Comp1130 Programming As Problem Solving(Advanced)

Assignment 3 Technical Report

Fri May 25,2018

Tutorial Group:Tuesday 2-4pm

Tutor:Debashish Chakraborty

University ID:u6683369

ANU College of Engineering & Computer Science

Introduction

In this assignment, I write code for how to build an AI by using minimax algorithm in backgammon game. When I attend the tournament, I got my highest rank of 5 and ends up with 22.

How can we determine whether an AI is good or not?

Let's say that we have 3 AI of A,B and C, the results of the game are A beats B, B beats C. We can't say that A is actually a better AI than C because some counter exists. Just as what we can see on the tournament, some bot with a relatively high rank loses to a relatively low rank bot with a score of 0-3, although the probability exists in this game and you can say that the low rank bot has a good luck. But such situations are occured from the day that the tournament has started.

Hence, apart from having a such large sample size(those bots fights with each other, like 1000 times), there's no exact conclusion can be made that a bot is really better than another one, especially when the rank between them is relatively close. Also, for the evaluation function, as the constant value is involved in the function, the best way of getting a evaluation function is to have a really fast computer to deal all the data(compete with all bots thousand times to get a value which has the best win ratio). Even if two students use the same method, the evaluation function that they write can have a big impact on the program's performance(efficiency is always related to the complexity of the evaluation function, win ratio depends on how good your evaluation function is).

LegalMoves Bot

In legalmoves bot part, first we need to generate a list of all possible states after a single move, so we map performsinglemove to legalmoves of the state, hence get a list of states. Then we map all the possible moves with all possible states, the reason why I am doing this is to get moves straight after I find the best state for me so that I can use it in makeMove function.

Data structure of tree

Then I am going to instruct the data structure of the tree, in this assignment I use the data structure of rosetree which is similar to what we did in Week 8 lab. First we can build a tree by recursing itself twice, then we can write a prune function which can generate all states to a certain depth(Which is lookahead in this assignment). Finally by call maximise function in minimise, minimise function in maximise, we can hence implicitly build a tree for the game. Finally we combine all these function together to generate a heuristic value of the best state that it can get to.

Greedy Bot

In regard to make a greedy bot, we just need to consider all the best possible move that a player can make regardless of your opponent's move and the lookahead value. (only care the current state)The reason why we are writing this is because sometimes in the tournament if I encounter with some enemy that I can't beat with my minimax, then I change my mode to greedy bot to make my score higher. Because if we look at the tournament carefully, there are two types of bot:minimax and greedy, initially all greedy bot place at the top of the rank, however after people come up with new evaluation function and use alphabeta pruning, they take place at the top instead. So in order to deal with that, as I can't make my bot smart enough to determine whether the opponent's bot is minimax or greedy, therefore I construct with a determine condition that if I am already 0-2 behind, then I switch my minimax mode to greedy bot, that might get me more scores. The concept behind that is that if my opponent uses alphabeta pruning or have better evaluation than me, there's very little

chance that I can beat them, so I would rather change my mode to greedy and bet that I am luckier than them.

Minimax Bot

To write minimax bot, first we need to figure out how it works. Minimax, as the word means, its function is to maximize the minimum gain. In this assignment, the maximum depth that we can search is only in depth 3, that's because there's two dices involves in the game as well as 24 different board in the state, that makes the sample space of all possible situations rather large and complicated, which means that every time it generate a state, if we are going to generate next state, the situation should increase dramatically. In that way, even if we use alphabeta pruning, there's no way to go deeper than a depth of 3.(you can still improve its efficiency though) In order to get to a depth of 3, first we need to get 3rd depth of the current state, then we evaluate all 3rd possible states of the current state, take the minimum of them, then go to the 2nd depth, take the maximum of states in 2nd depth. Finally return as a value.

Then, by finding the state and moves according to the minimax value, we can hence generate Moves to implement on our makeMove function.

AlphaBeta Bot

Although I didn't write alphabeta bot, but I do understand how alphabeta pruning works. Alphabeta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. The only difference between alphabeta pruning and minimax algorithm is that alphabeta pruning stores two values of alpha and beta. The algorithm updates those values constantly and if beta is less or equal than alpha then cut this node. It can improve the performance and efficiency of the AI dramatically, and as haskell is a lazy programming language, we can add a condition of beta <= alpha to filter the state list that we actually need to calculate. I have tried to write the algorithm for alphabeta pruning if you can see at the end of the PlayAsWhite.hs, although I have commented it off, but you still can see the process of doing alphabeta pruning.

Limitations

However, such a game as Backgammon depends heavily on the probability(luck), it's not a zero-sum game like tic-tac-toe or chess. You can't predict all states and moves because the number of cases becomes extremely large as you predict more into the depth. The best AI in backgammon now is TD-gammon which uses neural network and it can beats world champion in that game. So for that game I think it's better to collect a huge amount of data and construct a neural network to build an AI. But now I am not competent to build neural network, at least not on haskell.

Evaluation function

For the evaluation function, I wrote that if my score is less than my opponents', then get the score back by maximise the difference between my score and my opponents' score. If I am already in a leading position, then get to the bearing off as soon as possible.

In the evaluation function, I have tried several difference one, but just like what I mentioned before, the efficiency of your bot depends on the complexity of your evaluation function. And I have picked the one which perform best on my bot, and that's my current evaluation function, however you can check my last several evaluation function on gitlab.

Improvement on efficiency

For the improvement on efficiency, there are several ways of doing that. First we can write a simple but efficient(also powerful) evaluation function to make it more efficient. Second, obviously we can use alphabeta pruning instead of the minimax to improve efficiency. Finally, we can also nub the state list to get rid of the repeat states to reduce the time it takes to make moves.

Conclusion

Assignment 3 is the hardest assignment I have ever encountered in comp1130, but by doing the assignment, we get more understanding on rose tree and typeclasses, by writing recursion all the times, I feel I am more competent and prepared to coding.

Reference

Gérard M. Baudet, An analysis of the full alpha-beta pruning algorithm, ACM Symposium on Theory of Computing, 296–313, San Diego, California, May 1978, https://dx.doi.org/10.1145/800133.804359

Thomas Hauk, Michael Buro, and Jonathan Schaeffer, Minimax performance in Backgammon, International Conference on Computing and Games, 51–66, Ramat-Gan, Israel, July 2004, https://dx.doi.org/10.1007/11674399_4

John Hughes, Why functional programming matters, http://www.cse.chalmers.se/~rjmh/Papers/whyfp.html

Donald E. Knuth, and Ronald W. Moore, An analysis of alpha-beta pruning, Artificial Intelligence 6(4):293-326, Winter 1975 https://dx.doi.org/10.1016/0004-3702(75)90019-3

Ervin Melkó, and Benedek Nagy, Optimal strategy in games with chance nodes, Acta Cybernetica 18(2):171–192, January 2007, http://www.inf.u-szeged.hu/actacybernetica/edb/vol18n2/Melko_2007_ActaCybernetica.xml

Minimax,Wikipedia https://en.wikipedia.org/wiki/Minimax

Alpha-Beta Pruning,Wikipedia https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

Lecture’s powerpoint provided on Tony’s lecture