Comparisons Among Different Gomoku AI Approaches

Project 1 Report of

CS303 Artificial intelligence

Department of Computer Science and Engineering

BY

Ting Sun

11710108

December 2019

Table of Content

1 Preliminaries

Gomoku, also called Five in a Row or Gobang. It is a kind of traditional chess game with a long history and is welcome in many countries. Its board are usually use 15×15 of the 19×19 grid intersections[1]. The game is known in several countries under different names.

Players alternate turns placing a stone of their color on an empty intersection. The winner is the first player to form an unbroken chain of five stones horizontally, vertically, or diagonally.

In this report, I will describe and compare several detailed algorithms I used in the project.

1.1 Problem Description

The task for Artificial Intelligence for gomoku is to find the most important position which has no piece before in the chessboard. The thought of the algorithm is simple but deep and it is not hard to find that we can achieve it by searching. Limited by the rule of this project, the AI should finish every consideration in 5 seconds and the total time for all the game is 180 seconds. The oracle is the definition of "importance" and time cost.

1.2 Problem Application

The artificial intelligence for gomoku is not only a way to entertain users, but also can show the power of artificial intelligence and emphasize the importance of AI.

Additionally, students can enhance the skill for adjusting parameter and they can learn various algorithms about gomoku. They may further get hang of these algorithm using heuristic learned by machine learning methods on data containing matches of human players.

This algorithm will be used to create the first competitive player of the game gomoku.

1.3 Software & Hardware

This project is written in Python3.7 with editor Visual Studio Code.
The main testing platform is Windows 10 Professional Edition (version 1803) with Intel® Core™ i5-8250U @ 1.60GHz 1.80GHz.

2 Methodology

2.1 Representation

1) blank_list:

List of all the positions that have no piece on.

2) get_blank_list():

A function that push all the empty positions into the blank_list.

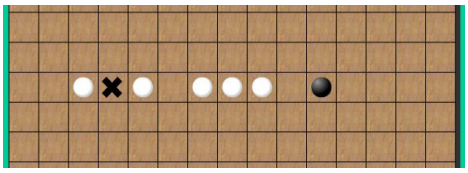
3) evaluation(x,y):

To evaluate the value of a position (x,y).

4) Type_tuple:

Use 7 numbers to show what the type looks like. The output type_tuple is (left_space2, left_length, left_space,centre_length, right_space, right_length, right_space2), where the centre_length means the pieces number that just close to the very point.

For example, to evaluate the node marked by black cross in the picture, we first get its type tuple: (left_space2, left_length, left_space,centre_length, right_space, right_length, right_space2)=(0, 0, 2, 3,1,3,0).



5) get_type(x, y, direction_x, direction_y) :

Require to input an index of a point and the wanted direction, and then output the type_tuple.

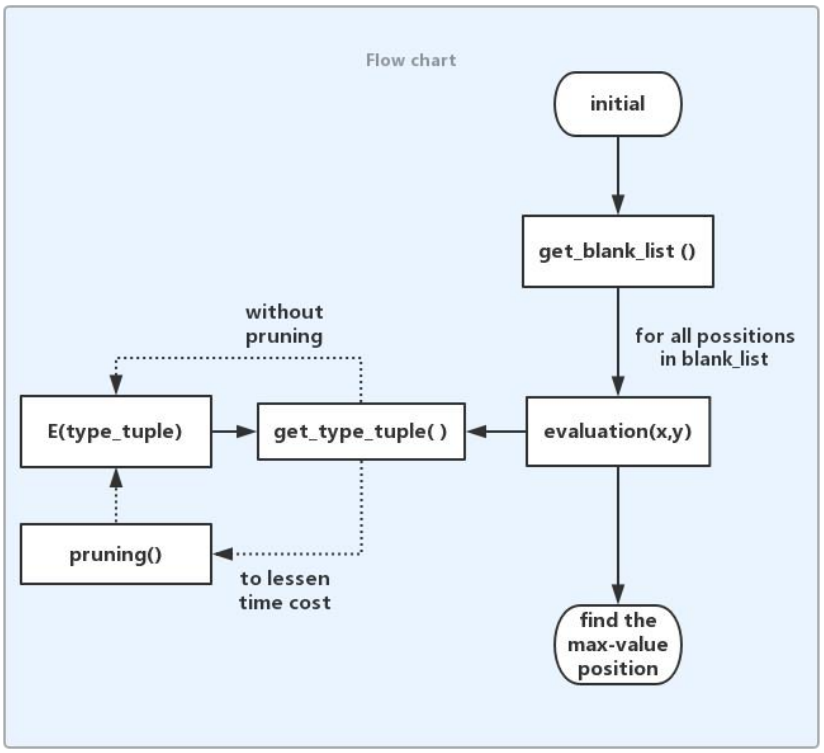
6) evaluation(type_tuple):

To evaluate the value of type_tuple.

7) Candidate_list:

candidate_list stores the valuable positions, last of which stores the candidates for the next points the AI will pose a piece and the last one in it is just the one that will be chosen as the next point by limitation of this project.

2.2 Model Design



First, the program calls get_blank_list() to get the blank_list. There may be adjacent-first pruning in get_blank_list().

After that, it uses evaluation(x,y) to evaluate the positions. If I use single-step search, then in the evaluation(x,y), I use get_type(x, y, direction_x, direction_y) to obtain the pieces types in the blank_list and call E(type_tuple) to assign a score to the type_tuple, the AI will add up all the scores. Else, I will use alpha-beta pruning. Finally, it will chooses the most valuable positions and put them into the candidate_list.

2.2.1 Pseudo-code

```
AI( ): return (x,y)
  find all empty positions to generate blank_list
  value_list=list[ chessboard_length][ chessboard_length ]
  for (x,y) in blank_list:
    type_list= get_type((x,y))
    for type in type_list:
      e= E( type )
      value_list[x][y]+=e
  return the most valuable position in value_list
```

2.3 Details of Algorithm

I developed two approaches for the evaluation function in this project. The first one is using dict(means dictionary) in Python, while the other one is using if-else clause, and both of them are clear and easy to understand.

2.3.1 get_blank_list()

This function has several editions to adapt different conditions. The simplest one is just select all the empty positions in the chessboard. However, this will bring disaster if we decide to use consideration for multi-depth. So I developed some optimized function for get_blank_list().

2.3.1.1 Adjacent-prior Pruning

According to experience and analysis, positions which are near the previous pieces are more likely to be choose, especially when the positions are near to the just previous piece. In this project, I used lambda expression to sort the blank list, which is the container for all the blank positions.

2.3.1.2 Adjacent-only Pruning

This means the chance left for these positions that far away from previous pieces is 0. We will ignore these no-nearby-pieces positions at the beginning. We can use convolution function in scipy to achieve that, or just iterate all the nearby positions to check whether there is a piece on it.

2.3.2 evaluation(x,y)

This function is designed to evaluate all the positions in blank_list, and again we have several approaches to implement it.

2.3.2.1 Single-step Search

Single-step search calculates values of positions on the view of black hand as well as white hand just once. It is intuitive that multi-step search is more advanced and should be far more intelligent than single-step search. However, one significant detail is that in this project many top coders used single-step search approach.

The advantage of single-step search algorithm is high speed and thus we should not care about optimization towards the algorithm for lessening the time cost. Omitting all the optimization, the two-step search can reach to as many as 225*225*225= 11390625 and three-step search can reach to a even larger number 225*225*225*225= 2562890625 and this will result in a disaster and by no means such a poor algorithm can pass the example tests.

The accuracy should be discussed here, for there is a danger that the single step search algorithm is too weak to be called intelligence. Yet, many top coders' using single-step search shows that the accuracy of this model can be guaranteed if your parameters setting is appropriate.

2.3.2.2 Multi-step Search

We are aware of that some top coders still use multi-step model. In multi-step search, there is usually a parameter to indicate the number of steps that should be considered if choose one position as our decision. To avoid the disaster described in 2.2.1, we should cut off some branches during the search. And I will share some approaches I used in this project.

2.3.2.3 Alpha-beta Pruning

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.[2]

In this project, we can use alpha–beta pruning naturally and without alpha–beta pruning we can hardly run over the algorithm if we choose multi-step search.

2.3.2.4 get_type(x, y, direction_x, direction_y)

Here I use while loop to get every numbers in the type_tuple. And first I judge centre_length, and then there comes left_space, right_space, ... , and finally left_space2 as well as right_space2 are calculated.

2.3.3 Using dict for E(type_tuple)

According to my estimation, to evaluate a pieces type roughly, we need only 5 or 6 pieces. For instance, if the to-be-evaluated point is (x,y), then we can get all the

linear tuple(can also use string) that has a length 5 or 6 pieces and contains (x,y), then we judge the values of all the tuples by using these tuples as index to access the corresponding values in the dict.

Using dict, the procedure can be very straightforward, because every type can be shown in the dict directly and they are visible. The key of the dict is the type of tuple, and the value is the weight assigned. Nothing is clearer.

Straightforward as the dict approach is, it is too rough to handle all the types. Using dict, we have to state every type and all the details should be mentioned. That means we cannot merge these similar types, even they are different just for one unimportant piece. What's more, sometimes a type with 7 or even more pieces should also be included in our considerations, with which our model can be more accurate. Luckily, all the two drawbacks can be solved in another approach for modeling: if-else clause.

2.3.4 Using If-else Clause for E(type_tuple)

In my code, the AI will judge the centre_length first, and then it will judge another numbers in the tuple and get the value.

To prevent the model from getting terribly complicated and hard-to-manage, I merged some cases. For example, if the centre_length is 5, the AI will not judge another numbers and directly evaluates the type as highest scores.

Using if-else clause, the procedure can be very delicate, and nothing will be out-of-control. We can assign scores to every specified type. While in dict approach, it's better to keep the dict not to be too huge, otherwise the dict will be messy for the absence of the logic in the dict. The precise scores distribution skills and noticeable details can be found in slides of *Zhao Yao*.[3]

The main disadvantage of using if-else clause is it will not be as straightforward as dict approach is. Yet, if the size of all the if-else clause is not very huge, this feature is bearable actually.

3 Empirical Verification

3.1 Dataset

First of all, 25 basic examples are used to test the basic correctness of my gomoku AI as all the peers did.

Besides these basic examples, I also use choose *Yixin*, which is known as the strongest gomoku AI, to play with my AI and thus I can detect the distance between my AI and *Yixin*'s.

Moreover, I used "playto" function in the online website to play to AI of my students.

3.2 Performance measure

Among all the 25 basic examples, I surely choose the pass rate as the fundamental benchmark.

Also, I collected the data that produced by my AI and *Yixin* and tried to analysis them.

Finally, I watched the contest between my AI and other students' and want to find the weakness of my AI.

3.3 Hyperparameters

The main key parameters lie in the weights of each pieces types as well as the steps of considerations. The judgement of hyperparameters choosing is results of contests.

During the observing of every steps of my AI, I could be aware of the mismatch of some pieces type and their scores and then I adjusted them.

3.4 Experimental result and analysis

I finally choose to use single-step, if-else clause algorithm to finish the task and all the tested data is produced by it.

All the test shows that the speed of single-step and if-else clause algorithm is very fast and a step always toke less 1 second. All the 25 examples are passed, which verified the fundamental correctness of my AI.

However, in the all the contest with *Yixin*, my AI never won *Yixin*, indicting the improving space of my gomoku AI. As the contest between AIs of peers and my AI, roughly my win rate was almost 30% and fail rate was 30%, rest of which is my game draw rate: 40%, and its performance is hard to improve by adjusting parameters. May be another architecture is needed.

3.5 Conclusion

In this report, I state several approaches I tested, which can be picked to combine a appropriate algorithm to solve another problems including gomoku AI and son on. I think I provide several aspects for thinking.

Also, there are some other approaches I don not implement in this report and further work can be carried out in following other pruning methods and other architectures of the algorithms, such as deep learning and so on. What's more, some traditional chess manual can also be absorbed into our algorithm and help to adjust parameters.

Acknowledgement

I would like to appreciate my teaching assistance *Zhao Yao* for her vivid illustration of this project as well as her excellent teaching skills and devotion. Besides, I am grateful for *Wang Ziqin* and *Lu Zirui*, who spared no effort to discuss with me, inspiring me to try various approaches.

Reffernces

[1] "Gomoku - Japanese Board Game". Japan 101. Retrieved from https://en.wikipedia.org/wiki/Gomoku on 2019 Oct 9th.

[2] Russell, Stuart J.; Norvig, Peter (2010). "Artificial Intelligence: A Modern Approach (3rd ed.)". Upper Saddle River, New Jersey: *Pearson Education, Inc*. p. 167. ISBN 978-0-13-604259-4.

[3] Zhao, Yao. (2019). 基本搜索之于五子棋.pdf. Retrieved from https://sakai.sustech.edu.cn/access/content/group/f6baff92-a8d1-4805-a353-c737451f5a9a/LAB/Lab1 on 2019 Oct 9th.