RL Assignment 2

Stephan van der Putten s1528459 | Murad Bozik s2619822 | Yuchi Zhang s2724952

March 28, 2020

1 Introduction

This is the second part of a series on applied reinforcement learning using the game Hex. Our focus is on exploring Monte Carlo Tree Search. In the first part we implemented Hex with 2 agents. An alphabeta agent using the Dijkstra algorithm for heuristic planning; The aforementioned agent is an improved agent using the alphabeta in combination with iterative deepening and transposition tables. In this second part we are going to implement an agent using a stochastic approach for finding optimal moves in the game Hex. The optimal policy is chosen using Monte Carlo Tree Search and an upper confidence bound applied to trees for searching the tree. Our goal is to tune the agent and report our findings in this research.

A brief overview of the approach and tools is shown in section 2. The report starts with section 3 explaining the theoretical background. We follow by introducing our experimental setup in section 4. In section 5 we show our results comparing and tuning the agent. Section 6 contains our discussion. We close off in section 7 where we conclude our findings.

2 Preliminaries

In this section, we describe the approach and tools we use in this report. Our primary tool is Python 3. For the experiments we use the Trueskill library. Our game does not include a Pie Rule, that was created to balance the first mover advantage. The base Hex skeleton code is provided by the course Reinforcement learning at University Leiden, given by Aske Plaat.

2.1 General Approach

We expand on an earlier version of the game Hex to use MCTS. We have 5 files in total. The main game could be run using the *game.py* file, where our interface of the game is. The *util.py* contains the utilities of the game, an *hex_skeleton.py* contains the board and hexes. The *mcts.py* has the MCTS algorithm implementation. The *node.py* creates the game states as nodes and the necessary attributes for the utilities and agents to be performed. At the start of the game the board size is chosen, also choosing if experiments or manual play will be performed. If experiments are chosen the user can choose between MCTS or ID-TT computers that can be tuned and the amount of experiments. The MCTS has CP, N, itermax, process time: exploration, rollouts, how many steps (nodes) to explore, time limit. The ID-TT agent has process time and maxdepth: time limit and the amount of moves to look ahead. Our report will be played on different board sizes, however we focus on $3 \times 3, 4 \times 4$ for early testing and later also some bigger 9×9, 11×11 boards.

3 Algorithms

In this section, we talk about the theoretical background of Monte Carlo Tree Search, the UCT score and the trade-off between exploration and exploitation.

3.1 Monte Carlo Tree Search

In our Hex game, we implement MCTS based on the explanation of MCTS main loop on page 136 in the book[2]. The MCTS algorithm keeps a path to the leaf nodes. Once we reach the leaf node we stop and playout the leaf. The leaf node with the highest visit count and probability will be chosen as move. In practice, how do we search the full answer for MCTS algorithm? We decide if there are more children to expand. Then we will playout these nodes in our Hex game. From the current game state, we copy the game state and check if the player wins. If not, we will backpropagate, otherwise we will remember the child we have visited. Following, we choose a child we never visited. Finally, if we want to add a child to this problem, we add the basic child information (e.g., id, name, visit). Then, we can use the information to build our MCTS algorithm and choose the best move.

In contrast with MCTS, minimax works from the leaf nodes of the tree. If it is its own turn, choose max, and if it is the opponent's turn, it chooses min. In fact, it is also assumed that the opponent is smart and will use the minimax algorithm so that it will reach a Nash equilibrium. Minimax is computationally expensive and as such infeasible. As result MCTS was created which can find subtrees in large search spaces. MCTS need to avoid locally optimal solutions and must be able to explore enough space. The Upper Confidence Bounds for Trees (UCT) algorithm gives priority to nodes that have not been explored when selecting child nodes. If all have children been explored, select the next node based on a score.

3.2 Exploration vs Exploitation

As stated in the previous sector a score is used to select which strategy to apply. In RL, we use an UCT to solve the problem.

The definition of UCT is the upper confidence interval algorithm which is a game tree search algorithm. This algorithm combines the MCTS method with the UCT formula. Search algorithms have time and space advantages. Equation 1 describe the UCT equation we need to use.

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

3.3 Summary

The MCTS algorithm is divided into four steps. The first step is Selection, which is to find the best node worth exploring in the tree. The general strategy is to first select the child nodes that have not been explored. If all the nodes have been explored, the one with the highest UCT value is selected as the child node. The second step is Expansion, which is to take a step out of the previously selected child nodes to create a new child node. The general strategy is to perform an operation randomly and this operation cannot be repeated with the previous child node. The third step is Simulation, which is to start the simulation game at the new Expansion node until the end of the game is reached. In this way, you can receive the score of the node from this expansion. The fourth step is Backpropagation, which is to feedback the node scores of the previous expansion to all the previous parent nodes and update the quality value and visit times of these nodes to facilitate later calculation of the UCT value.

4 Experiments

In this section we explain two types of experiments. We compare MCTS agents against MCTS agents and MCTS to ID-TT agents using Trueskill [1].

4.1 General notes

In section 5 the following two questions will be answered: (1) How do you test an algorithm with a non-deterministic play out? (2) How many games should be played for the rating to stabilise statistically?

Table 1: Default settings MCTS and ID-TT

	Process Time (s)	Depth	CP	N	Itermax
MCTS	20	NA	0.8	100	10
ID-TT	20	3	NA	NA	NA

During manual evaluation we noticed the depth parameter of our ID-TT player takes around 1-2 seconds for each depth in the first 3 depths, further exploration can take up to minutes. For MCTS we noticed that with a simulation count of 100, each exploration step takes around 1-2 seconds. As such we conclude that 20 seconds should be enough for the MCTS to explore enough nodes on a small 5×5 board and for ID-TT to find a decent approach. Our default settings can be found in Table 1. Furthermore we realise that certain board sizes can influence how the agents might perform, this will also be accounted for.

4.2 MCTS implementation vs ID-TT

In this first experiment we compare multiple agents using the default settings unless stated elsewhere. We compare the MCTS implementation to our ID-TT agent. We assign a rating to both player and play in total 50 or 100 games, which either player will start of with. First 20 tests includes these experiments.

4.3 Hyperparameter tuning

We create an experimental setup to compare different MCTS settings to compare both MCTS vs MCTS and MCTS vs ID-TT agents. We focus the tuning of our MCTS agents on the N, rollout parameter and CP parameter for exploration. The higher CP the more our algorithm favors exploration and the lower the more it favors moves that have been visited before. The higher the N parameter is set the more 'sure' the computer is that a move is favourable vice versa the lower the more 'unsure' the computer is of the consequences of a certain move; As we also keep process time in consideration, increasing N also leaves less time over for exploration. As stated earlier we compare $3 \times 3, 4 \times 4, 9 \times 9$ and 11×11 boards. In the results we will include the winrate and elo rating.

5 Results

5.1 Questions Answers

Answer to questions (1): A non-deterministic random playout infers the win rate from a certain board state. As the amount of playouts is increased the more the win rate will converge to a 'true' win rate, as is statistically sound by the law of large numbers. If the amount of experiments is increased to a high enough number with a high enough playout, one is actually calculating the win rate from each given board state or expected win rate and the algorithm should on average perform like that with enough experiments. Once again the amount of experiments should also follow the law or large numbers. In short random playouts should be high, such as 100 and the amount of experiments can be 30+.

Question (2): As we stated in the previous questions a random sampling to find a true distribution of each player is based on the law of large numbers. As the rating follows a gaussian distribution [1] we can state that the amount of games should at least be 30. If we compare them 1 on 1. However with the correct parameter tuning (of the rating convergence can be quicker with around 10-15 games. This however requires about 10-15 games based on the right weighting factor .

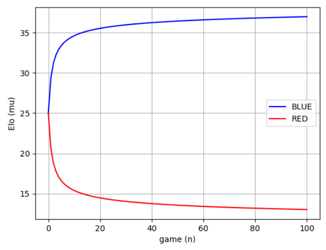
5.2 MCTS vs ID-TT

According to our experiments MCTS is clearly provides better results than iterative deepening. In our experiments the only experiments provides higher winrate for IDTT are Test0.2, Test16_swapped and Test19_swapped. All these experiments are performed on 5×5 board. In the Test0.1(4×4 board with 2 sec time limit) MCTS has higher winrate, but IDTT also can win %33 of games. When the board size becomes smaller MCTS almost always wins. Time required for exploration is logarithmic increasing for ID-TT, due to the time bounds, ID-TT cannot reach more than the depth 4. In order to give better results ID-TT algorithm needs more time than MCTS. Experiments showed that overall tests MCTS is mostly better than AlphaBeta Pruning algorithm with iterative deepening enhancement.

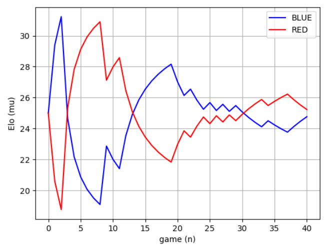
Our program doesn't include Pie Rule which eliminates the first players advantage. For this reason the experiments have been rerun by swapping the agents to see the difference among them. Even the first player is ID-TT agent the results showed MCTS agent mostly wins. The results can be found in IDTT_vs_MCTS Experiments folder.

5.3 MCTS-MCTS

In these experiments we tried to understand the effects of the hyperparameters particularly exploration parameter CP and number of simulation N. There is also other parameters 'itermax' which controls how many nodes will be searched, and 'process time' which is a limit for search process. We conducted controlled experiments to understand the effects. First, we keep all parameters same except one changing. Then we run experiments by increasing the changing parameters. But we preferred to use same process time for each agents. Higher CP increases the exploration tendency. While in small board size (3x3) the agents which has higher CP performs quite well compared to lower one, However this effect decreases if the board size increases. In Test29, for 9x9 board size we chose CP values 1.0 and 0.25 respectively. We used 50 for 'itermax'. This means 50 nodes can be searched within time limit. In 40 different the agent with higher CP has more win counts than lower one. However they were not so different as expected, this can be seen in Figure 1. On the other hand, In Test30, we performed the experiments with lower itermax value. In this case the agent with lower CP performed better than the other. Test30 showed us that if the number of nodes to be searched is not enough, then exploitation performs better that explorations. When other parameters all the same except N, the agent with higher simulation number provides better results. With higher simulation number the agents can differentiate good and bad moves easily. Increasing simulation numbers N brings less improvement than the Cp change. When the proper CP value chosen win rate highly increase, but increasing N doesn't provide results as much as CP change. Experiment results can be found in MCTS_vs_MCTS folder.



(a) 3x3 board



(b) 9x9 board

Figure 1: Effects of CP for different board sizes

6 Discussion

In this discussion we will talk about our report and what part could be improved.

In the first part of our experiments we compared ID-TT vs MCTS. One simple limitation of our experiments is the board size, for which 5 might not be enough to give conclusive results. Larger board experiments with smaller experiment count show that our current results are contradicted. In order to get expected results we need to choose higher itermax value. We leave this for future work.

During the hyperparameter tuning we noticed that our standard paramaters, process time and itermax, are very influential for the results. Actually Itermax, the amount of nodes explored, is bottle-necked as it takes far too long to explore all possible nodes. When our board size is small MCTS has a better result using a low CP to exploit as there is not much to explore. If we keep the node limit (itermax) small, CP actually gives different results when the board is big; A lower CP provides better results. This shows that if the node limit is small then exploitation gives better results then exploration. It represents the exploration/exploitation trade-off.

7 Conclusion

In this report we implemented Monte Carlo Tree Search (MCTS) for the game Hex. We explained how MCTS works and how an 'optimal' move is chosen using random playouts and the upper bound confidence for trees (UCT) search score. Our MCTS implementation has two hyperparmeters to tune N and CP, playout amount and the exploration/exploitation rate. To analyse how an MCTS player compares, we use an Iterative Deepening with Transposition Table (ID-TT) alphabeta player to compare with. In the first experimental setup we compared MCTS vs ID-TT and in the second experiment we tune the hyperparameters of the MCTS player using different board sizes.

In the first experiment we noticed that the winrate of ID-TT player is higher compared to the MCTS player only the board size is big enough. When the boards become smaller the MCTS player has a higher winrate compared to the ID-TT player. The ID-TT player has a clear advantage to be able to think ahead and as such do smarter moves on large boards, compared to the more random approximation of MCTS. The random component of MCTS starts the show improvements as the board becomes smaller or with enough itermax value. The smart ID-TT can not go as deep anymore within a certain time bound as the complexity increases with each increasing step, while the MCTS can use the random component to improve approximate better moves. As such we conclude for the first experiment that MCTS is a better player for complexer game if itermax value is high enough, due to its stochastic insights about the game compared to the 'smart' but less deep insight of the ID-TT player. In the second experiment we compared the results of the CP parameter on different board sizes with CP and N. We compared two MCTS computer against each other with different setups. From these setups we generalise our following conclusion. When comparing the smallest board of size 3×3 we notice that the player with an higher N has a better result, while increasing CP does not show any noticeable change. This is probably because increasing the play outs on a smaller board gives an better estimation how to win as each move on a smaller board is more influential for the player to win. The CP does not influence the end results a lot as exploration on a small board can not greatly improve the moves as only a few moves can actually influence whether you win or not. On the higher boards of size 5 and higher we can better see how CP and N compare. On bigger boards we noticed that the higher CP parameter gives the MCTS an improvement if itermax is high enough while increasing N does not improve the player noticeable. Our explanation is that CP on bigger boards is more important to find moves that are more optimal and as such N does not need to be really high our standard N does give a fair approximation. As such we find that our optimal CP and N are a trade-off, on smaller boards we prefer a higher N and on a bigger board we need CP to be higher. We conclude saying that the optimal MCTS agent needs to explore more on bigger and complexer game states and for simpler and smaller game states need to perform more rollouts to approximate the best moves.

7.1 Future Work

For future work, we consider that we can modify some part of MCTS to get a better time performance. If the time becomes more feasible, we can run larger experiments on bigger boards. If these limitations still stands, we should experiment using game states that have been partially filled.

7.2 Acknowledgements

This report is based on the Reinforcement Learning graduate course at University Leiden. The program is created in combination with the knowledge gathered from the course book [2], the lectures, the help of the lecturer or teaching assistants.

References

[1] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill™: a bayesian skill rating system. In *Advances in neural information processing systems*, pages 569–576, 2007.

[2] Aske Plaat. *Learning to play.* 2020.