the above experiments. However, we see not for all players in Figure 7 a statistically stabilized convergence line. In order to find out, at what point the convergence is reached, 4 more experiments, between 2 players, were done. In Figure 8 a perfect scenario is shown, where player1 always wins over player2. After around 200 matches convergence is reached. Because our experiment can vary in wins and loses. Each new experiment consists of 400 (2x200) head to head games.

The results can be seen in Figure 9. It shows that between the best IDTT(sec 15) & best MCTS($0.5C_p$), the convergence line stabilized after the 200th match. However, between the MCTS($0.5C_p$) & the closest IDTT(2 sec), this is much earlier around the 70 match. This also the case between the 2 best found MCTS: $0.5C_p$ & $2.0C_p$. Between the 2 closest IDTT (10 and 15 seconds), we see a stabilization around the 120th match.

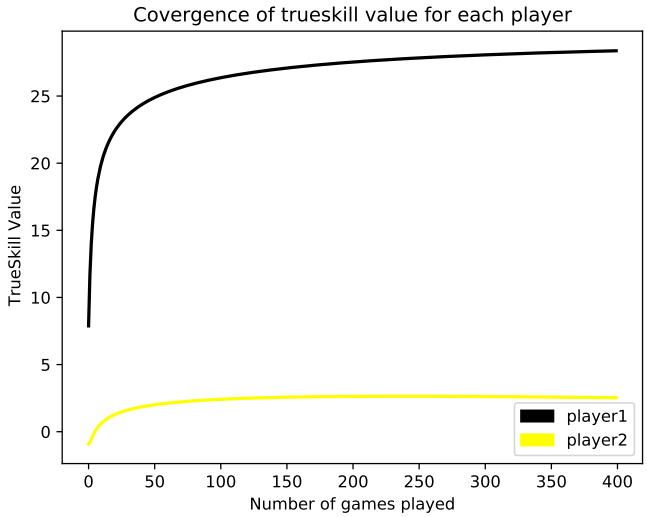
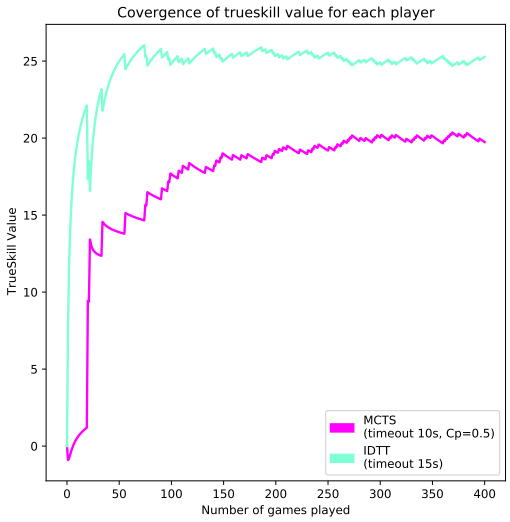
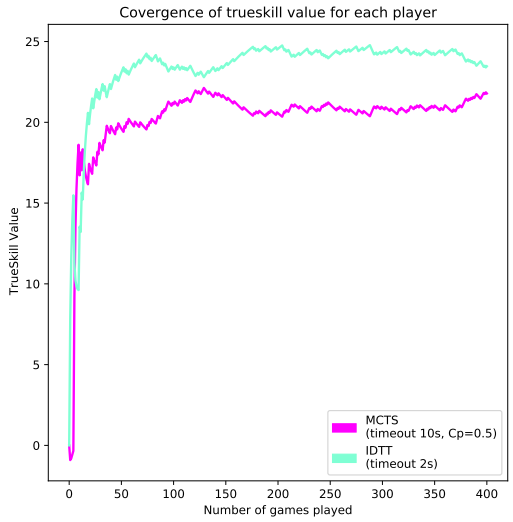


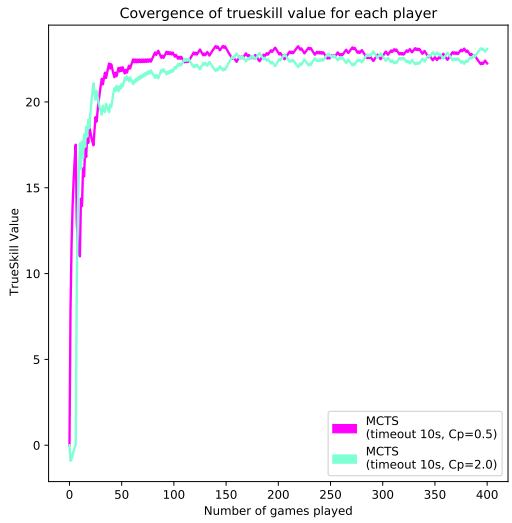
Figure 8: *The TrueSkill convergence of a scenario where player1 always wins against player2 (n=400). The x-axis represents the number of games played for that moment. The y-axis represents the TrueSkill value.*



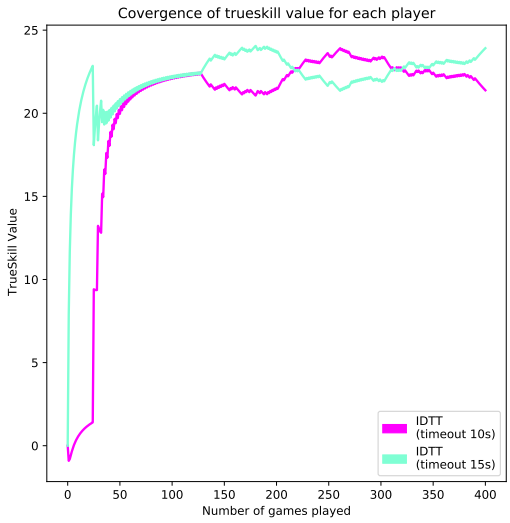
(a) *400 matches between MCTS(0.5C_p) and IDTT(15sec)*



(b) *400 matches between MCTS(0.5C_p) and IDTT(2sec)*



(c) *400 matches between MCTS(0.5C_p) and MCTS(2.0C_p)*



(d) *400 matches between IDTT(15sec) and IDTT(10sec)*

Figure 9: *Convergence plots. The x-axis represents the number of games played for that moment.The y-axis represents the TrueSkill value.*

4 MCTS tuning

In this section multiple parameters are tested with the MCTS on a hexboard of 4 by 4. All the combinations of the following parameters were used: N=[2, 4 , 8, 16, 32, 64, 128, 256, 512, 1024] and C_p=[0.0, 0.33, 0.67, 1, 1.33, 1.67, 2.0]. Each combination of parameters is defined as a player, resulting in 70 players (p). Taking all the players and calculating all the permutations results in 4830 games. All these players then play against each other and each game is played $2 \times \lceil 2 \times \log_2(p) \rceil$ times to corrects for randomness. This results in: $4830 \times (2 \times \lceil 2 \times \log_2(70) \rceil) = 62790$ games. This creates a distribution of games (elo and time) for each player. For each player the average Elo rating and execution time is taken of that distribution. This is all plotted in Figure 10. This figure shows the performance of our MCTS algorithm and the performance of each individual player. This figure shows that the implemented MCTS can do around 512 simulations in ∼4 minutes and 1024 simulations in ∼14 minutes, which corresponds to ∼1-2 simulations per second or ∼73-128 simulations per minute. It all heavily depends on the number of simulations.

To determine which parameter settings provide the best results a high Elo rating and a low execution time is used as an indication. Unfortunately, the execution time increase as the Elo rating increases. This figure shows that overall the rating increases as N increases. It also shows that MCTS with a C_p of 0 result in a much lower Elo rating when N stays the same, see for example N=1024. The parameters which result in the highest Elo rating is the MCTS with N=1024 and C_p=1 or C_p=0.67. After N=512 the Elo rating seems to increase slowly, but the average game time increases exponentially. This indicates that N=512 is more optimal than N=1024. For N=512, a C_p of 1.33 performed the best.

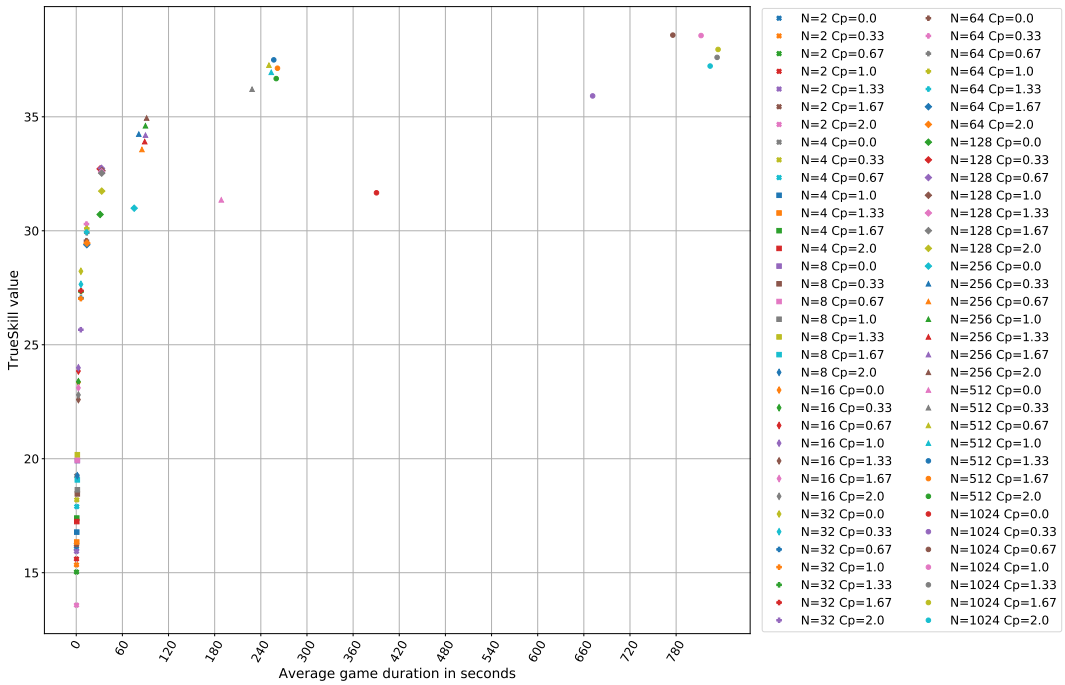


Figure 10: *The average performance of all the samples of all players that competed with each other. On the x-axis the average duration of the players game can be seen and on the y-axis the average Elo value. Here a high Elo value indicates that the player performs better and the time needed for that performance is the duration on the x-axis.*

5 Conclusion

In the end MCTS was tested and successfully implemented. The Elo rating showed that IDTT performs better than MCTS (C_p=0.5-2) when the timeout is between 5 to 15 seconds. Due to a time limit it was not possible to increase the timeout, to show if MCTS would perform better in the long run. Also, a hyperparameter tuning test was done on the MCTS algorithm to find the most optimal parameters with 4 by 4 board. It was shown that N=512 performed the best when taking into account it needs to be done in a "reasonable time" (∼4-5 minutes). The highest Elo rating for N=512 was achieved with C_p=1.33, which was around 38. In conclusion, the MCTS algorithm was successfully implemented and could sufficiently run on a board size of 4 taking around 0.5-1.0 seconds per simulation, depending on the number of simulations.

References

[1] Istvan Szita Guillaume Chaslot Sander Bakkes and Pieter Spronck. "Monte-Carlo Tree Search: A New Framework for Game AI". In: (2006). URL: https://www.aaai.org/Papers/AIIDE/2008/AIIDE08-036.pdf.

[2] T.P. Edwards D.J.; Hart. "The Alpha-Beta Heuristic". In: (1961). URL: https://dspace.mit.edu/bitstream/handle/1721.1/6098/AIM-030.ps?sequence=1&isAllowed=y.

[3] Eric A. Hansen and Rong Zhou. "Anytime Heuristic Search". In: *CoRR* abs/1110.2737 (2011). arXiv: 1110.2737. URL: http://arxiv.org/abs/1110.2737.

[4] Albert L. Zobrist. "A New Hashing Method with Application for Game Playing". In: *ICGA Journal* 13.2 (1990), pp. 69–73. DOI: 10.3233/ICG-1990-13203. eprint: 1110.2737. URL: https://content.iospress.com/download/icga-journal/icg13-2-03?id=icga-journal%2Ficg13-2-03.

[5] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1.1 (1959), pp. 269–271. ISSN: 0945-3245. DOI: 10.1007/BF01386390. URL: https://doi.org/10.1007/BF01386390.

[6] B. Schölkopf, J. Platt, and T. Hofmann. "TrueSkill^{TM}: A Bayesian Skill Rating System". In: *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. MITP, 2007, pp. 569–576. ISBN: 9780262256919. URL: https://ieeexplore.ieee.org/document/6287323.