FIT3080 Assignment 1 Report

# Backtrack

The bound on the backtrack algorithm was set to be 132. This is 110% of the highest count reached through testing. While there could still be combinations above this limit, it covers the majority of beginning puzzle states.  
The order of preference of operations, in decreasing order is:

2R, 2L, 3R, 3L, 1R, 1L

The one distance operations are the lowest priority as they do not change the order of Bs and Ws, meaning that they cannot lead to a success state. Two distance operations are the next priority due to their low cost. Performing right operations before left ones is arbitrary, but retains consistency across the different distances.

# Treesearch

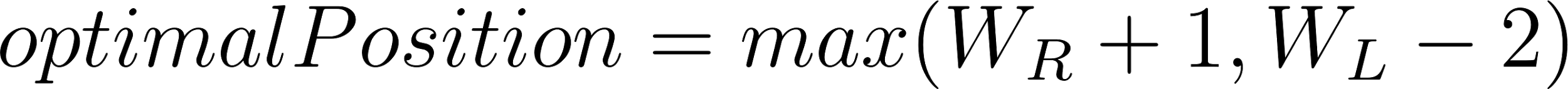
## DLS

DLS proved to be a straightforward task, only requiring a few tweaks and a (fairly simple) sorting algorithm to be added to the generic treesearch algorithm. We did not encounter any issues specific to DLS (i.e., only minor issues that were related to treesearch in general). We chose a depth limit of 55 for similar reasons as the Backtrack bound. The highest depth reached in our tests was 50, so 55 accounts for a 10% margin of error.

## Algorithm A\*

The heuristic function used consists of two parts:

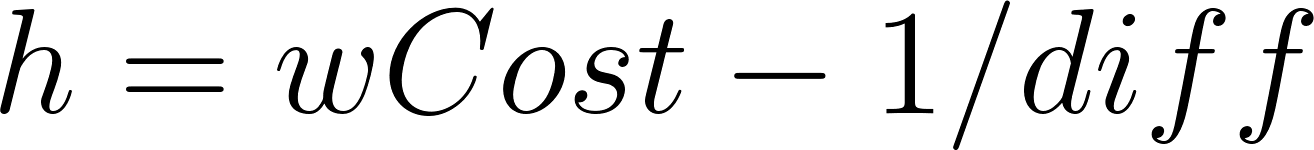
1. The cost required to get each W to the left of the leftmost B. This ends up being the sum of the amount of Bs between the leftmost B and each W (eg. BBBWWWE would have a distance of 9, as there are 3 Bs before each W). This simple algorithm is possible due to the exploitation of the fact that E can be moved to any position at no cost simply by taking 1R and 1L operations. Hence, we are able to position E in the perfect position to perform as few swaps as possible, as described in part 2.
2. An evaluation of the current position of E in comparison to the optimal position of E. This is calculated to differentiate two states of the puzzle which may have the same distance from solution as described in part 1, but where one may have E in a more optimal position (eg. BBBEWWW is better than BBBWWWE). Though it changes the value of h, the ordering of OPEN is only affected for the specific cases where the distance from solution is equal. The optimal position of E is decided using the following formula:



Where WR is the index of the rightmost W that is before the leftmost B and WL is the leftmost W that is after the first B. If there is no W before the first B, WR = -1. If no W exists after the first B, that means the state is already a success state, which means h will be 0.

The difference between the current position of E and the optimal position is then calculated. If E’s position is optimal, the difference, diff, is given as 0.9.

To keep h lower than the actual cost of reaching a success state, we decided on this final heuristic formula:



This heuristic formula is monotonic and admissible for all the test cases we ran. Hence, the algorithm implemented is A\*.

Implementing the code for the heuristic function presented many difficulties, especially in obtaining the values for our variables. Producing diagnostics at run time to show us exactly what was happening at any given step helped greatly in the debugging process of many of these issues.

# Testing

Each algorithm was tested with both boundary cases (such as invalid inputs and pre-solved puzzles), as well as a suite of five test cases. These were used to test both cost and runtime. The results of these tests can be found in the TestResults.docx file.

# Results

Algorithm A\* produced far lower costs than backtrack and depth limited search for our test cases. DLS was worse than BK for simple states, but BK was worse for more complicated ones.

Backtrack takes considerably less computational power than the other two due to the lower amount of calculations. Despite the small path costs, Algorithm A had relatively long run times due to the need to calculate h for each node. Furthermore, this negatively affected the time-efficiency of DLS as it used the same node system as A, leading it to always calculate h unnecessarily.

# Peer assessment

We worked well together, utilising a subversion repository to share our progress throughout the project. None of the complicated methods were done by just one member of the group (although some simple ones, like isSolution(), inevitably were), so both of us had to be familiar with every aspect of the program. Additionally, both members contributed to this report.  
Overall, the workload was evenly split.