Scotland Yard AI

Julian Loscombe and Ben Milne

May 5, 2015

1 Game Tree

We have set up our game tree to use Alpha - Beta pruning in order to increase the depth that we can search to in the allotted time. In addition we are allowing our tree to generate throughout the game, pruning off branches of unreachable nodes as moves are made.

1.1 Iterative Depth Search

Because we have a strict time limit in order to make our move and given that the initial conditions change the complexity of the game tree, we have decided to use an iterative depth search, updating the best move on each iteration and simply grabbing the one that is available when the time limit is near.

1.2 Scoring

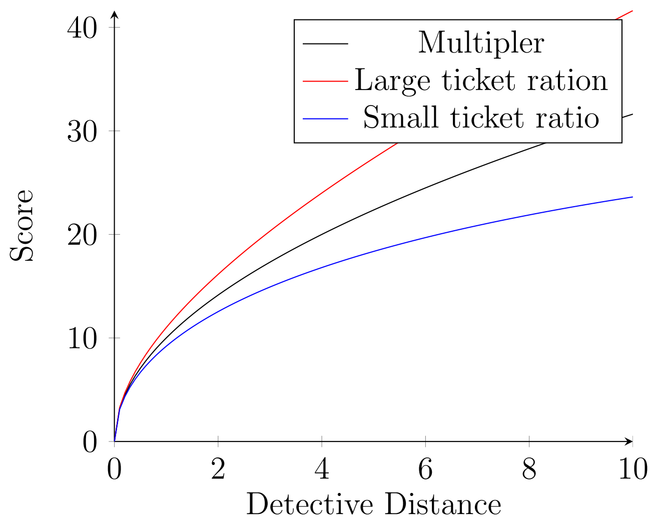
To score our game state we use three main sources of information, the ratio between the PageRank of the detectives' locations and Mr X's location, the ratio between the value of the detectives and MrX's tickets, and the average distance between the detectives and Mr X (using Dijkstra's Algorithm). We consider the average distance to be the most important factor and as such its influence is weighted much higher than the other two. Because it is particularly important to avoid very small distances we use a root function to transform the distance score so that is is much steeper when Mr X is close to being caught. We then normalise the ratios around 0 and apply them to the distance score as below:

```
double score = (10 * Math.pow(detDistance, 0.5));
       score += ((ticketRatio - 1) * kTicketInfluence) * score;
       score += ((pageRankRatio - 1) * kPageRankInfluence) * score;
```

This ensures that the average ratio will have no effect on the score and that when the distance is very small it's main objective is to increase that distance. We also added a special case to boost secret move scores when the distance was very small. The below graph illustrates part of the score function.

1.3 Pruning

Our game keeps the tree running throughout the game in order to use the work we have already done to extend the depth we are able to reach. In order to do this though, we must prune branches that are made impossible by the game advancing in order to make any real progress. We achieve this by setting the root of the tree to be the node containing the last played move, thus making unneeded nodes unreachable. However, what if we are currently in one of those



pruned branches? We would waste time evaluating nodes that are unnecessary. So, to avoid this, we perform a quick when we enter our alphabeta() function to see if we have pruned and break out if this is the case. The results of the pruning extends our search depth a few levels after which it reaches an equilibrium between the moves being pruned and the expanding search tree.

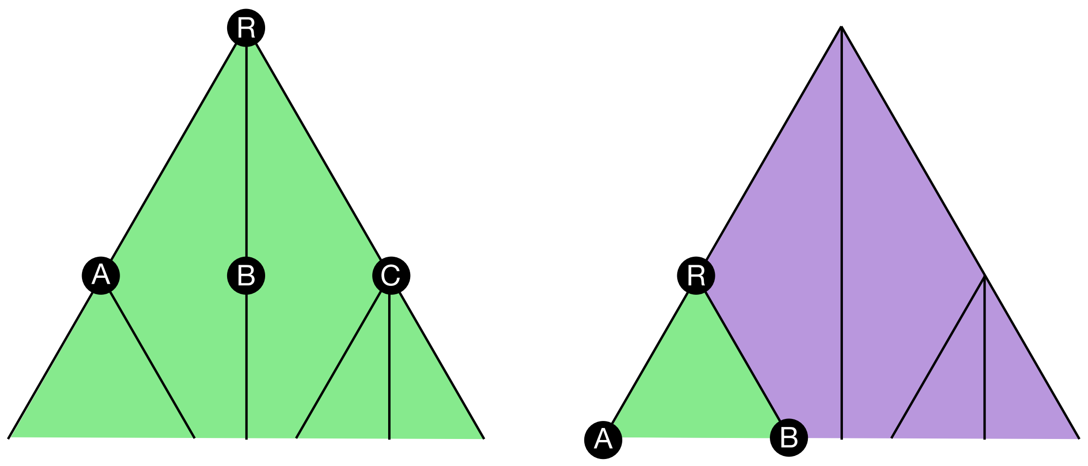


Figure 1: After making move A, we can prune all other moves and their subsequent moves.

2 Threading

3 Results

The AI we have produced has very promising results and does make smart moves. However, we are still commonly coming up against errors which we believe are a result of threading issues. This means that they are very difficult to debug and in retrospect we believe that given the short amount of time provided, it would have been wise to keep our tree simple and focus on reaching to greater search depths. This would mean that we could consider moves further ahead, something that is the hallmark of a good player.

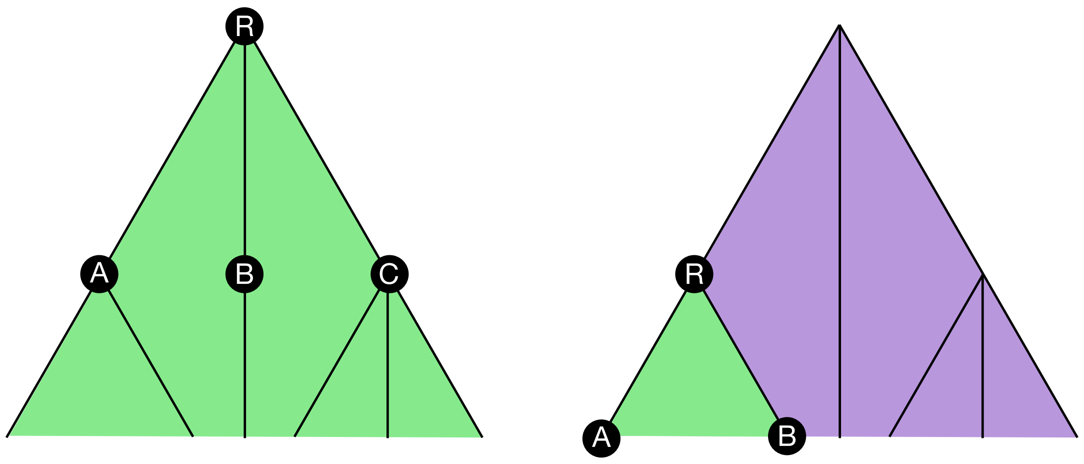


Figure 2: After making move A, we can prune all other moves and their subsequent moves.

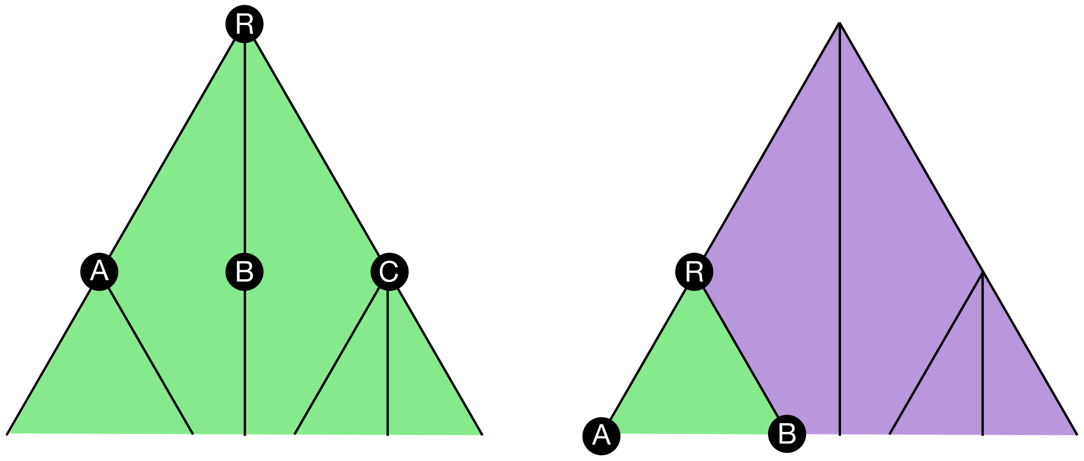


Figure 3: After making move A, we can prune all other moves and their subsequent moves.