Scotland Yard AI

Julian Loscombe and Ben Milne

April 29, 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 doing so we reduced the time to get to depth 4 from ???ms to ??? ms. In addition we are removing the double and secret moves from the game tree to further reduce the time to move through the trees. This is important because below a depth 6 search we would not even be considering some detectives moves when deciding what to do. A higher search depth also allows us to make mores with much greater foresight, something that is the hallmark of a competent player.

1.1 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 Dijkstras). 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 to that is is much steeper close to zero. 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;
```

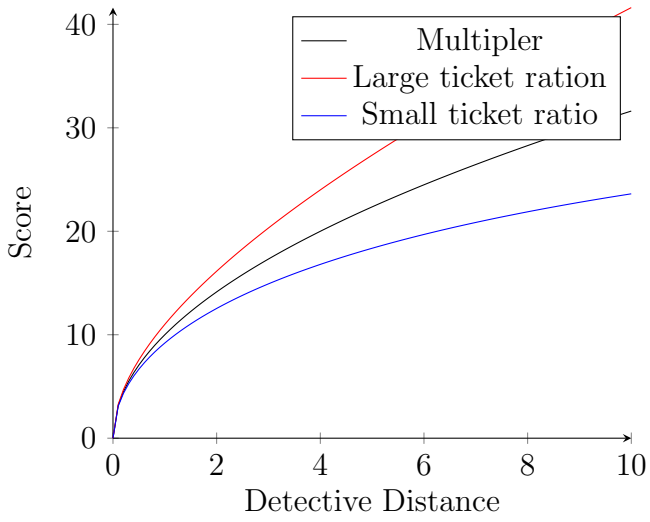
This ensures that the average ratio will have no effect on the score and that when the distance is very small it's main objective is to increase that distance. We also added a special case to boost secret move scores when the distance was very small. The below graph illustrates part of the score function.

1.2 Threads

Because we have multiple threads acting on our game tree at the same time we have had a smorgasbord of difficult to trace errors. To fix this we have ...

1.3 Iterative Depth Search

Because we have a strict time limit in order to make our move and 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 approaching.

1.4 Pruning

We are hoping to keep 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. Clearly 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 quickly traverse up the list when we enter our alphabeta() function to see if we are connected to the root node, if we are not we can simply return 0 immediately as it will make no difference to our final score.

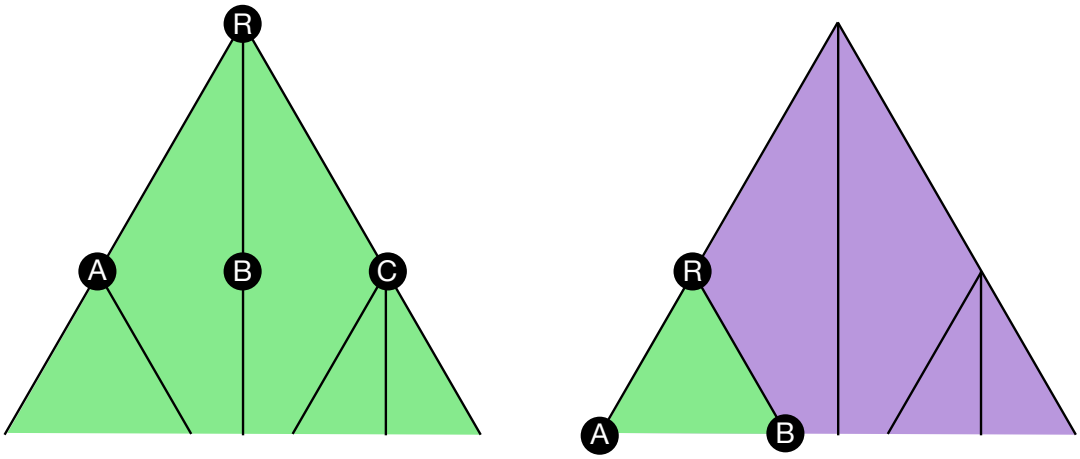


Figure 1: After making move A, we can prune all other moves and their subsequent moves.