CS3243 PROJECT 1 REPORT

Jeremy Loye, She Jiayu, Sebastian Lie, and Tan Yuanhong

National University of Singapore

1 Problem Specification

We model the k-puzzle Problem as follows:

State A $n \times n$ array, with cells numbered by unique integers from 0 to $k = n^2 - 1$ inclusive, where 0 represents an empty cell. Initial state is the array specified by input. Goal state is the array with integers 1 to $n^2 - 1$ occupying the first $n^2 - 1$ cells in order and 0 occupying the last cell.
Actions `LEFT`: Swap 0 with number to its right. `RIGHT`: Swap 0 with number to its left. `UP`: Swap 0 with number below it. `DOWN`: Swap 0 with number above it.
Transition Model `move(current_state, action)` where `action` is one of `LEFT`, `RIGHT`, `UP`, `DOWN` and `current_state` is the array representing the current arrangement of numbers in the grid.
Goal test Test if the current state is equal to the goal state for every cell.
Cost Function Cost equals to the number of moves required for the initial state to transit into the current state.

2 Technical Analysis

All of our search algorithms are graph based, i.e. we maintain an `explored` set to keep track of the states that have been goal-tested and we do not add nodes whose states are in the `explored` set. We adopt the following convention for notations:

n': successor of node n.
$g(n)$: cost from root to n.
$h(n)$: estimated cost from n to goal node.
$f(n)$: estimated cost from root to goal node passing through n.
$c(n, n')$: step cost from node n to n'.

2.1 Uninformed Search - Breadth-First Search (BFS)

Correctness

Completeness: Since BFS explores nodes in non-decreasing order of depth, as long as the branching factor b is finite, BFS is able to find the goal node. In the context of k-puzzle, $b = 4$ which is finite, so BFS is complete.
Optimality Since BFS explores nodes in non-decreasing order of depth, it outputs the shallowest goal node. In the context of k-puzzle, since step cost is one, the depth of the node is equal to its cost. Hence, BFS outputs the goal node with the least cost, which is optimal.

Complexity

Time complexity of BFS is equal to the number of nodes visited before yielding the result, which can be calculated layer-by-layer by $1+b+\cdots+b^d = O(b^d)$, where d is the depth of the goal node.
Space complexity of BFS is equal to the maximum size of the frontier plus the maximum size of the explored set. Since BFS saves every generated node, the maximum size is $O(b^d)$ for both the explored set and the frontier, leading to a total complexity is $O(b^d)$.

2.2 Informed Search - A* Search

We assume that the heuristic used in A* is consistent, i.e. $h(n')+c(n,n') \geq h(n)$ for every pair of node n and its corresponding successor n'.

Correctness

Completeness

Lemma 1. *Any finite graph search algorithm using a frontier queue is complete [1].*

Proof. By induction on n, the number of steps from root s to goal node g. *Base case*: When $n = 1$, s is one step away from g, at the first step of such an algorithm g will be added to the queue, so it is complete. *Inductive step*: Suppose the lemma is true for $n = k$, consider the case for $n = k + 1$. Since the g is reachable from s, when adding the neighbors of s to the queue, there exists at least one neighbor that can reach g and is k steps away from g. By the inductive hypothesis, the algorithm is complete.

Optimality

Lemma 2. *If h is consistent, $f(n)$ is non-decreasing along any path.*

Proof. $f(n') = g(n') + h(n') = g(n) + c(n, n') + h(n') \geq g(n) + h(n) = f(n)$

Lemma 3 (Graph-separation property). *In graph search, the frontier always separates the explored region of the search graph from the unexplored region. In other words, every path from the root to an unexplored node must include a node on the frontier [3].*

Theorem 1. *When A* (graph-based with consistent heuristic) selects a node n for expansion, the shortest path to n has been found.*

Proof. Suppose otherwise. By graph-separation property, any path from the root s to n (since n is not explored yet, otherwise A* will not select it for expansion) must include a node in the frontier. Since the shortest path to n is not found yet, there must exist some node t in the frontier such that it lies on the optimal path from s to n. Since t is on the path from s to n and is not n, by Lemma 2, $f(t) < f(n)$. As both n and t are in the frontier, A* will select t to expand instead of n, which is a contradiction.

By **Theorem 1**, we have that when goal node g is reached, the shortest path from root s to g is found. This proves the optimality of A*.

Complexity

Time complexity: $O(b^{f^*(s)-f(s)})$, where b is the branching factor and $f^*(s)$ is the actual cost from root s to goal node g.
Space complexity: $O(b^d)$, where d is depth of the goal node.

2.3 Heuristic 1 & 2 - Misplaced Tiles & Manhattan Distance

Proof of consistency is given in AIMA, page 105 [3].

2.4 Heuristic 3 - Linear Conflict

The definition of linear conflict, the algorithm, and the idea for the proof is taken from [2]. The full algorithm for calculating the linear conflict heuristic is given in the appendix.

Definition 1. *Two tiles t_j and t_k are in a linear conflict if t_j and t_k are in the same line, the goal positions of t_j and t_k are both in that line, 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.*

The estimated cost of node n by Linear Conflict heuristic is $h(n) = LC(n) + MD(n)$, where $MD(n) = \sum md(n, t_j)$ is the estimated cost given by Manhattan Distance heuristic, $LC(n) = 2\sum[lc(n, r_i)+lc(n, c_j)]$ where $lc(n, r_i)$ is the number of tiles to be removed from row i to attain 0 linear conflicts.

Proof. Consider the successor of a state obtained by moving a tile t left or right along row r_i of the k-puzzle. Since along r_i tile t is merely swapping places with a blank, number of linear conflicts of that row remains the same. Denote c_i as the initial column and c_j as the goal column. Now we consider 3 mutually exclusive cases for horizontal moves.

Case 1: The goal position of tile t is neither on column c_i nor c_j

Since neither column is the goal column, we have $LC(n') = LC(n)$ and $MD(n') = MD(n) \pm 1$, so $f(n') = 1 + g(n) + LC(n) + MD(n') \geq g(n) + LC(n) + MD(n) = f(n)$.

Case 2: The goal position of tile t is column c_i

Since tile t moves out of its goal column, $LC(n') = LC(n)$ or $LC(n') = LC(n) - 2$ since $lc(n, c_i) - 1$, and $MD(n') = MD(n) + 1$. Thus we have $f(n') = 1 + g(n) + LC(n') + MD(n) + 1 \geq g(n) + LC(n) + MD(n) = f(n)$.

Case 3: The goal position of tile t is column c_j

Since tile t moves into its goal column, $LC(n') = LC(n)$ or $LC(n') = LC(n)+2$, since $lc(n, c_j)+1$, but $MD(n') = MD(n)-1$. Thus we have $f(n') = 1+g(n)+LC(n') + MD(n) - 1 \geq g(n) + LC(n) + MD(n) = f(n)$.

A similar argument can be made for vertical moves. Hence, $f(n') \geq f(n)$ for any possible successor n' of any state n. Thus the linear conflict heuristic is consistent.

3 Experimental Setup

To examine the empirical time and space complexity of the 4 different algorithms implemented in this project, we wrote a generator to generate 10 random **solvable** [4] test cases for each dimension 3×3, 4×4 and 5×5, and then run the 4 algorithms on these test cases. For practicality of the experiment, we set a time limit of 120 seconds for each test case. To avoid too many time limit exceeded (TLE) cases, the generator only generates test cases whose solution depth is at most 100.

To assess the empirical space complexity, we measure the number of generated states and the maximum size of the frontier, both of which constitute memory consumption.

To assess the empirical time complexity, we measure the search depth, runtime and pass rate. Additionally, we measure the number of explored states and solution depth to further evaluate the efficiency [1] of the informed search heuristics.

4 Results and Discussion

Dimension	3×3				4×4				5×5			
Solution	1	2	3	4	1	2	3	4	1	2	3	4
Pass Rate(%)	100	100	100	100	0	80	100	100	0	50	100	100
Avg Runtime(s)	0.178	0.0840	0.0693	0.0789	-	17.4	1.83	1.09	-	19.4	13.4	13.5
Runtime Stdev(s)	0.153	0.0211	0.00851	0.0191	-	35.0	4.39	1.58	-	28.4	25.2	26.8
Max Runtime(s)	0.598	0.141	0.0853	0.124	-	109	14.9	5.36	-	74.5	73.4	86.6
Avg Solution Depth	14.6	14.6	14.6	14.6	-	26.25	28.2	28.2	-	29.2	34.8	34.8
Avg Search Depth	14.6	14.6	14.6	14.6	-	26.25	28.2	28.2	-	29.2	34.8	34.8
Avg Explored States	10461.9	588.9	119.5	70.0	-	339809.6	24690.7	3628.9	-	320016.6	125896.0	25246.0
Avg Generated States	10461.9	939.3	197.0	118.9	-	645075.1	46474.5	6970.0	-	727278.6	273628.9	54830.9
Avg Frontier size	3516.4	352.5	79.1	50.1	-	306215.1	22223.5	3397.7	-	408309.8	149535.2	29974.7

Table 1. Experiment results

From the results, we can see that while all 4 algorithms manage to solve all ten 3×3 puzzles within time limit. However, BFS had TLE for all 4×4 and 5×5 test cases.

Among the 3 informed search heuristics, Manhattan Distance (MD) and Linear Conflict (LC) performed better than Misplaced Tile (MT) as shown by the higher pass rate and smaller space complexity, since MT measures the cost to the most relaxed problem [2]. Between MD and LC, LC outperformed MD in terms of space complexity. This is consistent with the fact that LC dominates MD. However, the difference in runtime is negligible and MD even performed slightly better than LC for 3×3 and 5×5. This is most likely due to the overhead incurred in computing the estimated cost using LC, as it has a complexity of $O(n^3)$, where n is the dimension of the grid.

[1] It should explore as few nodes in the frontier as possible.

[2] Tiles can move to any place in one move.

References

1. Karaman, S.: Soundness and completeness of state space search (2010), https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16_410F10_lec04.pdf
2. Othar Hansson, A.E.M., Yung, M.M.: Generating admissible heuristics by criticizing solutions to relaxed models (1985). https://doi.org/10.7916, https://academiccommons.columbia.edu/doi/10.7916/D8154QZT/download
3. Russell S. J., N.P., E., D.: Artificial intelligence: A modern approach. Prentice Hall (2010)
4. Ryan, M.: Solvability of the tiles game https://www.cs.bham.ac.uk/ mdr/teaching/modules04/java2/TilesSolvability.html

5 Appendix

5.1 Linear Conflict Algorithm

Definition 2.
s is the current state.
$c(t_j, r_i)$ is the number of conflicts in row r_i that tile t_j is in conflict with.
$lc(s, r_i)$ is the number of tiles that need be removed from row r_i in order to resolve the linear conflicts of that row; $lc(s, c_i)$ is the number of tiles that need be removed from column c_i.
$LC(s)$ is the minimum number of additional moves necessary to resolve the linear conflicts in s.
$md(s, t_j)$ is Manhattan Distance of tile t_j.
$MD(s)$ is the sum of Manhattan Distances of all the tiles in s.

Algorithm 1: Linear Conflict Heuristic

```
for $r_i \in \{1, ..., n\}$ do
    $lc(s, r_i) = 0$;
    for $t_j$ in row $r_i$ do
        compute($c(t_j, r_i)$);
    while not all $c(t_j, r_i) = 0$ do
        $t_{max} = t_k$ where $c(t_k, r_i) = max(c(t_j, r_i)) \forall t_j$ in row $r_i$;
        for $t_k$ in conflict with $t_{max}$ do
            $c(t_k, r_i) = c(t_k, r_i) - 1$;
        $c(t_{max}, r_i) = 0$;
        $lc(s, r_i) = lc(s, r_i) + 1$;
for $c_i \in \{1, ..., n\}$ do
    repeat above;
$LC(s) = 2(\sum_{i=1}^{n} lc(s, r_i) + \sum_{j=1}^{n} lc(s, c_j))$;
$MD(s) = \sum_{j=1, t_j \neq 0}^{n^2} md(s, t_j)$;
$h(s) = MD(s) + LC(s)$;
```

Intuitively, we examine the puzzle state, row by row and column by column, and add to Manhattan Distance the minimum number of additional moves necessary to resolve the conflicts within each row and column.

5.2 Full Proof of Consistency of Linear Conflict

We partition the proof of consistency into 2 parts. Since there are at most 4 successors of each node n, we partition the proof into when the action is along the row (left/right successors) and along the column (up/down successors).

We assume that the Manhattan Distance is consistent.

Let us first consider the successors obtained by moving along the row r_i of the k-puzzle. We observe that since along row r_i, tile t is merely swapping places with an empty tile, it will not cause any change to the number of linear conflicts of that row.

WLOG, we denote c_i as the initial column of a tile n to be moved, and c_j as the column tile n moves to. In particular, if the move is valid, $c_j = c_i - 1, c_i + 1$.

Now we consider 3 mutually exclusive cases for tile n and prove consistency for each of them.

Case 1: The goal position of tile t is neither on column c_i nor c_j

Since neither column is the goal column, the number of linear conflicts does not change, so no extra tile needs to be moved, and thus $LC(n') = LC(n)$. Besides, $MD(n') = MD(n) \pm 1 \implies MD(n') + 1 \geq MD(n)$. Thus we have:

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= c(n,n') + g(n) + LC(n') + MD(n') \\ &= 1 + g(n) + LC(n) + MD(n') \\ &= g(n) + LC(n) + [MD(n') + 1] \\ &\geq g(n) + LC(n) + MD(n) \\ &= g(n) + h(n) \end{aligned}$$

Case 2: The goal position of tile t is column c_i

We only need consider the linear conflicts along the column c_i. Since tile t moves out of its goal column, the number of linear conflicts could decrease if tile t was in conflict with another tile along c_i, or stay the same if tile t had no conflicts. This could reduce the number of tiles to be removed from column i, thus $lc(n', c_i) = lc(n, c_i)$ or $lc(n', c_i) = lc(n, c_i) - 1$. Thus $LC(n') = LC(n)$ or $LC(n') = LC(n) - 2 \implies LC(n') + 2 \geq LC(n)$. Since it will take 1 more move to move tile n back to its goal, $MD(n') = MD(n) + 1$. Thus we have:

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= c(n,n') + g(n) + LC(n') + MD(n') \\ &= 1 + g(n) + LC(n') + MD(n) + 1 \\ &= g(n) + [LC(n') + 2] + MD(n) \\ &\geq g(n) + LC(n) + MD(n) \\ &= g(n) + h(n) \end{aligned}$$

Case 3: The goal position of tile t is column c_j

We only need to consider the linear conflicts along the column c_j. Since tile t moves into its goal column, the number of linear conflicts could increase if tile t is now in conflict with some tile along c_j, or stay the same if tile t does not create conflicts in its new position. This could increase the number of tiles to be removed from column i, thus $lc(n', c_j) = lc(n, c_j)$ or $lc(n', c_j) = lc(n, c_j) + 1$. Thus $LC(n') = LC(n)$ or $LC(n') = LC(n) + 2 \implies LC(n') \geq LC(n)$. Since it takes 1 less move to move tile n back to its goal, $MD(n') = MD(n) - 1$. Thus we have:

$$\begin{aligned}
f(s') &= g(n') + h(n') \\
&= c(n, n') + g(n) + LC(n') + MD(n') \\
&= 1 + g(n) + LC(n') + MD(n) - 1 \\
&= g(n) + LC(n') + MD(n) \\
&\geq g(n) + LC(n) + MD(n) \\
&= g(n) + h(n)
\end{aligned}$$

Therefore for all cases along the row of the puzzle, $f(n') \geq f(n)$.

A similar argument can be made for the actions along the column of the k-puzzle, by rotating the grid and interchanging the roles of rows and columns .

Thus $f(n') \geq f(n)$ for all possible successors of any state. This proves the consistency of Linear Conflict heuristic.

5.3 Solvability of k-puzzle [4]

Definition 3. *Flatten the $n \times n$ grid to a sequence of numbers, a pair of numbers (a, b) is called an inversion if $a > b$ but a comes before b in the sequence. Inversion number of k-puzzle is the total number of inversions in the grid.*

Let b be the number of the tile that moves.

Lemma 4. *For any grid, a horizontal move does not change its inversion number.*

Proof. Since the order of tile b relative to other tiles in the grid remains the same after the move, the inversion number will not change.

Below is an example of a horizontal move in 8-puzzle.

x	x	x
a	b	0
x	x	x

x	x	x
a	0	b
x	x	x

Lemma 5. *For any grid with odd size $n \times n$, the parity of the inversion number is invariant.*

Proof. When tile b moves horizontally, by **Lemma 4**, the inversion number does not change.

When tile b moves vertically, it either moves in front of, or behind $n-1$ tiles. Of these $n-1$ tiles, suppose there are p numbers that are greater than b, where $0 \leq p \leq n-1$, then $n-1-p$ of them are smaller than b. The change in inversion number after b has moved vertically is $p-(n-1-p) = 1-n$ if b moves down, or $-p+(n-1-p) = n-1$ if b moves up. Since $|n-1|$ is even, the move does not affect the parity of the inversion number.

Hence the parity of the inversion number never changes.

Below is an example of a vertical move in 8-puzzle.

a	x	x
0	c	d
b	x	x

a	x	x
b	c	d
0	x	x

Lemma 6. *For any grid with odd size $n \times n$, if its initial state has an odd number of inversions, it is unsolvable.*

Proof. First, we note that the goal state has an inversion number of 0. Since any legal move does not change the parity of inversion number, any legal move on a grid with an odd inversion number will result in an odd inversion number. Therefore legal moves on a grid with an odd inversion number will never result in 0 inversions, and thus the puzzle is unsolvable.

Lemma 7. *For any grid with even size $n \times n$, moving a tile vertically always results in the change in parity of the inversion number.*

Proof. When b moves vertically, it either moves in front of, or behind $n-1$ tiles. Of these $n-1$ tiles, suppose there are p numbers that are greater than b, where $0 \leq p \leq n-1$, then $n-1-p$ of them are smaller than b. The change in inversion number after b has moved vertically is $p-(n-1-p) = 1-n$ if b moves down, or $-p+(n-1-p) = n-1$ if b moves up. Since $|n-1|$ is odd, the move always changes the parity of the number of inversions.

Lemma 8. *For any grid with even size $n \times n$, if the puzzle's initial state has an odd inversion number and its empty cell is on an odd-numbered row counting from the bottom, or has an even inversion number and its empty cell is on an even-numbered row counting from the bottom, it is unsolvable.*

Proof. First we note that the goal state has an inversion number of 0, with the empty cell on the first row from the bottom. Thus the goal state has an even inversion number with the empty cell on an odd-numbered row counting from the bottom.

We also note that having an odd inversion number and the empty cell on an odd-numbered row counting from the bottom is equivalent to an even inversion number and the empty cell on an even-numbered row counting from the bottom.

By **Lemma 4**, we know that horizontal moves do not change the inversion number.

If the initial state has an odd inversion number and the empty cell is on an odd-numbered row, then any vertical move made will result in an even inversion number with the empty cell on an even-numbered row, and vice versa. Therefore, if an initial state starts with an odd inversion number and the empty cell is on an odd-numbered row, it can never have an even inversion numbers with the empty cell on an odd-numbered row, and thus never reach the goal state. Therefore such puzzles are unsolvable.