On Variations of Gomoku AI Algorithms

Jiaxiang Liu, Siyuan Xia, Yong Hwan Kim
{j632liu, s9xia, yh35kim}@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

Gomoku is a popular board game. Despite its rules' simplicity, Gomoku is a complex game for AI because of its large state space complexity (10^{105}). Thus, Gomoku has become a suitable testbed for artificial intelligence algorithms' development. In this paper, we investigate how different algorithms for Gomoku AI perform against human players. In particular, we want to determine which of the two AI algorithms (MCTS vs CNN) poses a greater challenge to an amateur human player. We conducted a user study with 3 participants, let them played against the two Gomoku AIs, and collected statistics such as the number of steps taken and average game duration to evaluate the algorithms' performance. We found that CNN performs better than MCTS algorithm. On average, CNN takes more steps, takes a longer time to finish the game, and has a higher win rate against human players (50% vs 0%), compared to MCTS.

Introduction

The game of Gomoku (five-in-a-row) is a strategy board game that has drawn much interest from AI researchers. Originated in China, Gomoku is a turn-based game in which two players compete to play five connected pieces in a line before their component does. Despite the simplicity of its rules, Gomoku is a very "complex" game with a large state-space complexity (10^{105}), comparing to many other board games, including Chess (10^{47}), Chinese Chess (10^{40}), and Reversi (10^{28}) (wik 2020). Thus, Gomoku can serve as an ideal test-bed for the design and evaluation of many different AI paradigms and algorithms. In fact, a large number of algorithms have already been proposed by researchers, attempting to find the "best" solution to play the game. If Gomoku can successfully be solved, it will mark another major success in human's conquest of intellecual mind sports, after chess and Go. Moreover, the findings and techniques developed in creating a good AI for Gomoku can potentially have profound impacts, transcending the game itself. For example, DeepMind, team that created AlphaGo, has been working together with UK's National Health Services to tackle problems related to healthcare. This is just one example illustrating how AI's developed for board games can have large impacts on real-world problems.

Currently, there exist three common approaches that AI researchers have taken to tackle the game of Gomoku. The first approach for designing an AI for Gomoku, or more generally, for games with winners and losers, is to use reinforcement learning. Reinforcement learning enables AIs to learn both the rules of the game, and the the way to play optimally under the rules that it has learned, by itself. Another approach is to use MCST (Monte Carlo search tree), which is a popular method for playing board games using some search heuristic. Lastly, an AI for Gomoku can be also be built by a supervised learning algorithm. That is, learning from a huge dataset of possible scenes that could happen as the game progresses, and try to mimic what an expert would do in that case.

In this paper, we are interested in evaluating and comparing the performance of these different approaches against humans. We address the following research question: **algorithm from which of these three approaches can build the Gomoku AI that poses the greatest challenge to an amateur human player?** To carry out our evaluation, we will implement three Gomoku AIs, each using an algorithm from one of the three approaches mentioned above. More specifically, we will implement the specific Monte Carlo Tree Search algorithm with Upper Confidence Bounds with Tree (UCT) for the search category, best possible move in the threat space with curriculum learning pipeline for reinforcement learning, and deep convolutional neural network with the Renju dataset for supervised learning. Then, we will evaluate the performance of these AIs by conducting a user experiment. We will recruit a number of participants, let them play against all three AIs, and ask them to fill out a short survey in the end. The performances of the AIs are measured both qualitatively and quantitatively. For quantitative measures, we will record statistics including the win rate of each AI, number of steps taken for each game, etc. For qualitative measures, we will ask the participants a series of survey questions, such as "Which AI do you think is the toughest to play against?", and "On a scale from 1 to 5, how much do you think AI number 2 resembles a real human player?". We aim to come up with unbiased, and comprehensive assessments of the AIs by the end of our experiment, by combining these metrics together.

Our main contributions in this project are:

- We are the first research paper to explicitly compare the performance of both rule-based Gomoku AI and machine learning Gomoku AI.
- We are one of the few research papers that controlled the level of user's Gomoku skill in the user study to ensure the fairness of results.
- We are one of the few research papers to measure the performance of Gomoku AI using both quantitative and qualitative statistics.

Related Work

Many previous studies used search algorithms as their bases to study Gomoku. For example, (Tan et al. 2009) used Minimax algorithm with alpha-beta pruning to search for the move associated with the highest value of a heuristic function in 2009. Later, (Wang and Huang 2014) developed an adaptive AI that aimed to match human players' level based on a genetic algorithm, which is another heuristic-based search method. Among these search algorithms, Monte Carlo Tree Search (MCTS) is a heuristic search algorithm that tries to make optimal decisions by taking many simulations. It has demonstrated outstanding performance in tackling the problem of GO. As a result, people started to apply MCTS, alone with existing search algorithms, to study Gomoku. In 2016, (Tang et al. 2016) combined a neural network trained by Adaptive Dynamic Programming (ADP) with MCTS to enhance the probability that decisions made by the AI are optimal. In 2018, (Mohandas and Nizar 2018) applies a variant of MCTS combined with genetic algorithms in Gomoku.

To build an AI engine for games involving wins and losses, researchers also use ideas from reinforcement learning. In 2012, (Zhao, Zhang, and Dai 2012) studied the last 2 steps before the game terminates, and assign rewards and penalties to the winner and loser, respectively. In 2018, (Xie, Fu, and Yu 2018) proposed a procedure of training the AI by introducing the rules, mimicking another Gomoku AI, and performing self-playing reinforcement learning to achieve improvements. More recently, an AI named AlphaGo Zero is implemented in (Srivastava and Yang), which uses reinforcement learning to search for the highest probability in each move based on self-played games.

In contrast to reinforcement learning, in which the AI engine develops its own methodology of playing the game, supervised learning takes advantage of expert knowledge in training the AI. In 2016, (Shao et al. 2016) trained the AI by the concept of supervised learning with a dataset containing moves made by expert Gomoku players, with the RenjuNet Dataset, and achieve the level of expert Gomoku players. Similarly, (Yang et al.) also use supervised learning in their approach to develop an AI agent to play a professional variant of Gomoku, called Renju. They develop a double dual ResNet with tree search that determines the next move in a game of Renju, trained on their dataset. Overall, their approach achieves reasonably competitive performance among other established Renju AI's.

To the best of our knowledge, there is no consensus to which approach is the clear best or worst for Gomoku. Thus, we believe that our work makes valuable contributions to the AI community, by helping researchers working on Gomoku, and possibly other areas in AI, to identify challenges and opportunities that they may have not perceived before.

Methodology

We would like to implement the following 3 algorithms – Monte Carlo Tree Search, Convolutinal Neural Network, and Curriculum Learning based Reinforcement Learning. For each of the three algorithms, we will first talk about its major benefits (and thus our rationale for choosing the algorithm), and then describe each algorithm in details. At the end of this section, we describe the RenjuNet dataset, which is the dataset we will be using for our algorithms.

Algorithm 1: Monte Carlo Tree Search:

Given the complicated nature of Gomoku and limited computational power, exhaustive search is infeasible. (Tang et al. 2016) reduced the size of the search space based on the heuristic knowledge that there exists unique optimal response that can be easily determined as the game proceeds. For example, if there exists a line of 4 stones of the same color where one end is open, the optimal response is to connect it to 5 stones or block it from being connected to 5 stones. The following algorithm is inspired by the idea of that paper. Characteristic patterns will be mapped to unique optimal solutions which will help us reduce the number of possible simulations in the search space.

Now, we extend the general 4-step process for Monte Carlo Tree Search – selection, expansion, simulation, and backpropagation to fit our case. For the selection section, define the Amadeus distance between two positions, $\mathcal{A}(s,t)$, on the board to be the minimum number of moves required to move from position s to position t where we are able to move from one position to any of the 8 neighbors within 1 step. Provided the current state s_0, root of the Monte Carlo Tree, we will loop over all possible moves where there exists a stone, p_2, on the board such that the stone we are trying to place, p_1, satisfies $\mathcal{A}(p_1, p_2) \leq 2$.

For the expansion and simulation sections, we will set an upper bound for the maximum number of simulations performed, N. For each simulation, if the current state of the game consists of any pattern described by the heuristic knowledge, we would use the mapped optimal response instead of exploring the search space. Otherwise, we would randomly select a move in the search space, which has not been explored yet, while satisfying the Amadeus distance requirement. Moreover, we may set a limit on the number of steps for each simulation, so that we can avoid the simulations where both sides are not responding correctly. Define a hyperparameter α, and let the step limit to be α times the average number of steps for each Gomoku game in the RenjuNet dataset. If a simulation exceeds that upper bound, we will terminate it, and conclude that the current game is a draw.

For the backpropagation step, WLOG, we will assume the last placed stone is of color white. For each of the simulation, we will assign weight 1 for those that result in black wins, and 0 otherwise (white wins or draw). All these simulations will roll back up to the first move of black piece given state s_0. We define variable r_i corresponding to the i^{th} possible move from $s0$, $move_i$, to be the total weight gained from all simulations. Then, we will have an array of tuples where each tuple is $(move_i, r_i)$, and we will select the move with the largest corresponding r_i.

Algorithm 2: Deep Convolutional Neural Network:

Another feasible method is to use a convolutional neural network, so that we do not have to limit ourselves to searching algorithms. (Shao et al. 2016) suggested that Gomoku is a game embedded with patterns in subsections of the board, and the better the player, the more accurate the judgement that player makes. Using this idea as the foundation, we would like to train an AI that makes judgements as accurate as those experts in the dataset. As a result, we will preprocess each game in the dataset into an array of tuples where each tuple consists of a state within the game and the response by the expert.

Since a board has 15×15 possible positions, a 15×15 matrix will be sufficient to represent each board. We will use 3 matrices, M_1, M_2 and M_3, to represent the positions of white, black and empty positions on the board, respectively. Then, for each hidden layer, we use the filter method to discover hidden features. That is, we will define p 5×5 matrices, q 3×3 matrices, and one 1×1 matrix, denoted as kernels. For each pair of kernel k_i and featured matrix M_j, let $\{m_1, \ldots, m_\ell\}$ be the set of all possible submatrices of M_j that have the same size as k_i. We compute the dot product between m_j and k_i, and place them in the sub-featured matrix F based on the relative position of the m_i in M_j for $i \in \{1, \ldots, \ell\}$. We limit the kernel size to be smaller than 5×5 because 5 consecutive stones in a row terminates the game, so it will not be much more meaningful to obtain hidden features on a sub-board with size greater than 5×5.

Consider the pseudocode provided below to construct layers in neural network

After we set up all the hidden layers, we train our model. In our model, each record is a (state, response) pair. The loss function is defined by $-\log(s(z))$ where $s(z)$ is the softmax function $s(z) = exp(z_i)/\sum_{i=j}^{m} z_j$, where $m = 15 \cdot 15 = 225$.

For each of the possible states, we will use the CNN described above to train the model, i.e, we will run the algorithm above m times for m possible states in the dataset. The hyperparameters we would need to tune over the training process will be N, the number of hidden layers for the 3×3 kernel, p, the number of 5×5 kernels, and q, the number of 3×3 layers.

Algorithm 3: Reinforcement Learning Based on Curriculum Learning:

A big problem with training an AI algorithm to play the game of Gomoku is that there is a problem of asymmetry, which is caused by the fact that the player who goes first (player black), has a much greater advantage. This problem of asymmetry is the reason why the Alphago algorithm doesn't perform well for the game of Gomoku. Therefore, in order to account for this problem of asymmetry, we will train 2 different policy value networks separately, one for player black and one for player white (using ideas from the Reinforcement Learning algorithm with Curriculum Learning describe in (Xie, Fu, and Yu 2018)). The policy value network for player black, hereby referred to as the black network, will be trained solely on the black moves. Similarly, the policy value network for player white, hereby referred to as the white network, will be trained solely on the white moves.

The curriculum learning pipeline consists of the following 3 parts, Learn Basic Rule, Imitate Mentor AI, and Self Play Reinforcement Learning. For the first stage, we will teach our algorithm the basic rules and basic strategies of Gomoku. We will do this by first randomly generating 80000 basic moves that includes both attack moves and defence moves. An instance of a move will be specified by a three-element-tuple, $\{s, p, w\}$, where w indicates specifies the type of the move, attack or defence, s represents the current state of the board in which exists a line of 4 consecutive stones of the same color with at least one opening end, so for either the attacker and defender, there exist a respond to win the game/ prevent losing the game. Based on the type of move, p is the one-hot vector(matrix) representing the board where all entries are 0 except the best response.

For the second stage, Imitate Mentor AI, we first create a mentor AI, $\mathcal{M}$, that our AI, $\mathcal{C}$, will learn from. $\mathcal{M}$ will be a simple rule based tree search. Some examples of the rules include the "3-3", "4-4" and "3-4" strategies. (for example, a 3-3 move bans a move that simultaneously forms 2 open rows of 3 stones, and the same idea for other moves). Afterwards, we will train $\mathcal{C}$ based on the results of $\mathcal{M}$ playing

Algorithm 1 Convolutional Neural Network for Gomoku

Initialize 3 15×15 matrices, M_1, M_2, M_3.
Given a state of the board B, use M_1 to represent the positions of white pieces, M_2 to represent the positions of black pieces, and M_3 to represent the positions of empty pieces
for $i \leftarrow 1 \ldots p$ **do**
    Obtain matrix f_i by 5×5 kernel k_i
end for
The first hidden layer $\leftarrow [f_1 \ldots f_p]$
for $i \leftarrow 1 \ldots N$ **do**
    for $j \leftarrow 1 \ldots q$ **do**
        Obtain matrix f_{ij} by 5×5 kernel k_{i_j}
    end for
end for
The $(i+1)^{th}$ hidden layer $\leftarrow [f_{i1} \ldots f_{iq}]$ for $i \in \{1 \ldots N\}$
Obtain f by 1×1 kernel k
The last hidden layer $\leftarrow f$

against itself, using mini batch stochastic gradient descent with momentum. Repeat this process until we notice that $\mathcal{C}$ can successfully defeat $\mathcal{M}$ by a significant margin.

For the final stage, Self Play Reinforcement Learning, we will train the 2 policy value networks of $\mathcal{C}$ through self play (making it play against each other), by using a self stochastic policy. After each move is made, we will alter the search tree by changing its root node to the new move it makes, and then discard the remainder of the tree (until the game ends). We will initially set the first newly trained network be our currently strongest model. Then, after each training segment, we will evaluate the current training effect by playing the newly trained network play against the currently strongest model, once again using a semi stochastic policy. Then, if the newly trained model wins, we will replace the currently strongest model with the newly trained model. Otherwise, we will keep the current strongest model. After the final iteration, the final remaining model will be our final and strongest model.

As mentioned in the introduction section, we want to compare the 3 different AI algorithms to play the game of Gomoku, in each of the search, reinforcement learning, and supervised learning categories. We chose the curriculum learning algorithm for the reinforcement learning category because curriculum learning has been shown to be a very effective reinforcement learning technique or paradigm for various use cases, including board games. The idea of curriculum learning is to mimic the way human beings learn via our education system. That is, curriculum learning tries to teach its AI algorithm simple tasks first, and then gradually move onto more difficult and complicated tasks. Similar to how neural networks resumes a human's brain, curriculum learning paradigm resembles how a human being learns via our education system. This allows curriculum learning to accelerate the convergence of non convex training and improve the quality of the local optimum obtained. With these advantages, as explained above, curriculum learning has been used to be a very effective reinforcement learning paradigm for various use cases, including board games, and therefore, we have chosen to use the curriculum learning paradigm to build and train our AI for our reinforcement learning category (the reinforcement learning category from the 3 categories of search category, reinforcement learning category, supervised learning category).

Dataset:

In order to train our deep convolutional neural network (deep CNN), we will use the RenjuNet dataset, referred to in the research paper (Srivastava and Yang). Note that the term RenJu refers to a professional variant of Gomoku. The RenjuNet dataset can be downloaded directly from (ren 2020a), which is used in the research paper (Srivastava and Yang). This RenjuNet dataset was collected by the Renju International Federation, and is available on the download page of (ren 2020b), which is the official website of the Renju International Federation. The RenjuNet dataset was collected by recording the moves made by professional Renju players in certain situations of the game (states of the board). The RenjuNet dataset was collected to study the moves made by professional Renju players, and specifically to be used in studies like ours that involve trying to predict the best possible move in a game of Renju (Gomoku).

We chose the RenjuNet dataset because it is the largest dataset that is publically available which stores moves made by professional Renju players in different situations of the game (states of the board).

The RenjuNet dataset contains 1247582 data points (Srivastava and Yang). We will use the methodology used in (Srivastava and Yang) to parse and use the RenjuNet dataset. The RenjuNet dataset is stored in a .rif file format, and the data inside is stored as an xml format. The Renju Dataset contains as many as 66000 Gomoku games played by professional Gomoku (RenJu) players. Each game contains a set of moves made by these professional Renju players, where some games have 10 moves up to other games having 100 moves. From each of these moves made by these professional RenJu Players, we will extract a (s, m, c) triplet where s represents the current state of the board, m represents the move made by a professional player, and c represents the colour of which player's turn it is. Each s will be represented by 15×15 matrix, which will represent the current state of the board. Each m will be represented by a pair element (x, y), representing the coordinate of the new piece placed. Each c will be either 1 if the current player colour is black, and 0, if the current player colour is white. We will split the RenjuNet dataset into a training set, validation set and test set, and use a k fold validation strategy.

Results

To evaluate the performance of the three algorithms, we will conduct a user study. The general purpose of the study is to investigate which of the three algorithms achieve the best performance against human players, from both a qualitative and quantitative perspective. We choose to conduct a user study for the following reasons. Since we are interested in measuring how the algorithms perform against human players, it's natural for us to analyze actual games between AI and humans, rather than games between AI's. The game results can provide us with valuable statistics and insights, such as the number of moves in each game, and the users' comments on each of the algorithms. These empirical findings, along with the existing theoretical results from literature, can help us formulate a more comprehensive answer to our research question.

We will first get a list of volunteers for our user study, from our peers and friends. For any follow-up studies, we can get the list of volunteers by sending out emails to each university faculty instead. Then, we will ask the list of volunteers the following screening question, "On a scale from 1 to 5, with 1 being completely unfamiliar with Gomoku, and 5 being very good at Gomoku, how good of a Gomoku player do you think you are?". The purpose of the screening question is to keep the participants who are moderately familiar with Gomoku (those who give a rating between 2 and 4). We want to filter out people who are either completely novices, or experts at Gomoku, because they may find all three AI's too difficult/easy to play against, thus failing to

provide a good comparison between the 3. We also believe that the participants who pass our screening question will introduce less bias to our results, because they have comparable skill levels.

From those who pass the screening question, we will recruit 6 final participants based on their availability for the study. We will record some basic demographic information, including their age and gender. During the study, each of the 6 participants will play against all 3 Gomoku AI's that we have implemented. For each algorithm, so that a total of 18 games will be played. For the quantitative results, we will log information including the number of moves taken for each game, the average time taken per move for the human player, and the win rate of each AI. For the qualitative results, we will ask each participant to fill out a short survey at the end of the study. The survey will include questions such as "Which of AI_1, AI_2, AI_3 is the most mentally demanding to play against?", and "On a scale from 1 to 5, how often do you think AI_2 makes unreasonable moves?". Together, the qualitative and quantitative results help us calculate the performance of each algorithm, and allow us to determine the best performing algorithm. To reduce the amount of bias, the participants will not be told which algorithm corresponds to which Gomoku AI they play against, at any stage of the experiment. Moreover, we will randomly permute the order of the 3 Gomoku AI's for each participant to play against. This helps eliminate potential bias that comes from the order of playing.

We use the following procedure to construct the performance metrics. For each qualitative question and quantitative question, we will assign a score of 1 to the best performing algorithm, and a score of 0 to all other algorithms. For example, if 4 out of 6 participants answer that the AI corresponding to the reinforcement learning algorithm is the most mentally demanding to play against, then the reinforcement learning AI gets a score of 1 for this question, and the other two algorithms get a score of 0. (Since we believe that being "mentally demanding" is a positive measure of an algorithm's performance). We will manually assign weight c_j to each question j, and the performance p_i of algorithm i is calculated by the formula

$$\sum_j c_j \cdot s_j$$

(with s_j being 1 if algorithm i is the best performing algorithm for question j, and 0 otherwise)

Note, for simplicity, in this paper we will set the weights for all questions to be 1. This can be easily extended in the future.

We believe that these performance metrics capture the results from users studies in a comprehensive manner, as it includes all the questions and statistics that we record from each of the games. Not only is this performance metric efficient to compute, it is also highly flexible. If needed, we can easily give more emphasis to one aspect of the results over another by giving more weights to a particular question. Overall, this performance metric allows us to construct a meaningful, and robust answer to which algorithm is the best performing one.

Implementation

We managed to implement our Gomoku AI using two different algorithms: Monte Carlo Tree Search and Convolutional Neural Network.

To implement Monte Carlo Tree Search, we started by defining basic threats (how to not lose the game) and basic attacks (how to win the game) in certain game states. For example, one basic attack specifies that at any state, if we have four connected pieces in a line, we place the last piece on the same line to form five connected pieces. This leads to a win. We use these threats and attacks as the basic guiding policies for our AI. We also restrict the search space of the algorithm to places adjacent to existing pieces on the board. This restriction improved the efficiency of our algorithm, without sacrificing much performance (most of the reasonable moves in a Gomoku game are adjacent to existing pieces). At any given game state, we use the following strategy to determine the placement of the next step: we let the machine self-play for 5 more steps, using the basic guiding policies and the current game state. This process is repeated for every possible location for the next step. Each of these placements corresponds to a unique game result, which is a win, a loss, or no outcome. We create three sets, each representing one of the three game results, then put each placement in its corresponding set. If the winning set is not empty, we select the placement of our next step as the placement from the winning set that takes the fewest number of steps to achieve victory. Else, if the "no outcome" set is not empty, we randomly select a placement from this set. Lastly, if all placements lead to a loss, we select the placement that takes the most number of steps that leads to the loss.

To implement Convolutional Neural Network (CNN), we made use of the Keras Python package. We first transformed each expert game record in Renju.Net into k states. We use k to denote the number of steps of a game state, with 0 representing the empty board. Because the Renju dataset contains a large number of games, we randomly select a subset of games for training to improve efficiency while maintaining fairness. To represent the output of CNN for each game state, we use a matrix of 0's and 1's of size 15*15, with 1 presenting the position of the next move and 0 otherwise. We can now pass these training examples into CNN. We use three hidden layers: the first hidden layer consists of eight 5*5 random kernels, and the second layer consists of eight 3*3 kernels. We chose these dimensions because 5 pieces in a row terminates the game, and 3 pieces in a row forms a potential attack/threat. Finally, since the output matrix of CNN can contain any real number, we apply the softmax function to the output matrix to transform it to a matrix of probabilities that sum to 1. The entry with the largest probably is the placement for our next move. Our implementation is in Python3.

Lessons Learned

We learned the two following lessons during the implementation of our algorithms. First: for Gomoku, specifications of basic attacks and threats can significantly impact the performance of Monte Carlo Tree Search. Initially, our MCTS

AI makes many moves that seem unreasonable. For example, it would try to block its opponent, when all it needs to do is place one more piece to complete five in a line. We found out that this is because we did not give any priorities or orders to our basic guiding policies. For example, in any game state, if the AI is playing next, and it has four connected, unblocked pieces in a line already, then completing five-in-a-row should take the highest priority, because such a move leads to immediate victory. Carefully specifying the priorities of the basic guiding policies is a simple, yet effective method to improve the performance of our MCTS AI. Second: making reasonable restrictions on the search space is important for games with a large state space, such as Gomoku. In our current implement, we decided to let our algorithms only explore locations adjacent to existing pieces on the board. This is because our algorithms were running very slowly before this restriction was added. We did some inspections to figure out the reason, and found out that our AI's are searching many locations that are unreasonable. For example, without the restriction, when the play places the fist move at the center of the board, our AI is actually wasting a lot of time exploring options on the first row of the board, which is not sensible. Adding the restriction on the search space prevents our algorithms from exploring states that are unlikely to result in wins, thus improving both efficiency and performance.

Performance

We ran a pilot study with three participants to evaluate the performance of our algorithms. All participants are male, university students. All of them are familiar with the rules of Gomoku, and have comparable skill levels (e.g. none of them have received special training in Gomoku and are casual players). For each of the two algorithms, each participant played two games with the algorithm, once going first and once going second, resulted in a total of 3*2*2 = 12 games. For each game, we collected the following statistics: length of game in seconds, number of total steps, number of four pieces in a row, and the final result. We have summarized our results in Table 1. All the entries are averaged over all six games.

	Steps Taken	Time(s)	# of 4's	W/R (in %)
CNN	40	148	5	50
MCTS	26.5	75.5	2.7	0

Table 1: Average Statistics of CNN and MCTS

From Table 1, we see that CNN clearly outperforms MCTS, across all performance metrics. On average, games using CNN take 40 steps (51% increase from MCTS) and $148s$ (95% increase from MCTS) to complete, indicating a much higher effort from the players. Games using CNN form five 4 connected pieces in a row on average, indicating CNN's superior offensive and defensive performance compared to MCTS. The most obvious statistics that shows the effectiveness of CNN is the average win rate, which is 50%, compared to MCTS's 0%. We used the one-tail t-test to test the null hypothesis that the average win rate of CNN is the same as the average win rate of MCTS, and obtained a p-value of 0.0071. Since this value is much less than 0.05, we can conclude that the difference in the average rate is statistically different. Although we can use hypothesis testing to verify whether the difference in each of the other statistics is different, we only show the p-value for the win rate, as all the other p-values are statistically significant as well. Note that CNN achieves a 50% win rate on average, over the six games it played. This shows that the CNN algorithm actually achieves performance comparable to human players, which is quite impressive. In the actual user study, we will also include the overall performance score for each algorithm in the table, using the formula defined in the Results section. In Figure 1 and 2, we show two line plots, plotting the number of steps and time taken for both algorithms for all the games. Overall, the blue line (CNN) stays above the orange line, and the distance between the two lines are significant. These two line plots provide visual support to our observation that CNN is the better performing algorithm.

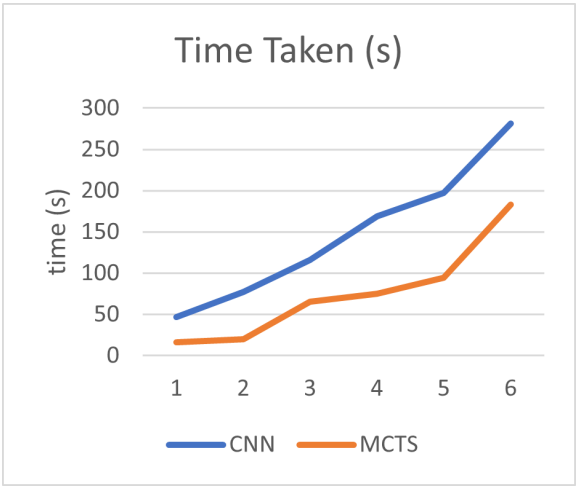

Figure 1: Time taken for CNN and MCTS

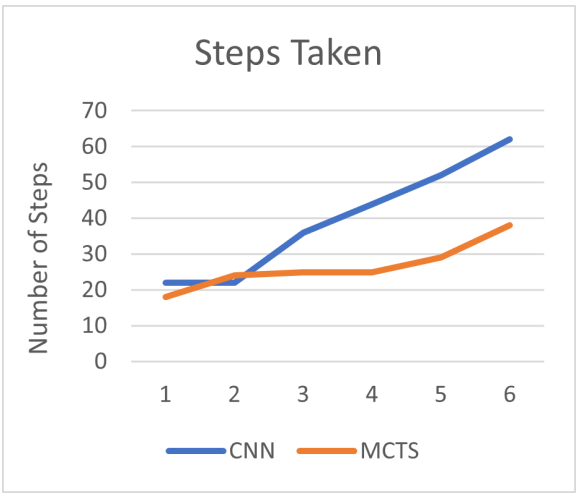

Figure 2: Steps taken for CNN and MCTS

Discussion

Implication of Results and Comparison with Prior Work Recall that our research question was to determine which of the two algorithms, MCTS vs CNN, will build the Gomoku AI that poses the greatest challenge to an amateur human player. The results of our user study showcases that the CNN algorithm poses a greater challenge to an amateur human player, compared to the MCTS algorithm. This result was expected because the CNN algorithm was trained on a dataset containing moves made by professional Gomoku players (Renju players) under different situations, while the MCTS algorithm used simple basic rules (such as win if there's 4 in a row, block opponent's 4 in a row) to create the Monte Carlo tree. One unexpected result is that the MCTS algorithm had a 0% win percentage, because we expected our MCTS algorithm to be able to win at least a few games against amateur human players. However, one explanation of this result could be that we only played a total of 6 games against the MCTS algorithm (in fact 6 games for each of the CNN algorithm and the MCTS algorithm).

The paper that implemented the MCTS algorithm for Gomoku (Tang et al. 2016) conducted a case study for their MCTS algorithm against the 5 star Gomoku, a Gomoku commercial program. However, this research paper doesn't conduct any type of research against human players. In this case study against 5-star Gomoku, it reported a win percentage of 46% against (ratio of 46:54) against the beginner level, a win percentage of 13% (ratio of 13:87) against the moderate level, and a win percentage of 0% (ratio of 0:100) against the expert level.

In our user study, our 3 participants were causal players, which correspond to the moderate player category in (Tang et al. 2016). The win percentage of our MCTS algorithm was 0%, which is lower than the 13% win percentage of the MCTS algorithm from the paper, in their case study. However, this discrepancy can be attributed to the fact that we only played 6 games while the research paper conducted 100 games (against the moderate difficulty), and a small sample size of 6 games (which was due to a limitation of number of available volunteers and time, as mentioned in the limitation section) would obviously lead to slightly more inaccurate results. Therefore, the winning percentage of 0% for the MCTS algorithm agrees with the previous research conducted by the research paper regarding the MCTS algorithm.

For our CNN algorithm, as mentioned in the implementation section and the limitation section, because the RenJu dataset contained a large number of games, and because of our lack of computational resources, we selected a random subset, 50% of the games, of the original RenJu dataset to use as our training, validation, and test datasets. Despite selecting a subset of the original RenJu dataset, the statistics of training accuracy, validation accuracy, and the training accuracy corresponded with these statistics given by the research paper that implemented the CNN algorithm. Following the research paper, we trained our CNN algorithm on 3 different sets of hyper parameters. Our statistics of training accuracy, validation accuracy, and test accuracy from these 3 different sets of hyper parameters are close and match the statistics of training accuracy, validation accuracy, and test accuracy given by the paper for the CNN algorithm. In particular, the research paper reported the highest test accuracy with the network hyper parameters of *depth* to be 10, *filter number* to be 128, and *batch size* to be 32. With these 3 hyper parameters, the research paper reported the statistics of Training Accuracy to be 48.81%, Validation Accuracy to be 42.78%, and Test Accuracy to be 42.04%, while training on the entire RenJu dataset. With the same hyper-parameter values (depth=N=10, Filter Number=128, Batch Size=32), we recorded similar results of Training Accuracy=43.23%, Validation Accuracy=39.47%, Test Accuracy=37.68%, despite training on only a random subset of the original RenJu dataset. Similar to our MCTS algorithm and the the results given in the research paper for the MCTS algorithm, we have that the statistical data given by the training of our CNN algorithm matches and agrees with the statistical data given by the training of the CNN algorithm from the research paper with the CNN algorithm.

Limitations. For the MCTS algorithm, our computational power limited our recursion depth, and we believe with more computational power, our MCTS algorithm would have been more intelligent.

For our CNN algorithm, we didn't have enough computational resources to fully test our hyperparameters. This might cause us to miss some better combinations of hyperparameters, which means that our final setting may not be optimal.

Moreover, for our CNN algorithm, we had access to only one dataset (the Renju dataset mentioned above), and hence we couldn't compare and contrast different datasets to pick the best one for the training of our CNN algorithm. Another limitation is that we couldn't get enough volunteers to play our game within the timeframe that we had. Due to busy schedules during near finals or finals week, many of our potential participants were extremely busy with last assignments, finals, research papers, coop applications, and grad school applications. With more volunteers, we would have been able to collect more data on our two AI algorithms (MCTS vs CNN), which would have led to more accurate results and a more accurate overall conclusion.

Also, if we had more volunteers, we could have selected volunteers of different skill levels to compare our two AI algorithms (MCTS vs CNN) against players of different skill levels, which would have led to a more sophisticated analysis and conclusion.

Another limitation, similar to the one above with not having enough volunteers, is the lack of time per volunteer. Again, due to our volunteers being extremely busy because of the reasons mentioned above, our volunteers could only play 4 games each (2 per each AI algorithm (MCTS vs CNN)). If each of our volunteers had more time, we would have had more data points, and with more data points, as mentioned above, we would have had more accurate results, and a more accurate overall conclusion.

Conclusion

Developing novel AI algorithms for Gomoku and evaluating their performance has caught many researches' interest. In

this paper, we conducted research on how MCTS and CNN for Gomoku AI perform against amateur human players. Our results showed that CNN is the better performing algorithm among these two algorithms. The superiority of CNN in comparison to MCTS is shown both qualitatively and quantitatively. CNN not only had a higher win rate and game duration on average, but were also described by most participants to be more challenging to play against than MCST. We believe that our paper is an important first step in helping both AI researchers and Gomoku enthusiasts to get an idea of how MCTS and CNN compare, and we look forward to seeing new results or extensions to this project. For future work, we plan to:

- Assign adaptive weights to elements in the search space instead of strict rules to allow more complexity.
- Conduct user studies with participants in different ranges of Gomoku skills.
- Evaluate similarities and differences of models based on generated quantitative results.

References

Mohandas, S., and Nizar, M. A. 2018. Ai for games with high branching factor. In *2018 International CET Conference on Control, Communication, and Computing (IC4)*, 372–376. IEEE.

2020a. Renjunet download.

2020b. Renjunet website.

Shao, K.; Zhao, D.; Tang, Z.; and Zhu, Y. 2016. Move prediction in gomoku using deep learning. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, 292–297. IEEE.

Srivastava, A., and Yang, L. A reimplementation of alphagozero on a game of gomoku.

Tan, K. L.; Tan, C. H.; Tan, K. C.; and Tay, A. 2009. Adaptive game ai for gomoku. In *2009 4th International Conference on Autonomous Robots and Agents*, 507–512. IEEE.

Tang, Z.; Zhao, D.; Shao, K.; and Lv, L. 2016. Adp with mcts algorithm for gomoku. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1–7. IEEE.

Wang, J., and Huang, L. 2014. Evolving gomoku solver by genetic algorithm. In *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, 1064–1067. IEEE.

2020. Game complexity.

Xie, Z.; Fu, X.; and Yu, J. 2018. Alphagomoku: An alphago-based gomoku artificial intelligence using curriculum learning. *arXiv preprint arXiv:1809.10595*.

Yang, G.; Chen, N.; Niu, R.; Patel, V.; Ni, Y.; Jiang, H.; and Weinberger, K. Learning the game of renju with neural network and tree search.

Zhao, D.; Zhang, Z.; and Dai, Y. 2012. Self-teaching adaptive dynamic programming for gomoku. *Neurocomputing* 78(1):23–29.