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.

Throughout this report, the following variable definitions are used:

Let d be the depth of the shallowest goal node g.

Let b be the maximum branching factor of the problem space. $b = 3$, see A.1.

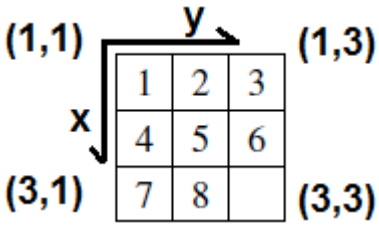

Fig. 1. The coordinate system on the goal state, in the case of $n = 3$.

2 Technical Analysis

2.1 Breadth-First Search (BFS) (Uninformed)

BFS explores all reachable nodes of depth d' before it starts exploring nodes of depth $d' + 1$. Hence, there are at most $\sum_{i=0}^{d} b^i = \frac{b^d - 1}{b-1} = \frac{3^d - 1}{3-1} \in O(3^d)$ nodes that BFS will explore before it explores g.

Correctness The algorithm first checks for solvability (see Appendix A.2), terminating if it is not. If a goal node g exists, BFS explores a finite number of nodes before finding the shallowest goal node g. Then, it backtracks through its search tree to derive the sequence of actions that leads from s to g. All actions that BFS has explored are valid, and the sequence leads from s to g, therefore BFS returns a correct sequence.

Efficiency Each node is explored in constant time, thus BFS has $O(3^d)$ time complexity. BFS tracks all explored and seen nodes in memory, Which amount to at most $\sum_{i=1}^{d+1} b^i \in O(b^{d+1}) = O(b \cdot b^d) = O(3 \cdot 3^d) = O(3^d)$, thus BFS has $O(3^d)$ space complexity.

2.2 A* Search (Informed)

A* explores the node v with the least estimated cost $f(v) = g(v) + h(v)$ to the shallowest goal node g, where $g(v)$ is the cost from the initial state s to v, and $h(v)$ is an admissible estimate of the cost from v to g.

Let $d(v, v')$ be the optimal distance between any two nodes v and v'.

Assuming g exists, let p be an optimal path from s to g. For any node v in p, note that $f(v) = g(v)+h(v) \leq g(v)+h^*(v) = d(s, n)+d(n, g) = d(s, g) = d$. Also note that at least one node in p will be in the frontier of A* at any given point, hence there is always at least one node v in the frontier with $f(v) \leq d$. Therefore, A* will explore at most all reachable nodes v such that $f(v) \leq d$ before finding g. In particular, it will search at most all nodes v of depth $d' = d(s, v) = g(v) \leq g(v)+h(v) = f(v) \leq d$. Hence, there are at most $\sum_{i=0}^{d} b^i = \frac{b^d-1}{b-1} = \frac{3^d-1}{3-1} \in O(3^d)$ nodes that A* will explore before it explores g.

Correctness Same as with BFS above.

Efficiency Same as with BFS above, A* has $O(3^d)$ time and $O(3^d)$ space complexity, in the worst case (i.e. constant heuristic). The time complexity may be improved depending on the heuristic, generally taking on the form $O(b^{d-h(s)})$, or $O((b^*)^d)$ (where b^* is the effective branching factor of A* using h, which has to be empirically measured), see [1]. At best, under the optimal heuristic h^* (where $h^*(v)$ returns the actual cost from v to g), A* runs in $O(d)$ time.

Heuristics For any (valid) state v and any successor v' of v, their difference is the position of a single tile T (the empty cell is not counted), which has swapped places with the adjacent empty cell. Their absolute difference in cost $d(v, v') = 1$. Let $d_h(v, v') := h(v') - h(v)$ be their difference in heuristic cost.

h_1: Sum of Manhattan distances of each (non-empty) tile from its final position. This heuristic is consistent.

h_2: Number of misplaced (non-empty) tiles. This heuristic is consistent.

h_3: Heu3 Heu3 Heu3 Heu3 Heu3 Heu3 Heu3 Heu3 Heu3 Heu3 Heu3. 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. □

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 and plotted in Figure 2.

Inputs are drawn with a uniform random distribution from the set of $n \times n$ solvable initial states. The optimal depth d is measured using BFS, and the initial state is categorized accordingly. 2000 initial states for each listed depth are tested. The metrics used are the absolute time taken to obtain an output, 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 2.6 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]`.

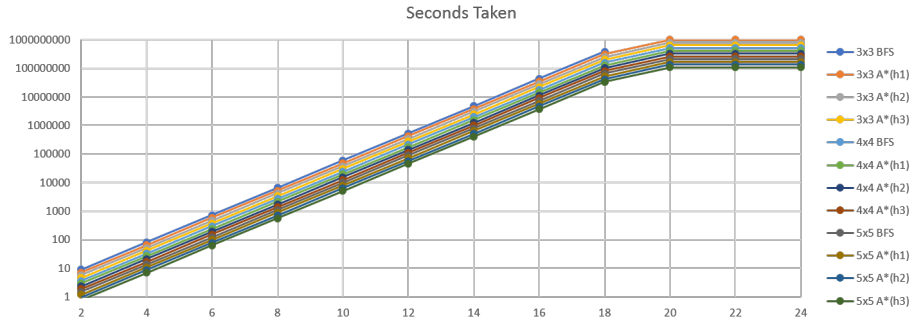

Fig. 2. Plot of the performance results of the proposed algorithms.

4 Results and Discussion

The average time and space complexities for all algorithms are $O(1.667^d)$ for 3x3, $O(2^d)$ for 4x4, and $O(2.2^d)$ for 5x5. (See see A.1 for the average branching factor.) The empirical results in Figure 2 supports this analysis.

Table 2 also shows that the algorithms in order of most to least time-efficient is A* with h_1, then A* with h_2, then BFS. This agrees with our theoretical analysis regarding the dominance of one heuristic over another.

Table 1. Performance results of the proposed algorithms.

3x3	BFS		A* with h_1		A* with h_2		A* with h_3	
Depth d	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen
2	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1
22	1	1	1	1	1	1	1	1
24	1	1	1	1	1	1	1	1
4x4	BFS		A* with h_1		A* with h_2		A* with h_3	
Depth d	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen
2	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1
22	1	1	1	1	1	1	1	1
24	1	1	1	1	1	1	1	1
5x5	BFS		A* with h_1		A* with h_2		A* with h_3	
Depth d	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen	Time taken (s)	#Nodes seen
2	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1
22	1	1	1	1	1	1	1	1
24	1	1	1	1	1	1	1	1

A Appendix: Proofs

A.1 Maximum and Average Branching Factors

Maximum branching factor. 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 $b = |A| - 1 = 3$. □

Note: In the case $n = 2$, each cell is a corner cell and thus has only 2 actions, and 1 of them backtracks and thus is discounted. Thus $b = 1$ in this case. (Only the initial node has 2 branches.) The complexities evaluated as $O(b^d)$ become wrong in this case as $O(1^d) = O(1)$, and they should be $O(d)$ instead.

Average branching factor. Given n, there are 4 corners of 2 branches each, $4(n-2)$ side cells of 3 branches each, and $(n-2)^2$ center cells of 4 branches each. We may subtract 1 from each of the total branches as we discount backtracking actions. (This is not applicable only to the initial state, which incurs only a constant factor.) Assuming all states are equally likely to occur, the average branching factor b_{avg} is thus $\frac{1\cdot 4+2\cdot 4(n-2)+3\cdot(n-2)^2}{n^2} = \frac{4+8n-16+3n^2-12n+12}{n^2} = \frac{3n^2-4n}{n^2} = 3 - \frac{4}{n}$. □

Table 2. Average branching factor for some values of n.

n	2	3	4	5
b_{avg}	1	5/3	2	2.2

A.2 Solvability of an Initial State [2]

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.*

($\Longrightarrow$) 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. □

($\Longleftarrow$) 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. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, pp. 98,103. Prentice Hall, 3rd edn. (2010)
2. Ryan, M.: Solvability of the tiles game, https://www.cs.bham.ac.uk/ mdr/teaching/ modules04/java2/TilesSolvability.html