MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Artificial Intelligence (CSE 2225) MINI PROJECT REPORT ON

Implementation of N-Puzzle Problem

SUBMITTED TO

Department of Computer Science & Engineering

by

Suchit Gupta
220962003
6
Ambuj Shukla
220962238
52
Arnav Jagia
220962324
50
J. Karthikeyan
220962272
41

Name & Signature of Evaluator

(Jan 2024 - May 2024)

Table of Contents

1. Introduction

1.1 Introduction

The N-puzzle, a generalization of the classic 8-puzzle game, presents a challenging problem in state-space search. In this game, players manipulate a grid of numbered tiles with one empty slot, aiming to reorder the tiles into numerical sequence.

This exploration delves into various algorithms and heuristics to efficiently solve the N-puzzle problem, focusing on strategies to navigate the vast state space and optimize performance.

1.2 Problem statement

The puzzle starts with a square frame consisting of (n – 1) numbered tiles and one empty space. The problem involves placing the numbers on tiles to match the final configuration using the empty space.

Four adjacent tiles (left, right, down, and up) can slide into the empty space. Each state will be represented as a 2D matrix and the transition between states will be through moving the empty square, which will have the value of 0.

1.3 Objectives

- Formulate a simple polynomial-time algorithm to determine if the initial state is solvable. If found not solvable, a search is not required.
- Explore different search techniques including both informed and uninformed methods with an emphasis on the A* algorithm.
- Investigate techniques for pruning the search space, thereby mitigating computational overhead and enhancing the efficiency of solution discovery.
- Provide insights into the effectiveness of different heuristic functions, comparing their performance in terms of solution optimality and computational efficiency.
- Develop an interactive N-puzzle game that applies dynamic state-space searching to offer hints to the user.

2. Literature survey

The problem of finding the minimal path for solving an N-puzzle is known to be NP-complete (Demaine et al., 2018). Most proposed solutions typically require steps on the order of $\Omega(n!)$ or higher (Ratner et al. 1986).

2.1 Uninformed search methods

- **Depth-first search**

In the context of the N-puzzle problem, DFS exhaustively explores each possible move from the initial state until it reaches a terminal state (either the goal state or a dead end). While DFS is straightforward to implement and requires minimal memory overhead, it suffers from following limitations:

– Completeness: Not complete as it may fail to find a solution if the search space is infinite or contains cycles.

– Time complexity: DFS exhibits an exponentially high time complexity of $O(n!)$ for N-puzzle.

- **Breadth first search**

BFS unintelligently explores all possible moves from the initial state before progressing to deeper levels. While BFS guarantees completeness and ensures optimality, its limitations include:

– Memory Consumption: BFS consumes significant memory resources, as it stores all generated nodes in memory until a solution is found. This can lead to scalability issues in scenarios with large state spaces.

– Time complexity: BFS exhibits a similar time complexity of $O(n!)$ for N-puzzle.

For both DFS and BFS, the run time will be proportional to the number of nodes in the graph.

2.2 Informed search methods

- **A***

In the N-puzzle, A* algorithm intelligently explores the state space by balancing the cost of reaching a state from the initial state with an estimate of the cost to reach the goal state. It selects nodes for expansion based on their combined cost and heuristic value, efficiently guiding the search towards the optimal solution with minimal computational overhead. Some details include:

– Optimality: A* guarantees the discovery of optimal solutions, making it ideal for scenarios where finding the shortest path is critical.

– Heuristic Guidance: The integration of heuristic guidance allows A* to efficiently explore the state space, leading to faster convergence towards the solution.

– Time complexity: A* improves the average-case performance, but the asymptotic bound will remain on the order of $O(n!)$. However, the average-case time complexity is greatly reduced when the number of edges separating the initial and goal state is within reason.

2.3 Heuristics

- **Hamming distance**

For an n-puzzle, hamming distance is the total number of tiles which are not in their correct position.

- **Euclidean distance**

The sum of the Euclidean distances of each tile from their goal position.

- **Manhattan distance**

For an n-puzzle, the Manhattan distance is the sum of distance between tiles and their correct position.

Clearly, the heuristic value stemming from the Manhattan distance will necessarily be greater than the heuristics based on Hamming or Euclidean distance. It is desirable to pick a heuristic that closely approximates the actual distance to goal (Hart P.E. et al., 1968). Knowing that Manhattan distance is admissible while producing the highest values, it is the most compelling choice for our use.

3. Methodology

3.1 Representation

Each unique configuration of the N-puzzle board is treated as a state in the search space of the problem. All nodes that are separated by a single move in the game are connected by an edge.

The permutation of the tiles can be conveniently represented in a square matrix with N cells, each corresponding to a tile on the N-puzzle board.

What states a particular permutation of tiles can transition to depends on the position of its empty space. To speed up the generation of neighboring states, the position of the empty space is cached during object instantiation. This avoids an expensive O(n^2) search through the matrix.

A wrapper class `BoardInstance` is built to encapsulate the matrix along with the blank tile position and to implement methods such as `__hash__`, `__eq__`, and `__str__` for making the type compatible with built-in data types.

3.2 Search

Half of all N-puzzle states are known to not be solvable (Johnson, W.W., 1879). These instances can be interpreted as being on a different connected component of the state space from their goal.

To avoid searching for a path between these disjoint nodes, we count the number of inverted pairs in the current state with respect to the goal state. The problem is solvable if and only if the count of inversions is even. [5] This validation happens in O(n) time and saves us from traversing the entire subgraph containing the initial state.

If the problem is found to be solvable, the search for an optimal path is undertaken by A* with a min-heap. For the reasons outlined in section 2.3, the heuristic is computed as the sum of the Manhattan distances of corresponding tiles in the current state and the goal state. When the goal state is popped from the frontier, the path is backtracked and returned.

3.3 Testing

For test cases, the initial state and goal states are generated randomly from the set of all possibilities. This is done without regards to the solvability of the problem. The search utility is then dispatched in a separate thread with a 3 second timeout.

3.4 Interactive demo

The game starts by generating a goal state with a random permutation of tiles, implemented by shuffling the configuration matrix in `BoardInstance`. The initial state is generated by a random walk in the subgraph containing the goal state. The number of steps in the random walk can be adjusted to control the “difficulty” of the game.

The user has to play to reach the goal state from the current state. At each move, the user has the option to shift the blank tile to any of its valid neighbors. Alternatively, the user may request a hint.

The hint simply returns the next state in the optimal path from the current state. This path is recomputed after every move.

When the user arrives at the goal state, the game concludes. A score is awarded based on how many extra moves were made relative to the optimal path and how many hints were requested.

4. **Results and Discussion**

The software package generates the search graph and expands only those nodes that promise a solution at the next step. Our implementation can solve any randomly generated N-puzzle problem for values of N up to 15.

Beyond that point, due to limitations on the computational power of our systems and the NP-hard nature of the problem, we are unable to get solutions in a healthy range of time.

However, with some modifications to the problem generator, the interactive demo remains responsive for values of N up to 63.

5. Future Enhancements

Iterative Deepening A* (IDA*)

Variant of A* algorithm performing depth-first search strategy iteratively with increasing depth limits until a solution is found. IDA* maintains a threshold value, representing the maximum solution cost found. If a path's cost exceeds this threshold, the search backtracks to explore alternatives.

The IDA* algorithm's performance advantage over A* is due to its ability to handle tied f(n) values more effectively, reducing the number of nodes that need to be expanded.

IDA* outperformed A* in all tested cases, taking significantly less time to solve 8, 15, and 24-puzzles and was 70-100% faster than A* in solving N-puzzles.

Parallel Breadth First Heuristic Search (PBFSH)

Parallel Breadth-First Heuristic Search aims to efficiently explore the state space by considering multiple possible moves from the current state in parallel and distributes the computational workload across multiple cores or processors, enabling simultaneous exploration.

This heuristic guidance helps prioritize the exploration of more promising regions of the state space, leading to faster convergence towards the optimal solution.

The scalability of Breadth First Search (BFS) and Iterative Deepening A* (IDA*) is limited to a single machine due to hardware constraints, making PBFHS a better choice for large combinatorial problems like N-puzzle.

Using Artificial Neural Networks (ANN) with A* Algorithm

ANN can be utilized to design heuristics for near-optimal solving of N-puzzle, in conjunction with the A* search algorithm.

The ANN-distance heuristic, based on a deep artificial neural network, provides better accuracy in estimating the minimum number of moves needed to reach the goal compared to conventional heuristics like Manhattan distance.

Deep ANN-distance heuristics trained with the Mean Squared Error (MSE) cost function can lead to more optimal solutions and consume less memory than pattern database heuristics.

References

1. Demaine, E.D., & Rudoy, M. (2018). *A simple proof that the (n^2 – 1) puzzle is hard*. doi.org/10.1016/j.tcs.2018.04.031

2. Ratner., D., & Warmuth, M. (1986). *The (n^2 – 1)–Puzzle and Related Relocation Problems.* doi.org/10.1016/S0747-7171(08)80001-6

3. Hart, P.E., Nilsson, N.J., & Raphael B. (1968). *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* doi.org/10.1109/TSSC.1968.300136

4. Johnson, W.W., & Story, W.E. (1879). *Notes on the "15" Puzzle.* https://www.jstor.org/stable/2369492

5. "How to check if a 8-puzzle is solvable?" – math.stackexchange.com https://math.stackexchange.com/questions/293527/how-to-check-if-a-8-puzzle-is-solvable

6. Evaluating Search Algorithms for Solving n-Puzzle. n-puzzle.pdf (sumitg.com)

7. Implementation and Analysis of Iterative MapReduce Based Heuristic Algorithm for Solving N-Puzzle. jcp0902.pdf (jcomputers.us)

8. Near Optimal Solving of the (N2–1)-Puzzle Using Heuristics Based on Artificial Neural Networks. 2021-near-optimal-solving.pdf (cahlik.net)