Solving the 8-Puzzle

Brad Shook and **Wilbert Garcia**
{brshook,wigarcia}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

We are recreating the experimental results of Russell and Norvig (RN) in the 8-puzzle domain. We are implementing the A* search algorithm and investigating the impact of heuristic quality on search performance. In particular, we are comparing using different heuristics such as the misplaced tiles heuristic, the Manhattan Distance heuristic and the Linear Conflict Heuristic.

Our results show that our implementation of the experiment results in lower search costs represented by the number of nodes traversed for the misplaced tiles heuristic function using A* search, similar search cost using the Manhattan Distance heuristic and significantly better search cost using the Linear Conflict heuristic. In terms of Effective Branching Factor (EBF), our results show low EBF at depths lower than 8. EBF is highest in heuristic h1 and is lowest in heuristic h3.

1 Introduction

We are recreating the results from Russel and Norvig's (RN) 8-puzzle experiments. These experiments investigated the impact of heuristic functions on the efficiency of A* search on the sliding 8-puzzle.

In an attempt to recreate the experimental results of Russel and Norvig's analysis of the efficiency of using different heuristic functions with A* search, we implemented the misplaced tiles (h1), Manhattan Distance (h2) and Linear Conflict (h3) heuristic functions to solve the 8-puzzle. We have recorded the nodes traversed, the effective branching factor and the time elapsed in seconds on 100 different board states for each even depth between 2 and 24. We are comparing the search cost and the effective branching factors using these different heuristic functions on the A* search algorithm.

We are attempting to better understand the A* search algorithm by determining the relative efficiency of three heuristics while also comparing our own results to those of Russel and Norvig. We are trying to understand how the heuristic functions optimize search performance represented by minimizing the number of nodes traversed and also minimizing the Effective Branching Factor (EBF).

The experiment that we are attempting to recreate comes from *Artificial Intelligence: a Modern Approach* by Stuart Russell and Peter Norvig (Russell and Norvig 2003). The Linear Conflict heuristic function is explored in depth in "Generating Admissible Heuristics by Criticizing Solutions to Relaxed Models" by Othar Hansson, Andrew E. Mayer and Mordechai M. Yung (Hansson, Mayer, and Yung 1985).

The rest of the paper will be structured in the following format. In the next section, background will be provided that contextualize and clarify what we hope to achieve. Then, we will describe the logistics of the experiment that we performed. We will then present the results of the experiment performed followed by the conclusions drawn from the experiment.

2 Background

Before the search cost of heuristic functions in A* search can be analyzed, we must determine that the puzzle board that we have instantiated is a solvable board. Random starting configurations for the 8-puzzle do not work because the state space for the 8-puzzle consists of two disconnected components where starting from the wrong component results in an unsolvable board. To assure that the board is solvable, we have written a method that determines whether the board is solvable. The method uses the inversion counts to determine whether or not the board is solvable. Since the 8-puzzle has an odd number of columns and rows, if the number of inversions is even, then the board is solvable. If the number of inversions is odd, the board is not solvable. An inversion is when two tiles are in the reversed order of their goal state. For example, if the goal state for two tiles is [a,b], then the tiles represent an inversion if their current state is [b,a] as is seen in Woolsey Johnson's article Notes on the "15" puzzle (Johnson 1879).

The misplaced tiles heuristic counts how many tiles are not in their goal state positions. The Manhattan Distance heuristic function counts the sum of the distance of tiles from their goal positions by counting vertical and horizontal distances. The Linear Conflict heuristic addresses the pitfalls of the Manhattan Distance heuristic. According to Nilsson, the Manhattan Distance heuristic function

underestimates "the difficulty of exchanging the position of two adjacent tiles" (Russell and Norvig 2003).

The Linear Conflict heuristic attempts to find the minimum number of moves needed to resolve the conflict in a row or column thus resolving conflict in an optimal manner and better estimating the cost of switching the positions of adjacent tiles. In this heuristic function, a linear conflict is defined as two tiles t_j and t_k being in conflict if they are both in the same line and both of their goals are also in that same line and t_j is to the right of t_k and the goal position of t_j is to the left of the goal position of t_k (Hansson, Mayer, and Yung 1985). The algorithm for the Linear Conflict heuristic takes a state of an 8-puzzle board s and searches each row counting the number of linear conflicts for each tile. The tile with the most conflicts is moved out of the row. The counts for linear conflict are adjusted and the linear conflict is calculated for the row. The same process is done for columns in s. The sum of these linear conflicts for each line becomes the linear conflict for the board and this is summed with the Manhattan Distance to give a heuristic estimate (Hansson, Mayer, and Yung 1985).

3 Experiments

We wanted to determine which heuristic function is optimal in solving the sliding 8-puzzle using A* search. We are determining the efficiency using search cost which is the nodes traversed by the algorithm for each heuristic function. We are also using effective branching factor or b^* as a measure of the effective quality of the heuristic. To analyze these heuristic functions, we coded in Python. We created a Puzzle class which recreated the functionality of the sliding 8-puzzle. We also implemented a Node class that helped perform A* search on the puzzle board.

We initialize a puzzle board at the goal state. Then, we apply a specific number of moves that correspond with specific solution depths. For example, if the desired solution depth is d then d moves are applied to the board that was initialized to result in d moves to the goal. Once we have this board, we run A* search on the board using each of the three heuristic functions. We run A* search for each heuristic function on an arbitrary number of boards until we achieve 100 instances of boards at each desired depth d. In this experiment, we are measuring the number of nodes traversed which corresponds to the length of the explored set and the time to run each search is collected. This process is repeated for even numbered depths, $d = 2$ to $d = 24$. Then, we average the number of nodes collected at each desired depth, d, to calculate the EBF for A* search at d. EBF is calculated with $N + 1 = 1 + b^* + (b^*)^2 + ... + (b^*)^d$ where N is the number of nodes traversed, d is the solution depth, and b^* is the EBF (Russell and Norvig 2003). Lastly, we use the EBF and the number of nodes traversed to compare search costs and determine relative efficiency between heuristics.

4 Results

Nodes Traversed

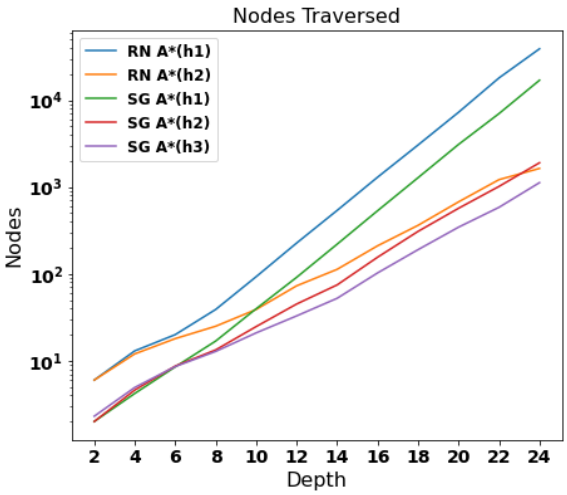

Figure 1: Comparison of search costs for our and RN's A* search with h1, h2, and h3. Search costs are averaged over 100 instances of initial puzzle states for every depth.

The nodes traversed data for A* search using each heuristic is shown in Figure 1. RN's h1 traversed, on average, twice as many nodes as our h1. Our nodes traversed values more closely related to those for the previous even depth for RN's h1 results. Contrary to the h1 results, both RN's and our h2 performed similarly for the number of nodes traversed at each depth. Our h2 still outperformed RN's at most depths except at depths 22 and 24. Overall, our h3 greatly outperformed the rest of the heuristics. This is to be expected since h3 is taking extra time to determine its next move, meaning it should be a better move, in turn leading to a shorter solution path and less nodes traversed. Any disparities between RN's and our results are likely due to a difference in what RN and we are counting as nodes traversed. RN does not clarify how they determine a node has been traversed.

Effective Branching Factor (EBF)

For EBFs, our heuristics followed vastly different trends compared to RN's. As seen in Figure 2, RN's h1 and h2 both have high EBFs at depths 1 through 8 and then level off. Contrary to this trend, our h1, h2, and h3 have low EBFs at these depths. For example, all three of our heuristics have an EBF of 1 at depth 2. The reasoning behind this gap at low depths is that our heuristics traversed a lower number of nodes at these depths compared to RN's heuristics. As expected, RN's and our h1s have the highest EBFs, then the h2s have the second highest EBFs, and our h3 has the lowest EBF. At higher depths (14 through 24), both h2s have similar, if not exact, EBFs. Since our h2 traverses more nodes at depth 24 compared to RN's h2, our h2 EBF for that depth is

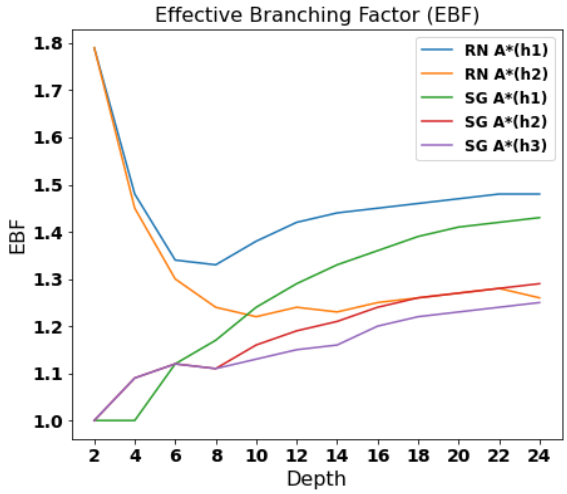

Figure 2: Comparison of effective branching factors for our and RN's A* search with h1, h2, and h3. Effective branching factors are averaged over 100 instances of initial puzzle states for every depth.

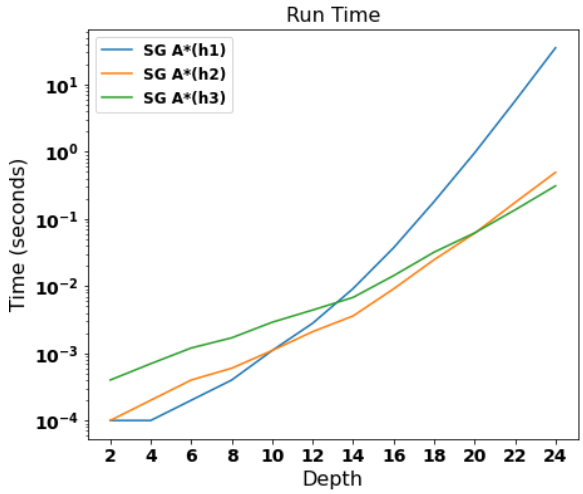

Figure 3: Comparison of run times for our A* search with h1, h2, and h3. Run times are averaged over 100 instances of initial puzzle states for every depth.

higher. Interestingly, our h1 produces lower or at least equal EBFs compared to the other heuristics at depths 2 through 6.

Run Time

The average run times of A* search using each of our heuristics is shown in Figure 3. As predicted, h1's run time exponentially increases with respect to depth. In contrast, h2 and h3 both ran much faster, with run times not exceeding 1 second, for depths 24 and under. At depth 24, h1 runs 72 and 113 times slower, respectively, than h2 and h3. At depths up to depth 12, h3 runs slower than h1 and h2. This is due to the extra calculations that h3 is performing to determine which nodes to look at next. As the depths increase, however, h3's extra calculations increase performance over h1 and h2.

5 Conclusions

In this paper, we were interested in investigating the efficiency of three different heuristic functions using the A* search algorithm to solve the 8-puzzle. We found that the Linear Conflict heuristic (h3) optimized efficiency both in number of nodes traversed and in EBF.

According to our results, the Manhattan Distance heuristic (h2) outperformed the misplaced tiles heuristic (h1) in terms of both number of nodes traversed and EBF. Though our results varied from the experimental results of Russel and Norvig, we have drawn similar conclusions and further asserted the dominance of the Linear Conflict heuristic in terms of search cost performance of the A* search algorithm on the 8-puzzle.

Avenues for further investigation can be to recreate our experiment and run for greater values of desired solution depths. Other avenues of further investigation would be to recreate our experimental results but also implement the n-Swap and n-Max Swap heuristic functions to determine the relative search cost performance pertaining to the A* search algorithm.

Further reading on the n-Swap and the n-Max Swap heuristic functions can be found in "Implementation and Analysis of Iterative MapReduce Based Heuristic Algorithm for Solving N-Puzzle" by Rohit P. Kondekar, Mohit Modi, Akash Gupta, Parag S. Deshpande, Gulshan Saluja, Richa Maru and Ankit Rokde (Kondekar 2014).

6 Contributions

BS and WG practiced paired programming for all coding. WG set up the board representation and the finding of the next states. BS implemented a function to check for the solvability of the board. BS also implemented the h1 and h3 heuristic functions. WG implemented the h2 heuristic function. WG implemented A* search. BS ran the experiments and collected data.

WG wrote the introduction and background sections. BS wrote the Experiments and Results. Additionally, BS prepared figures. Lastly, WG wrote the Conclusion and Contributions. Both authors proof-read the entire document.

7 Acknowledgements

We would like to thank Dr. Raghu Ramanujan for insight and guidance throughout the process.

References

Hansson, O.; Mayer, A. E.; and Yung, M. M. 1985. Generating Admissible Heuristics by Criticizing Solutions to Relaxed Models.

Johnson, W. 1879. Notes on the "15" Puzzle. *American Journal of Mathematics* 2(4).

Kondekar, R. P. 2014. Implementation and Analysis of Iterative MapReduce Based Heuristic Algorithm for Solving N-Puzzle . *Journal of Computers* 9(2).

Russell, S. J., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education.