Go-Moku Solved by Some New Approachs

Yuxiang Wang
wang5350@umn.edu
Joseph Dinh
dinhx068@umn.edu

May 11, 2016

Contents

1 Introduction

1.1 Abstract

This report briefly introduces the game of Five-in-a-row, also known as Go-Moku, and investigates some related issues. First off, we will introduce some other algorithms and heuristics that other people have used in the past for Five-in-a-row. Next off, we will analyze the fairness of the game Five-in-a-row and the new search algorithm - threat-space search algorithm to solve this game. Then, we will introduce the approach that we are going to solve this game by using different threats and show some examples and figures on how each threat looks like Five-in-a-row. Meanwhile, we will also show how we use the mini-max search algorithm with alpha-beta pruning with threat space search to build our agent for this project. Finally, experiment results will be presented to readers at the end of the research.

1.2 Introduction to Go-Moku

Five-in-a-row (Go-Moku) is a pure strategy chess game. It is originally from the Heian period of Japan and is very popular nowadays especially in east Asia and some western countries. Five-in-a-row is traditionally played with two players, where one player uses white stones and the other uses black stones, on a board of n horizontal lines and n vertical lines. In order for one player to win, he or she needs to form an unbroken row of five stones horizontally, vertically, or diagonally. In this project we will use the game board of 15x15 intersections. If the board is completely filled, and no one has a five-in-a-row condition, then game ends with a draw. This game is not easy to learn and play at first, but it can improve an individuals strategic skills and enhance their thinking skills as the person keeps playing the game. Some players have claimed that the first person that moves has the advantage to winning the game, but there is no program that has proven otherwise. The reasoning for choosing this topic for our research is because in class we learned about the algorithms mini-max and alpha beta pruning. It also interesting to see that a humans strategic ability would compared to a computer programs searching ability would make one wonder if a computer program can beat a human in Go-Moku. Also, there was a World Champion program held for Go-Moku in 1991 where white won against the black. [8] I would be interesting to see how our agent would perform against the agent that won won the championship in 1991.

1.3 Outline

Overall, this paper is broken up into six different chapters. The first chapter is the introduction to the game which was previously introduced. The second chapter presents some previous works that was done by other computer scientists and researchers. The third chapter explains the the algorithms and heuristics that was used to implement Go-Moku in this project. The fourth chapter explains the experiment design and some problems that raised during our project, and the last chapter of this paper is our conclusion and explains what we think can be done in future for improving our algorithm for Go-Moku.

2 Literature Survey

2.1 Mini-Max and Alpha Beta Pruning

- Mini-Max Search
  A value is associated with each position or state of the game from an evaluation function that will be provided later. In our research project our group will use the standard 15x15 intersection game board which has a total of 225 positions. This value is computed by means of a function evaluation and it indicates how much of a threat it would be for an agent to take that position. The game-tree search gets very large as the game progresses. The agent then makes the best move according the the mini-max algorithm and that maximizes the minimum threat of the position. In our game the search depth of the agent is very important, since the search depth will determine which moves the agent sees as possible threats or winning moves. The shorter the depth search is the less threats and possible winning states the agent can detect, and in vice versa. It is possible that the mini-max algorithm can go search unexplored parts of the tree and are prone to finding bad moves, for example early in the game where there are not many stones placed on the board.

- Alpha Beta Pruning
  Alpha beta pruning will stop evaluating a move if the move found is worse than the previous move because if the algorithm were to continue evaluating then the following moves would also bad moves and thus stopping the algorithm early. In general, the more cut-offs, the fewer positions searched and the more efficient the search. The number of cut-offs is maximized if at each position the move found best by the search is actually the first move examined.[1] In Adaptive Game AI they found out that, Before implementing alpha-beta pruning, the first five AI moves take 56 seconds to execute. After implementation, the same first five moves only take 14 seconds.[11] As the game progresses the search space for the tree will become bigger and will make the search time becomes increasingly large as well, but with the help of alpha beta pruning that should decrease the amount of time to explore unwanted parts of the tree.

2.2 Iterative Deepening and Transposition Tables

- Iterative deepening
  Iterative deepening is a search where it expands the best node and iteratively go one level deeper to finding a solution. Transposition tables are sometimes used with iterative deepening to eliminate duplicated paths and so that there is no redundant data in the table. Transpositions are also used to recall the best move from previous search in the position and try it first when searching deeper so as to maximize cut-off.[2] In the early iterations moves are not assessed accurately. Some initially good moves for game agents may return with a poor expected score for one or two iterations. Later the score may improve, but the move could remain at the bottom of a list of all moves of equal score – not near the top as the initial ranking recommended. Should this move ultimately prove to be best, then far too many moves may precede it at the discovery iteration, and disposing of those moves may be inordinately expensive.[9] The method of searching can increase the probability of searching the best move first at the root of the tree. This algorithm can be useful if time is a constraint

since if the algorithm needs to be terminated it will return the best move that it has found up to where it terminated.

- Transposition Tables
  Transposition tables will keep record of multiple paths that are the same, so that it does not need to do the same search over again. The tables keep a unique number for every possible state on the game board. Where every unique position will have the best move, current evaluation, Alpha and Beta will be stored with the unique position. Transportation tables tend to be large from keeping track of all the possible paths and or positions.

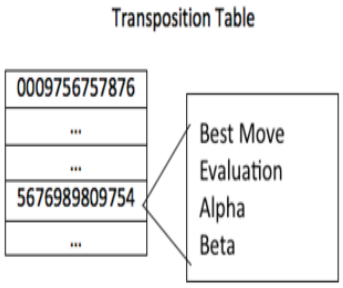

Figure 1: Transposition Table

2.3 Killer Heuristic and History Heuristic

- Killer Heuristic
  This heuristic, involves remembers at each depth of search the move(s) which seem to be causing the most cutoffs (killers) [cutoffs are sub-trees that are not in the range of the alpha-beta search and are eliminated]. The next time the same depth in the tree is reach, the killer move is retrieved and used.[10] This can be used to quickly eliminate the same sub-trees that caused earlier cutoffs and thus improving the search time of our game.

- History Heuristic
  Our agent will start at an initial state and will try to win the game by repeatedly looking for a goal state. The history heuristic implementation is based on two parameters, one is mapping the moves to the history and a number associated with how sufficient move was. Mapping the moves in our case can be be mapped to compact tables. Calculating a sufficient move can be done by counting every time there is a good move, the history score for that move will be increased. The greater the score the better the moves are. Though that does not mean that the move with the higher score is the best move, as it may lead into unforeseen moves that may appear in the future. The knowledge gathered can be used to order the moves based on the scores from highest to lowest.[**?**]yeah The information gathered previously can also be used and distributed throughout the tree for quicker calculation.

- Other heuristic techniques
  [6] focused more on analyzing the different heuristic values for Five-In-Row game. They also used alpha-beta search to approach this game. Moreover, some different heuristics were presented by authors, and they discussed the performance of how each heuristic value affects

their agent in detail. They concluded that the Simple [6] and SimpleDef heuristics [6] performed the best among the rest of heuristics. Furthermore, using some ordering heuristics [6] can also increase the effectiveness of alpha-beta pruning. They showed that the line-based heuristic [6] and the neighbor-based heuristic [6] performed very well when their agent used them altogether.

3 Approach

3.1 Mini-Max and Alpha Beta Pruning with Threats

Mini-max and alpha beta pruning was introduced earlier in this paper. In this section, we will show how those two algorithms will be used in the five-in-a-row agent that we used for our project. In the example provided (figure 2), at the root of the mini-max tree the root would be a maximum, and the next depth would be a minimum and the next would be maximum and continuing that pattern so forth down the tree. The reasoning why it would make sense in this game is because when it is our move, starting from the root of the tree, we want to maximize our threat to winning the game. The next depth would be nodes that are minimizing their threat if our agent made a move at that position. The pattern continues down the tree as far the search depth is implemented. As there are more possible moves as the games progresses the longer it takes to search for the better solution.

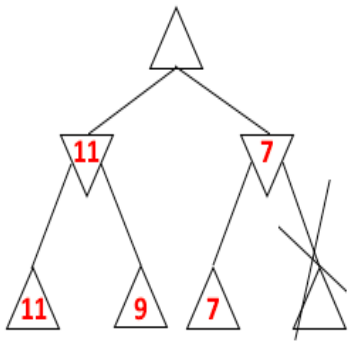

Figure 2: Small example of mini-max tree

The alpha beta pruning that our agent uses will prune the tree when conditions are met. When there is a evaluation at a maximum, root node in this case, will evaluate the from left to the right tree. So if the left minimum of the tree was evaluated a 12 while the right minimum tree for example say the left maximum, third maximum on the bottom is a 7, one can see that the evaluator above it is a minimum so the right of the right minimum evaluation will be ignored because the root node is looking for a maximum number. The right of the root node was found to be 7 while the left of the root node was a 11, therefore the right of the right minimum does not need to explored any farther and pruned since the maximum was already found.

3.2 Threat-Space-Search

There is also a new search technique named threat-space search algorithm which can solve Go-Moku problem very efficiently. Threat-space search [6] will be used in our project. This algorithm

is concentrating on the space of different threats. Threat-space search algorithm itself will not win the game. This algorithm looks for opportunities to create hazards, i.e. a situation in which the opponent will have to fend off an attack, not to lose. Threat-space search algorithm is designed to find such a string of threats, which will be a sequence of victories. As mentioned below in section 4.1 [8], there are different kinds of threats in this game and a winning threat sequence [8] consists of different threat. So, the agent needs to focus on the space of all threats. The main idea of threat-space search is instead of choosing all defensive moves for the agent, they allow to make all possible moves against their opponent at the same time. The agent can eliminate some moves that is impossible for them to reach the winning threat, so our agent can simply reduce the search space to a space of attacking moves and make our agent run faster. Figure 6 shows the idea of a conventional threat space search [7]. Since our project is only focus on five-in-a-row, the line 3 of this algorithm should be (if own-side can connect five or 2 threats).

```
procedure THREAT-SPACE-SEARCH(attacker)
    if defender has a win or a threat then
        return FAIL
    end if
    if own-side can connect six or create 3 threats then
        return SUCCESS
    end if
    Movegen(list)
    if list is empty then
        return FAIL
    end if
    result ← FAIL
    for Every move in list do
        Move()
        result ← Threat-Space-Search(attacker)
        Undo()
        if result = SUCCESS then
            return SUCCESS
        end if
    end for
    return FAIL
end procedure
```

Figure 3: threat space search

4 Experiment Design and Problem Focusing

4.1 Different Threats in Go-Moku

We are going to implement this game by using a new mathematical technique named threats [8] as our heuristic value for this game.

There are, in total, three distinct types of threat in this game. A winning threat [3] appears when a player can win the whole game in the next move. This includes straight four (Figure 4) which means there are four consecutive pieces either in horizontal, vertical and or diagonal position.

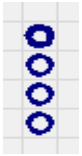

Figure 4: Straight Four

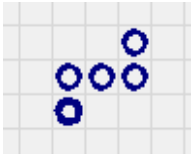

Figure 5: Double three in row

Non-Refutable threats [3] can happen when a player can definitely win a game within a limited move(s). This includes double-three-in-row (Figure 5) and straight four (Figure 6).

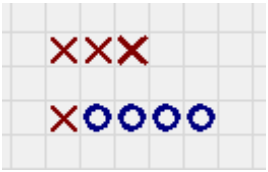

Figure 6: Example

Refutable threats [3] are made when a player tries to attempt to win the game, but its not assured to win. This threat forces your opponent to either take defensive moves, to stay in the game, or create more powerful threats in order not to lose. Examples are on the next page, four in row (Figure 7) and broken three (Figure 8).

In the example above (Figure 6), if it is blue's turn blue should consider to win this game by using the straight four condition instead of making a defensive move since the opponent can block to win and create a winning threat for themselves.

Then, we will assign a specific weight to each threat and use the formula listed below to calculate the total weight as our heuristics value for our agent:

$$\begin{aligned} EVAL = {} & \omega 1 \times \#\,straight\,four + \omega 2 \times \#\,double\,three + \omega 3 \times \#\,Broken\,three + \\ & \omega 4 \times \#\,four\,in\,row + \omega 5 \times \#\,three\,in\,row + \omega 6 \times \#\,two\,in\,row + \omega 7 \times \#\,single\,marks \end{aligned}$$

[3]

Then, using this equation to calculate the total weight for each move. Clearly, the winning threat should be assigned the largest value, and the non-refutable threat has the greater value than the refutable threats. Finally, we plan to use mini-max search algorithm with alpha-beta pruning to determine which move is the best action for the agent in each stage of the game.

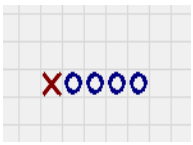

Figure 7: Four in row

Figure 8: Broken three

4.2 The fairness of the game

Some of the professional players that play Five-in-a-row claim that the first person to move has the advantage to win the game. Therefore, the fairness of Five-in-a-row is a major issue for players even though there is no proof from existing Five-in-a-row programs that can show that the above statement is true. However, this game is known to favor the player who goes first over the player who goes next if there are no certain restrictions. The definition of fairness [5] of this game is given: A game is considered fair if it is a draw and both players have a roughly equal probability of making a mistake. Its clear that in the game of five-in-a-row, the second player always has the higher probability to make a mistake [4]. Since the first player can place a stone randomly at the beginning and the second player needs to make his or her choice according to the first players move. As the game goes on, the second player usually needs to make a defensive move if the first player does not make any mistakes. In order to reduce this unfairness, some new rules can be added and they can restrict the first player to make some certain moves like double three and double four. However, adding more rules will make agents harder to implement a search algorithm for the game [13]. The second way to limit the first player to have a higher chance to win the game is to reduce the game board since some professional players argue that a larger board increases the first players probability to win the game. Moreover, a smaller board reduces the complexity of the game. Consequently, it becomes easier to solve the game [13].

In the experiment results section, a graph is created and it contains the different winning probabilities for the AI that move first with the same search depth and our goal is try to find the best rule to make both players to have an equal probability to win a game.

4.3 Threat-Space-Search

First of all, we make the initial game. The first four movements are based on a database of opening game Go-Moku. Then, we are looking for opportunities to create a winning threat. The order of searching threats is important. Threats which in turn can be created are as follows: If we can find such an opportunity to create a threat but it can be saved by the opponent, then another winning threat need to be created. In this case, we can win if we produce a double winning threat sequence or four in a row. If we cannot create a winning threat sequence, we then need to search for another winning threat sequence. The following is a way that we implemented our threat-space-search algorithm.

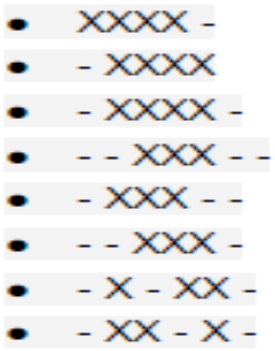

Figure 9: threat space search

When our agents turn to their movement:
ATTACK: Agents are looking for the sequence '- X X X X' or 'X X X X -'. If they can be found then the agents can win.

DEFENSE: Our agent will look for the sequence '- O O O O' or 'O O O O -'. If these sequences are found then they need to make a defensive move to block the opponents move. If the sequence is - O O O O - agent will claim it has lost.

ATTACK: Our agent will search for unbroken winning threat sequences. If our agent can find this, then they can keep attacking to win the game.

DEFENSE: If agent do not stop the opponents threat then agent will lose.

ATTACK: If it is impossible for agent to create winning threat sequence(s), then try to explore another way to create a winning sequence.

DEFENSE: Our agent will look for some threats that has potential to cause them to lose. If there are any, then they must stop the attack, preferably in such a way to create a threat.

ATTACK: Agents can move to a closer area that will enable them to create a winning threat sequence. The best places to move to are in the intersections between potential opponent squares or to line up next to them. This movement must be done in a way that it places the opponent sufficiently close to the top of the box occupied by them.

The following is the way our agent to create a winning threat sequence:
First, they look for opportunities to create a winning threat sequence
All such possibilities are listed above.
For each potential winning threat sequences, they must then simulate the movements of the opponents defences to vaporise their threat.
After this, our agent double check each potential winning threat sequence against each possible defence move. For every threat they must know whether or not it is part of a sequence of threats

that will lead to a win.

If this is the case then our agent create a winning threat sequence.

5 Experiment Results

5.1 Fairness

As we mentioned in section 4.2, fairness has been a major issue for Go-Moku. In the free style of rules ,without any restriction on the first player, the game favors the first player. In practice, based on our experiments on our agent, we have also concluded that the first player always has a bigger chance to win the game regardless of how many search depths we applied on our agent. Figure 10 has shown the fact that the first agent in our game always had the larger probability to win the game. Both of two players have the same search depth in this experiment. However, we found a way to change the rule of Go-Moku to make our game become fair by using pie rule [12]. The pie rule, sometimes called swap rule, is a rule used to balance abstract strategy games like Go-Moku where the first-move advantage has been demonstrated. By using pie rule, we can modify this game as follows: after the first players first move, we can give the second player a choice. He or she can either choose to remain his or her role of the second player or to switch places, in such a case the second player actually becomes the first-moving player. Then, he or she can make his or her first move. After that, this rule can apply throughout the whole game until one of the players can win the game. To prove this rule can ensure both players have the same chance to win, we can assume that the first player makes a first-move and guarantees one a win. For example, he or she places a move to make a straight four. The second player can simply choose to switch his or her place and take the winning position to win the game. Moreover, the first player cannot make a bad move since the second player also can remain in his or her place. For instance, the first player makes a move and created a double three. In order to not lose the game for the second player, he or she can simply remain in his or her place and not to create a straight four position, so the first player still needs to make a move to put him or her in a tie position again. Therefore, the best strategy for both players is to make it into a position where both players can tie with other player, thus a fair game. If the second player choose to switch, he or she can only make a move to tie the game. This rule satisfies the definition of fairness which is both players have the equal probability of making a mistake.

Figure 10: The Chance of Winning

5.2 Mini-max and Alpha beta, Depth of the search

In this search game, the deeper the depth that agent has searched, the better solutions(A threat that can lead agent to win the game) can be found compared to smaller depths used. The reasoning for this is because the higher the depth the deeper the agent can search which can help the agent to detecting other threats and better moves for itself. The one downside to using a larger depth for searching is that it can take a longer time to calculate. When using a low value for the depth of the search the agent will not get the best results because it will not able to see what other possible threats or winning positions may occur. The larger the depth of the search the more search space it has to go through and thus taking more time to calculate every position's value. The use of mini-max will determine the best move available from those values obtained from the evaluation function refer to the end of Section 4.1.

In our experiment when we will be using the standard 14 by 14 game board (15x15 intersections). We assigned a agent to a search depth of two and matched it against another agents of search depths from two to seven. For example, agent of search depth two vs search depth two and search depth two vs search depth three until the search depth of seven. The agent that had a search depth two would gradually lose more if the search depth was higher than the search depth of two as shown below in Figure 11. As the agent reaches the limit of its depth for every possible position it can make it is then uses the evaluation function for all possible positions found. The mini-max tree will prune according the the numbers generated from the evaluation function. It was interesting to see that the second agent at depths four and five were similar. The ending results for depth two vs depth four and depth two vs depth five were returning the similar or the same end game board because both depths were pruning the same thing.

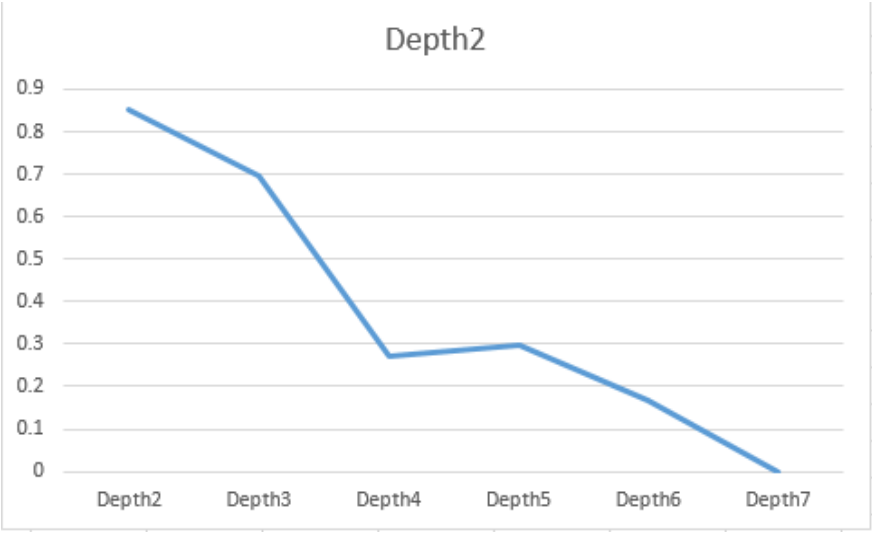

Figure 11: Depth of the search

5.3 The result of Threat-Space-Search

Unfortunately, the implementation attempted for this algorithm did not completely succeed. There are situations when the program makes "bad" moves. This depends on the implementation of some other errors and not the Threat-Space search algorithm. I'm not exactly able to determine what the problem is but it may possible that our agent is not able to detect another "special" kind of threat within the game that we do not know of. If the agent is not able to detect this special kind of threat then the agent would fail to make a correct move for Threat-Space search algorithm.

6 Conclusion and Future Scope

6.1 Conclusion

The content of this paper is summarized as follows.

- This paper introduces the game of Five-in-row and analyze the fairness of Five-in-row and claim the player who plays first has the higher probability to win the game. We also showed some methods to make this game more fair for both players, especially the pie rule which satisfy the definition of fairness which is both players have the equal probability of making a mistake .
- Multiple methods are presented to limit the search space of Five-in-a-row since the search space for the game would take a tremendous amount of time to find the best move for every move.
- The use of Mini-max and Alpha Beta pruning worked well in this experiment but could use some improvements. The larger the depth search the more calculations and pruning may needed to be done. It was shown that the agent with a higher depth search will have a higher chance to winning the game if the opponent has a lower depth search.
- The paper present different threats in the game and suggest a threat space search to play Five-in-row games and show an algorithm for Go-Moku, based on the search space search. Moreover, using threat-space search finds winning lines of play based on threats in averagely 0.1 CPU seconds which is a very efficient way to solve this game [8]. Unfortunately for us, we can't implement threat-space search flawless due to the time limitation.

6.2 Future Scope

- Sorting the mini-max tree after evaluating each position would have made pruning more effective than pruning a tree that is not in order.
- The agent could use sort of learning technique like implementation of the history heuristic to keep track of patterns to help the agent learn from its previous mistakes.
- We could make our agent search faster by improving the algorithm that the agent uses. For example, as stated earlier in the paper the use of the history heuristic and killer heuristic can be used for our future project. We can also try to speed up our agent by combining some heuristics that have the better performance or adjusting the ordering of heuristics.

- We plan to implement the pie rule to our agent and to show this game will always tie if two players follow the pie rule very carefully.
- We need to figure out what's wrong with our implementation of threat-space search and make it work flawless.

References

[1] S. G. Akl and M. M. Newborn. The principal continuation and the killer heuristic. In *Proceedings of the 1977 annual conference*, pages 466–473. ACM.

[2] T. Cazenave. Iterative widening. In *IJCAI*, pages 523–528.

[3] J.-H. Chen and A. X. Wang. Five-in-row with local evaluation and beam search, 2004.

[4] D.-Y. H. Chen Wu and H.-C. Chang. Connect6. *ICGA*, 28(4):234–241, 2005.

[5] U. J. a. R. J. Herik, H. J. van den. *Games solved: Now and in the future.Artificial Intelligence,*, volume 134. 2002.

[6] V. Kulev and D. Wu. Heuristics and threat-space-search in connect 5, 2009.

[7] Q. Liu. A defensive strategy combined with threat-space search for connect6. 2010.

[8] J. v. d. H. M. H. L.V. Allis, H.J. van den Herik. Go-moku solved by new search techniques. *COMPUTATIONAL INTELLIGENCE -OTTAWA-*, 12(1):7 – 23, 1996.

[9] T. A. Marsland. A review of game-tree pruning. 9(1):3–19.

[10] J. Schaeffer. The history heuristic and alpha-beta search enhancements in practice. 2011.

[11] K. L. Tan, C. H. Tan, K. C. Tan, and A. Tay. Adaptive game AI for gomoku. In *Autonomous Robots and Agents, 2009. ICARA 2009. 4th International Conference on*. IEEE.

[12] Wikipedia. Latex — wikipedia, pie rule, 2016. [Online; accessed 11-May-2016].

[13] C. Wu and D.-Y. Huang. A new family of k-in-a-row games, 2006.