Journal of Computer Science

Original Research Paper

A Comparative Study of Informed and Uninformed Search Algorithm to Solve Eight-Puzzle Problem

[1]Wahyu Hidayat, [2]Fitri Susanti and [3]Dedy Rahman Wijaya

[1,3]Department of Information System, School of Applied Science, Telkom University, Bandung, Indonesia
[2]Department of Multimedia Technology Engineering, School of Applied Science, Telkom University, Bandung, Indonesia

Article history
Received: 24-06-2021
Revised: 31-08-2021
Accepted: 09-10-2021

Corresponding Author:
Wahyu Hidayat
Department of Information System, School of Applied Science, Telkom University, Bandung, Indonesia
Email: wahyuhidayat@telkomuniversity.ac.id

Abstract: Problems in artificial intelligence can be solved using intelligent tracking methods through intelligent search mechanisms. Understandably, search algorithm performances are highly dependent on the problem solved. In this study, we evaluate and compare the performance of five uninformed and informed search (breadth-first search, depth first search, optimal search and best first search using two heuristic functions, namely mismatched tile and Manhattan distance) algorithms to solve the eight-puzzle game problem. For each algorithm, the numbers of raised and explored nodes were assessed and analyzed. Our experiment demonstrates that informed search with heuristic outperforms uninformed search significantly, both in terms of memory usage efficiency and computational power efficiency. On average, the informed search using heuristic requires only 5.33% of memory used by uninformed search and only 4.45% of computational power demanded by uninformed search. Boxplot analysis also confirms that informed search using heuristic also delivers more stable performance contrasted to uninformed search. These could be a concern for researchers and game developers to consider implementing the heuristically enhanced search algorithm to utilize memory and computational power efficiently to solve similar problems.

Keywords: Comparative Study, Uninformed Search, Informed Search, Heuristic, 8-Puzzle Game

Introduction

Artificial Intelligence (AI) is a branch of computer science that studies the behavior of intelligence, learning and adaptation in machines. The main purpose of AI is to mimic and emulate human intelligence then applied it to machines. Research on AI emphasizes the automation of machine processes that require intelligent behavior. Currently, artificial intelligence has promoted emerging technology integrations and revolutionize a wide range of applications and fields (Lu, 2019).

Problems in artificial intelligence can be solved by using intelligent tracking methods through an intelligent search mechanism. There are two types of search algorithms, namely uninformed search and informed search with heuristics. Uninformed search is a search mechanism that can only distinguish a goal state from a non-goal state, with no information on how far the goal state from the current state is. On the other hand, the informed search can estimate the cost of reaching the goal from a particular state through a function that calculates such cost estimates, called a heuristic (Russell and Norvig, 2020).

In the past five years, a lot of research has been focused on studies about uninformed and informed search. Some research focuses on finding and developing new search algorithms or improving well-known algorithms, while others focus more on implementing search algorithms in various fields.

However, search algorithm performances are highly dependent on the problem being solved. For one particular case, several algorithms show different performances. Therefore, it is necessary to study the comparison of search algorithm performance for various case studies.

In this study, a various search algorithm is applied to the 8-puzzle problem. We evaluate five search algorithms, namely breadth-first search, depth-first search, optimal search and best-first search using two heuristic functions; mismatched tile and Manhattan distance function. The purpose of this study was to evaluate and compare the performances of uninformed (represented by breadth and depth first search) and

informed search (represented by Best First Search) algorithms to solve the 8-puzzle game problem.

The 8-puzzle was one of the earliest heuristic search problems and it is a perfect case to compare informed with uninformed search. With an average solution of 20 steps and an average branching factor of 3, puzzle-8 can easily cause an exhaustive uninformed search to find a solution among almost 3.5 billion states (Russell and Norvig, 2020).

Using the heuristic is much more promising because the number of unique possible states in 8-puzzle is much lower than the number of states that are evaluated using an exhaustive uninformed search. Understandably, the number of evaluated states is highly related to memory and computational power requirements. Thus, experimenting on the 8-puzzle by evaluating the number of raised and explored nodes can show the different performances among algorithms in terms of memory and computational power efficiency.

The contribution of this study is to find which search algorithm that provides the best performance to solve the 8-puzzle problem both in terms of memory usage efficiency and in terms of computational power efficiency. By knowing the most efficient search algorithm, game developers can choose the right algorithm to solve 8-puzzle problems and similar searching and pathfinding problems in the field of computer game development. Thus, the games created will be more efficient in terms of memory requirements and computational power requirements and can be applied in devices with limited resources.

Related Works

Intelligent tracking methods through an intelligent search mechanism can be used to solve various problems and implemented in many different scenarios. Problems that can be solved using intelligent search are ranging from a very simple problem such as block architecture problem (Rahim *et al*., 2018a) to a more advanced polyhedra puzzle (Iordan, 2018) and even up to sophisticated problems such as to help an unmanned air vehicle navigate using as efficient energy as possible (Debnath *et al*., 2019).

Gaming is one of many scenarios where intelligence search is mostly used. For example, in a classic TicTacToe game (Hutahaean, 2018), an Indonesian popular traditional game of "Congklak" (Rahim *et al*., 2018b) and general pathfinding task in the game (Zafar *et al*., 2018).

While some studies implement intelligent search in gaming, other studies presented interesting implementations of intelligent search that are closer to a real-life scenario. Among them are the implementation of intelligence search to facilitate smart shopping experience in groceries using dynamic pathfinding (Ada *et al*., 2019), fast and safe food delivery by a robot (Navya and Ranjith, 2021) and even to find the path for the facility staff to travel with the minimum cumulative radiation dose within nuclear facilities (Chen *et al*., 2020). Meanwhile, some other research focus on attempting to solve multicriteria decision problem such as to assess the security (Kumar *et al*., 2021) (Kumar *et al*., 2021), durability (Sahu *et al*., 2020) and reliability(Sahu *et al*., 2021) of software.

Some research focuses on finding and developing new search algorithms or improving previously known algorithms. Meister (2020) proposed best first beam search, which is an upgrade from a basic Beam search that is derived from Breadth-First Search. It is claimed to perform ten times faster than basic Beam Search to solve a decoding problem in a natural language processing task. Hatem *et al*. (2018) propose a modification of the A* search algorithm to solve large general-purpose problems using parallel external memory and disk-based search. Iordan (2019) investigates Chebyshev distance, Hamming distance and Manhattan distance heuristic to improve the A* search algorithm. Meanwhile, (Hu and Sturtevant, 2019) optimizes Breadth-First Search with an external memory heuristic to build and store 5.8 trillion entries of heuristic pattern to solve the Rubik's Cube problem.

One of many ways to improve the performance of search algorithms is by adding a heuristic function to make the algorithm smarter by emulating the way humans think. Chowdhary (2020) describes various search algorithms that utilize heuristic, for instance, Hill-Climbing search, A* Search and genetic algorithm-based search. Some heuristics can be applied in real-time and (Ismail and Agwu, 2019) have investigated the effect of heuristic function properties on traditional and real-time heuristic search methods.

Due to so many variations among search algorithm and their enhancements, some studies have done comparison studies to find the most effective and efficient search algorithm. Pathak *et al*. (2018) compares breadth-first search, depth first search, uniform cost search, A* and greedy search based on their time and space complexity, optimality and completeness. Iordan (2016) and (Menon and Amali, 2018) compare various search algorithms to solve the 8-puzzle problem. Iordan (2016) measure the effective branching factor and running time to solve the puzzle while (Menon and Amali, 2018) focus on the number of nodes explored and the time required to solve the puzzle.

Methodology

Eight-puzzle is a simple game consisting of eight tiles that can be moved that are numbered 1 to 8 and placed on a "floor" measuring 3 × 3 tiles. One of the tiles of the "floor" is always empty and any tiles next to it (horizontally or vertically) can be moved into that empty tile. The object of the game is to start from a certain

configuration and end it with the tiles lying in order according to the number on them.

In this study, a various search algorithm is applied to solve the 8-puzzle problem. We evaluate five search algorithms, namely breadth-first search, depth-first search, optimal search and best-first search using two heuristic functions; mismatched tile and Manhattan Distance function.

For each algorithm, the numbers of raised and explored nodes are assessed and analyzed. The distribution and the ratio of explored to raised nodes are also assessed to measure the performance and efficiency of each algorithm in terms of memory usage and computational power requirements.

Breadth-First Search (BFS)

In the Breadth-First Search tracking method, all nodes at level *n* will be expanded first before visiting nodes at level *n*+1. Tracing starts from the root node is continued at the first level from left to right, then moves to the next level is done the same thing.

Each time a node is visited, the conditions at that node are matched with the conditions in the goal state. If the conditions at the visited node are different from the target conditions, it means that a solution has not been found. The tracking process is then carried out on all nodes to a predetermined depth.

On the other hand, if the condition of the visited node is the same as the target condition, it means that a solution has been found and the tracking is stopped. The illustration of the breadth-first search can be Fig. 1.

To apply the Breadth-First Search method, the pseudo-code that is executed is as follows:

1. Give the starting node to the open list *L*
2. Loop: If the open list *L* is empty, then tracking is stopped
3. Put *n* at the beginning of the open list *L*
4. If *n* is a goal, then the tracking has been successful
5. Remove *n* from the open list *L*
6. Put *n* on the closed list *C*
7. Expand *n*. Give the tail an open list *L* of all child nodes that have not appeared in open list *L* or closed list *C* and assign a pointer to *n*
8. Back to Loop

Depth First Search (DFS)

In the depth first search tracking method, the tracking process will be carried out on all the left nodes of the first child before tracking to nodes of the same level. This process is repeated to a certain depth. If the depth limit has not been found, then the tracking is continued at the node that is right next to it and has one parent with it. If the node of one parent is exhausted, the trace continues to the node that is right next to its current parent, as illustrated in Fig. 2.

To apply the depth-first search method, the pseudo-code that is executed is as follows:

1. Give the starting node to the open list *L*
2. Loop: If the open list *L* is empty, it means it failed
3. Put *n* at the beginning of the open list *L*
4. If *n* is a goal, then the tracking is successful
5. Remove *n* from the open list *L*
6. Expand *n* then assign all child nodes to the open head and add a pointer from the n^{th}-child node
7. Back to Loop

Optimal Search

Breadth-first search and depth-first search only track based on the position of the child of each level in the tree diagram. One strategy that can be used to improve this is by prioritizing the nodes explored using additional information. One of the additional information that can be used for tracking is the cost function calculation. The illustration of the optimal search method is shown in Fig. 3.

In this study, the cost function *C* is the sum of the node depth and the number of nodes explored to get from the initial state to the current node, computed as follows:

$$Cp = depth_{(p)} + E_p \quad (1)$$

where, *p* is a vector of the current state and E_p is the number of nodes explored to get from the initial state to *p*.

Based on the cost value information obtained in Fig. 3, the node chosen for the next situation is node *N*1, because it has the lowest cost. To determine the next node, all raised nodes are sorted in ascending order based on their cost function value, calculated using Eq. 1.

The node that has the lowest total cost is selected. In this case, *N*5 which has a total cost of 2+2 = 4 is chosen as the next explored node. This is repeated until the desired goal state is reached.

Best First Search

Best first search works very similarly with optimal search, but with a fundamental difference. In best first search, additional information that can be used to assist the tracking process is not just the value of the cost. Another additional information that can be used is the heuristic value.

Best first search tries to mimic and emulate the human's approach in solving problems by using a heuristic function that measures the likeliness of reaching the goal state from the currently evaluated state. In this method, it is possible to return to the previous state if a solution fails to be obtained. This process is called the backtracking mechanism.

An example of a heuristic evaluation process of three different states is given in Fig. 4.

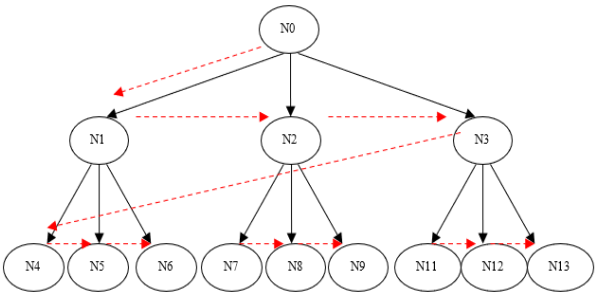

Fig. 1: Illustration of breadth-first search

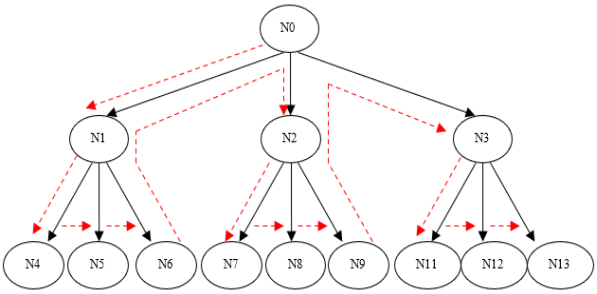

Fig. 2: Illustration of depth first search

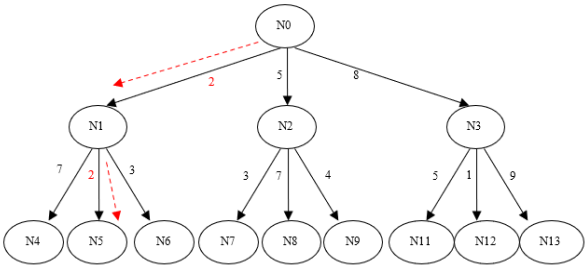

Fig. 3: Illustration of optimal search

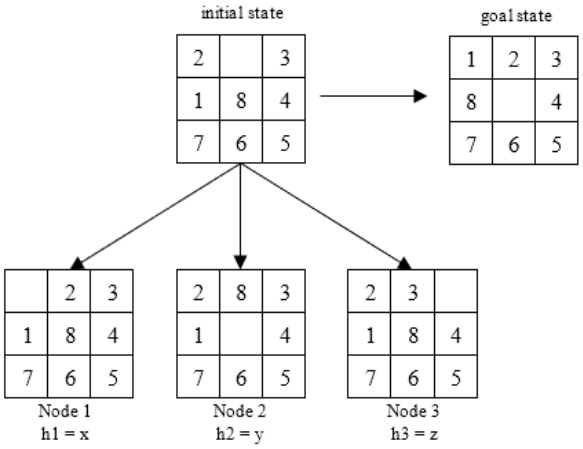

Fig. 4: Illustration of best first search with heuristic function h

In this study, two different heuristic information implemented to enhance the Best First Search algorithm are (Russell and Norvig, 2020):

a) *H1*: The number of boxes that do not match the target (mismatched tile), computed as follows:

$$m(p,q) = \sum_{i=1}^{n} x \tag{2}$$

where, p is a vector of the current state, q is a vector of the goal state and $x = 0$ if $p_i = q_i$ and $x = 1$ if $p_i \neq q_i$

b) *H2*: The sum of the vertical and horizontal distances of the boxes that do not match (Manhattan Distance) computed as follows:

$$d(p,q) = \sum_{i=1}^{n} |qi - pi| \tag{3}$$

where p is a vector of the current state, q is a vector of the goal state and $n = 9$ since there are nine available positions of tile in the 8-puzzle.

Based on the heuristic information obtained in Fig. 4, the node selected for the next state is node $N1$. This process is repeated until all nodes are found or all nodes are examined to a predetermined depth.

By using the heuristic function, it is possible to return to the previous state through a backtracking mechanism if a solution fails to be obtained.

Test Scenario and Performance Metrics

We prepared a total of 30 eight-puzzle cases to solve using five different algorithms. Each case has an initial state in the form of a vector p that consists of nine puzzle elements that are positioned randomly. Each initial state is ensured to be able to reach its goal state, which is a vector q that consists of nine puzzle elements with an arrangement of 123456780. The limit of node explored is set to 1000 and the limit of depth explored is set to 25 levels.

The total number of trials is 5 × 30 = 150 trials. During each trial t, a linked list data structure is constructed and the numbers of *Add Head()*, *Add Tail()* and *Get Head()* operations performed in that linked list determine how many Raised (Rt) and Explored (Et) nodes during the search process:

$$R_t = AH_t + AT_t \tag{4}$$

$$E_t = GH_t \tag{5}$$

where, AH_t is the number of *Add Head()* operations, AT_t is the number of *Add Tail()* operations and GH_t is the number of *Get Head()* operations that is performed during trial t.

Similar to (Zhang *et al*., 2021) and (Iordan, 2018), in this study, after the search algorithm has found a solution to solve the problem, the number of raised nodes R_t and the number of explored nodes E_t are recorded. A ratio r of explored to raised node for each trial is computed as follows:

$$r_t = \frac{E_t}{R_t} \quad (6)$$

To help visualize the distribution of each E_t and R_t across different trials and to detect outliers, all values of E_t and R_t are grouped and sorted in ascending order to construct a boxplot diagram using the following quartile formula:

$$Q1 = \left((n+1)/4\right)thTerm \quad (7)$$

$$Q2 = \left((n+1)/2\right)thTerm \quad (8)$$

$$Q3 = \left((n+1)/4\right)thTerm \quad (9)$$

$$IQR = Q3 - Q1 \quad (10)$$

where, $Q1$, $Q2$, $Q3$ *and* IQR are first quartile, second quartile, third quartile and interquartile range, respectively. Outliers are extreme values that are defined as any value that falls below $Q1$ - 1.5 IQR or any value that falls above $Q3$ + 1.5 IQR.

Results and Discussion

Our experiment results clearly show that on average, best first search raised and explore fewer nodes compared to Breadth-First Search, depth first search and optimal search. Table 1 below shows the number of raised nodes while Table 2 shows the number of explored nodes.

The number of raised nodes correlated with memory usage while the number of explored nodes correlated with the number of operations done by the processor. Thus, a significantly lower average number of raised nodes means that both best first search using mismatched tile and using Manhattan distance are far superior to breadth-first search, depth first search and optimal search in terms of memory usage. Additionally, an also significantly lower average number of explored nodes indicates that best first search outperforms other algorithms in terms of computational power requirement.

Uninformed search algorithms generate high numbers of raised and explored nodes but depth first search performs worst in both memory usage and computational power requirement. Meanwhile, best first search using both Manhattan distance and mismatched tile heuristic function perform best in both memory usage and computational power requirement.

Figure 5 above shows the distribution of the number of raised dan explored nodes in breadth-first search. It shows the uneven distribution of both raised and explored nodes, although the number of raised nodes seems to have more variance than the number of explored nodes.

Both boxplots are heavily skewed to the lower values while some outliers are detected at the top values. These heavily skewed boxplots indicate that in most cases, the number of raised and explored cases tend to be on the lower side but the variance is quite high and some occasional extreme values do occur, confirming that breadth-first search somewhat gives inconsistent results.

Figure 6 shows the uneven distribution of the number of raised dan explored nodes in depth first search. While breadth-first search shows a higher variance in the number of raised nodes compared to explored nodes, depth first search shows a similar variance between the number of raised dan explored nodes.

The number of raised and explored cases in Depth First Search tends to be on the lower side and although the boxplots are still skewed, they are not as skewed as in breadth-first search. Some extreme values are spotted on the top values, but not as much as in breadth-first search. These skewed boxplots indicate that depth first search also gives some inconsistent results, but what more important is the fact that depth first search has a very wide boxplot, indicating a very high variance among the results.

Figure 7 above shows the distribution of the number of raised dan explored nodes in Optimal Search which is quite similar to Breadth-First Search. It shows the uneven distribution of both raised and explored nodes, where the number of raised nodes seems to have more variance than the number of explored nodes.

It also shows that the number of raised and explored cases tend to be on the lower side but the variance is high and some extreme values do occur, indicated by both boxplots that are heavily skewed to the lower values and some outliers that are caught at the top values. These heavily skewed boxplots indicate that in most cases, optimal search somewhat gives inconsistent results. This should be confirmed by further research whether it is indeed caused by its cost function and whether it could have been avoided by choosing a better-suited cost function in optimal search.

Figure 8 above shows the distribution of the number of raised dan explored nodes in best first search using the mismatched tile heuristic function. Both boxplots are narrow and show no significant skewness although some outliers are spotted in the top and bottom values of the number of raised nodes.

A narrow boxplot indicates that best first search using the mismatched tile heuristic promises a stable result across different test cases. Some outliers are detected in the number of raised nodes, while no outliers are found in the number of explored nodes, indicating that best first search using the mismatched tile heuristic delivers more consistent results in terms of computational power requirements compared to its memory usage.

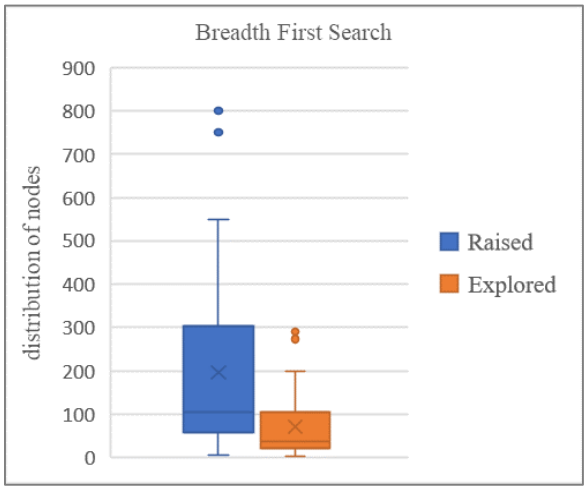

Fig. 5: Distribution of raised and explored nodes in breadth-first search

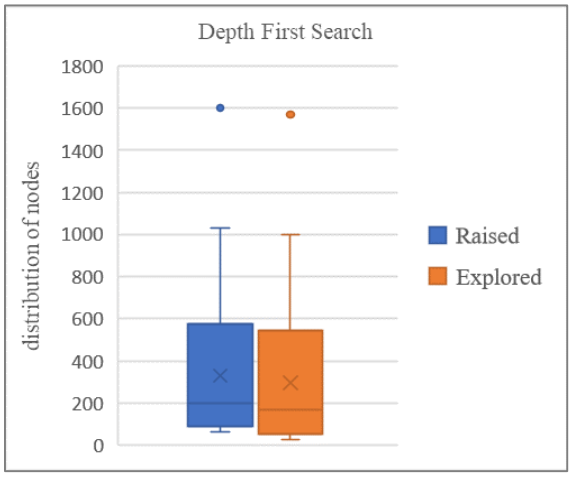

Fig. 6: Distribution of raised and explored nodes in depth first search

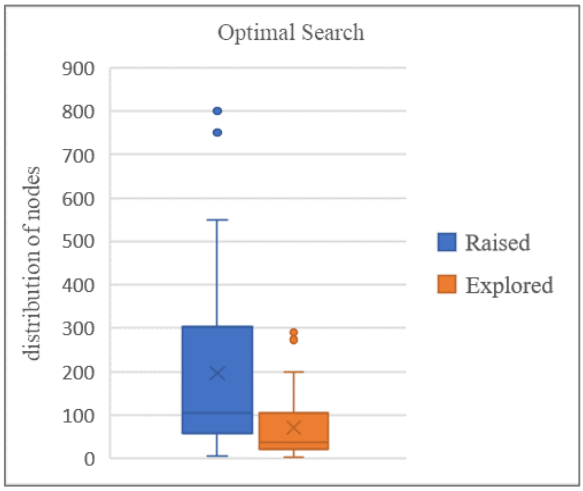

Fig. 7: Distribution of raised and explored nodes in optimal search

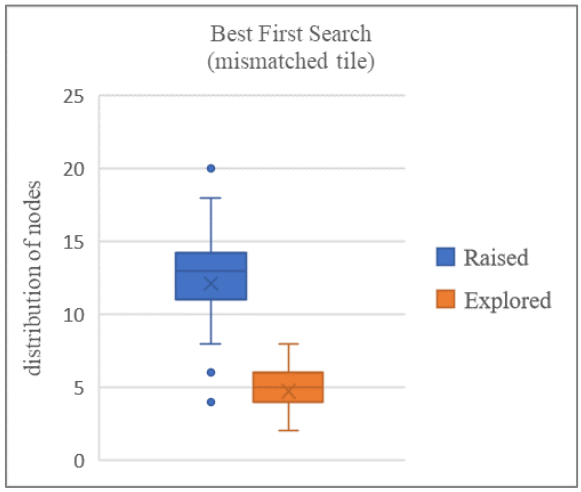

Fig. 8: Distribution of raised and explored nodes in best first search using mismatched tile

Figure 9 shows the distribution of the number of raised dan explored nodes in best first search using the Manhattan distance heuristic function. Both boxplots are narrower than the boxplots in the mismatched tile heuristic function. However, more outliers are spotted, both in the number of raised dan explored nodes. This indicates that the Manhattan distance heuristic function performs almost as well as mismatched tiles heuristic functions, albeit a bit more inconsistent at times. Further research with a larger number of cases and or improved functions should confirm whether these inconsistencies are permanent and/or avoidable.

To better illustrate the performance differences among the five search algorithms, Fig.10 and 11 depict the distribution of the number of raised and explored nodes, respectively.

Figure10 shows that breadth-first search, depth first search and optimal search raised significantly more nodes than Best First Search. Both heuristic functions used in Best First Search generate significantly fewer raised nodes. The same thing also happened with the number of explored nodes. Figure 11 shows that breadth-first search, depth first search and optimal search explore significantly more nodes than best first search. Both heuristic functions used in Best First Search also explore significantly fewer nodes.

Figure 10 and 11 reveal that uninformed search algorithms tend to have significantly wider boxplots. Uninformed search algorithms also appear with occasional outliers detected, indicating high variance in the result and somewhat inconsistent performances. Meanwhile, best first search using both heuristic functions show narrow boxplots and no outliers, confirming stable performance.

The comparison of the average number of raised and explored nodes among different algorithms is presented in the following Table 3.

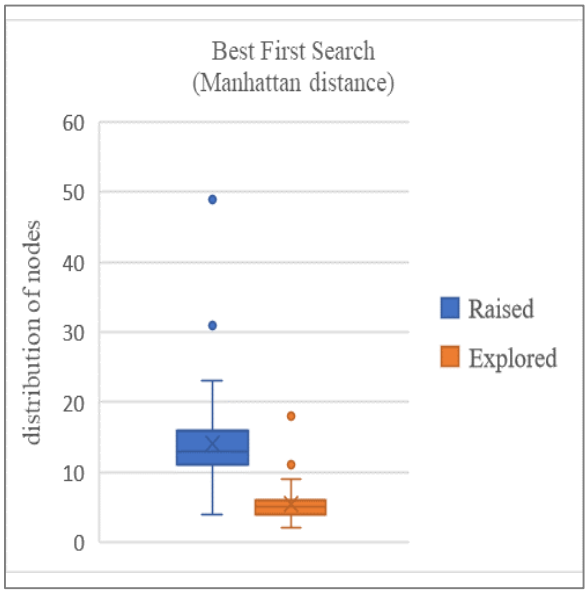

Fig. 9: Distribution of raised and explored nodes in best first search using Manhattan distance

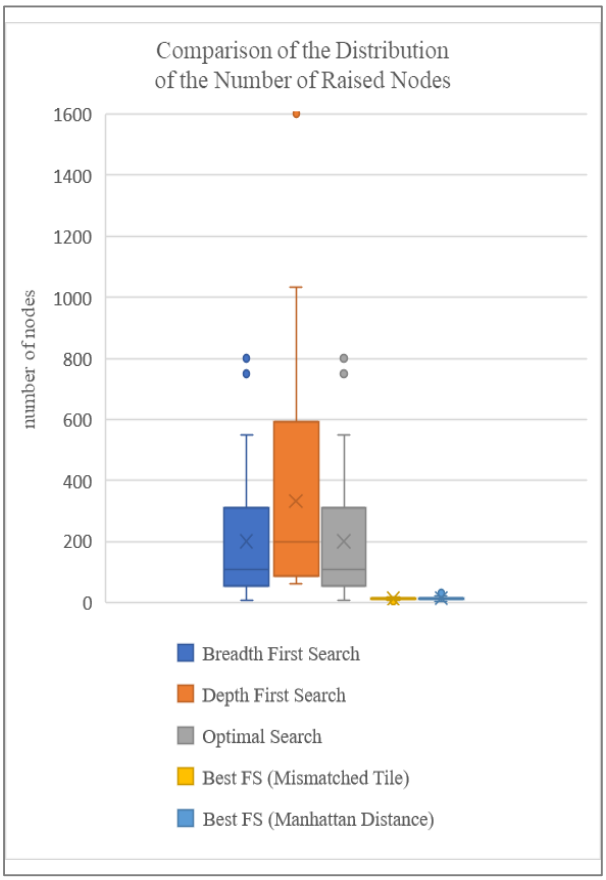

Fig. 10: Comparison of the distribution of the number of raised nodes among different algorithms

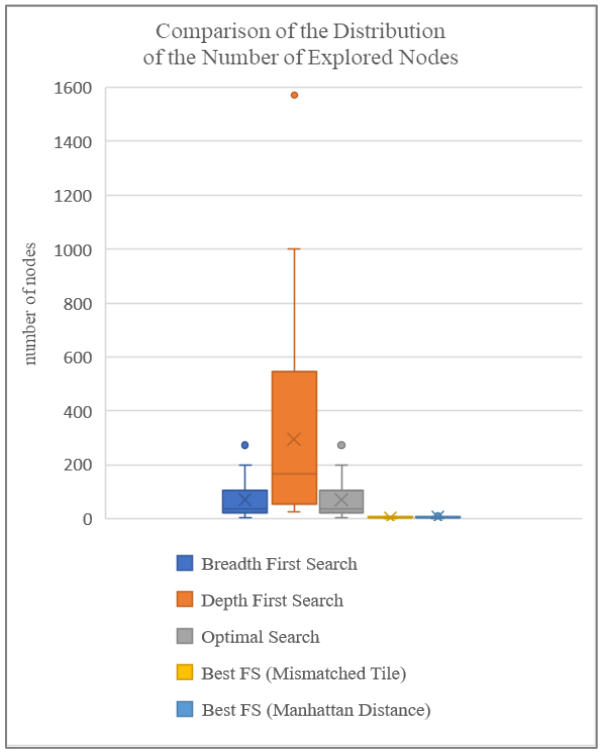

Fig. 11: Comparison of the distribution of the number of raised nodes

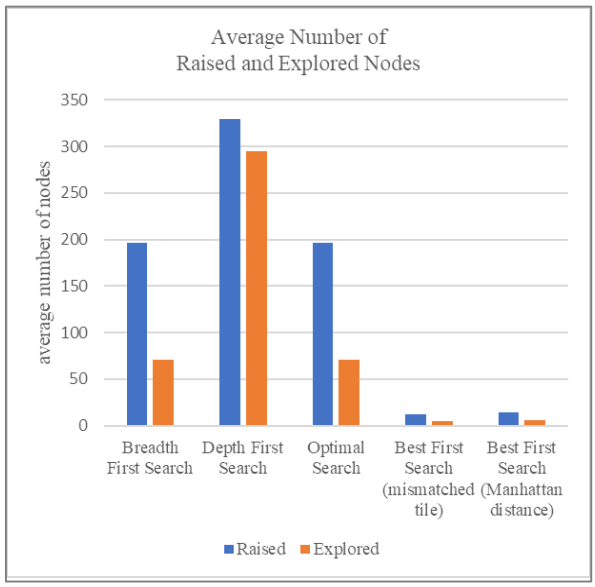

Fig. 12: The average number of raised and explored nodes

In general, all uninformed search algorithms raised and explored more nodes than informed search with heuristics. This confirms that the informed search algorithm with heuristic is far more efficient than uninformed search algorithms.

The ratio of explored to raised nodes in uninformed search also low, meaning a smaller fraction of raised

nodes is needed to be explored to solve the puzzle. This confirms that in solving 8-puzzle, in general, uninformed search performs worse than informed search with heuristics, both in terms of memory usage and computational power requirements.

Figure 12 shows the bar chart comparing the average number of raised and explored nodes. It is clearly shown that both best first search using mismatched tile and Manhattan distance heuristic functions raised and explored far fewer nodes compared to breadth-first search, depth first search and optimal search. This verifies that in terms of memory usage and computational power requirements, informed search algorithms represented by best first search significantly outperform breadth-first search, depth first search and optimal search that represent uninformed search algorithms.

To illustrate the efficiency, we calculate the percentage of raised and explored nodes in the informed search algorithm relative to uninformed search algorithms. Table 4 and 5 present the average percentage of raised and explored nodes in best first search using the mismatched tile heuristic function and manhattan distance heuristic function, respectively.

Table 4 shows that to solve the same 8-puzzle problem, best first search with mismatched tile heuristic function requires to raise only 4.92% nodes and explore 4.15% nodes compared to the uninformed search of breadth-first search and depth first search.

Meanwhile, Table 5 shows that Best First Search with Manhattan distance heuristic function requires to raise only 5.73% nodes and explore 4.76% nodes compared to the uninformed search of Breadth-first search and depth first search. Therefore, on average, best first search requires only 5.33% memory and 4.45% of computational power required by uninformed search.

Table 1: Comparison among the number of raised nodes

	Number of raised nodes		
Search algorithm	min	max	average
Breadth first search	6	800	196.57
Depth first search	63	1602	328.87
Optimal search	6	800	196.57
Best first search (mismatched tile)	4	20	12.1
Best first search (Manhattan distance)	4	49	14.1

Table 2: Comparison among the number of explored nodes

	Number of explored nodes		
Search algorithm	min	max	average
Breadth-first search	3	291	70.87
Depth first search	27	1571	294.63
Optimal search	3	291	70.87
Best first search (mismatched tile)	2	8	4.73
Best first search (Manhattan distance)	2	18	5.43

Table 3: Comparison of the average number of raised and explored nodes among different algorithms

Search algorithm	Average number of raised nodes	Average number of explored nodes	Average ratio of explored to raised nodes
Breadth-first search	196.57	70.87	0.368
Depth first search	328.87	294.63	0.748
Optimal search	196.57	70.87	0.368
Best first search (Mismatched tile)	12.10	4.73	0.399
Best first search (Manhattan distance)	14.10	5.43	0.396

Table 4: The average percentage of raised nodes in Best First Search using mismatched tile heuristic compared to uninformed search

Best first search (mismatched tile) compared to	The average percentage of raised nodes	The average percentage of explored nodes
Breadth first search	6.16%	6.68%
Depth first search	3.68%	1.61%

Table 5: The average percentage of raised nodes in best first search using Manhattan distance heuristic compared to uninformed search

Best first search (Manhattan tile) compared to	The average percentage of raised nodes	The average percentage of explored nodes
Breadth first search	7.17%	7.67%
Depth first Search	4.29%	1.84%

Conclusion and Future Work

Our research findings clearly show that to solve 8-puzzle, in terms of memory usage and computational power requirements, informed search algorithms represented by best first search significantly outperform breadth-first search, depth first search and optimal search that represent uninformed search algorithms. All uninformed search algorithms raised and explored more nodes than informed search with heuristics. The ratio of explored to raised nodes in uninformed search also low, meaning informed search requires a smaller fraction of raised nodes to be explored to solve the puzzle.

Our experiment shows that uninformed search algorithms deliver inconsistent performance. In most Breadth-First Search cases, the number of raised and explored cases tend to be on the lower side but the variance is quite high and some occasional extreme values do occur, confirming that breadth-first search somewhat gives inconsistent results. Depth first search performs worst in both memory usage and computational power requirement. Depth first search also delivers a very high variance among the results, indicating inconsistent results. Optimal Search performs similarly with breadth-first search, where the number of raised and explored cases tend to be on the lower side, high variance and some extreme values do occur, indicating inconsistent results.

On contrary, in informed search algorithms, best first search using the mismatched tile heuristic delivers consistent results in terms of computational power requirements compared to its memory usage. Meanwhile, Best First Search using the Manhattan distance heuristic performs almost as well as mismatched tiles heuristic functions, albeit a bit more inconsistent at times.

This study is limited by a single cost function implemented in optimal search. This might lead to optimal search somewhat gives inconsistent results. Further research whether it is indeed to verify whether it is caused by its cost function and whether it could have been avoided by choosing a better-suited cost function in optimal search.

Additionally, further research with a larger number of cases and or improved functions should confirm whether some slight inconsistencies in best first search using the Manhattan distance heuristic are permanent and/or avoidable.

Acknowledgment

We thank all members of Applied Information System (AIS) research groups and all System Information laboratory staff for their support.

Funding

This research was funded by Telkom University grant number PDT 2021-1.

Author's Contributions

Wahyu Hidayat: Collecting relevant literature, interpreting and analyzing test results, drafting and revising manuscript contents.

Fitri Susanti: Conducting test and reporting test results, reviewing manuscript contents.

Dedy Rahman Wijaya: Supervising and reviewing manuscript contents.

Ethics

We confirm that we have read and approved the manuscript and no ethical issues are involved.

References

Ada, A. H. D., Cortez, I. P. Q., Juvida, X. A. S., Linsangan, N. B., & Magwili, G. V. Dynamic Route Optimization using A* Algorithm with Heuristic Technique for a Grocery Store. In 2019 IEEE 11th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM) (pp. 1-6). IEEE. doi.org/10.1109/HNICEM48295.2019.9072759

Chen, C., Cai, J., Wang, Z., Chen, F., & Yi, W. (2020). An improved A* algorithm for searching the minimum dose path in nuclear facilities. Progress in Nuclear Energy, 126, 103394. doi.org/10.1016/j.pnucene.2020.103394

Chowdhary, K. R. (2020). Fundamentals of artificial intelligence. Springer Nature. doi.org/10.1007/978-81-322-3972-7_9

Debnath, S. K., Omar, R., Latip, N. B. A., Shelyna, S., Nadira, E., Melor, C. K. N. C. K., ... & Natarajan, E. (2019). A review on graph search algorithms for optimal energy efficient path planning for an unmanned air vehicle. Indonesian Journal of Electrical Engineering and Computer Science, 15(2), 743-749. doi.org/10.11591/ijeecs.v15.i2.pp743-749

Hatem, M., Burns, E., & Ruml, W. (2018). Solving Large Problems with Heuristic Search: General-Purpose Parallel External-Memory Search. Journal of Artificial Intelligence Research, 62, 233-268. doi.org/10.1613/jair.1.11209

Hu, S., & Sturtevant, N. R. (2019). Direction-optimizing breadth-first search with external memory storage. IJCAI International Joint Conference on Artificial Intelligence, 2019-Augus, 1258-1264. doi.org/10.24963/ijcai.2019/175

Hutahaean, H. D. (2018). Penerapan Metode Best first search Pada Permainan Tic Tac Toe. Jurnal Mantik Penusa, 2(2). doi.org/10.47709/cnapc.v1i1.3

Iordan, A. E. (2018). A comparative study of the A* heuristic search algorithm used to solve efficiently a puzzle game. In IOP Conference Series: Materials Science and Engineering (Vol. 294, No. 1, p. 012049). IOP Publishing. doi.org/10.1088/1757-899X/294/1/012049

Iordan, A. E. (2019). Comparative Analysis of Four Heuristic Functions that Optimizes the A* Search Algorithm. doi.org/10.9734/bpi/amacs/v2

Iordan, A. E. (2016). A Comparative Study of Three Heuristic Functions Used to Solve the 8-Puzzle. British Journal of Mathematics & Computer Science, 16(1), 1-18. doi.org/10.9734/bjmcs/2016/24467

Ismail, I. M., & Agwu, N. N. (2019, February 4). Influence of heuristic functions on real-time heuristic search methods. 14th International Conference on Electronics Computer and Computation, ICECCO 2018. doi.org/10.1109/ICECCO.2018.8634782

Kumar, R., Jamal Ansari, M. T., Baz, A., Alhakami, H., Agrawal, A., & Khan, R. A. (2021). A multi-perspective benchmarking framework for estimating usable-security of hospital management system software based on fuzzy logic, ANP and TOPSIS methods. KSII Transactions on Internet and Information Systems, 15(1), 240–263. doi.org/10.3837/TIIS.2021.01.014

Lu, Y. (2019). Artificial intelligence: A survey on evolution, models, applications and future trends. Journal of Management Analytics, 6(1), 1-29. doi.org/10.1080/23270012.2019.1570365

Meister, C. (2020). Best-First Beam Search. Transactions of the Association for Computational Linguistics, 8, 795–809. doi.org/10.1162/tacl a 00346

Menon, V., & Amali, G. B. (2018). Performance Analysis of Various Uninformed and Informed Search Strategies on 8 Puzzle Problems - A Case Study. World Wide Journal of Multidisciplinary Research and Development, 4(12), 96–99.

Navya, P., & Ranjith, R. (2021). Performance Analysis of BFS and DFS Algorithms for Food Serving Robot in an Eatery Performance Analysis of BFS and DFS Algorithms for Food Serving Robot in an Eatery. EasyChair Preprints, 5638. https://easychair.org/publications/preprint/zVBw

Pathak, M. L., Patel, R. L., & Rami, S. P. (2018). Comparative Analysis of Search Algorithms. International Journal of Computer Applications, 179(50), 40-43. doi.org/10.5120/ijca2018917358

Rahim, R., Abdullah, D., Simarmata, J., Pranolo, A., Ahmar, A. S., Hidayat, R., ... & Zamzami, Z. (2018a). Block Architecture Problem with Depth First Search Solution and Its Application. In Journal of Physics: Conference Series (Vol. 954, No. 1, p. 012006). IOP Publishing. doi.org/10.1088/1742-6596/954/1/012006

Rahim, R., Kurniasih, N., Hasibuan, A. andriany, L., Najmurrokhman, A., Supriyanto, S., ... & Abdullah, D. (2018b). Congklak, a traditional game solution approach with breadth first search. In MATEC Web of Conferences (Vol. 197, p. 03007). EDP Sciences. doi.org/10.1051/matecconf/201819703007

Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th ed.). Prentice Hall.

Sahu, K., Alzahrani, F. A., Srivastava, R. K., & Kumar, R. (2020). Hesitant fuzzy sets based symmetrical model of decision-making for estimating the durability of Web application. Symmetry, 12(11), 1770. doi.org/10.3390/SYM12111770

Sahu, K., Alzahrani, F. A., Srivastava, R. K., & Kumar, R. (2021). Evaluating the Impact of Prediction Techniques: Software Reliability Perspective. CMC-Computers Materials & Continua, 67(2), 1471-1488. doi.org/10.32604/cmc.2021.014868

Zafar, A., Agrawal, K. K., & Kumar, W. C. A. (2018). Analysis of multiple shortest path finding algorithm in novel gaming scenario. In Intelligent Communication, Control and Devices (pp. 1267-1274). Springer, Singapore. doi.org/10.1007/978-981-10-5903-2_132

Zhang, W., Sauppe, J. J., & Jacobson, S. H. (2021). Comparison of the number of nodes explored by cyclic best first search with depth contour and best first search. Computers & Operations Research, 126, 105-129. doi.org/10.1016/j.cor.2020.105129