Iterative Deepening A* for the 15-puzzle: An Analysis

Robert Ward II

September 2023

1 Introduction and Hypothesis

The primary objective of this project revolves around implementing, executing, and subsequently analyzing the outcomes derived from the application of the Iterative Deepening A* (IDA*) search algorithm, with a particular focus on solving the classic 15 puzzle. This puzzle is a sliding puzzle that consists of a frame of numbered square tiles in random order with one tile missing, and the aim is to place the tiles in order by making sliding moves that use the empty space[1].

Given the intricacy and sophistication inherent in algorithmic execution and analysis, a meticulous approach was adopted. Python was initially chosen as the programming language for this project due to its readability and ease of understanding. This choice enabled a quick grasp of the algorithm and its accompanying limitations, primarily in terms of execution speed, allowing for a smoother developmental transition and refined analytical procedure.

In terms of the scope, the project aimed to critically assess the algorithm across two distinct metrics: average execution time and the search depth required for the solution. To ensure the robustness and validity of the evaluation, a collection of 500 known solvable problems was constituted as the testing bank upon which the program would be executed. This exhaustive list of problems serves as a comprehensive framework to extensively evaluate the efficacy and efficiency of the algorithm, providing a multifaceted perspective on its operational capabilities.

For each run of the program on a problem from the problem bank, multiple facets of the process were recorded. These encompassed the initial state of the puzzle, the successive steps required to maneuver the blank tile to achieve a solved state, the run time in milliseconds, the overall search depth, and the number of nodes expanded. This extensive and detailed record-keeping was vital in fostering a more nuanced understanding of the algorithm's functioning and the variables influencing its performance, and all this information was meticulously documented in a simple text file.

Additionally, to incorporate a comparative dimension to the project, two versions of the program were developed, utilizing two closely related heuristics:

the Manhattan distance and the Manhattan distance augmented with linear conflict. The incorporation of these heuristics aimed at facilitating an analytical contrast, thereby illuminating the disparities in performance and efficiency between them.

The initial hypothesis posited that the version employing Manhattan distance coupled with linear conflict would exhibit superior performance relative to the one utilizing Manhattan distance alone. However, the overarching goal of the project was not merely to validate this expectation but to quantitatively and qualitatively gauge the extent of this potential enhancement in performance. By scrutinizing the different outcomes yielded by the application of these heuristics, the project sought to elucidate the intrinsic merits and possible limitations inherent in each approach, thereby contributing to a richer, more nuanced understanding of heuristic efficiency in the context of the Iterative Deepening A* search algorithm solving the classic 15 puzzle[2]

2 Iterative deepening A*

The Iterative Deepening A* (IDA*) search algorithm is a prominent strategy in the realm of artificial intelligence, developed to overcome the hurdles inherent in its counterparts, A* and depth-first search. IDA* amalgamates the space-efficiency of depth-first search with the optimality and completeness of A*. It scrutinizes paths in the search tree by progressively intensifying the depth limit until a solution is found, enabling the algorithm to always find the optimal solution with reduced memory overhead, especially vital for problems with large search spaces like the 15-puzzle [2].

3 The 15-puzzle or the NxN puzzle

The 15-puzzle,an instance of the general NxN puzzle, is a classic puzzle game, renowned for its combinatorial challenge. It comprises fifteen sequentially numbered tiles ensconced within a four-by-four frame, leaving one space vacant. The goal is to transition from a randomized initial state to a solved state, where the tiles are in ascending order, utilizing the vacant space to slide tiles. The game exemplifies a meticulous permutation puzzle and serves as a fertile ground for algorithmic experimentation, especially in exploring search algorithms and heuristic evaluations [1].

4 First attempt and Language Change

The project initially embraced Python as the chosen language, leveraging its user-friendly and intuitive nature for writing and implementing accurate code for the Iterative Deepening A* (IDA*) algorithm. However, Python's interpreted nature soon revealed inherent limitations and bottlenecks, particularly evident when addressing more intricate problems. Specifically, when the required search

depth surpassed 30, the program experienced substantial slowdowns, with execution times often approaching an hour.

Questions regarding whether these slowdowns were attributable to implementation inefficiencies were considered; however, detailed scrutiny on this front was not undertaken. A prevailing skepticism existed, questioning whether such inefficiencies, even if present, could substantially contribute to the pronounced delays experienced. The approach employed in the Python implementation of IDA* also introduced a distinct method, incorporating a list to manage the nodes set for expansion. This method, upon reflection, was identified as a suboptimal choice, resulting in significant memory inefficiencies and a departure from the core principles of IDA*. This departure could, lead one to argue whether the developed solution could truly be classified as Iterative Deepening A* or perhaps as a closely related algorithm.

Given that this was an exploratory and initial venture, exhaustive research into the nuances of IDA* was not conducted prior to the implementation, leading the first attempt to be marred by several technical inaccuracies, inefficiencies, and unintentional mistakes, underscoring the imperative need for more refined and accurate subsequent attempts. And as such all further discussion will be in relation to the C++ implementation compiled with level three compiler optimizations enabled.

5 Generating the initial Puzzle states

Initially, the puzzles for testing the algorithm were sourced from an online game.[5], but later a Python script was developed to generate initial puzzle states for convenience. These states were represented by a random string of numbers ranging from 0 to 15, with each number being unique, arranged in a single line, and with spaces between each number. The 0 within this sequence represents the blank tile

This Python script also determined the solubility of the puzzles by calculating inversion count coupled with the parity of the row number of the blank tile. The parity was determined by counting from the bottom row to the top starting at one,and an inversion in this context is whenever the state is considered in the form of a 1D array such as 15 7 11 8 14 3 2 4 12 6 9 10 0 5 1 13 each pair of numbers considered left to right with no loop back.

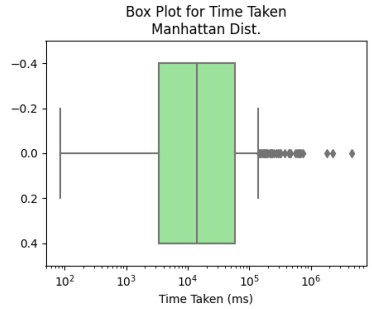
If the first number in the pair is greater than the second number in the pair then the pair is considered an inversion.An example of an inversion pair from the above sequence would by (15, 7) as 15 comes before 7 and 15 is greater than 7 conversely (7,11) is not an inversion as 7 comes before 11 but it is not greater than 11. If the pair contained zero in either place it was automatically not considered an inversion as the zero just represents a space that could simply be filled by the next (or previous) number in the list. These inversions were counted and the parity of the row number and inversion count was used to determine if a problem was solvable or not.[1]

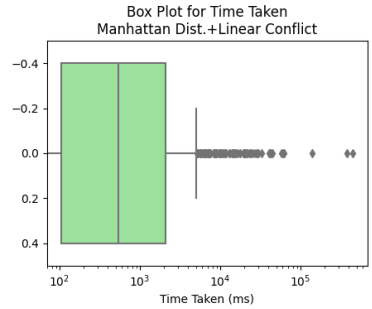
6 Experiment

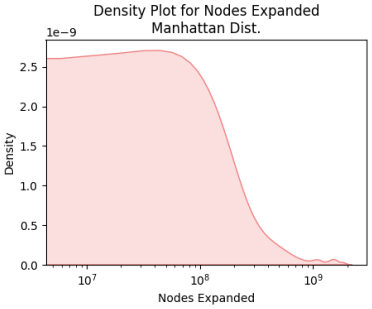
The experiment involved running the initial algorithm, which solely used Manhattan distance, on five hundred randomly generated solvable instances of the 15-puzzle. Subsequently, the next implementation was executed on the same five hundred puzzle instances. Data collected from both implementations included: runtime for each puzzle, maximum search depth reached, number of nodes expanded, and the sequence of moves to solve a particular instance. This data was then saved to JSON file for easy parsing by python to later be visualized and analyzed using matplotlib and seaborn among other python libraries

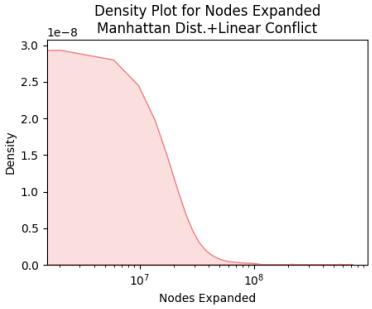
7 Analysis

Several plots were created to illuminate the performance characteristics each heuristic had on the algorithm. Several of these are illustrated below.



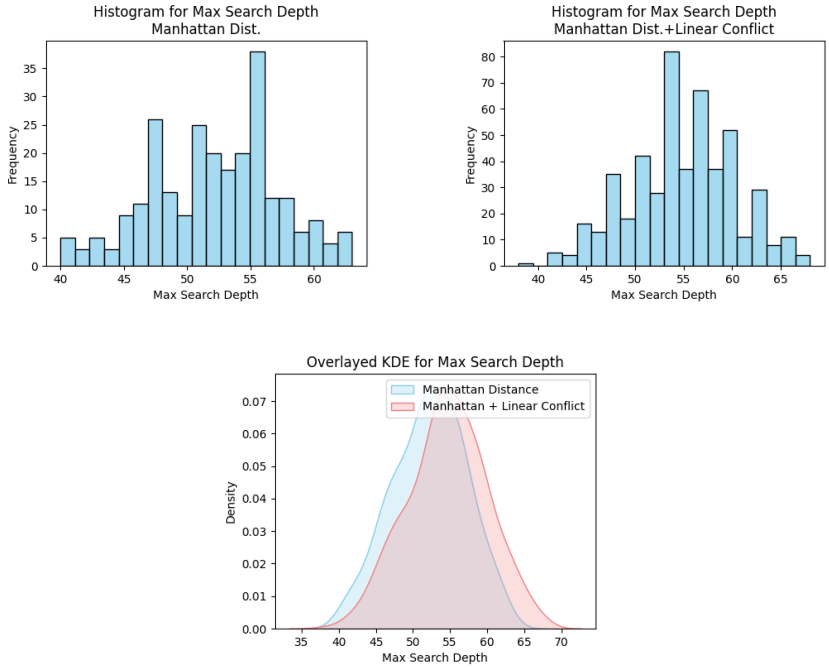






In the above figures it can be seen that the initial hypothesis is likely correct as Manhattan distance consistently under performed Manhattan distance with linear conflict in both overall average run time and the number of nodes

expanded. Most notably is the chart for Manhattan distance shows that the density of times more than 10^8 nodes were expanded vastly outpaces that for Manhattan distance with linear conflict. This likely follows from the fact that linear conflict improves upon Manhattan distance by making it aware of conflicts in both the rows and columns elucidating more accurately how far a given tile is from its goal more so that solely MD.



The charts above illustrate a slightly higher search depth frequency for MD+LC compared to MD, possibly influenced by the limited sample size of 500 puzzles. This does not alter the optimal solution depth, which remains constant for a given puzzle regardless of the heuristic. However, the difference in the average max search depth can be attributed to the distinct exploration strategies of the heuristics.

MD, being less informed, explores more nodes at shallower depths, which might make it seem closer to the solution but may require more time due to extensive exploration of less promising nodes. In contrast, MD+LC, more informed, bypasses these less promising shallow nodes, exploring deeper, more promising paths more quickly, which is why it might reach higher search depths before finding the optimal solution.

This illustrates the differing natures of MD and MD+LC within the IDA* algorithm and emphasizes the significance of heuristic choice in impacting the search efficiency and the paths explored to reach the solution.

8 Conclusion

In conclusion, this project embarked on an exploratory journey to investigate the effectiveness of the Iterative Deepening A* algorithm in solving the classic 15-puzzle, through a systematic comparative analysis between Manhattan Distance (MD) and Manhattan Distance with Linear Conflict (MD+LC) heuristics. The insights gleaned from the experimental outcomes substantiate the hypothesis, underscoring the superior efficiency of MD+LC in both execution time and nodes expanded, albeit with a nuanced exploration strategy leading to higher search depths. These findings serve to elucidate the pivotal role of heuristic selection in optimizing search algorithms, fostering a deeper comprehension of their operational intricacies and potential improvements, thus offering a robust foundational understanding of heuristic search algorithms.

References

[1] Wm. Woolsey Johnson and William E. Story. Notes on the "15" puzzle. *American Journal of Mathematics*, 2:397, 12 1879.

[2] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 9 1985.

[3] Richard E. Korf, Michael Reid, and Stefan Edelkamp. Time complexity of iterative-deepening-a*. *Artificial Intelligence*, 129:199–218, 6 2001.

[4] Numberphile. Why is this 15-puzzle impossible, 2020. YouTube: https://www.youtube.com/watch?v=YI1WqYKHi78.

[5] Shubham Singh. 15 puzzle. `https://15puzzle.netlify.app/`. Accessed: 2023-09-01 to 2023-09-26; Git Repository: `https://github.com/imshubhamsingh/15-puzzle`.