Evaluation of K-Puzzle Search Algorithms

Group 26: Harsh Goel, Ho Hol Yin, Shradheya Thakre, and Wang Shiyao

1 K-Puzzle Problem Specification

1.1 Problem Introduction

The k-puzzle problem involves k numbered cells and a blank space placed on a square grid. The goal is to arrange the numbered cells sequentially across the grid, followed by a blank space at the end, through a valid sequence of moves. A valid move is defined by moving a numbered cell that is adjacent to the blank space along a row or column.

1.2 State Representation

The state space of the puzzle is discrete. Each state of the puzzle is represented by a 2D array of integers from 0 to k (0 represents the blank space). The action space of the puzzle is defined by a valid move 'Up','Down','Left' or 'Right' that moves cells directly below, above, to the right or to the left of the blank space respectively. Each state transitions to another state by applying a valid action to the numbered cells adjacent to the blank space.
Each node in the search graph represents a state and its children represent the state derived from applying one of 4 valid actions on the original state. The cost to reach a certain node is the number of actions taken to reach that node from the root node represented by the initial state.

2 Technical Analysis of Algorithms

2.1 Uninformed search - Iterative Deepening Search (IDS)

IDS is complete and takes advantage of both the space efficiency of Depth-First-Search and the optimality of Breadth-First-Search. The total time complexity is $\mathcal{O}(4^d)$ and space complexity is $\mathcal{O}(4d)$ where the branching factor is 4 and d is the depth of shallowest goal state. Since the branching factor is finite and all step costs are identical, this search algorithm gives a complete and optimal solution.

2.2 Informed Search: A* Search

We have implemented the graph-search version of A* Search, which given our chosen heuristics is optimal and complete. The algorithm's time complexity depends on the heuristic used and is denoted by $\mathcal{O}(b^{h^*(s_0)-h(s_0)})$ where $h^*(s_0) - h(s_0)$ denotes error between true and estimated cost for heuristic. The space complexity is $\mathcal{O}(b^d)$ as it will have to store all generated nodes in memory in the worst case.

Heuristic 1: Manhattan Distance (MD)
This heuristic estimates the cost of the current state to the goal by summing up the Manhattan distance of each tile with respect to its goal location. This heuristic relaxes the condition that a tile must be moved to a blank space. The proof of consistency for this heuristic can be found in Appendix 5.1.

Heuristic 2: Linear Conflict (LC)
This heuristic further scrutinises the movement of cells observed in Manhattan Distance heuristic by evaluating the number of linear conflicts across a row or column. Two cells t_1 and t_2 are in a linear conflict along a row if t_1 and t_2 are in the same line, the goal positions of t_1 and t_2 are both in that line, t_1 is to the right of t_2, and the goal position of t_1 is to the left of the goal position of t_2 or vice versa. The same definition extends to linear conflicts along a column. We evaluate the total linear conflicts for all cells, across all rows and columns of a given state.
Define the linear conflict function $l : v \rightarrow \mathbb{Z}^+ \cup \{0\}$.
Letting the Manhattan Distance heuristic be h_0, and the Linear Conflict heuristic be h_1, we have:

$$h_1(v) = h_0(v) + 2 * l(v)$$

We prove consistency on three distinct cases:
1) A cell moving into the goal row
2) A cell moving from a non-goal row to another non-goal row
3) A cell moving away from a goal row.

In each of the cases we prove the consistency of the heuristic (see Appendix 5.2). In addition, we do not lose generality, and these cases can be extended to moving across columns as well.

Heuristic 3: Euclidean Distance (ED)
This heuristic estimates the cost of the current state from its goal location by summing the Euclidean distances of the current position of their tiles from their goal location. This heuristic is consistent as each movement of tile reduces the euclidean distance from its target position by at most one. See Appendix 5.3 for a more detailed proof.

Table 1. Comparison between IDS and the different heuristics used in A* Search

Algorithm	IDS	A*:Manhattan Dist.	A*: Linear Conflict	A*: Euclidean Dist.
Completeness	✓	✓	✓	✓
Optimality	✓	✓	✓	✓
Time Complexity	$\mathcal{O}(4^d)$	$\mathcal{O}(b^{h^*(s_0)-h(s_0)})$	$\mathcal{O}(b^{h^*(s_0)-h(s_0)})$	$\mathcal{O}(b^{h^*(s_0)-h(s_0)})$
Space Complexity	$\mathcal{O}(4d)$	$\mathcal{O}(b^d)$	$\mathcal{O}(b^d)$	$\mathcal{O}(b^d)$

3 Experimental Setup

We evaluate our search algorithms using two metrics, *efficiency* and *memory-efficiency*. We define an *efficient* search algorithm to be one that explores a small number of states in the state space to reach a goal state. Additionally, we define a *memory-efficient* search to be one that has a small total number of nodes in the frontier in the memory at a given search depth. These statistics are computed using the puzzles provided for testing(See Appendix 5.7).
Our choice of metric can be explained as follows:

3.1 Number of Nodes Explored (*efficiency*)

We can directly determine if a search algorithm is *efficient* using this statistic. A small number of nodes explored will translate to an efficient algorithm. One can also infer the effective branching factor at a given depth using this statistic which is crucial in determining how effective the search algorithm is in navigating through the state space. An *efficient* search will consequently take a less amount of time, thus giving us an indication of the **time complexity**.

3.2 Number of Nodes in Frontier (*memory-efficiency*)

We define an *node in the frontier* to be a node that is generated during the search but has yet to be explored. A small **number of nodes in the frontier** during the search at a given depth will mean that a search is *memory-efficient* and, therefore, indicating to us the **space complexity** of the search. This statistic is only applicable for A* Search, since in IDS all nodes are explored.

Thus, through this experiment one can evaluate how many nodes are explored by the implemented IDS and A* search algorithms. In order to replicate the experiment, detailed instructions can be found in the **CS3243_P1_26_5.py** file comments.

4 Results and Discussion

Table 2. Results from running on the given **n_equals_3/input_2.txt** input

Algorithm	IDS	A*: MD	A*: LC	A*: ED
No. of Nodes Explored	512,922	1264	697	2288
Max No. of Nodes in Frontier	$6 * 10^{12}$	1263	696	2284
Time Taken (s)	15.99	0.08	0.16	0.14

4.1 Number of Nodes Explored (*efficiency*)

Referring to results for an 8 puzzle problem in Appendix 5.8 (Summarized in Table 2.), we can see that IDS has a vastly greater number of nodes explored (512922) than any of the A* Search heuristics (1264, 697, 2288 respectively). A* Search also vastly outperforms IDS for all 3 heuristics in terms of time taken in seconds. These also results hold true for larger puzzles.

Comparing the heuristics, it is observed that at any given depth, the **LC** heuristic explores fewer nodes than the **MD** heuristic, which in turn explores fewer nodes than the **ED** heuristic. This is an indication of the order of dominance among the heuristics, which we have formally proven in the Appendix Sections 5.4, 5.5 and 5.6.

While lower number of nodes generated implies lower computation time, A* search with **LC** has no substantial improvement in the time taken as compared to A* search with **MD** and **ED** heuristic for the the 8 puzzle problem. This is due to the computation overhead incurred on calculating the heuristic cost. Although, for difficult puzzles with high solution depths, the **LC** heuristic is expected to perform much faster.

4.2 Number of Nodes in Frontier (*memory-efficiency*)

In **Fig. 2** (See Appendix 5.7), we can see that, generally the **LC** heuristic has the smallest frontier size and is therefore the most *memory-efficient*, followed by the Manhattan Distance heuristic and finally, the Euclidean Distance heuristic. We reason that a more dominant heuristic will likely generate fewer new nodes, thus resulting in a smaller frontier.

Interestingly, we have observed that the number of nodes in frontier seems to plateau and drop after a certain depth for any given heuristic as it approaches the goal state.

Overall, we can see how A* in general performs much better than IDS time-wise due to informed node exploration. On comparing the three heuristics, the most dominating Linear Conflict heuristic gives the optimal solution with the greatest *efficiency* and *memory-efficiency*.

References

1. Othar Hansson, Andrew Ma Yer. Moti Yung: Criticizing Solutions to Relaxed Models Yields Powerful Admissible Heuristics. In: INFORMATION SCIENCES vol. 63, pp. 207–227. Elsevier Science Publishing Co., Inc, New York (1992).

2. Mark R.: Solvability of the Tiles Game . The University of Birmingham, https://www.cs.bham.ac.uk/ mdr/teaching/modules04/java2/TilesSolvability.html. Last accessed 5 Mar 2020

5 Appendix

5.1 Consistency for Manhattan Distance Heuristic

Proof. Let the heuristic for the Manhattan Distance Heuristic be h_0. By the definition of Manhattan Distance, we know that:

$$|h_0(v) - h_0(v')| = 1$$
$$h_0(v) - h_0(v') \leq 1$$

Rearranging, we get:

$$h_0(v) \leq h_0(v') + 1$$

showing that h_0 is consistent and consequently also admissible.

5.2 Consistency for Linear Conflict Heuristic

Proof. Let the heuristic for the Linear Conflict Heuristic be h_1. We shall focus on the behaviour of a single cell which can then be extended to all cells. Suppose, without a loss in generality, we move a cell from row r_1 to r_2, where the cell is in r_1 in state v and in r_2 in state v'. There are three cases:

1) The cell's goal position is along r_2: Since the cell starts on r_1, there is no linear conflict for this cell in state v

$$h_1(v) = h_0(v) \tag{1}$$

Moving into the row of the goal position will reduce the Manhattan Distance by 1.

$$h_0(v') = h_0(v) - 1 \tag{2}$$

Moving into the row of the goal position will either create no linear conflict (can slide into goal directly without moving other cells), or will create a new linear conflict (of +2)

$$h_1(v') = h_0(v')$$
$$or$$
$$h_1(v') = h_0(v') + 2$$

so that by (2), we have that

$$h_1(v') = h_0(v) \pm 1$$

and from (1),

$$h_1(v') = h_1(v) \pm 1$$

It follows that

$$|h_1(v) - h_1(v')| = 1$$

$$h_1(v) \leq h_1(v') + 1$$

showing that h_1 is consistent and consequently also admissible.

2) The cell's goal position not in r_1 and r_2:
There is no change in linear conflict between v and v' for this cell. Therefore,

$$h_1(v) = h_0(v)$$

$$\textit{and}$$

$$h_1(v') = h_0(v')$$

and since h_0 is proven to be consistent in 5.1 above, it follows that h_1 is also consistent in this scenario.
3) The cell's goal position is along r_1: Before moving, there is either no linear conflict, or there will be a linear conflict (of +2):

$$h_1(v) = h_0(v) \tag{3}$$

$$\textit{or}$$

$$h_1(v) = h_0(v) + 2 \tag{4}$$

Moving away from the row of the goal position will increase the Manhattan Distance by 1.

$$h_0(v') = h_0(v) + 1 \tag{5}$$

Moving into the row that is not that of the goal position remove all linear conflict.

$$h_1(v') = h_0(v')$$

and by (5),

$$h_1(v') = h_0(v) + 1$$

Following from (3) and (4), we have

$$h_1(v') = h_1(v) \pm 1$$

It follows that

$$|h_1(v) - h_1(v')| = 1$$

$$h_1(v) \leq h_1(v') + 1$$

showing that h_1 is consistent and consequently also admissible.
In all three cases, we have shown that h_1 remains consistent and consequently admissible when moving cells between rows. Repeating the proof on columns will yield the same result.

5.3 Consistency for Euclidean Distance Heuristic

Proof. Let the heuristic for the Euclidean Distance be h_2. The euclidean distances from v to the goal position $(h_2(v))$, from v' to the goal position $(h_2(v'))$ and from v to v' (of length 1), forms a triangle. By cosine rule, it follows that:

$$(h_2(v))^2 = (h_2(v'))^2 + 1^2 - 2h_2(v')cos\theta$$

where θ is the angle between the edge from v to v' and the edge from v' to the goal position.

We know that

$$-1 < cos\theta < 1$$

so that

$$(h_2(v))^2 \leq (h_2(v'))^2 + 1^2 - 2h_2(v')(-1))$$

$$(h_2(v))^2 \leq (h_2(v'))^2 + 1^2 + 2h_2(v'))$$

$$(h_2(v))^2 \leq (h_2(v') + 1)^2$$

$$h_2(v) \leq h_2(v') + 1$$

showing that h_2 is consistent and consequently also admissible.

5.4 Linear Conflict is more dominant than Manhattan Distance

Proof. Define the linear conflict function $l : v \rightarrow \mathbb{Z}^+ \cup \{0\}$.
Given that V is the set of all valid states, we know that

$$\forall v \in V, l(v) \geq 0$$

From the definition of h_0 and h_1 above, we have:

$$h_1(v) = h_0(v) + l(v)$$

Consequently,

$$\forall v \in V, h_1(v) \geq h_0(v)$$

so that h_1 dominates h_0.

5.5 Manhattan Distance is more dominant than Euclidean Distance

Proof. Denote δx and δy as the horizontal and vertical displacement from v to the goal position respectively. Therefore, $\delta x \geq 0$ and $\delta y \geq 0$, and so

$$(h_0(v))^2 = (\delta x + \delta y)^2 = (\delta x)^2 + (\delta y)^2 + 2(\delta x)(\delta y)$$

Also,

$$(h_2(v))^2 = (\delta x)^2 + (\delta y)^2$$

so that

$$(h_0(v))^2 = (h_2(v))^2 + 2(\delta x)(\delta y)$$
$$(h_0(v))^2 \geq (h_2(v))^2$$
$$h_0(v) \geq h_2(v)$$

giving us the required inequality that h_0 dominates h_1.

5.6 Linear Conflict is more dominant than Euclidean Distance

Proof. Given that V is the set of all valid states, from 5.4 and 5.5, we know that $\forall v \in V$,

$$h_1(v) \geq h_0(v)$$
$$and$$
$$h_0(v) \geq h_2(v)$$

By transitivity,

$$h_1(v) \geq h_2(v)$$

so that h_1 dominates h_2

5.7 Results Plots

The following scatter graphs are generated to show how the heuristics scale with difficulty of puzzle(i.e. $h^*(s_0) - h(s_0)$ is higher for difficult puzzles and they require more steps to reach the goal state).
We took four different puzzles and used the x-axis to denote the number of steps required. The y-axis in the first graph represents the number of explored nodes[in closed list] and number of nodes in frontier[in open list] in order to determine the time and memory efficiency as the puzzles get harder.

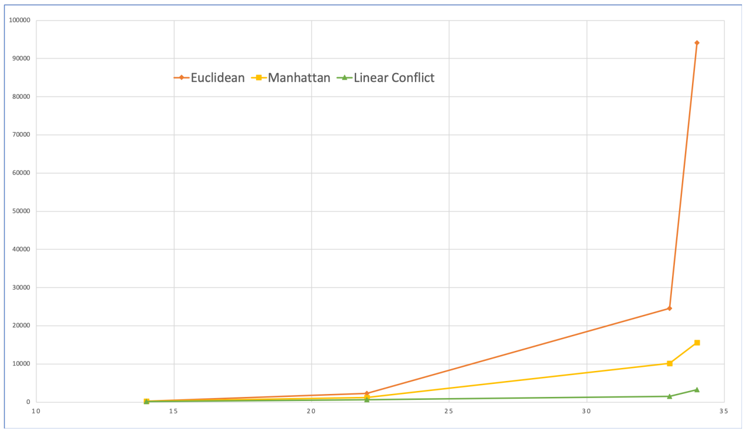

Fig. 1. Solution Depth vs Nodes explored

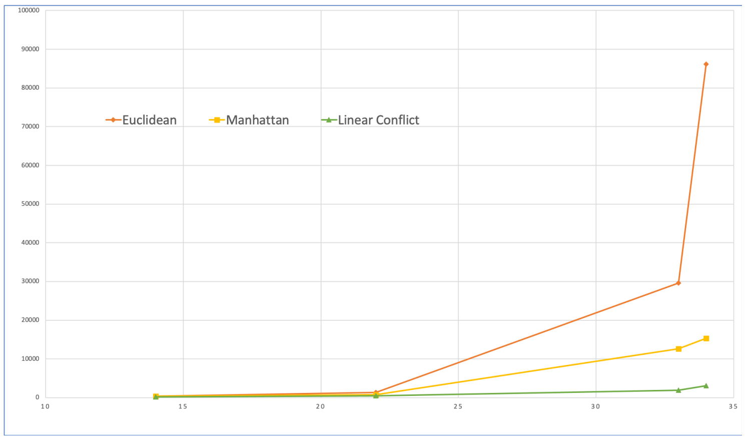

Fig. 2. Solution Depth vs Size of frontier

5.8 Result generated for Analysis of Search Algorithm Experiment

The result below is generated from running our experiments script **CS3243_P1_26_5.py** on the given input file **n_equals_3/input_2.txt**, from which we have drawn conclusions for our report.

```
Iterative Deepening Search
Nodes Explored: 512922
 Search Size at Depths: 0, 4, 20, 84, 340, 1364, 5460, 21844,
     ↪ 87380, 349524, 1398100, 5592404, 22369620, 89478484,
     ↪ 357913940, 1431655764, 5726623060, 22906492244,
     ↪ 91625968980, 366503875924, 1466015503700, 5864062014804,
     ↪ 23456248059220
 Number of steps: 22
 Time taken (s): 15.9904389381

Heuristic 1(Manhattan)
Nodes Explored: [0, 2, 4, 6, 9, 13, 22, 37, 52, 53, 55, 61, 147,
    ↪ 154, 188, 386, 465, 474, 911, 1258, 1260, 1262, 1264]
 Size of Priority Queue: 1, 3, 16, 40, 87, 256, 637, 1098, 1214,
     ↪ 1261, 1263, 1250, 1259, 1245, 1247, 1249, 1257, 1254,
     ↪ 1256, 791, 792, 793, 794
 Total Nodes: [1, 5, 20, 46, 96, 269, 659, 1135, 1266, 1314, 1318,
     ↪  1311, 1406, 1399, 1435, 1635, 1722, 1728, 2167, 2049,
     ↪ 2052, 2055, 2058]
 Number of steps: 22
 Initial H cost: 8
 Time taken (s): 0.0816991329193
 Average heuristic cost: 7.77272727273

Heuristic 2(Linear Conflicts)
Nodes Explored: [0, 2, 4, 5, 7, 11, 27, 30, 35, 64, 114, 148, 150,
    ↪  152, 154, 227, 329, 373, 534, 688, 691, 694, 697]
 Size of Priority Queue: 1, 3, 16, 37, 110, 193, 491, 624, 658,
     ↪ 696, 692, 695, 693, 687, 690, 680, 667, 682, 685, 456,
     ↪ 457, 458, 460
 Total Nodes: [1, 5, 20, 42, 117, 204, 518, 654, 693, 760, 806,
     ↪ 843, 843, 839, 844, 907, 996, 1055, 1219, 1144, 1148,
     ↪ 1152, 1157]
 Number of steps: 22
 Initial H cost: 12
 Time taken (s): 0.162168979645
 Average heuristic cost: 7.86363636364
```

```
Heuristic 3(Euclidean) Nodes Explored: [0, 2, 4, 6, 9, 15, 26, 29,
    ↪  56, 57, 59, 138, 164, 325, 330, 621, 680, 682, 1420, 2279,
    ↪  2282, 2285, 2288]
 Size of Priority Queue: 1, 3, 16, 27, 92, 151, 312, 603, 1102,
     ↪ 2233, 2150, 2242, 2272, 2275, 2284, 2277, 2287, 2271,
     ↪ 2276, 1350, 1351, 1351, 1354
 Total Nodes: [1, 5, 20, 33, 101, 166, 338, 632, 1158, 2290, 2209,
     ↪  2380, 2436, 2600, 2614, 2898, 2967, 2953, 3696, 3629,
     ↪ 3633, 3636, 3642]
 Number of steps: 22
 Initial H cost: 8.0
 Time taken (s): 0.139376163483
 Average heuristic cost: 6.98023255447
```