The Eight Puzzle and Performance of Various Heuristics

COMP.5430 Artificial Intelligence

Brendan Hertel, Bhanu Kanumuri

November 2022

Abstract–In this paper we examine the eight puzzle, a game played on a grid where a goal state must be reached from some start state by swapping tiles with a blank spot. We examine several different search methods for the eight puzzle, including uninformed (breadth-first search) and informed (greedy best-first search and A* search) methods. For informed methods, we use several different heuristics proposed in previous literature, as well as develop a novel heuristic and analyze the performance of all heuristics.

1 Introduction

In playing a game or solving a puzzle, the player must make several decisions, usually guided by some evaluation of how that decision progresses towards a goal. One particular interest is single player games, where the player does not have to worry about an adversary interfering with a path to the goal. In this situation, the objective is often to reach a goal as fast as possible through making optimal decisions. One such example of a game of this nature is the eight puzzle. The eight puzzle, although small, is difficult to solve optimally and efficiently. We examine different search methods as well as different heuristics (functions used to guide the search) for the eight puzzle.

2 The Eight Puzzle

The eight puzzle is a game played on a 3×3 grid as shown in Fig. 1. There are eight tiles, numbered 1 through 8, and a single blank, represented as 0. The goal of the game is to get from a start state (i.e., the state shown on the left in Fig. 1) to a goal state. In this paper, the goal state will always be the one shown on the right in Fig. 1. This is done through a series of actions where only the blank is moved. At different points on the grid, the blank can be moved such that it is swapped with the tile above, below, to the left, or to the right of itself. Previous work [5] has shown through computing all possible start states the length of every optimal path to the goal state, with the longest a length of 31.

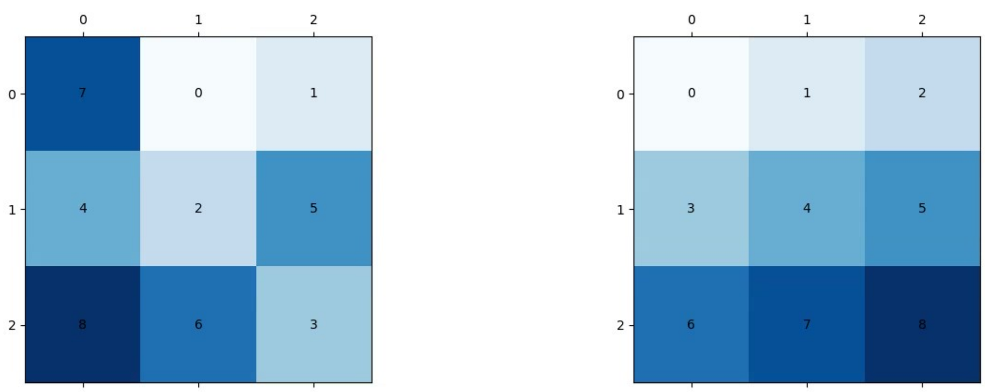

Figure 1: Example of starting position (left) and final position (right) of the eight puzzle. Tiles are denoted by number and color.

3 Search Methods

The solution of many problems is a path from an initial state towards a goal state, where the path consists of all in-between states. This path is found through searching, where the children of each state, starting with the initial state, are explored, looking for the goal. If no goal is found, the next state is explored, and so on, until a goal is found. Nodes to be searched are stored in a priority queue, Q, which is sorted according to weights w. The general psuedocode for search is shown in Algorithm 1. Search methods differentiate themselves through the order in which nodes are explored (how $u.w$ is determined). We examine three different search methods in this paper.

Algorithm 1: Search

```
Input: Initial State s, Goal State g, Rules R, Key Function f
Output: Path of States P
s.w = 0
s.π = NULL
Q = NULL
ENQUEUE(Q, s)
while Q is not NULL do
    u = DEQUEUE(Q)
    if u is g then
        P = GENERATE-PATH(u)
        return P
    for r ∈ R do
        c = APPLY(r, u)
        if c is not previously visited then
            c.w = f(c)
            c.π = u
            ENQUEUE(Q, c)
```

3.1 Breadth-First Search

Breadth-first search (BFS) is an uninformed search method, meaning that all states are treated as equally worth exploring, and the order they are explored is the same as the order added to the queue. For BFS, it is unnecessary to use a priority queue, as a simple first-in-first-out (FIFO) queue will order nodes correctly. However, given a priority queue, the weight of a node u can be determined as

$$f_{BFS}(u) = u.\pi.w + 1 \tag{1}$$

where $u.\pi.w$ is the weight of the parent node. This weight corresponds to the depth of a node, such that in a minimum priority queue, all lower depth nodes are explored before searching the next depth level. This ensures a complete search, but BFS may be slow for large search trees. In order to more efficiently explore the search space, other informed search methods may be used.

3.2 Greedy Best-First Search

Greedy best-first search (GBS) [6] is an informed search method, and uses a heuristic to evaluate which nodes are explored next. In GBS, nodes closest to the goal are expanded first. Using a heuristic function that evaluates states closer to the goal as lower and a minimum priority queue, the weight of nodes is determined as

$$f_{GBS}(u) = h(u) \tag{2}$$

where h is a given heuristic function. For more information on heuristics in the eight puzzle, see Sec. 4. This search expands nodes closest to the goal first, and therefore does not explore, which may lead to finding a path which is not optimal. However, GBS does not broadly explore and therefore tends to expand less nodes than other search methods.

3.3 A* Search

A* search [6], similar to GBS, is an informed search method. Unlike GBS, A* takes into account the length of the path already traveled to a state, rather than only the heuristic of a state to the goal. Determining weights in A* can be done by summing the heuristic and the distance already traveled, as

$$f_{A^*}(u) = u.\pi.w + h(u). \tag{3}$$

Note that A* is the same as BFS for $h(u) = 1$. This prompts broader search into paths which may be farther from the goal, but have not traveled a long distance. A* will always find the optimal path to the goal for admissible heuristics [2], but explores more than GBS.

4 Heuristics

Heuristics are functions which evaluates the "goodness" of a state. Heuristics are used to guide search algorithms such that the amount of exploration is less than uninformed search methods. In search, a heuristic provides an estimate of the distance to the goal, and a heuristic is admissible if it never overestimates the cost to the goal [6]. We investigate several heuristics previously proposed for the eight puzzle, as well as introduce a new heuristic, subdivided sequence.

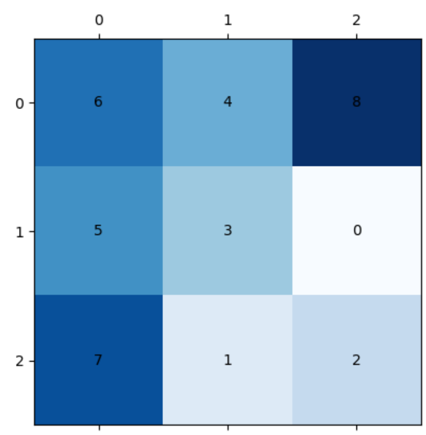

Figure 2: Example of a position of the eight puzzle.

4.1 Misplaced Tiles

The misplaced tiles heuristic, h_{MP}, counts the number of tiles which are in a different location than in the goal state [6]. Note that the blank is not counted, only tiles. This can be formally defined as

$$h_{MP} = \sum_{i=1}^{8} p_i, \tag{4}$$

where

$$p_i = \begin{cases} 0 & \text{if tile } i \text{ is correctly placed} \\ 1 & \text{otherwise.} \end{cases}$$

This guarantees that $h_{MP}(g) = 0$ and $h_{MP}(P[-1]) = 1$, where $P[-1]$ is the last state before the goal in the found path P. This heuristic is admissible, as it will always take more or equal actions to move tiles to proper positions than the number of misplaced tiles. Additionally it is equivalent to solving a simplified problem of removing the constraint of tiles only swapping with the neighboring blank, but instead being able to swap with the blank anywhere on the grid. For the example shown on the left in Fig. 1, $h_{MP} = 7$, as all tiles are misplaced except for 5, and for the example shown in Fig. 2, no tile is correctly placed, so $h_{MP} = 8$.

4.2 Manhattan Distance

The Manhattan distance heuristic, h_{MD}, sums the Manhattan distance for each tile from its current position to its goal position [6]. This is the number of actions it would take to solve the puzzle if a tile could be swapped with any other tile, not just the blank. As this heuristic solves a simpler version of the problem than the actual problem, the heuristic is admissible. This can be formalized as

$$h_{MD} = \sum_{i=1}^{8} \Delta x_i + \Delta y_i, \tag{5}$$

where Δx_i and Δy_i are the x and y distance between the current position and goal position of tile i, respectively. In Fig. 2, $h_{MD} = 13 = (0+2)+(0+2)+(1+0)+(0+1)+(2+0)+(0+2)+(1+0)+(0+2)$.

4.3 Sequence Score

The sequence score heuristic, h_{Seq}, evaluates the ordering (sequence) of tiles in the eight puzzle [4]. This heuristic aims to evaluate how "lined up" tiles are rather than their proper positions. The sequence score is found by evaluating if there is a tile where the blank should be, and if the other tiles are in the correct sequence. This can be shown as

$$h_{Seq} = n + 2m \tag{6}$$

where

$$n = \begin{cases} 0 & \text{if blank is in goal position} \\ 1 & \text{otherwise} \end{cases}$$

and m is the number of mismatching sequences. For example, the goal state in Fig. 1 has sequences $\{(1,2),(2,5),(5,8),(8,7),(7,6),(6,3),(3,4),(4,1)\}$, and the state shown in Fig. 2 has sequences $\{(4,8),(8,0),(0,2),(2,1),(1,7),(7,5),(5,3),(3,4)\}$. Since none of the eight sequences align, $m = 8$, and since there is a non-blank tile in the blank goal position, $n = 1$, so $h_{Seq} = 1 + 2 * 8 = 17$. It should be noted that sequence score is *not* admissible, and overestimates the cost to the goal. Additionally, this heuristic is proposed by Nilsson [4] to be used in combination with Manhattan distance, and using

$$h_{Nilsson} = h_{MD} + 3h_{Seq} \tag{7}$$

is suggested.

4.4 X-Y Distance

The X-Y distance heuristic [3], h_{XY}, evaluates separately the number of row swaps and column swaps needed in order to achieve the same row/column alignment as the goal. To find this the formulation

$$h_{XY} = x_{swap} + y_{swap} \tag{8}$$

is used, where x_{swap} and y_{swap} are the number of row and column swaps, respectively. To find these values a breadth-first search is used, introducing significant overhead. Although overhead time is not reported in Sec. 5, using the X-Y distance heuristic still significantly reduced the amount of time taken to solve the eight puzzle. For the state shown in Fig. 2, this heuristic would count the following row swaps as

1. Swap 0 with 8
2. Swap 0 with 3
3. Swap 0 with 1
4. Swap 0 with 8
5. Swap 0 with 6
6. Swap 0 with 1
7. Swap 0 with 2
8. Swap 0 with 6
9. Swap 0 with 3
10. Swap 0 with 2
11. Swap 0 with 4

and count column swaps using the same method. In this case 6 column swaps are needed, for a total score of 17. X-Y distance is admissible, and recommended to be used with Manhattan distance as

$$h_{Mostow} = h_{MD} + h_{XY} \tag{9}$$

in order to more efficiently guide the search [3].

4.5 Linear Conflict

The linear conflict heuristic, h_{LC}, measures linear conflicts between tiles moving towards their goal positions [1]. A linear conflict occurs when a tile (tile A) and its goal position are in the same row, a different tile (tile B) with its goal position is in the same row, and the goal of tile A is to the right or at tile B, and the goal of tile B is to the left of the goal of tile A. Linear conflicts also occur for columns, but replace left/right with down/up, respectively. In the example position shown in Fig. 2, 2 linear conflicts exist. One exists in the center row, where tiles 5 and 3 pose a linear conflict. Tile 5 has a goal position to the right of tile 3, tile 3 has a goal position where tile 5 currently is, and tile 3 is to the right of tile 5. The other linear conflict exists in the right column, with tiles 8 and 2. This heuristic is admissible, as it includes the amount of moves to account for moving tiles "out of the way" in Manhattan distance solutions. In fact, each linear conflict requires 2 actions to resolve (1 to move the conflicting tile out of the row/column, 1 to move it back), and is proposed to be used in combination with Manhattan distance [1] as

$$h_{Hansson} = h_{MD} + 2 * h_{LC}. \tag{10}$$

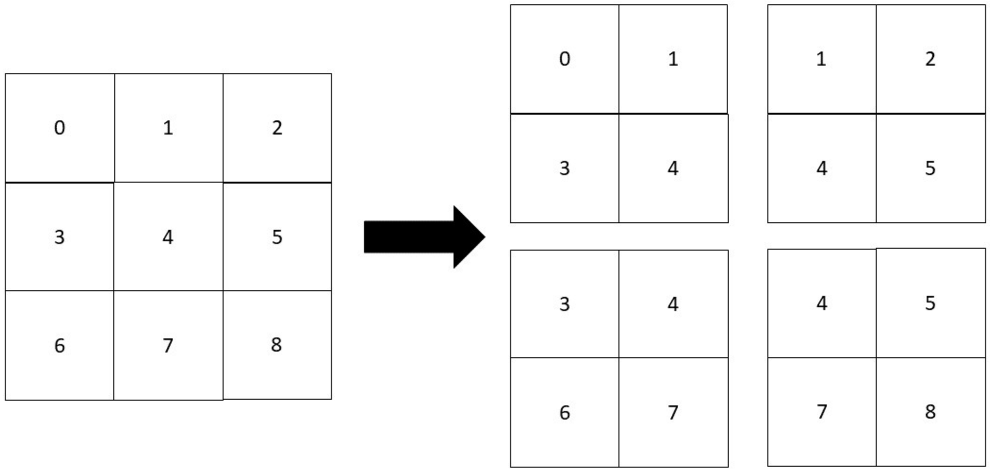

Figure 3: Sub-sequences for the eight puzzle goal. See Section 4.6 for details.

4.6 Subdivided Sequence

Additionally, we propose a new heuristic for the eight puzzle, known as subdivided sequence, and denoted by h_{Sub}. This heuristic is inspired by sequence score [4], but instead of looking at the entire sequence around the puzzle, breaks the puzzle into 4 smaller 2x2 regions, measuring the sequence in each sub-puzzle separately. An example of these subdivisions for the goal position is shown in Fig. 3. Once these smaller sequences are formed, the score is calculated the same as with sequence score and combined with Manhattan distance similarly, such that

$$h_{SS} = h_{MD} + 3 * h_{Sub}. \tag{11}$$

5 Experiments

All experiments were run using code implemented in Python running on a single thread of an AMD Ryzen 5 3600 CPU at 3.6 GHz. The implementation is not optimized for fast execution, and therefore we do not report the time of execution, but rather the amount of states explored (labeled as *visited*) and the length of the path found (labeled as *length*) in order to compare search methods and heuristics. Additionally, start states were generating using pre-selected random seeds which led to reasonable execution times. We evaluate performance according to two different metrics: penetrance [4] and optimality. Penetrance measures how "tight" a search is, and is defined as the length of the found path divided by total number of nodes explored. A penetrance of 1 indicates the search only explored nodes along the optimal path, and a penetrance close to 0 indicates the search explored many nodes not used by the optimal path. Optimality measures how optimal the final path found was, and is defined as the length of the optimal path divided by the length of the found path. An optimality of 1 indicates the search found the optimal path, whereas an optimality near 0 indicates the search found an extremely unoptimal path.

5.1 Comparison Between Search Methods

We first compare different search methods for the eight puzzle from a single starting state. We compare breadth-first search, greedy best-first search, and A* search. For the informed searches, we use the misplaced tiles and Manhattan distance heuristics. The results of this comparison are shown in Table 1. BFS does find the optimal path to the goal, but explores many nodes to find that path. GBS does not find the optimal path, but can find a near-optimal path depending upon the heuristic (Manhattan distance finds a near-optimal path for example), but also explores far less nodes than BFS and A*, and thus has a much higher penetrance. However, A* always finds the optimal path while exploring far less nodes than BFS. Even for the worst-performing heuristic, misplaced tiles, BFS explores more than 12 times as many states.

Table 1: Results of different search methods for a single start state. Effectiveness and optimality of search are shown across various methods.

Search Method	Visited	Length	Penetrance	Optimality
BFS	17471	17	0.000973041	1.0
Misplaced Tiles GBS	360	47	0.130555556	0.361702128
Manhattan Distance GBS	151	21	0.139072848	0.80952381
Misplaced Tiles A*	1431	17	0.011879804	1.0
Manhattan Distance A*	752	17	0.022606383	1.0

5.2 Comparison Between Heuristics in Greedy Search

We use greedy best-first search to compare all heuristics (for more information on heuristics, see Sec. 4). Although GBS does not gaurantee finding the optimal path, it does quickly find a path to the goal state without exploring many nodes. The average number of states explored, average found path length, average penetrance, and average optimality are shown over 10 start states are shown in Table 2. Full data is shown in the appendices (see Sec. 7.1). It can be seen that on average, misplaced tile is the worst-performing heuristic with the lowest penetrance and optimality. We will compare all other heuristics to Manhattan distance, as they all incorporate Manhattan distance to guide the search. These heuristics all improve on Manhattan distance in some way. Sequence score (Nilsson) and subdivided sequence both have very high penetrances and optimalities, with the lowest average visited and average path lengths, respectively. X-Y Distance (Mostow) has a lower average path length, but less penetrance and optimality, due to exploring more nodes on average. Finally, linear conflict (Hansson) has higher penetrance and optimality, but not as high as the sequence-based heuristics.

Table 2: Comparison of various heuristics across GBS. Values shown for states visited, final path length, penetrance, and optimality are averaged across 10 start states.

Heuristic	Visited	Length	Penetrance	Optimality
Misplaced Tiles	589.8	85	0.1734	0.293761589
Manhattan Distance	243.1	67.2	0.3099	0.429619439
Nilsson	93.3	43.2	0.5631	0.641733705
Mostow	266.6	62.6	0.2604	0.410041138
Hansson	167.8	59.8	0.389	0.453125717
Subdivided Sequence	101.5	38.4	0.4526	0.684699763

5.3 Comparison Between Heuristics in A* Search

Similarly, we compare all heuristics using A* search, with average performances shown in Table 3 (for full results, see Sec. 7.2). It is of note that A* should have an average optimality of 1 for heuristics which are admissible, but for some heuristics which meet this requirement this optimality was not achieved (misplaced tiles and linear conflict). The reason for this is unknown, possibly due to an implementation error. The sequence-based heuristics are not admissible, and perform very poorly in A* search. Misplaced tiles performs the worst of all, with a lower penetrance than even the non-admissible heuristics. This is likely because misplaced tiles provides little guidance towards the solution as a heuristic. Manhattan distance, X-Y distance (Mostow), and linear conflict (Hansson) all perform very well in A* search, with average penetrances of about 0.01. These heuristics provide good information about the path to the solution and effectively guide the search.

Table 3: Comparison of various heuristics across A* search. Values shown for states visited, final path length, penetrance, and optimality are averaged across 10 start states.

Heuristic	**Visited**	**Length**	**Penetrance**	**Optimality**
Misplaced Tiles	46686	24	0.0027	0.9862
Manhattan Distance	9377.1	23.6	0.0099	1
Nilsson	24254.5	24.8	0.0042	0.9589
Mostow	7964.9	23.6	0.0105	1
Hansson	7671.1	24	0.0112	0.9862
Subdivided Sequence	43490	23.8	0.0036	0.9926

6 Conclusion

We have examined the problem of the eight puzzle, a game played on a grid where the objective is to find a path to a goal state from some start state. We examine different search methods and different heuristics, and examine the effectiveness of each. Additionally, we propose a new heuristic for the eight puzzle, subdivided sequence, and analyze its performance across search methods. We found that in greedy searches, sequence-based heuristics perform best, whereas in A* search, admissible heuristics which provide sufficient information about the path to the goal perform best. Future work could include further analysis into why some heuristics did not perform optimally in A* search, as well as developing a better heuristic for A* search.

References

[1] Othar Hansson, Andrew Mayer, and Moti Yung. Criticizing solutions to relaxed models yields powerful admissible heuristics. *Information Sciences*, 63(3):207–227, 1992.

[2] Robert C Holte. Common misconceptions concerning heuristic search. In *Third Annual Symposium on Combinatorial Search*, 2010.

[3] Jack Mostow and Armand Prieditis. Discovering admissible heuristics by abstracting and optimizing: A transformational approach. 1989.

[4] Nils J Nilsson. *Principles of artificial intelligence*. Springer Science & Business Media, 1982.

[5] Alexander Reinefeld. Complete solution of the eight-puzzle and the benefit of node ordering in ida*. In *International Joint Conference on Artificial Intelligence*, pages 248–253, 1993.

[6] Stuart Jonathan Russell and Peter Norvig. Artificial intelligence: A modern approach. 1995.

7 Appendices

7.1 Appendix A: Greedy Best-First Search Results

Table 4: Full results for GBS with misplaced tiles heuristic.

RNG Seed	**Optimal Length**	**Misplaced Tiles**			
		Visited	Length	Penetrance	Optimality
1	26	446	58	0.130044843	0.448275862
2	28	1113	92	0.082659479	0.304347826
5	17	360	47	0.130555556	0.361702128
6	25	1078	105	0.097402597	0.238095238
10	23	428	73	0.170560748	0.315068493
11	25	630	113	0.179365079	0.221238938
14	24	281	94	0.334519573	0.255319149
15	17	450	79	0.175555556	0.215189873
17	23	341	115	0.337243402	0.2
28	28	771	74	0.095979248	0.378378378
Averages		589.8	85	0.173388608	0.293761589

Table 5: Full results for GBS with Manhattan distance heuristic.

RNG Seed	**Optimal Length**	**Manhattan Distance**			
		Visited	Length	Penetrance	Optimality
1	26	439	78	0.177676538	0.333333333
2	28	264	74	0.28030303	0.378378378
5	17	151	21	0.139072848	0.80952381
6	25	76	41	0.539473684	0.609756098
10	23	315	61	0.193650794	0.37704918
11	25	192	59	0.307291667	0.423728814
14	24	153	64	0.418300654	0.375
15	17	382	139	0.363874346	0.122302158
17	23	351	89	0.253561254	0.258426966
28	28	108	46	0.425925926	0.608695652
Averages		243.1	67.2	0.309913074	0.429619439

Table 6: Full results for GBS with sequence score (Nilsson) heuristic.

RNG Seed	**Optimal Length**	**Nilsson**			
		Visited	Length	Penetrance	Optimality
1	26	42	30	0.71429	0.86667
2	28	40	30	0.75	0.93333
5	17	298	79	0.2651	0.21519
6	25	57	33	0.57895	0.75758
10	23	108	61	0.56481	0.37705
11	25	100	61	0.61	0.40984
14	24	52	38	0.73077	0.63158
15	17	28	17	0.60714	1
17	23	88	39	0.44318	0.58974
28	28	120	44	0.36667	0.63636
Averages		93.3	43.2	0.56309	0.64173

Table 7: Full results for GBS with X-Y distance (Mostow) heuristic.

RNG Seed	**Optimal Length**	**Mostow and Prieditis**			
		Visited	Length	Penetrance	Optimality
1	26	380	108	0.28421	0.24074
2	28	409	58	0.14181	0.48276
5	17	239	55	0.23013	0.30909
6	25	408	69	0.16912	0.36232
10	23	227	55	0.24229	0.41818
11	25	130	31	0.23846	0.80645
14	24	271	76	0.28044	0.31579
15	17	96	51	0.53125	0.33333
17	23	250	63	0.252	0.36508
28	28	256	60	0.23438	0.46667
Averages		266.6	62.6	0.26041	0.41004

Table 8: Full results for GBS with linear conflict (Hansson) heuristic.

RNG Seed	**Optimal Length**	**Hansson et al.**			
		Visited	Length	Penetrance	Optimality
1	26	78	34	0.4359	0.76471
2	28	279	102	0.36559	0.27451
5	17	43	21	0.48837	0.80952
6	25	267	67	0.25094	0.37313
10	23	104	51	0.49038	0.45098
11	25	209	81	0.38756	0.30864
14	24	265	72	0.2717	0.33333
15	17	131	59	0.45038	0.28814
17	23	177	59	0.33333	0.38983
28	28	125	52	0.416	0.53846
Averages		167.8	59.8	0.38902	0.45313

Table 9: Full results for GBS with subdivided sequence heuristic.

RNG Seed	**Optimal Length**	**Subdivided Sequence**			
		Visited	Length	Penetrance	Optimality
1	26	234	76	0.32479	0.34211
2	28	158	48	0.3038	0.58333
5	17	53	27	0.50943	0.62963
6	25	72	27	0.375	0.92593
10	23	54	29	0.53704	0.7931
11	25	108	41	0.37963	0.60976
14	24	63	38	0.60317	0.63158
15	17	21	17	0.80952	1
17	23	74	29	0.39189	0.7931
28	28	178	52	0.29213	0.53846
Averages		101.5	38.4	0.45264	0.6847

7.2 Appendix B: A* Search Results

Table 10: Full results for A* search with misplaced tiles heuristic.

RNG Seed	Optimal Length	Misplaced Tiles			
		Visited	Length	Penetrance	Optimality
1	26	66805	28	0.00041913	0.9286
2	28	111139	28	0.000251937	1
5	17	1431	17	0.011879804	1
6	25	44987	25	0.000555716	1
10	23	6010	23	0.003826955	1
11	25	57235	25	0.000436796	1
14	24	33205	24	0.000722783	1
15	17	2827	17	0.006013442	1
17	23	9451	23	0.002433605	1
28	28	133770	30	0.000224266	0.9333
Averages		46686	24	0.002676443	0.9862

Table 11: Full results for A* search with Manhattan distance heuristic.

RNG Seed	Optimal Length	Manhattan Distance			
		Visited	Length	Penetrance	Optimality
1	26	18927	26	0.001373699	1
2	28	25438	28	0.001100715	1
5	17	752	17	0.022606383	1
6	25	11469	25	0.002179789	1
10	23	2425	23	0.009484536	1
11	25	5933	25	0.00421372	1
14	24	4381	24	0.005478201	1
15	17	435	17	0.03908046	1
17	23	1932	23	0.011904762	1
28	28	22079	28	0.001268173	1
Averages		9377.1	23.6	0.009869044	1

Table 12: Full results for A* search with sequence score (Nilsson) heuristic.

RNG Seed	Optimal Length	Nilsson			
		Visited	Length	Penetrance	Optimality
1	26	34619	28	0.000808804	0.9286
2	28	37046	32	0.000863791	0.875
5	17	2196	17	0.007741348	1
6	25	32651	25	0.000765673	1
10	23	11447	23	0.00200926	1
11	25	39167	29	0.000740419	0.8621
14	24	19004	26	0.001368133	0.9231
15	17	682	17	0.024926686	1
17	23	11306	23	0.002034318	1
28	28	54427	28	0.000514451	1
Averages		24255	24.8	0.004177288	0.9589

Table 13: Full results for A* search with X-Y distance (Mostow) heuristic.

RNG Seed	Optimal Length	Mostow and Prieditis			
		Visited	Length	Penetrance	Optimality
1	26	17469	26	0.001488351	1
2	28	17442	28	0.00160532	1
5	17	862	17	0.019721578	1
6	25	9714	25	0.002573605	1
10	23	2672	23	0.008607784	1
11	25	5484	25	0.004558716	1
14	24	3877	24	0.006190353	1
15	17	352	17	0.048295455	1
17	23	2279	23	0.010092146	1
28	28	19498	28	0.001436045	1
Averages		7964.9	23.6	0.010456935	1

Table 14: Full results for A* search with linear conflict (Hansson) heuristic.

RNG Seed	Optimal Length	Hansson et al.			
		Visited	Length	Penetrance	Optimality
1	26	15670	28	0.001786854	0.9286
2	28	21654	30	0.001385425	0.9333
5	17	565	17	0.030088496	1
6	25	8489	25	0.002944988	1
10	23	2918	23	0.007882111	1
11	25	6564	25	0.003808653	1
14	24	4197	24	0.00571837	1
15	17	368	17	0.046195652	1
17	23	2158	23	0.010658017	1
28	28	14128	28	0.00198188	1
Averages		7671.1	24	0.011245045	0.9862

Table 15: Full results for A* search with subdivided sequence heuristic.

RNG Seed	**Optimal Length**	**Subdivided Sequence**			
		Visited	Length	Penetrance	Optimality
1	26	68203	26	0.000381215	1
2	28	104619	28	0.000267638	1
5	17	2391	17	0.007109996	1
6	25	43042	25	0.000580828	1
10	23	14718	23	0.001562712	1
11	25	64520	27	0.000418475	0.9259
14	24	23102	24	0.001038871	1
15	17	786	17	0.021628499	1
17	23	10257	23	0.002242371	1
28	28	103262	28	0.000271155	1
Averages		43490	23.8	0.003550176	0.9926