Enhanced Knowledge in Sciences and Technology Vol. 3 No. 1 (2023) 097-107
© Universiti Tun Hussein Onn Malaysia Publisher's Office

EKST

Homepage: http://publisher.uthm.edu.my/periodicals/index.php/ekst
e-ISSN : 2773-6385

Benchmarking the solution time of N-puzzles: A* versus Iterative Deepening A* algorithms

Siow Pui Zhen[1], Lee Siaw Chong[2*]

[1]Pem Fasterner (M) Sdn. Bhd.
Lot 70, Lorong Senawang 4 Kawasan Perusahaan Senawang, Senawang, 70450 Seremban, Negeri Sembilan, MALAYSIA

[2]Department of Mathematics and Statistics,
Faculty of Applied Sciences and Technology,
Universiti Tun Hussein Onn Malaysia (Pagoh Campus),
84600 Pagoh, Muar, Johor, MALAYSIA.

*Corresponding Author Designation

DOI: https://doi.org/10.30880/ekst.2023.03.01.012
Received 15 January 2023; Accepted 9 May 2023; Available online 3 August 2023

Abstract: *N-puzzles* are difficult to solve manually due to the large number of potential paths that increase with the puzzle size. Intuitive solutions are likely to include repeated paths, and various strategies can help shorten the paths. In this research, the performance of the A* and IDA* algorithms, along with their respective heuristics, in solving the *N-puzzle* was benchmarked using 15 examples of puzzles with actual distances up to 30, using Manhattan and Hamming distances as heuristics. The experiments were conducted on a laptop with an AMD Ryzen 3 2200U processor and 8GB DDR4 RAM, programmed in Python. The results showed that A* and IDA* algorithms with the Manhattan distance were at least 90% faster than those with the Hamming distance, and that the IDA* algorithm was also faster than the A* algorithm, taking at least 75% to 98% less time in solving the 8-*puzzles* and 70-100% less time to solve the 15-*puzzle* and 24-*puzzle*. The longer runtime of the A* algorithm was attributed to its slower retrieval of a node with a smaller $\boldsymbol{f}(\boldsymbol{n})$ value from the open list, especially when multiple nodes had the same value. The research suggested that future work could focus on developing an approach to break ties when multiple nodes have the same $\boldsymbol{f}(\boldsymbol{n})$ value.

Keywords: *N-Puzzle*, Admissible Heuristic, IDA* Algorithm, And A* Algorithm

1. Introduction

The *N-puzzle* is a puzzle game composed of *N* number of tiles where *N* can be 8, 15, 24, and so on. For instance, the 15-*puzzle* consists of 15 tiles numbered from 1 to 15 and a blank space on a square board. To solve the puzzle, the player must arrange all tiles to match the goal configuration by sliding them horizontally or vertically into the blank space without removing any tiles. However, some initial

*Corresponding author: sclee@uthm.edu.my puzzle configurations are unsolvable due to not adhering to the rules of solvability, which are described in Section 2.1. For example, Sam Loyd's 15-*puzzle* cannot be solved due to tiles 15 and 14 of the puzzle are arranged in reverse order. Solving *N-puzzles* manually is a challenging process because there are numerous potential paths that may lead to the solution, and as the puzzle size increases, so does the number of paths. Although a player might solve the puzzle intuitively, the solution is likely to include repeated paths. Various strategies can help shorten the paths, but our goal is to find the shortest or optimal path for a general *N-puzzle*.

Pathfinding techniques are one of the ways to get the shortest path between two points in a graph. It widely applied in many real-life scenarios like transportation [1], logistics, intelligent navigation system [2], robotics and others. There are two types of pathfinding techniques: informed and uninformed searches. Informed search uses domain-specific knowledge to guide the search process, while uninformed search does not. The A* and IDA* algorithms are examples of informed search algorithms that explore fewer nodes [3] and are more efficient than uninformed search algorithms like Dijkstra's algorithm.

Dijkstra's algorithm is a pathfinding technique that calculates the shortest path through a graph from a given vertex to all other vertices [4]. The algorithm finds the shortest path by continually expanding the network from the start to adjacent vertices and selecting the node with the shortest path from the start to that node, until the goal node is reached. If a better path with a shorter distance is discovered, Dijkstra's algorithm updates the nodes in the path. However, it becomes ineffective for large-scale problems [5]. This is because Dijkstra’s algorithm may visit a huge number of the network’s nodes to get the shortest path between vertices that are reasonably far from each other, thereby consuming a lot of time and wasting necessary resources [6, 7].

Another pathfinding technique that can be used is the A* algorithm. The A* algorithm is one of the Best First Search algorithms (BFS) that explores a graph by expanding the most promising node first, according to some evaluation function like heuristics. The A* algorithm gets the shortest path like the Dijkstra’s algorithm and using heuristic information like the best first search to guide itself [4]. The heuristic information helps to reduce the search effort and estimate the most promising node to explore next [8], but it needs to be a close approximation to the target to work well for the A* algorithm [5]. There are various heuristics that can be used with the A* algorithm such as Hamming distance and Manhattan distance, which are explained in detail in Section 2.2. The A* algorithm is more efficient in finding the shortest path compared to Dijkstra's algorithm [9, 10] , but it also requires more memory to prevent duplicated searches [11, 12] . Because of this limitation, other algorithms have been considered, and one of them is the Iterative Deepening A* (IDA*) algorithm.

The IDA* algorithm is a combination of two other algorithms, the Depth-First Iterative Deepening (DFID) and A* algorithms. The DFID algorithm sets a limit on the depth of nodes to be expanded, and it expands all nodes up to that limit before discarding them and starting a new search with a deeper limit. This process repeats until the goal node is found. In the IDA* algorithm, the A* algorithm's cost function $f(n)$ is used as a threshold, and the algorithm applies the Depth-First Iterative Deepening algorithm's technique to search. The details of how the IDA* algorithm works are explained in Section 2.4.

The aim of this research is to benchmark the time taken by the A* and IDA* algorithms to solve 8, 15, and 24-*puzzles* and to benchmark the time taken for different heuristics used in each algorithm. A total of 15 puzzles with an actual distance up to 30 were used for the benchmarking process, using the A* and IDA* algorithms, and the heuristics of Hamming and Manhattan distances. The evaluation was conducted on three different *N-puzzles*, specifically the 8, 15, and 24-*puzzle*, without considering the pattern, type, or difficulty of the puzzles.

2. Methodology

We will first check the solvability of the puzzle before solving it. As discussed in the previous section, two heuristics are considered, and we will explain how both heuristics are calculated for *N-puzzle*'s case. Then we will discuss the implementation details of the A* and IDA* algorithms.

2.1 Solvability of the puzzle

The solvability of sliding puzzles can be determined based on two rules. The first rule is applicable to odd-sized puzzles, which can be solved if the initial state of the puzzle has an even number of inversions. An inversion number refers to the number of times a tile in a sliding puzzle is ahead of another tile with a lower number, when counting from left to right, top to bottom, and excluding the blank tile. For even-sized puzzles, the second rule applies if satisfy one of the followings:

- If the puzzle has an even number of inversions, then the blank must be on an odd-numbered row counting from the bottom row.
- If the puzzle has an odd number of inversions, then the blank must be on an even-numbered row counting from the bottom row.

To demonstrate how the inversion numbers are calculated, see Figure 1. Figure 1 shows different *N-puzzles*, such as 8-*puzzles* and 15-*puzzles*, each with its own numbered tiles, for example, the 8-*puzzle* has eight tiles numbered from 1 to 8. The tile 0 represents the blank tile or space in the puzzle. In Figure 1, puzzle (b) is the goal state of puzzle (a), while puzzle (d) is the goal state of puzzle (c). The tile 5 in the goal puzzle (b) is located at row 2 and column 2. Since there are four tiles with lower numbers (1, 2, 3, and 4) positioned after tile 5 in puzzle (a), the inversion number of tile 5 is 4. On the other hand, for Figure 1(c), the inversion number of tile 5 is 0 as there are no tiles with a lower number following it. By adding up the inversion numbers for each tile in both puzzles, the total inversion number for Figure 1(a) is 8, while for Figure 1(c) is 33.

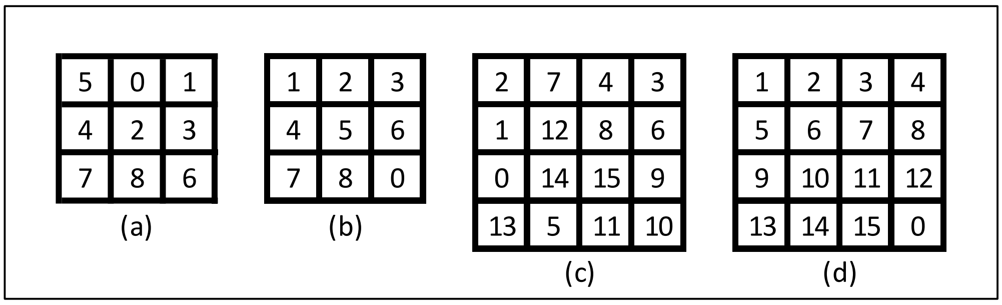


Figure 1: Examples of *N-puzzle* (a) 8-*puzzle*. (b) goal node of the 8-*puzzle*. (c) 15-*puzzle* (d) goal node of 15-*puzzle.*

Since the puzzle in Figure 1(a) is an odd sized puzzle and the number of inversions is even, it is solvable. For the even sized puzzle in Figure 1(c), the blank tile 0 is located at an even-numbered row. As a result, the puzzle in Figure 1(c) is also solvable.

2.2 Admissible heuristic

Heuristics are the functions that estimate the actual cost of the path. Heuristic is also an estimation that provides information to guide the algorithm to get the goal in the right direction [13]. A heuristic that overestimates the actual cost might end up finding a path that is not optimal. On the other hand, a heuristic that underestimates the actual cost is guaranteed to find the optimal path, and such heuristic is known as an admissible heuristic. Furthermore, the closer the heuristic approximates the actual cost, then the faster the optimal path can be found. We consider two types of heuristics here, the Hamming distance and the Manhattan distance. The blank tile 0 is excluded in the calculation since it represents the blank space in the puzzle.

Generally, the Hamming distance is defined as the number of elements which two objects differ. For example, the difference between the two words ‘Jimmy” with ‘Jammi’ is 2 because there are two differences between them. Therefore, we can formulate the Hamming distance for an *N-puzzle* as

$$h = \sum_{i=1}^{N} d(x_i, y_i), \qquad Eq. 1$$

where

$$d(x_i, y_i) = \begin{cases} 1, & x_i \neq y_i, \\ 0, & x_i = y_i, \end{cases}$$

and x_i is i^{th} tile in the first puzzle, and y_i is i^{th} tile in the second puzzle. For example, the number of misplaced tiles for Figure 1(a) is 5 because tiles 1, 2, 3, 5, and 6 are misplaced compared to the goal node in the Figure 1(b). For Figure 1(c), the Hamming distance is 14 compared to the goal node of the Figure 1(d) because a total of 14 tiles are misplaced, except tile 13.

The Manhattan distance is the total distance of each tile of current state to its goal position [14]. The formula of Manhattan distance for each tile is

$$h = \left| x_1^{(M)} - x_2^{(N)} \right| + \left| y_1^{(M)} - y_2^{(N)} \right| \qquad Eq. 2$$

where x is and y are the row and column position of the tile in puzzle. Using the examples in Figure 1, tile 14 in the Figure 1(c) is located at row 3 and column 2, while in the goal node of Figure 1(d), tile 14 is located at row 4 and column 2. By applying the *Eq.* 2, we get the Manhattan distance for tile 14 between Figures 1(c) and 1(d) as

$$h = |3 - 4| + |2 - 2| = 1.$$

The Manhattan distance between two *N-puzzle* is

$$h = \sum_{i=1}^{N} \left| x_i^{(M)} - x_i^{(N)} \right| + \left| y_i^{(M)} - y_i^{(N)} \right| \qquad Eq. 3$$

where $x_i^{(P)}$ refers to the i^{th} tile in puzzle P. Using *Eq.* 3, the total Manhattan distance between the puzzles in Figure 1(c) and Figure 1(d) is 24.

2.3 A* Algorithm

The A* algorithm expands each node from start to adjacent node where each adjacent node is evaluated with a cost function $f(n)$. The function represents the estimated cost for any node n to the goal node by adding the n’s heuristic value $h(n)$ to $g(n)$, the number of nodes currently in the path to the node n. In other words,

$$f(n) = g(n) + h(n). \qquad Eq. 4$$

To prevent revisiting the same node, the A* algorithm utilizes two lists, the open list and the closed list. The open list contains frontier nodes that need to be explored and a closed list holds the visited nodes to prevent revisiting. The A* algorithm will choose and expand the node with minimum f value from the open list. Then, the node that the A* algorithm selected will shift from the open to closed list while the nodes that just explored will added into the open list. If the A* algorithm found that new same node exists in open or closed lists, the node will be excluded and will not add into the open list. This step is repeated until the goal node is found. The overall flow of the algorithm is shown in Appendix A.

2.4 IDA* Algorithm

The Iterative Deepening A* (IDA*) algorithm is a search technique that search iteratively with increasing cost of thresholds until the goal node is found [15]. The algorithm starts by setting a threshold value, denoted as $f(n)$ in *Eq.* 4, and using the initial node's $f(n)$ value as the initial threshold value. If the node's $f(n)$ value is greater than the given threshold, the search halts at that node and moves on to the next node or branch. Unlike the A* algorithm, the IDA* algorithm does not keep track of visited nodes, only checking for duplicates of the previous two of its current nodes. Once all branches have been explored, the threshold is updated with the minimum $f(n)$ of all nodes that exceeded the current threshold [16]. This process is repeated until the goal node is found. The overall flow of the algorithm is presented in Appendix B.

3. Results and Discussion

This research was conducted using a Python test program on an Acer Aspire 3 laptop equipped with an AMD Ryzen 3 2200U processor and 8GB DDR4 RAM. Two major comparisons were made in this section. The first comparison was between the performance of the Hamming and Manhattan distances in the A* and IDA* algorithms. The second comparison was between the performance of the A* and IDA* algorithms in 5 sets of 8, 15, and 24-*puzzles*. This research only focused on the time taken to solve the puzzles using the A* and IDA* algorithms, disregarding the pattern, type, or difficulty of the *N-puzzles*.

3.1 Hamming vs Manhattan distances

In this section, we compare the performance of Hamming and Manhattan heuristics in both the A* and IDA* algorithms using the puzzles in Figure 2. In Section 2.1, we noted that both heuristics are admissible, and this is justified in Table 1, which shows the actual distance of the three puzzles from their goal. As we can see, both the Hamming and Manhattan distances are less than the actual cost (making them admissible), and the Manhattan distance is closer to the actual distance than the Hamming distance. For puzzle A3, the Manhattan heuristic even provides an accurate approximation.

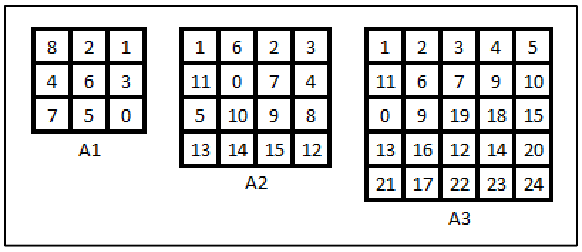


Figure 2: *N-puzzles* that tested in Section 3.1

Table 1: The actual, Hamming and Manhattan distances for different puzzle A1-A3

Puzzle	A1 (8-*puzzle*)	A2 (15-*puzzle*)	A3 (24-*puzzle*)
Actual distance	18	18	20
Hamming distance	5	9	14
Manhattan distance	8	12	20

Table 2 shows the time elapsed (in seconds) for the A* and IDA* algorithms to find the optimal path using the Hamming and Manhattan distances for the puzzles in Figure 2. The percentage reduction column measures the reduction rate of the solution time of Manhattan distance compared to the Hamming distance. For all three puzzles, the times for Manhattan distance are shorter by at least 90%

compared to the Hamming distance in both A* and IDA* algorithms. As we expect, for both the A* and IDA* algorithms, the Manhattan distance performs better than the Hamming distance. This is because the Manhattan distance approximates the actual distance much better than the Hamming distance. When the Manhattan distance accurately approximates the actual distance (in the case of puzzle A3), the reduction rate of the Manhattan distance is 100% and 98% for the A* and IDA* algorithms respectively.

Table 2: Time used by different heuristics for different size puzzle using the A* and IDA* algorithms.

	A1 (8-*puzzle*)			A2 (15-*puzzle*)			A3 (24-*puzzle*)		
	H(s)	M(s)	% R	H(s)	M(s)	% R	H(s)	M(s)	% R
A* algorithm	6.72	0.32	95	28.13	0.65	98	38.66	0.08	100
IDA* algorithm	0.34	0.03	92	0.69	0.04	94	0.48	0.01	98

Variations: H = Hamming distance; M = Manhattan distance; % R = percentage of reduction rate

3.2 A* vs IDA* algorithms

In this experiment, a total of 15 examples with three different *N-puzzles* were tested, including 8-*puzzles* in Figure 3, 15-*puzzles* in Appendix C, and 24-*puzzles* in Appendix D. The goal of the experiment was to compare the performance of A* and IDA* algorithms on *N-puzzles* with different sizes. All the *N-puzzles* were randomly generated by moving the blank tiles of goal puzzles with 30 random moves to ensure that the *N-puzzles* tested were solvable. Manhattan distance was applied in both the A* and IDA* algorithms since it performs better than the Hamming distance. Each experiment is repeated 100 times and the average time is recorded in Table 3.

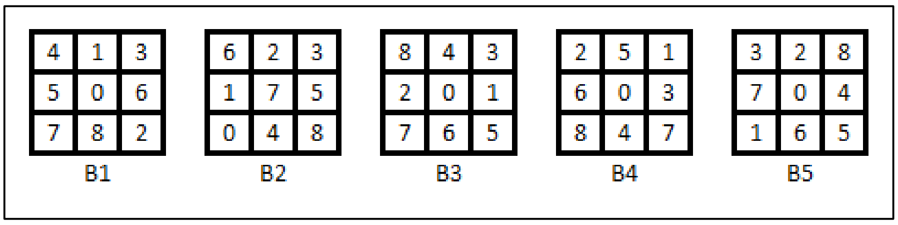


Figure 3: Test 8-*puzzles*

Table 3: Time taken by the A* and IDA* algorithms for puzzles B1-B5

Puzzle	Actual distance	A* algorithm (s)	IDA* algorithm (s)	% Reduction
B1	14	0.08	0.01	78
B2	16	0.12	0.02	83
B3	18	0.31	0.02	92
B4	22	7.68	0.25	96
B5	24	21.81	0.45	98

Table 3 presents a comparison of the time required by the A* and IDA* algorithms to solve puzzles B1-B5. The data in Table 3 demonstrate that the IDA* algorithm was able to solve the 8-*puzzle* examples shown in Figure 3 at least 78%, and at most 98% faster than the A* algorithm.

The reason that the A* algorithm finds the optimal path much slower than the IDA* algorithm is that the A* algorithm must expand all nodes in the open list that have the minimum value of $f(n)$. When there are many nodes with tied values of $f(n)$, the A* algorithm must expand all of them first. This situation is exacerbated when the $f(n)$ values for the newly expanded nodes continue to tie with each other. On the other hand, for the IDA* algorithm, the search is "deepened" until the threshold value increases. When the goal node is found, there might be a lot of nodes that have tied threshold value that remain unexplored. This scenario is illustrated in Figure 4 and 5.

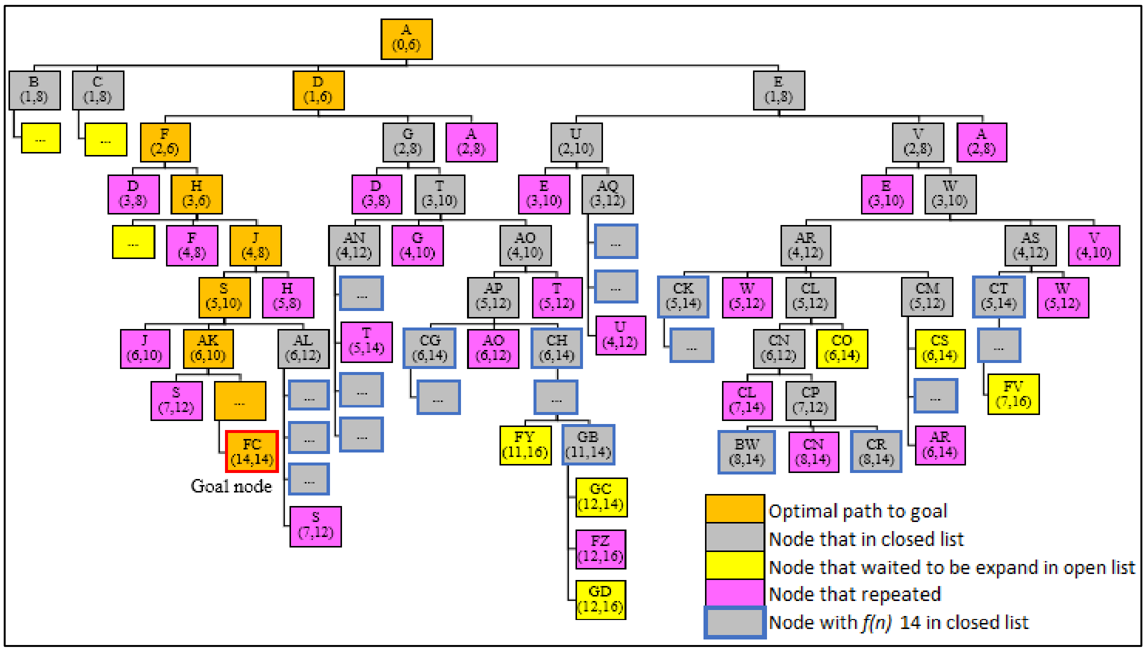


Figure 4: Branch network of the puzzle B1 in the A* algorithm

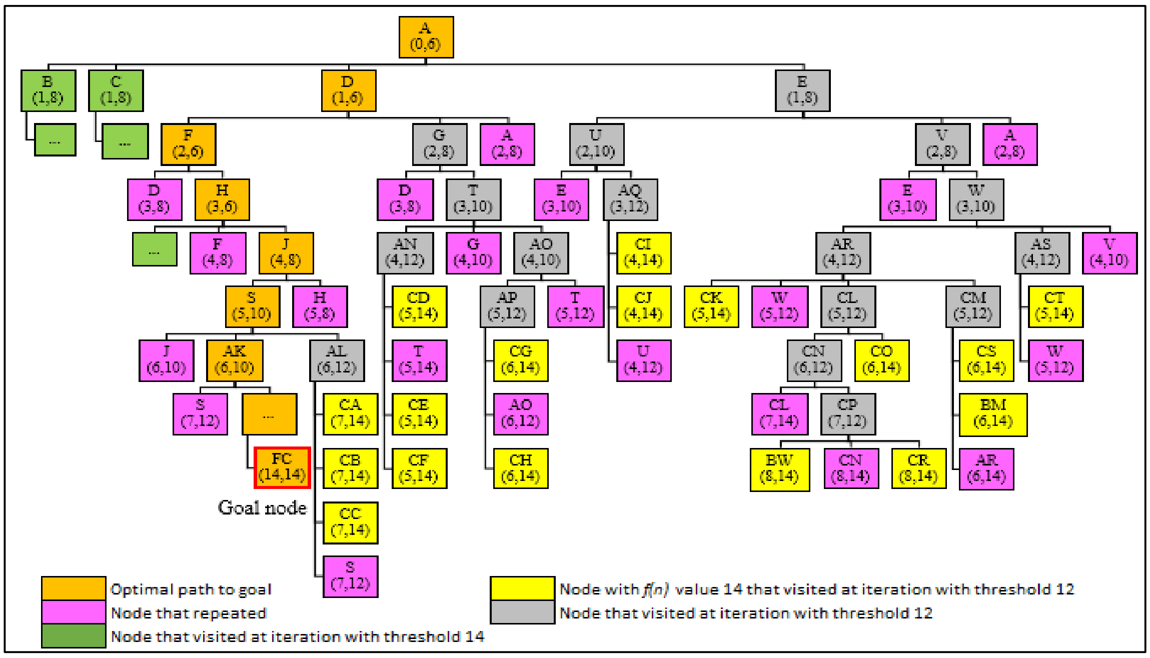


Figure 5: Branch network of the puzzle B1 in the IDA* algorithm

Figure 4 shows how the A* algorithm gradually expands the search up to the goal node, which is highlighted with a red border. Each node is represented by a square with a label that includes its g and f values for example $A(0, 6)$ for node A with g value 0 and f value 6. Currently, the search is completed, and the nodes are expanded up to f value equal to 14, which is the f value of the goal node. The nodes that are in the closed list are coloured in grey, and the grey nodes that have blue border are the ones with f value equal to 14. The nodes in the open list that are awaiting to be expanded are coloured yellow. We counted 67 blue-bordered nodes in total, which took a long time to expand because some expansions required traversing multiple levels. For example, the node with label $CH(6,14)$ has expanded several nodes with an f value of 14 from level 6 to level 11, as there are many newly expanded nodes with tie f value of 14.

Figure 5 shows the expansion tree of the IDA* algorithm for the same puzzle. Unlike Figure 4, which depicts the A* algorithm, Figure 5 differs slightly as the two algorithm search nodes differently. The A* algorithm searches nodes based on the open list while the IDA* algorithm searches nodes by depth through iteration with a threshold limit. In Figure 5, the green nodes indicate the ones searched by the IDA* algorithm when the threshold was 14. Among these green nodes, a total of 51 nodes with an f value 14 were expanded in the IDA* algorithm. In comparison, the yellow nodes in Figure 5, which are not expanded by the IDA* algorithm, were expanded by the A* algorithm, as depicted by the blue bordered nodes in Figure 4. The yellow node in Figure 5 is expanded during threshold 12 but not expanded during the threshold was 14 since the path containing the goal node is found. Therefore, Figure 4 shows that the A* algorithm took more time on these yellow nodes in Figure 5 compared to the IDA* algorithm.

Furthermore, we also run the benchmark of larger-sized puzzles such as 15-*puzzle* in Appendix C and 24-*puzzle* in Appendix D and the result is summarized in Table 4.

Table 4: Time used by the A* and IDA* algorithms for the 15-*puzzle* and *24-puzzle.*

Puzzle	Actual distance	A* algorithm (s)	IDA* algorithm (s)	% Reduction
C1 (15-*puzzle*)	15	0.19	0.02	91
C2 (15-*puzzle*)	18	0.26	0.01	95
C3 (15-*puzzle*)	18	9.55	0.16	98
C4 (15-*puzzle*)	24	4.72	0.07	98
C5 (15-*puzzle*)	27	28.77	0.11	100
D1 (24-*puzzle*)	14	0.06	0.01	80
D2 (24-*puzzle*)	16	0.02	0.01	70
D3 (24-*puzzle*)	18	0.46	0.03	92
D4 (24-*puzzle*)	22	2.33	0.06	97
D5 (24-*puzzle*)	28	11.54	0.13	99

Table 4 presents similar results to those in Table 3, demonstrating the superiority of the IDA* over A* algorithms. Table 4 data shows that the IDA* algorithm is 70% to 100% faster than A* algorithm in solving puzzles in Appendices C and D, and consistently requires less time overall. The reason for the slower performance of the A* algorithm is the same as those observed when solving the 8-*puzzles*.

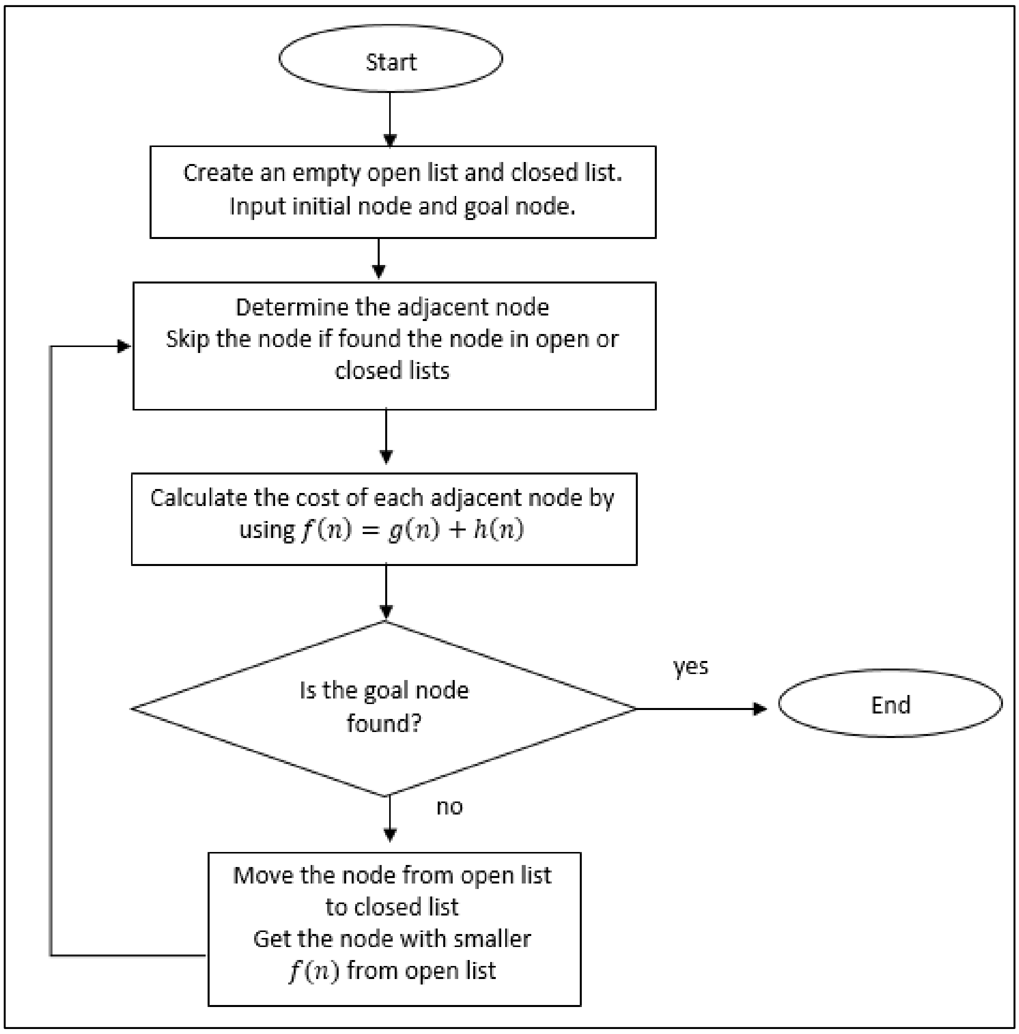
4. Conclusion

Based on our experiments, we found that the Manhattan heuristic is 90% faster than the Hamming heuristic for our test puzzles. Moreover, the IDA* algorithm outperformed the A* algorithm in all tested cases, taking 78-98% less time to solve the 8-*puzzles* and 70-100% less time to solve the 15-*puzzles* and 24-*puzzles*. This suggests that the IDA* algorithm is a more efficient approach than the A* algorithm for solving *N-puzzle* problems. The A* algorithm's limitation is that it expands all nodes with the minimum value of $f(n)$, leading to a huge increase in the number of nodes that must be expanded when there are many new nodes with tied $f(n)$ values. For this reason, it is worthwhile to look for a better heuristic function so that tied $f(n)$ values happen less frequent.

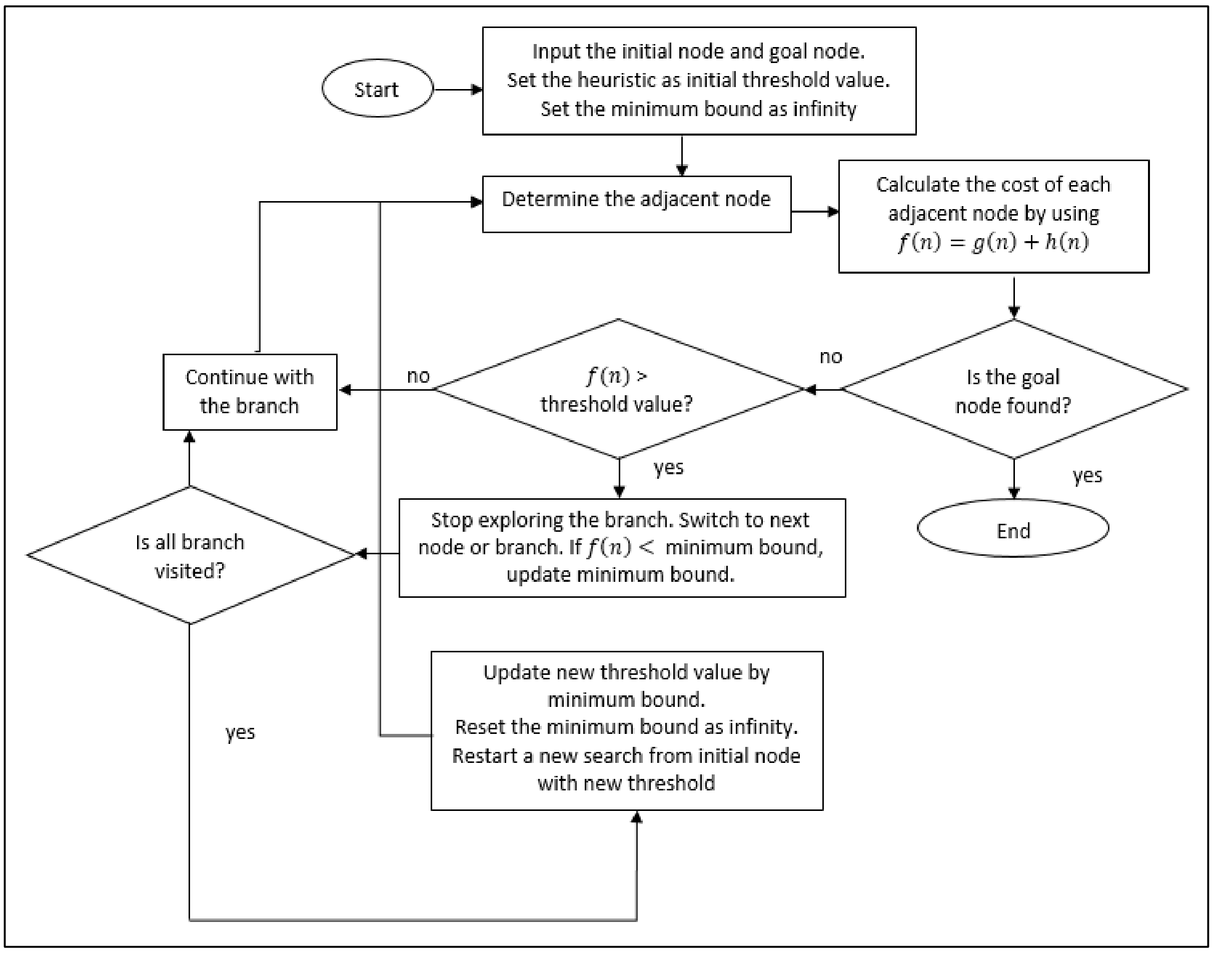
Acknowledgement

The authors would like to thank the Faculty of Applied Sciences and Technology, Universiti Tun Hussein Onn Malaysia for its support.

Appendix A


Start
Create an empty open list and closed list. Input initial node and goal node.
Determine the adjacent node
Skip the node if found the node in open or closed lists
Calculate the cost of each adjacent node by using $f(n) = g(n) + h(n)$
Is the goal node found?
yes
End
no
Move the node from open list to closed list
Get the node with smaller $f(n)$ from open list

Appendix B


Start
Input the initial node and goal node.
Set the heuristic as initial threshold value.
Set the minimum bound as infinity
Determine the adjacent node
Calculate the cost of each adjacent node by using $f(n) = g(n) + h(n)$
Is the goal node found?
no
yes
End
$f(n)$ > threshold value?
no
yes
Continue with the branch
Stop exploring the branch. Switch to next node or branch. If $f(n) <$ minimum bound, update minimum bound.
Is all branch visited?
yes
Update new threshold value by minimum bound.
Reset the minimum bound as infinity.
Restart a new search from initial node with new threshold

Appendix C

1	0	6	4
5	3	2	7
9	10	11	12
13	14	8	15

C1

0	7	5	3
2	1	10	4
9	6	11	8
13	14	15	12

C2

1	2	3	4
6	5	11	8
0	10	7	12
9	13	14	15

C3

1	2	4	8
9	7	5	3
10	13	0	12
11	14	6	15

C4

5	6	10	3
9	1	0	2
11	7	12	4
13	14	8	15

C5

Appendix D

6	1	3	4	5
11	2	8	0	10
7	12	13	9	14
16	17	18	19	15
21	22	23	24	20

D1

1	2	3	4	5
6	7	8	14	9
12	17	18	13	10
11	0	19	24	15
16	21	22	23	20

D2

1	2	3	4	5
6	7	13	8	9
0	11	12	15	10
17	21	18	14	20
16	22	23	19	24

D3

1	2	3	5	10
0	6	7	8	4
17	12	14	9	18
11	16	13	19	15
21	22	23	24	20

D4

7	13	1	4	5
6	0	3	8	10
11	14	2	9	15
21	12	18	19	20
17	16	22	23	24

D5

References

[1] I. E. Salem, M. M. Mijwil, A. W. Abdulqadar, and M. M. Ismaeel, "Flight-schedule using Dijkstra's algorithm with comparison of route findings," *International Journal of Electrical and Computer Engineering,* vol. 12, no. 2, pp. 1675-1682, 2022.

[2] C. J. Sandra, S. Dileep, C. S. Sithara and L. E. Sunny, "Indoor navigation system using augmented reality," *International Research Journal of Engineering and Technology,* vol. 8, no. 6, pp. 2408-2412, 2021.

[3] W. Hidayat, F. Susanti, and D. R. Wijaya, "A Comparative Study of Informed and Uninformed Search Algorithm to Solve Eight-Puzzle Problem," *Journal of Computer Science,* vol. 17, no. 11, pp. 1147-1156, 2021.

[4] B. M. Sathyaraj, L. C. Jain, A. Finn and S. Drake, "Multiple UAVs path planning algorithm: a comparative study," *Fuzzy Optimisation and Decision Making,* vol. 7, no. 3, pp. 257-267, 2008.

[5] A. R. Soltani, H. Tawfik, J. Y. Goulermas, and T. Fernando, "Path planning in construction sites: performance evaluation of the Dijkstra, A*, and GA search algorithm," *Advanced Engineering Informatics,* vol. 16, no. 4, pp. 291-303, 2002.

[6] A. S. Naik., "User Query Processing Using Distance Oracle For Road Networks," *International Journal of Engineering Research & Technology,* vol. 2, no. 2, pp. 1-6, 2013.

[7] N. Gupta, K. Mangla, A. K. Jha, and M. Umar, "Applying Dijkstra's Algorithm in Routing Process," *International Journal of New Technology and Research,* vol. 2, no. 5, pp. 122-124, 2016.

[8] Z. Cai, "Chapter 3: General Inference Principle," in *Intelligent Control: Principles, Techniques and Applications*, vol. 7, China, World Scientific Publishing Company, 1997, pp. 64-85.

[9] D. Andiwijayakusuma, A. Mardhi, I. Savitri, and T. Asmoro, "A Comparative Study of the Algorithms for Path finding to Determine the Adversary Path in Physical Protection System of Nuclear Facilities," *Journal of Physics: Conference Series,* vol. 1198, no. 9, p. 092002, 2019.

[10] D. Rachmawati and L. Gustin, "Analysis of Dijkstra's Algorithm and A* algorithm in Shortest Path Problem," in *4th International Conference on Computing and Applied Informatics 2019, ICCAI 2019, November 26-27, 2019,* Medan, Indonesia, 2019.

[11] W. Jiang, "Analysis of Iterative Deepening A* Algorithm," in *8th Annual International Conference on Geo-Spatial Konowledge and Intelligence, December 18-19, 2020*, Xi'an Shaanxi, China, 2021.

[12] O. R. Chandra and W. Istiono, "A-star Optimization with Heap-sort Algorithm on NPC Character," *Indian Journal of Science,* vol. 15, no. 13, pp. 1722-1731, 2022.

[13] W. Zhang, "Algorithms of Combinatorial Optimization," in *State-Space Search: Algorithms, Complexity, Extensions, and Applications*, New York, Springer, 1999, pp. 13-31

[14] R. Dechter, H. Geffner, and J. Y. Halpern, "Heuristic, planning and cognition," in *Heuristics, Probability and Causility. A Tribute to Judea Pearl*, London, College Publications, 2010, pp. 23-42.

[15] K. Tierney, D. Pacino and S. Voß, "Solving the pre-marshalling problem to optimality with A* and IDA*," *Flexible Service and Manufacturing Journal,* vol. 29, no. 2, pp. 223-259, 2018.

[16] R. E. Korf, "Depth-Fist Iterative-Deepening: An Optimal Admissible Tree Search," *Artificial Intelligence,* vol. 27, no. 1, pp. 97-109, 1985.