Project idea

* The n puzzle is a classical problem for modelling algorithms involving heuristics. Commonly used heuristics for this problem include counting the number of misplaced tiles and finding the sum of the taxicab distances between each block and its position in the goal configuration.
* N-Puzzle consists of N tiles where N can be 8, 15, 24 and so on. The puzzle is divided into √(N+1) rows and √(N+1) columns eg. 15-Puzzle will have 4 rows and 4 columns, an 8-Puzzle will have 3 rows and 3 columns and so on. The puzzle consists of one empty space where the tiles can be moved
* An instance of the **n-puzzle game** consists of a board holding n^2-1 distinct movable tiles, plus an empty space. The tiles are numbers from the set 1,..,n^2-1. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, the blank space is going to be represented with the number 0. Given an initial state of the board, the combinatorial search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order 0,1,… ,n^2−1. The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions *{‘Up’, ‘Down’, ‘Left’, ‘Right’}*, one move at a time. The cost of moving from one configuration of the board to another is the same and equal to one. Thus, the total cost of path is equal to the number of moves made from the initial state to the goal state.

A puzzle problem with a number and a number

Description automatically generated with medium confidence

**Main Functionalities:**

**Game Initialization:**

The game is initialized with a default puzzle size of 3x3.

Pygame window and display settings are configured.

**User Interface:**

The Pygame window displays the puzzle grid, buttons for different puzzle sizes, and buttons for selecting different heuristics.

The user can choose the puzzle size (3x3, 4x4, or 5x5) and select a heuristic for solving the puzzle.

**Puzzle Solving Algorithms:**

The code includes various heuristic functions for solving the N-Puzzle, such as

1. number of misplaced distance

**F(n)= h(n)= number of misplaced tiles**

1. Euclidean distance

The distance between two points measured along axes at right

angles. In a plane with p1 at (x1, y1) and p2 at (x2, y2), it is

F(n)= |x1 - x2| + |y1 - y2|.

1. Manhattan distance

The distance between two points measured along axes at right

angles. In a plane with p1 at (x1, y1) and p2 at (x2, y2), it is

F(n)= |x1 - x2| + |y1 - y2|.

1. , and Permutation inversion.

The best first search algorithm is implemented to find the optimal solution based on the selected heuristic.

**Board Representation:**

The game board is represented as a 2D array, where each tile has a unique number, and an empty space is represented as None.

**Game Controls:**

The user can interact with the game using the mouse to click on tiles and buttons or use keyboard arrow keys to move tiles.

The game provides options to reset the puzzle, generate a new random puzzle, solve the puzzle

**Animation:**

The code includes animations for tile movements and puzzle solving.Animations are implemented using the Pygame library to visualize the sliding of tiles.

**Heuristic Functions:**

The Heuristic class provides implementations of different heuristic functions used in the best first search algorithm.

**Search Space and Solution Path:**

The search space is represented by a dictionary (Space) containing information about each state explored during the search process.

The solution path is generated based on the search space, representing the sequence of moves to solve the puzzle.

**User Interaction:**

The user can choose the puzzle size, select a heuristic, and interact with the game using mouse clicks or keyboard arrow keys.

**Termination:**

The user can close the Pygame window to terminate the program.

**Similar applications in the market:-**

**Klotski:**

* is a type of sliding block puzzle where the player moves blocks to clear a path for a specific block to exit the board. It involves a different type of movement than the N-Puzzle but shares the sliding puzzle concept.



**Rush Hour:**

* Rush Hour is a sliding block puzzle that involves moving red vehicles within a grid to clear a path for the target vehicle to exit. The goal is to shift cars and trucks in a way that opens up a route.



**Unblock Me:**

* Unblock Me is a sliding puzzle game where the player needs to move blocks within a grid to clear a path for a special block to exit. It comes with various difficulty levels.

A screenshot of a game

Description automatically generated

**Finding( paper / book /article ):**

**15-Puzzle Problem Solving with the Artificial Bee Colony Algorithm Based on Pattern Database**

Abstract

The N-puzzle problem is one of the most classical problems in mathematics. Since the number of states in the N-puzzle is equal to the factorial of the number of tiles, traditional algorithms can only provide solutions for small-scale ones, such as 8-puzzle. Various uninformed and informed search algorithms have been applied to solve the N-puzzle, and their performances have been evaluated. Apart from traditional methods, artificial intelligence algorithms are also used for solutions. This paper introduces a new approach based on a meta-heuristic algorithm with a solving of the 15-puzzle problem. Generally, only Manhattan distance is used as the heuristic function, while in this study, a linear conflict function is used to increase the effectiveness of the heuristic function. Besides, the puzzle was divided into subsets named pattern database, and solutions were obtained for the subsets separately with the artificial bee colony (ABC) algorithm. The proposed approach reveals that the ABC algorithm is very successful in solving the 15-puzzle problem.

<https://pdfs.semanticscholar.org/0d26/9677cb337ff6f7647599600c077926113f67.pdf>

### **E**[**valuating Search Algorithms for Solving n-Puzzle‏**](http://sumitg.com/assets/n-puzzle.pdf)

Abstract‏

Artificial game playing has gathered significant attention in past few decades. Several games are used to evaluate various algorithm. n-puzzle is a classical problem in computer science in evaluating search heuristics which is a central problem in artificial game playing and artificial intelligence in general. In this paper we formulate n-puzzle as an undirected graph problem and evaluate search algorithms with various heuristics. We find out relation between a random puzzle state and its time complexity and also provide evidence that finding a shortest path to an n-puzzle is an NP-Hard problem.

‏

Link of web page: <https://scholar.google.com.eg/scholar?q=Evaluating+Search+Algorithms+for+Solving+n-Puzzle&hl=ar&as_sdt=0&as_vis=1&oi=scholart>

url of paper:

<https://sumitg.com/assets/n-puzzle.pdf>

# **Using Uninformed & Informed Search Algorithms to Solve 8-Puzzle (n-Puzzle) in Python**

This article of

An instance of the **n-puzzle game** consists of a board holding n^2-1 distinct movable tiles, plus an empty space. The tiles are numbers from the set 1,..,n^2-1. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, the blank space is going to be represented with the number 0. Given an initial state of the board, the combinatorial search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order 0,1,… ,n^2−1. The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions {‘Up’, ‘Down’, ‘Left’, ‘Right’}, one move at a time. The cost of moving from one configuration of the board to another is the same and equal to one. Thus, the total cost of path is equal to the number of moves made from the initial state to the goal state.

URL :

<https://www.datasciencecentral.com/using-uninformed-informed-search-algorithms-to-solve-8-puzzle-n/>

# **A Distributed Approach To N-Puzzle Solving**

Abstract and Figures

We present in this paper a distributed approach for solving the N-puzzle. This approach is based on the decomposition of a problem into the set of the smallest independent subgoals that can be described, be they serializable or not. These subgoals are at their turn decomposed into agents whose task is to satisfy the subgoal. We have chosen the Eco-Problem-Solving model for describing the internal functioning of these agents. Each of them is then characterized by a state, a goal, a satisfaction behavior and a flight behavior. Those kernel behaviors invoke domaindependent actions that must fit with the domain in which the agent is involved. The description of these actions is made by presenting their basic algorithm and the heuristics used in it, namely the MGB and VMGB distance computations. A simple example of solving is presented, followed by an example of what can be called an "emergent solution" to the problem of the corners. We prove, then, that the method is complete

URL:

<https://www.researchgate.net/publication/2710914_A_Distributed_Approach_To_N-Puzzle_Solving>

**Analysis and Implementation of Admissible Heuristics in 8 Puzzle Problem**

Thesis submitted in May 2014 to the department of Computer Science and Engineering of National Institute of Technology Rourkela in partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science and Engineering by Debasish Nayak

**Abstract**:

N-puzzle problem has been one of the basic problem since the beginning of artificial intelligence. The most popular version of n-puzzle among people is 8- puzzle problem. It consists of an area divided into 3x3 grid containing 8 numbered (to identify) tiles and one empty grid. We are given an initial state and we have to reach the goal state which is also specified. In this project, we have used various informed search methods like a\*algorithm, ida\* algorithm to solve the puzzle. Various heuristic involved in the informed search like number of misplaced tiles, Manhattan distance were analyzed; Manhattan distance being one of the most popular ones. Drawbacks of the heuristics are mentioned and an improvement in Manhattan distance heuristic is implemented.

URL :

<https://core.ac.uk/download/pdf/53190059.pdf>

Details of algorithms :

**Best-first search:**

 is a class of [search algorithms](https://en.wikipedia.org/wiki/Search_algorithm), which explores a [graph](https://en.wikipedia.org/wiki/Graph_(data_structure)) by expanding the most promising node chosen according to a specified rule.

[Judea Pearl](https://en.wikipedia.org/wiki/Judea_Pearl) described the best-first search as estimating the promise of node *n* by a "heuristic evaluation function �(�) which, in general, may depend on the description of *n*, the description of the goal, the information gathered by the search up to that point, and most importantly, on any extra knowledge about the problem domain."[[1]](https://en.wikipedia.org/wiki/Best-first_search#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Best-first_search#cite_note-RN03-2)

Some authors have used "best-first search" to refer specifically to a search with a [heuristic](https://en.wikipedia.org/wiki/Heuristic_function) that attempts to predict how close the end of a path is to a solution (or, goal), so that paths which are judged to be closer to a solution (or, goal) are extended first. This specific type of search is called [*greedy*](https://en.wikipedia.org/wiki/Greedy_algorithm)*best-first search*[[2]](https://en.wikipedia.org/wiki/Best-first_search#cite_note-RN03-2) or *pure heuristic search*.[[3]](https://en.wikipedia.org/wiki/Best-first_search#cite_note-3)

Efficient selection of the current best candidate for extension is typically implemented using a [priority queue](https://en.wikipedia.org/wiki/Priority_queue).

The [A\* search algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm) is an example of a best-first search algorithm, as is [B\*](https://en.wikipedia.org/wiki/B*). Best-first algorithms are often used for path finding in [combinatorial search](https://en.wikipedia.org/wiki/Combinatorial_search). Neither A\* nor B\* is a greedy best-first search, as they incorporate the distance from the start in addition to estimated distances to the goal.

Advantage and disadvantage of best first search algorithm

Advantage :

1. **Efficiency:** Best-First Search can be more efficient than uninformed search algorithms, especially in large state spaces. It focuses on exploring the most promising paths early on, potentially reaching the goal faster.
2. **Optimality:** With an admissible heuristic (a heuristic that never overestimates the true cost to reach the goal), Best-First Search is guaranteed to find an optimal solution. This is a significant advantage in applications where finding the best solution is crucial.
3. **Scalability:** It is well-suited for problems with a large number of states, as it selectively explores nodes based on heuristic information, avoiding unnecessary exploration of unpromising paths.
4. **Memory Efficiency:** Best-First Search doesn't require storing the entire search space in memory. It focuses on the most promising nodes, which can be more memory-efficient compared to some uninformed search algorithms.

Disadvantage

1. **Completeness:** Best-First Search is not guaranteed to find a solution even if one exists. Its performance heavily depends on the quality of the heuristic function. If the heuristic is poorly designed, the search may get stuck or fail to find a solution.
2. **Heuristic Dependence:** The effectiveness of Best-First Search is highly dependent on the choice and quality of the heuristic function. If the heuristic does not accurately estimate the cost to reach the goal, the algorithm's performance may suffer.
3. **Space Complexity:** In some cases, Best-First Search can have high space complexity, especially if the open list (priority queue) becomes large. This can lead to significant memory requirements.
4. **Optimality Concerns:** While Best-First Search is optimal with admissible heuristics, not all heuristics are admissible. If an inadmissible heuristic is used, the algorithm may not find the optimal solution.
5. **Sensitivity to Initial Conditions:** The performance of Best-First Search can be sensitive to the initial conditions of the problem. Small changes in the starting state or the heuristic function may lead to different search patblock

**Lets know how best first algorithm implement :**

1-create a priority queue

2-insert the initial state in the queue

3-calculate heuristic distance for child states

4-select the state with lowest heuristic distance

5- if the state is the goal state

solution is found;

**else**

repeat process 3 until the end

**flow chart of “Best-first search”**

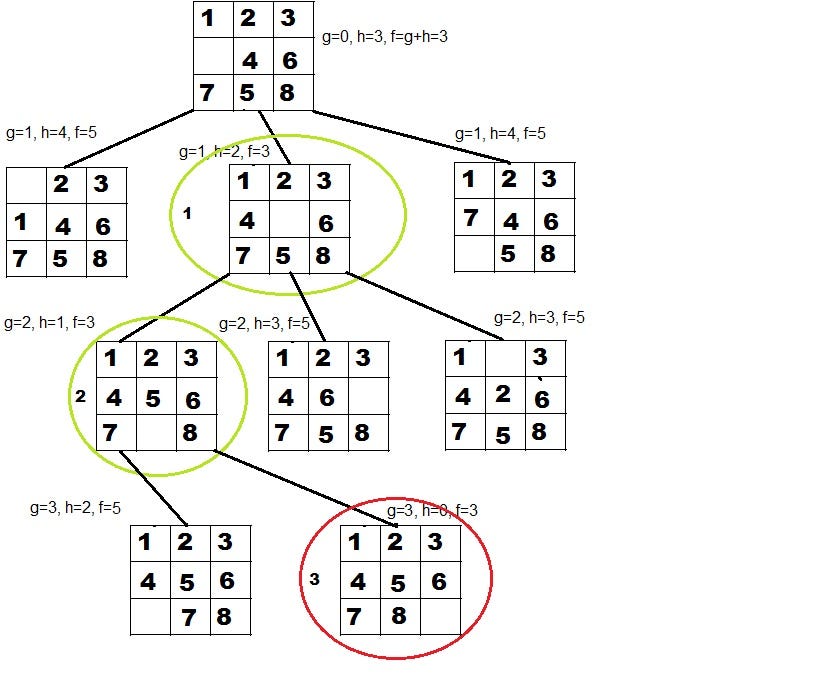
Diagram

Description automatically generated

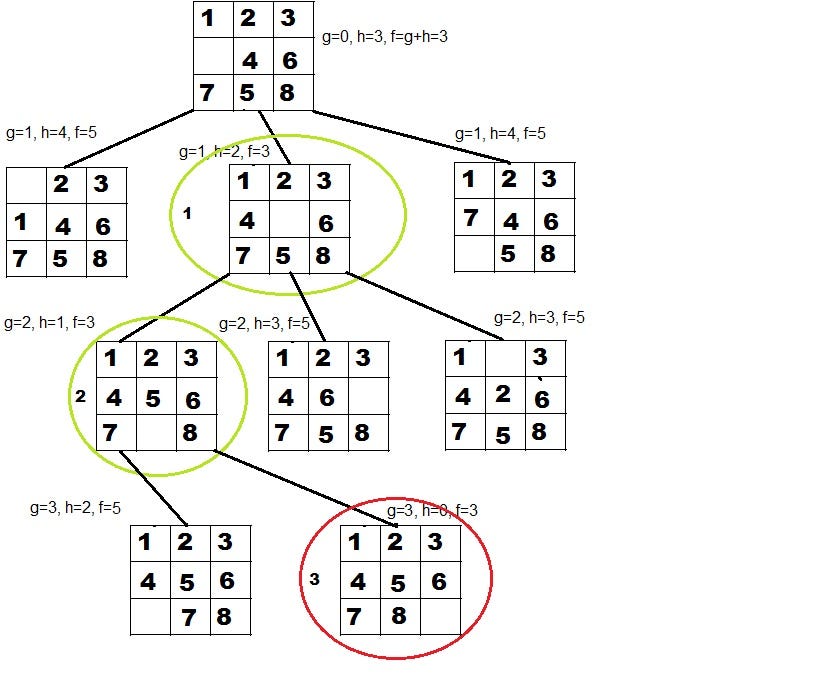
Explain of heuristics:

misplaced tiles :  
f(n)=h

h = no.of misplaced



Euclidean distance :

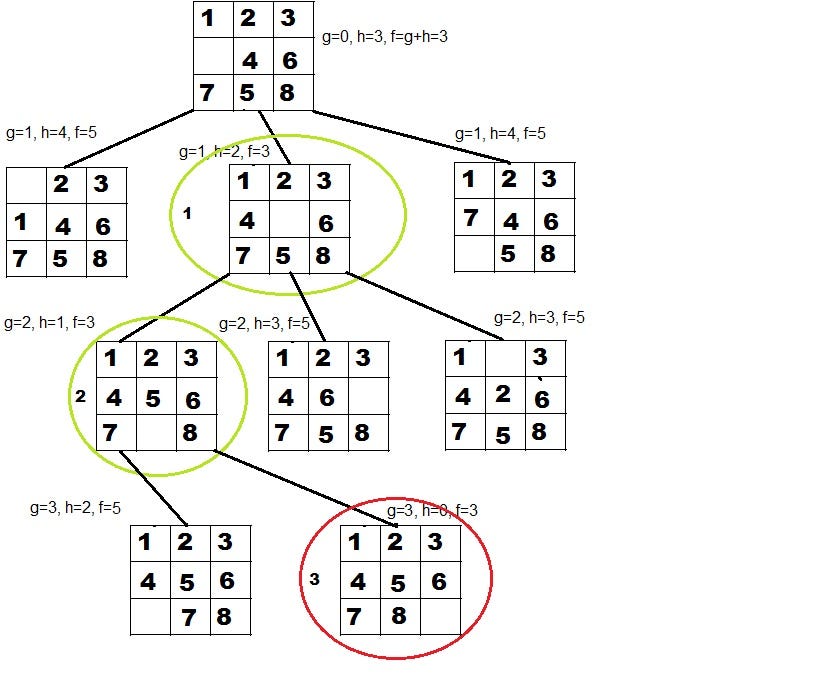
**F(n) = √[(x2– x1)2 + (y2– y1)2]** 

Manhattan distance

The distance between two points measured along axes at right

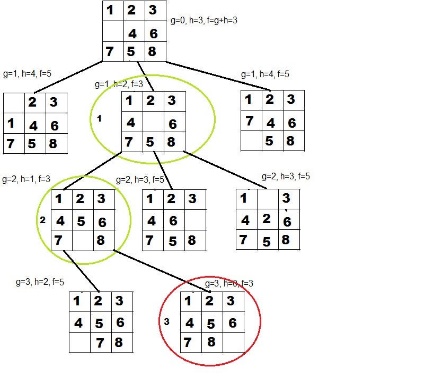
angles. In a plane with p1 at (x1, y1) and p2 at (x2, y2), it is

F(n)= |x1 - x2| + |y1 - y2|.

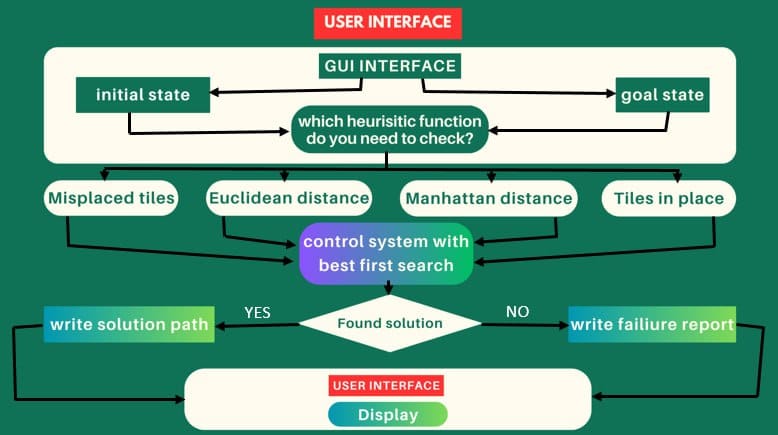


Tiles in place :

F(n)=h

Where h= no of placed tiles 

Block diagram



Use case of main functionalities of users :

