An Evaluation of Uninformed and Informed Search Algorithms on the k-puzzle Problem

AY19/20 Semester 2 CS3243 Project 1, Group 37

Lim Fong Yuan, Zhuang Xinjie, Otto Alexander Sutianto, and Mario Lorenzo

School of Computing, National University of Singapore

1 Problem Specification

Fix $n \in \mathbb{Z}_{\geq 2}$. Let $k = n^2 - 1$. A valid state v is a $n \times n$ array with the entries $v_{(x,y)}$ containing a permutation of the integers $[0, k]$. V is the set of valid states. Each valid state has a unique coordinate $e = (x_e, y_e)$ such that $v_e = 0$. Call v_e the empty cell of v. An initial state s is a valid state with $e = (n, n)$. The goal state g is where $g_{(x,y)} = (x-1)n + y$ for all $(x, y) \in (\mathbb{Z}_{[1,n]})^2$, except for $g_{(n,n)}$ which is 0. Let $A = \{\texttt{up} = (-1, 0), \texttt{down} = (1, 0), \texttt{left} = (0, -1), \texttt{right} = (0, 1)\}$ be the set of actions. The transition function $T : V \times A \to V$ is where $T(v, a) = v'$ with v' being identical to v except with $(v'_{e-a}, v'_e) = (v_e, v_{e-a})$.

The problem is to determine if g is reachable from a given initial state s, and if so, specify a sequence of actions $p \in A^*$ that leads from s to g.

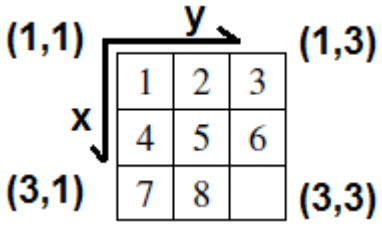

Fig. 1. The coordinate system on the goal state, in the case of $n = 3$.

2 Technical Analysis

All the evaluated algorithms initially check for the solvability of the given puzzle, using the criterion that was proposed and proved in [1]. This is to guarantee to the search algorithms that the goal state is reachable, so they do not need to exhaust an extremely large solution space. The criterion is restated and reproved in Appendix A.1.

2.1 Breadth-First Search (BFS) (Uninformed)

Correctness [BLAH]

Efficiency Given any valid state, there are up to $|A| = 4$ possible actions. However, the initial state only has 2 possible actions, and every state thereafter has an action that backtracks and can thus be discounted, thus the maximum branching factor b of the problem space is $|A| - 1 = 3$. // This algorithm thus has $O(3^d)$ time and $O(3^d)$ space complexity.

2.2 A* Search (Informed)

Correctness [BLAH]

Efficiency [BLAH]

Heuristics In this subsection, let v be any (valid) state and v' be any state reachable from v by a single (valid) action. Their difference is the position of a single tile T (the empty cell is not counted), which has swapped placed with the empty cell which it is adjacent to. Their absolute difference in cost $d(v, v') = 1$. Let $d_h(v, v') := h_1(v') - h_1(v)$ be their difference in heuristic cost.

h_1: Sum of Manhattan distances of each (non-empty) tile from its final position. Proof of consistency: T is moved either closer to ($d_{h_1} = -1$) or farther away from ($d_{h_1} = 1$) its final position by a Manhattan distance of 1. In any case, $|d_{h_1}(v, v')| \leq 1 \implies h_1(v) \leq 1 + h_1(v') = d(v, v') + h_1(v')$, $\therefore h_1$ is consistent. □

h_2: Number of misplaced (non-empty) tiles. Proof of consistency: T is either placed on ($d_{h_2} = -1$), moved out of ($d_{h_2} = 1$), or remains away ($d_{h_2} = 0$) from its final position. In any case, $|d_{h_2}(v, v')| \leq 1 \implies h_2(v) \leq 1 + h_2(v') = d(v, v') + h_2(v')$, $\therefore h_2$ is consistent. □

h_3: Heu3 The 3 heuristics should be distinct and non-trivial; for example, your heuristic should not be a constant, a simple linear transformation of other heuristics, or any simple function of another heuristic like a square root of another

Consistency/Admissibility: Your heuristic must be provably admissible, or (preferably) consistent. In either case you must formally prove this property. Note that consistency implies admissibility but not the other way around. Thus, if you show admissibility, you should provide a (preferably simple) counterexample where your heuristic violates consistency.

3 Experimental Setup

The performances of BFS and A*-search with each of the three heuristics have been experimentally measured, and consolidated in Table 1. The inputs used draw from a specified subset of valid initial states, under a uniform random distribution, and the average results from 200 inputs were taken for each experiment. The metrics used to evaluate each algorithm are the absolute time taken to execute the algorithm, and the number of nodes seen. These metrics directly reflect the time and memory requirements of each algorithm respectively.

The algorithms were coded in Python 3.x.x and the experiments were run on a [OS] machine with a [processor] and 16 GB of RAM. Absolute time was measured with `[Python time library]`.

Table 1. Experiment results.

	BFS		A* with h_1		A* with h_2		A* with h_3	
Inputs	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen
3x3 Solvable	1	1	1	1	1	1	1	1
3x3 All	1	1	1	1	1	1	1	1
4x4 Solvable	1	1	1	1	1	1	1	1
4x4 All	1	1	1	1	1	1	1	1
5x5 Solvable	1	1	1	1	1	1	1	1
5x5 All	1	1	1	1	1	1	1	1
5x5 Unsolvable	1	1	1	1	1	1	1	1

4 Results and Discussion

As Table 1 shows, [A* with h_1 is better than BFS]. This agrees with our theoretical analysis.

A Appendix: Proofs

A.1 Solvability of an Initial State [1]

Lemma 1. *It always takes an even number of vertical actions to get from any (solvable) initial state s to the goal state g (or equivalently, from g to s).*

Proof. Colour the board black and white in alternating rows, with the bottom row coloured white. On every vertical action, the empty cell either moves from a white row to a black row, or from a black row to a white row. The empty cell starts on a white row and ends on a white row, which requires an even number of vertical actions. □

Definition 1. *For any given state v, write out its entries in a sequence $S(v)$, ignoring the empty cell. i.e. $S(v)_i = v_{(\lceil i/n \rceil, ((i-1) \mod n)+1)}$ for $i \in \mathbb{Z}_{[1,n^2]}$, then remove the empty cell from $S(v)$ (so it is now indexed from 1 to k). An **inversion** is a pair of indices (i, j) such that $i < j$, but $S(v)_i > S(v)_j$.*

Every state v has a well-defined number of distinct inversions, denote it $I(v)$. $I(v)$ can be calculated by tallying how many distinct pairs in v are inversions, which takes $O(k^2)$ time as there are ${}^kC_2 = \frac{k(k+1)}{2}$ pairs to check.

Theorem 1. *An initial state s is solvable if and only if $I(s)$ is even.*

($\implies$) A solvable state is one that can reach the goal state g in a finite number of actions. Since each action is reversible by an opposite action, this can be restated as: A state is solvable if and only if it is reachable from g in a finite number of actions.

Now given a solvable initial state s, consider a sequence of actions taken to move from g to s, and how $I(v)$ changes across it. For each action, consider its start state v and result state v'. If an action moves horizontally, $S(v) = S(v')$, thus $I(v') = I(v)$.

If an action moves vertically, then $n-1$ swaps between adjacent elements occur from $S(v)$ to $S(v')$, and each swap either introduces or resolves an inversion, which flips the parity of $I(v')$. If n is odd, then the number of swaps $n-1$ is even and thus the parity of $I(v')$ is always the same as $I(v)$. If n is even, then $n-1$ is odd and thus the parity of $I(v')$ is always different from $I(v)$.

It has thus been shown that the parity of $I(v)$ flips only when n is even and a vertical action is taken. Even then, by Lemma 1, there are an even number of vertical actions, and thus the parity of $I(v)$ flips an even number of times over the entire path from g to s. Therefore, $I(s)$ has the same parity as $I(g)$. And $I(g) = 0$ is even. □

($\impliedby$) This is essentially a tedious proof by construction, and the proof is deferred. The general strategy is to move 1 to the right place and never move it again, then do the same with 2, then 3, and so on, up until the puzzle is solved. □

References

1. Ryan, M.: Solvability of the tiles game, https://www.cs.bham.ac.uk/ mdr/teaching/modules04/java2/TilesSolvability