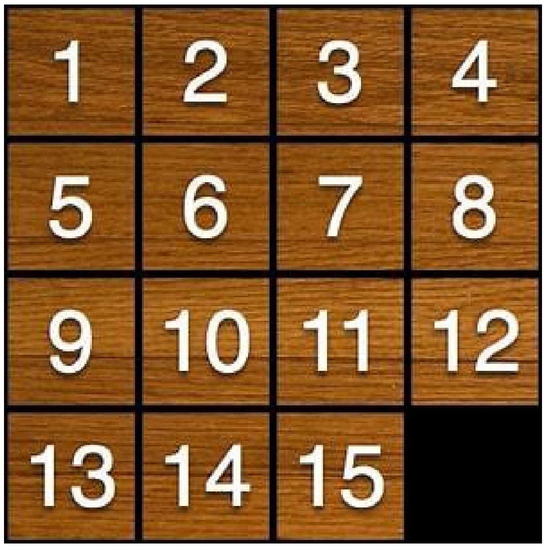
The 15 Puzzle

and its Extension for a N*N Grid

In this project, we've used maths to solve the famous 15 puzzle game. We've even applied this to bigger challenges - any NxN grid. We've used in this the A* algorithm, proven to work great at solving problems, and also made sure the puzzle can actually be solved!

Focus is put on making this method work for bigger and bigger puzzles. Before putting the A* algorithm to work, we check first to make sure the puzzle can be solved. This makes our puzzle solver more dependable and trustworthy.

We looked into and demonstrated the ability in solving varying sized puzzles. Our method works for various difficulties, adding flexibility to this puzzle-solving system. We pushed the boundaries of traditional search algorithms and puzzle-solving methods, offering a functional approach to tackling even the most complicated puzzles without compromising their solvability.



Representation of final state of the 15 puzzle

Here the bottom right Square is empty and can be switched with its adjacent block

Introduction

We looked into what makes a puzzle solvable by examining its width, inversion, and the intricate relationship between them. Our main goal is to establish clear criteria for determining solvability and demonstrate how the puzzle can be a reliable assessment tool.

For puzzles with odd widths, we confidently assert that a solvable state must have an even number of inversions. However, for puzzles with even widths, the conditions for solvability may vary a bit. Specifically, we need to consider the position of the blank space and how it affects the number of flips required to reach a solvable state. If the blank space is in an odd-numbered row from the bottom, the solvable state must have an even number of flips. Conversely, if the blank space is in an even-numbered row from the bottom, the solvable state should have an odd number of inversions. This subtle distinction emphasises the need to carefully think about solvability conditions.

To crack these tricky puzzles, we've set up the A* algorithm. This super search tool mixes the plus points of both even cost searches with the ambitious best-first searches, creating a really effective way to find answers. Because it uses clues as part of its search, the A* algorithm has done a great job at finding perfect solutions without hogging computer resources. By blending in-depth textbook study and hands-on examining, our aim is to clear up the rules for solving and highlight how well the A* algorithm works in finding the finish line.

History & Previous Literature

For more than two decades, puzzle 8 and puzzle 15 have been central to the study of search strategies. Created by Sam Lloyd in 1959, these puzzles served as an important model for experimenting with theoretical methods of exploration. Because of the versatility of these puzzles, they have helped develop and evaluate algorithms. Back in 1966, visionaries Dolan and Michie incorporated these puzzles into their comprehensive problem-solving program, the Graph Traverser. Based on this pioneering work, Paul used 15 puzzles as the basis for his research on two-way search and dynamic loading. In 1979, Gaschnig described the 8 puzzles as simple and fun, sharing features believed to extend beyond the puzzles.

This puzzle has received numerous noteworthy contributions, and here are some of the significant ones.

- Sam Loyd's Invention and Early Popularity (1878 - 1914)

  In 1878, Sam Loyd claimed credit for creating the 15-puzzle. In his publication "Cyclopedia of Puzzles" from 1914, he featured numerous puzzles, including the 15-puzzle, contributing significantly to its widespread popularity as an enjoyable game.

- Mathematical Analysis Begins (1879)

  Noyes Chapman's paper, "On the Unsolvability of the Fifteen Puzzle" (1879), provided the first significant mathematical analysis of the 15-puzzle, demonstrating that half of the configurations are unsolvable. This marked the beginning of a more rigorous examination of the puzzle's mathematical properties.

- Evolution into a Mathematical Challenge (Late 19th Century)

  Researchers such as Chapman, Gaston Tarry, William Story, and William Alonzo Rogers expanded on Chapman's work. They provided more systematic approaches to determining the solvability of the 15-puzzle, evolving it from a newspaper pastime to a subject of mathematical investigation.

- Graph Theory and Algorithmic Developments (1960s - 1980s)

  In the 1960s and 1970s, algorithmic concepts developed by Dijkstra and Bellman, coupled with exploration by the artificial intelligence community, set the stage for applying graph theory to combinatorial problems.

- A* Algorithm Adaptation (1968 - 1980s)

  The A* algorithm was introduced by Peter Hart, Nils Nilsson, and Bertram Raphael in "A Formal Basis for the Heuristic Determination of Minimum Cost Paths" (1968). While not initially applied to puzzles, its principles were later adapted for puzzle-solving, including the 15-puzzle and N-puzzle.

- Puzzle-Specific Adaptation and Integration with Graph Theory (1980s onward)

  Puzzle enthusiasts and researchers in artificial intelligence began adapting A* specifically for solving puzzles. The integration with graph theory became common practice, representing puzzle states as nodes in a graph and using A* to traverse the graph efficiently.

- Algorithmic Refinements and Widespread Application (1980s - present)

  Over time, researchers introduced refinements and variants of the A* algorithm to enhance its efficiency in solving puzzles. The algorithm became widely used in puzzle-solving software, marking its widespread application in both academic and practical settings.

Topics of Study

Solvability

The solvability of the puzzle depends on the size of the grid (N). Here are the rules:

- For Odd N:

  The puzzle is solvable if the number of inversions (pairs of tiles where a tile appears before another, but has a higher number) is even.

- For Even N:
  - If the blank space is on an even row from the bottom (second-last, fourth-last, etc.), the puzzle is solvable if the number of inversions is odd.
  - If the blank space is on an odd row from the bottom (last, third-last, fifth-last, etc.), the puzzle is solvable if the number of inversions is even.
  - In all other cases, the puzzle is not solvable.

Here, **Inversion** refers to a pair of tiles (a, b) where a appears before b in a linear arrangement, but a > b.

Eg - In a 4×4 board with 15 numbered tiles and one empty space, in the arrangement: 2 1 3 4 5 6 7 8 9 10 11 12 13 14 15 X, there is only one inversion: (2, 1).

PROOF -

Defining Terms -
I is number of inversions in the permutations
R is the Row number of the blank space from the bottom

Claim - (by using given terms)

The n*n puzzle is solvable if R+I is even

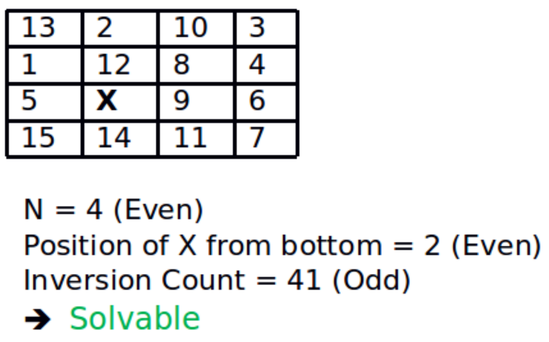
Proof -
Assume S is solvable Configuration . There are sequences of moves M1, M2 …. Mk and That transforms S into a goal state.

Define I(i) as the inversions after Mi
Each move changes I parity and maybe R.

Hence, Now parity preservation step -
If a move involves moving the blank space vertically, the change in inversions is odd. And change in r is +-1 which is odd so I+R is even
If a move involves horizontal movement, the change in inversions is even.

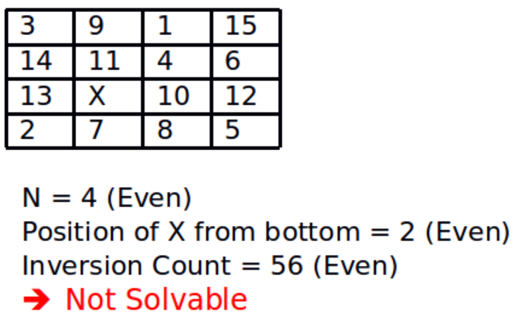
& Now finally -
The total number of moves k is even since each move changes the parity of I and R.
At the final state ,I(k) =0 and R(k) is even => I+R = I(k) + R(k) is even

Illustration for 15 puzzle ie 4 x 4 grid -



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

N = 4 (Even)
Position of X from bottom = 2 (Even)
Inversion Count = 41 (Odd)
➔ Solvable



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

N = 4 (Even)
Position of X from bottom = 2 (Even)
Inversion Count = 56 (Even)
➔ Not Solvable

In Conclusion,

- When the width is odd, every solvable state must exhibit an even number of inversions.
- When the width is even
    - A solvable state requires an even number of inversions if the blank is on an odd-numbered row when counting from the bottom.
    - A solvable state necessitates an odd number of inversions if the blank is on an even-numbered row when counting from the bottom.

Solving Algorithm

Our Code :-https://github.com/ANGADJEET/DM_PROJECT

Introduction

A* is recognized as one of the most effective informed search algorithms, and its success hinges on the quality of its heuristic function. Extensive experiments have consistently highlighted the algorithm's efficiency in solving N Puzzle problems of different sizes.

These findings not only underscore A*'s effectiveness but also affirm its superiority when compared to other search algorithms. The algorithm's optimality and performance make it a compelling choice for a wide range of problem-solving scenarios.

What exactly is A*?

A* is a more efficient version of Dijkstra's algorithm, especially useful for solving problems like the N puzzle. In this algorithm, the puzzle states form a graph, and A* aims to find the best path from the initial state to the goal state.

Unlike Dijkstra's algorithm, A* uses a heuristic function to estimate the cost of reaching the goal from a given state. This additional information guides the algorithm to explore paths that seem more likely to lead to the goal, improving efficiency.
The algorithm starts by initialising a priority queue with the initial state and its associated cost. This cost combines the actual cost from the start state and the heuristic estimate to the goal. A* then iteratively selects and expands nodes from the priority queue until it reaches the goal or the queue is empty.
A* prioritises paths that appear promising, striking a balance between exploration and exploitation. This heuristic-guided approach makes A* highly effective for navigating large state spaces, such as those found in the N puzzle problem.

Heuristic function

When attempting to solve the N-puzzle problem, employing a strategic approach is essential. By utilising heuristic functions within the A* algorithm, we can evaluate the desirability of a specific state.
This framework requires consideration of both a cost function and a heuristic to guide decision-making at each step. The cost function, referred to as "cost," takes into account the heuristic cost as well as the number of steps needed to reach the current state.

For instance, in the case of an unsolved 8-puzzle board like the one below:

0 | 3 | 8
4 | 1 | 7
2 | 6 | 5

Two heuristics are employed for evaluating the desirability of a given state -

1. Misplaced Tiles Heuristic (h1):

This heuristic assesses the number of tiles that are out of their intended positions. For the provided example, the number of misplaced tiles is calculated as 8.

2. Manhattan Distance Heuristic (h2):

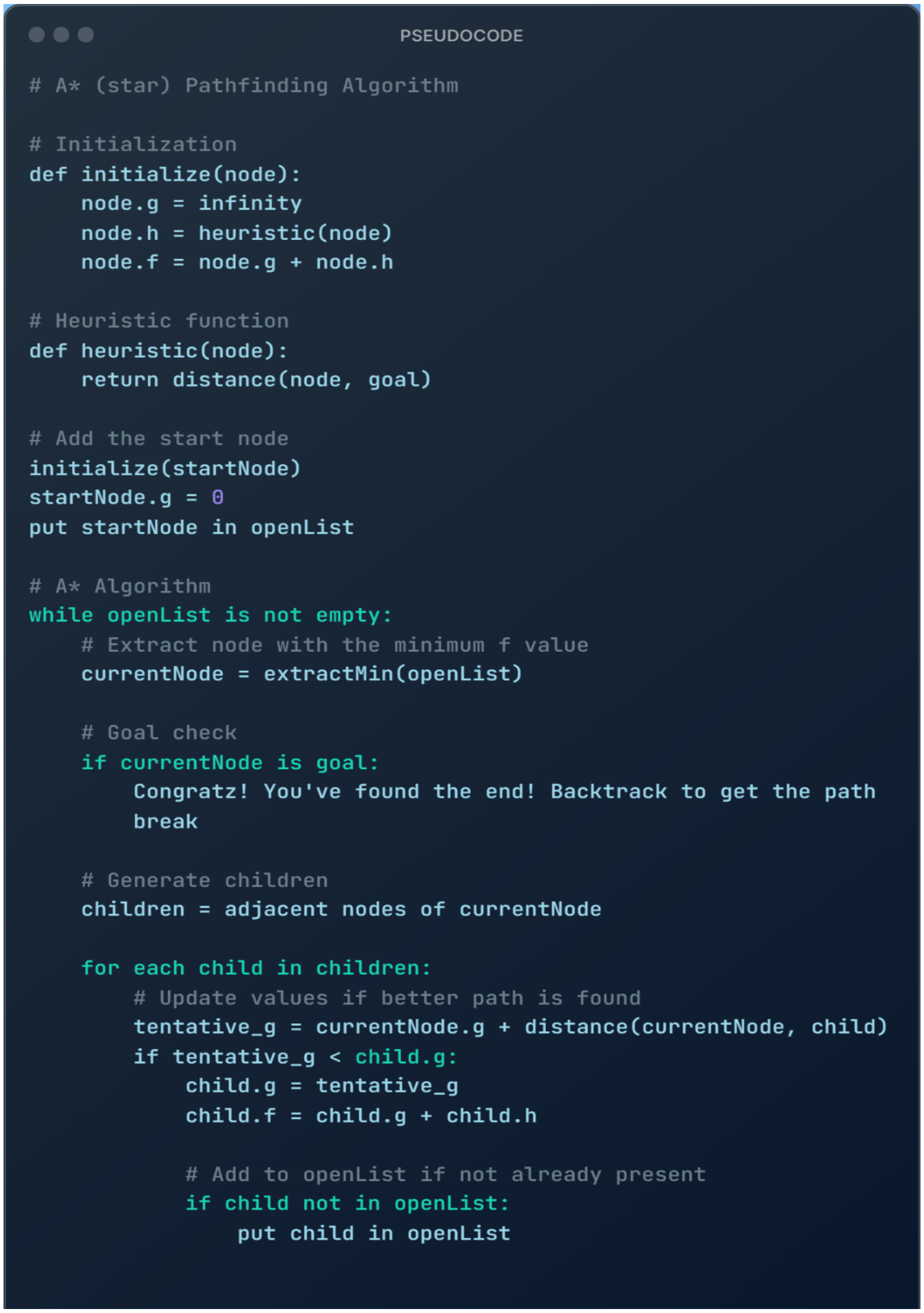
This heuristic computes the Manhattan distance for each tile, measuring the cumulative displacement from its original position to its current location. For the illustrated configuration, the Manhattan distance is determined individually for each tile and then summed, yielding a value of 14.

In adherence to the principles of the A* algorithm, the maximum of the two heuristics is selected, as the maximum heuristic ensures that the cost function never underestimates the actual cost of reaching the desired end state, i.e., the initial configuration of the 8-puzzle.
Illustration - if it took 10 moves to reach the current state, the cost function for a given state cost is computed as follows -

cost =max(h1, h2) + 10 = max(14, 8) + 10 = 24

In order to thoroughly investigate all possible avenues and strive towards achieving the most advantageous solution, the implementation utilises a priority queue. Within this framework, each potential state is meticulously documented, and the state with the most competitive cost function is carefully chosen for further investigation. The incorporation of the A* algorithm guarantees a methodical and successful approach to effectively resolving the challenging N-puzzle conundrum, meeting the uncompromising criteria of academic and professional scholarship.

Pseudocode



PSEUDOCODE

```
# A* (star) Pathfinding Algorithm

# Initialization
def initialize(node):
    node.g = infinity
    node.h = heuristic(node)
    node.f = node.g + node.h

# Heuristic function
def heuristic(node):
    return distance(node, goal)

# Add the start node
initialize(startNode)
startNode.g = 0
put startNode in openList

# A* Algorithm
while openList is not empty:
    # Extract node with the minimum f value
    currentNode = extractMin(openList)

    # Goal check
    if currentNode is goal:
        Congratz! You've found the end! Backtrack to get the path
        break

    # Generate children
    children = adjacent nodes of currentNode

    for each child in children:
        # Update values if better path is found
        tentative_g = currentNode.g + distance(currentNode, child)
        if tentative_g < child.g:
            child.g = tentative_g
            child.f = child.g + child.h

            # Add to openList if not already present
            if child not in openList:
                put child in openList
```

Beyond A* : Using AI

1. Using Subgoal analysis -

Solving puzzles is like exploring with the A* algorithm, a skilled guide armed with a detailed map. However, when faced with more complex challenges, even A* can struggle. This leads to a fresh approach, breaking big goals into smaller, achievable tasks.

Unlike A*'s rigid structure, the new method involves independent agents handling smaller tasks, making puzzle-solving a collaborative effort. Inspired by natural systems, this strategy prioritises adaptability and efficiency, allowing real-time adjustments based on the evolving problem dynamics.

Take the journey from Delhi to Mumbai as an example. While A* might stick to a fixed route, the innovative method breaks it down into subgoals like reaching Surat or Nashik, each assigned to an autonomous agent. If the journey doesn't efficiently hit these subgoals, it indicates an ineffective route.

2. Using Reinforcement Learning (RL) -

In the N-puzzle problem, where the number of possible states grows quickly as the puzzle size increases, Reinforcement Learning (RL) offers a solution. RL allows an agent to learn how to make decisions, like choosing the next puzzle move, based on the current state. This is particularly useful when traditional search algorithms struggle with large state spaces.

RL agents learn through iterations, employing algorithms like Q-learning or policy gradient methods. Q-learning assigns a value (Q-value) to each possible move in a given state, representing the expected payoff. The reward function plays a crucial role, positively reinforcing correct moves and discouraging wrong ones. Striking a balance between exploring new options and relying on known strategies is vital for effective agent training.

Conclusion

So as to conclude everything has been stated so far . Now we can say we can successfully check the solvability of the 8 , 15 and the n puzzle(n^2 -1) and also analyse how to solve it using concepts of Discrete maths like Graph theory. With help of this project we get to know more about A* algorithm and how efficient it is in graph traversing to find an optimised way to reach the final solved state. This projects help us know everything from concept of inversions and parity to concept of heuristics , graph , and A* algorithm.

Additional comments

We can extend this study by finding a better solution in n puzzle as even though a* is faster than many algorithms but still it is very slow to find the final solution. We can even extend this puzzle in not just 2 dimension but in 3 dimensions N*N*N . we can even try to extend this by using Machine and Reinforced Learning algorithms

---

Contributions

Angadjeet Singh - 2022071
Aryan Jain - 2022111
Parth Sandeep Rastogi - 2022352

References :

1. Research Paper:
   - Title: A Distributed Approach To N-Puzzle Solving
   - URL:
https://www.researchgate.net/publication/2710914_A_Distributed_Approach_To_N-Puzzle_Solving
2. Online Demo:
   - Title: N-Puzzle Demo by Tristan Penman
   - URL: https://tristanpenman.com/demos/n-puzzle/
3. Academic Paper:
   - Title: Finding a Shortest Solution for the N x N Extension of the 15-Puzzle Is Intractable
   - Source: AAAI
   - URL:
https://aaai.org/papers/00168-AAAI86-027-finding-a-shortest-solution-for-the-n-x-n-extension-of-the-15-puzzle-is-intractable