Comparison of Heuristic Search Algorithms in Solving 11-puzzle Problems

Abstract

This paper presents a comparative analysis of theA* and Iterative Deepening A* (IDA*) search algorithms to solvethe 11-puzzle problems using the Manhattan distance heuristic.Both algorithms were implemented and evaluatedbased on performance metrics including nodes generated, nodesexpanded, solution depth, effective branching factor, and CPUtime. The results indicate that A* consistently outperforms IDA*in computational efficiency and scalability with A* reducing the node generation by 62.86%, node expansion by 61.60%, and CPU time by 51.46%, though IDA* remainsmore memory efficient. These findings validate the broaderapplicability of heuristic search strategies and reinforce the roleof the Manhattan distance heuristic in optimal path finding.

Index Terms: A* algorithm, IDA* algorithm, Manhattan distanceheuristic, 11-puzzle

Introduction

Heuristic search algorithms play a pivotal role in artificialintelligence, especially in solving combinatorial optimizationand path finding problems. Among these, A* and IterativeDeepening A* (IDA*) search algorithms have emerged astwo of the most prominent informed search strategies due totheir ability to find optimal solutions using heuristic guidance.A* is known for its efficient exploration of the search spacethrough the use of an evaluation function that combines path cost andestimated cost of distance to the goal, while IDA* offers a memoryefficientalternative by using iterative deepening to limit spacecomplexity. Both algorithms have been widely applied andtested in standard search problems, particularly in puzzlesolvingtasks.One such widely studied domain is the sliding tile puzzle,with the 8-puzzle and 15-puzzle being the most commonbenchmarks for evaluating the performance of search algorithms.These puzzles offer a controlled and well-understood environmentfor measuring metrics such as node generation,node expansion, and computational efficiency. However, thereremains a lack of research focusing on mid-complexity puzzleconfigurations like the 11-puzzle, which has a state space of more than 200 million nodes, and sits between thesimplicity of the 8-puzzle and the greater complexity of the 15-puzzle. Exploring this under-represented puzzle variant offersa valuable opportunity to assess how algorithmic behavioursscale with increasing problem size and complexity.

This research bridges this gap by doing a comparativeanalysis of the A* and IDA* search algorithms using the 11-puzzle as the test domain. The Manhattan distance heuristic,an admissible and widely used metric based on tile movement estimating cost, is used as the heuristic function for the twoalgorithms. A custom puzzle generator is developed in Pythonto produce a set of randomly generated, solvable 11-puzzleproblems. Each algorithm is then evaluated using five keyperformance indicators; number of nodes generated, numberof nodes expanded, effective branching factor, solution depth,and CPU time.

Our objective in this research is to determine which algorithmprovides superior performance in terms of computationalefficiency and scalability while maintaining solution optimality.The findings of this research would not only reinforce theoreticalexpectations about heuristic search algorithms but also validatethe applicability of the Manhattan distance heuristic to midcomplexitypuzzle problems. Furthermore, the research contributesto the broader field by confirming whether the results observedin traditional puzzles would extend to the 11-puzzle, supporting itsuse as a valid benchmark for future studies.

Literature Review

Heuristic search algorithms are essential tools in artificialintelligence (AI) for solving state-space problems wherethe search space can be vast and computationally intensive.Heuristic search algorithms employ domain-specificknowledge to guide the search towards the goal state moreefficiently than uninformed algorithms such as Breadth-FirstSearch (BFS) or Depth-First Search (DFS). Among the manyheuristic-based methods, the A* search algorithm can beconsidered a fundamental method due to its optimality andcompleteness when coupled with admissible heuristics It usesan evaluation function $f(n) = g(n) + h(n)$; where $g(n)$represents the cost to reach the current node from the initial state, and $h(n)$ is theheuristic estimate to the goal. IDA* (Iterative Deepening A*)is a variant that combines the depth-first nature of iterativedeepening with the heuristic-informed approach of A*, aimingto reduce memory usage while still finding optimal solutions.

The sliding tile puzzle, particularly the 8-puzzle and 15-puzzle, has served as a benchmark for evaluating such heuristicsearch algorithms due to its clear state space, optimal solutions, and practical complexity. Prior research has extensivelyexamined how A* and IDA* perform on theseproblems. [2] conducted a comparative study demonstratingthat the Manhattan distance heuristic significantly improvesthe efficiency of A* over simpler heuristics such as Hamming distance. [1] found that A* using the Manhattan distanceheuristic dramatically reduced node expansions and improvedruntime compared to Uniform Cost Search and Euclideanbasedheuristics, achieving over 99% improvement in averageperformance metrics. [3] compared A* and Greedy Best-FirstSearch on the 15-puzzle and observed that while GreedyBest-First was faster, A* consistently produced more optimalsolutions.

Additional studies have examined enhancements and limitationsof heuristic approaches. [4] proposed hybrid heuristics,such as combining Manhattan distance heuristic with LinearConflict, to improve node expansion rates. [5] exploredhow less consistent heuristics might still outperform moreconsistent ones under certain conditions, particularly in largeproblem spaces. Meanwhile, [9] introduced additive patterndatabase heuristics as a more powerful alternative, althoughthey also come with higher memory requirements. [13] and[14] further analysed the behaviour of IDA*, particularlyhighlighting its tendency for redundant node re-expansion dueto the lack of memory structures like open and closed lists.

Other empirical studies, such as [6] and [7], reinforce theadvantages of informed algorithms such as A* in solving8-puzzle configurations. [8] emphasized the importance ofselectingsuitable heuristics, demonstrating how Manhattandistance heuristic balances efficiency and accuracy. The benefitsof run-time adaptability were highlighted in [10], wherethe rational deployment of multiple heuristics in IDA* wasexplored. Hybrid approaches such as A*+IDA* [12] andA*+BFHS [11] have been proposed to combine memoryefficiency with faster convergence.

Prior studies consistently show that A* minimizes node expansions when sufficient memory is available, while IDA* trades runtime efficiency for space savings. However, scalability trends across mid-sized puzzles remain unclear.Despite the depth of existing research, most studies have focusedon the 8-puzzle and 15-puzzle domains. Mid-complexityconfigurations, have received little attention in the literature.This research addresses that gap by evaluating the performanceof A* and IDA*search algorithms on the 11-puzzle, using theManhattan distance heuristic. By doing so, it provides newempirical insights into whether algorithmic trends observed insmaller puzzles scale to more complex configurations, and itvalidates the general applicability of heuristic strategies in abroader state-space search context.

Methodology

This study was designed to investigate and compare theperformance of the A* and Iterative Deepening A* (IDA*)search algorithms in solving the 11-puzzle problem using theManhattan distance heuristic. The methodology consists offour main stages; the generation of puzzle instances, implementationof algorithms, heuristic function definition, evaluationof performance of each metric.

Puzzle Instance Generation

To ensure a balanced and unbiased assessment, a Pythonbasedpuzzle generator was developed to create a large setof randomly shuffled but solvable 11-puzzle instances. Eachpuzzle consisted of 12 tiles arranged in a 4x3 grid (Fig 1), includingone blank tile (denoted by 0). The solvability of each puzzleinstance was verified using the inversion rule adapted for evensized
grids. A puzzle is solvable if the sum of the number of inversions and the row number of the blank tile (from the bottom) is even.This ensured all instances produced had valid solutions sothat both algorithms could reach an optimal goal state foreffective comparison.

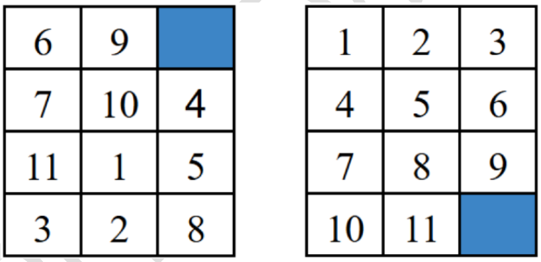

Fig 1:A solvable problem instance (left) and goal (right)

Algorithm Implementation

Both A* and IDA* were implemented in Python. Each algorithmused the same state representation, node expansion logic,and goal-checking mechanism to eliminate implementationbias. The algorithms differ primarily in their search strategyand memory usage.

- A* uses an evaluation function: $f(n) = g(n) + h(n)$; where
  $g(n)$: cost to reach node n from the initial state.
  $h(n)$: estimated cost from n to the goal, computed usingthe Manhattan distance heuristic.
- IDA* performs iterative deepening depth-first search guided by the same $f(n)$ evaluation. It repeatedly searches with increasing threshold limits until a solution is found.

Heuristic Function

The Manhattan distance heuristic was chosen due to itsadmissibility, simplicity, and effectiveness in guiding searchalgorithms on sliding tile puzzles. It calculates the sum of thehorizontal and vertical distances each tile must move from itscurrent position to its goal position:
Heuristic function:

$$h_M(S) = \Sigma_{k \in \{1, 2, \ldots, N\}} MD(k) \quad (1)$$

where:

$$MD(k) = |x_k - x_{kg}| + |y_k - y_{kg}| \quad (2)$$

$(x_k - y_k)$: current position of tile k

$(x_{kg} - y_{kg})$: goal position of tile k

N: number of tiles excluding the blank tile

This heuristic guides both A* and IDA* to explore statesthat appear closest to the goal.

Performance Metrics

The effectiveness of each algorithm was evaluated using thefollowing metrics.

1. Nodes Generated: The total number of nodes (states) generated during the search.
2. Nodes Expanded: The number of nodes from which successors were created.
3. Effective Branching Factor (EBF): A measure of the average number of child nodes generated per expanded node, computed as:

$$N + 1 = 1 + b + b^2 + \cdot\ \cdot\ \cdot + b^d \quad (3)$$

where:

N: total number of nodes generated

d: depth of the optimal solution

b: effective branching factor

4. CPU Time: The total execution time required to solve each instance.
5. Solution Depth: The number of moves required to reach the goal from the initial configuration.

All metrics were averaged over a large number of testcases to ensure statistical reliability and to identify consistentpatterns in algorithm behaviour.

Evaluation Procedure

The experimental design ensured that every algorithm solved thesame instances of puzzles. Results were recorded for everymetric per instance and then aggregated. Graphs and tableswere used to visualize trends across varying solution depths.Special attention was given to the problem instances having solution depth 36, which has been found to be the average solution depth in the dataset. Thiscomprehensive methodology allowed for a fair, reproducible,and insightful comparison of A* and IDA* under controlledconditions, using the Manhattan distance heuristic as theguiding function.

Results and Discussion

This section presents the comparative performance analysisof the A* and IDA* search algorithms when applied to the 11-puzzle problem using the Manhattan distance heuristic. The

results were obtained from solving over two million randomlygenerated, solvable 11-puzzle problem instances. Each algorithm wasassessed using five performance metrics; number of nodesgenerated, number of nodes expanded, effective branchingfactor, CPU time, and solution depth.All experiments were executed on a PC having a 4.0 GHz quad core processor, 24 GB GPU, and 64 GB RAM.

Nodes Generated

The number of nodes generated reflects how broadly eachalgorithm explores the state space (Fig 2). A* generatedsignificantly fewer nodes on average compared to IDA*.Specifically, A* reduced node generation by approximately62.86%, highlighting its efficiency in pruning irrelevant pathsearly during the search. This efficiency is attributed to A*'suse of the Manhattan distance heuristic to prioritize pathsthat are closer to the goal, reducing unnecessary expansions.IDA*, in contrast, repeatedly regenerates nodes across multipleiterations due to its iterative deepening structure.

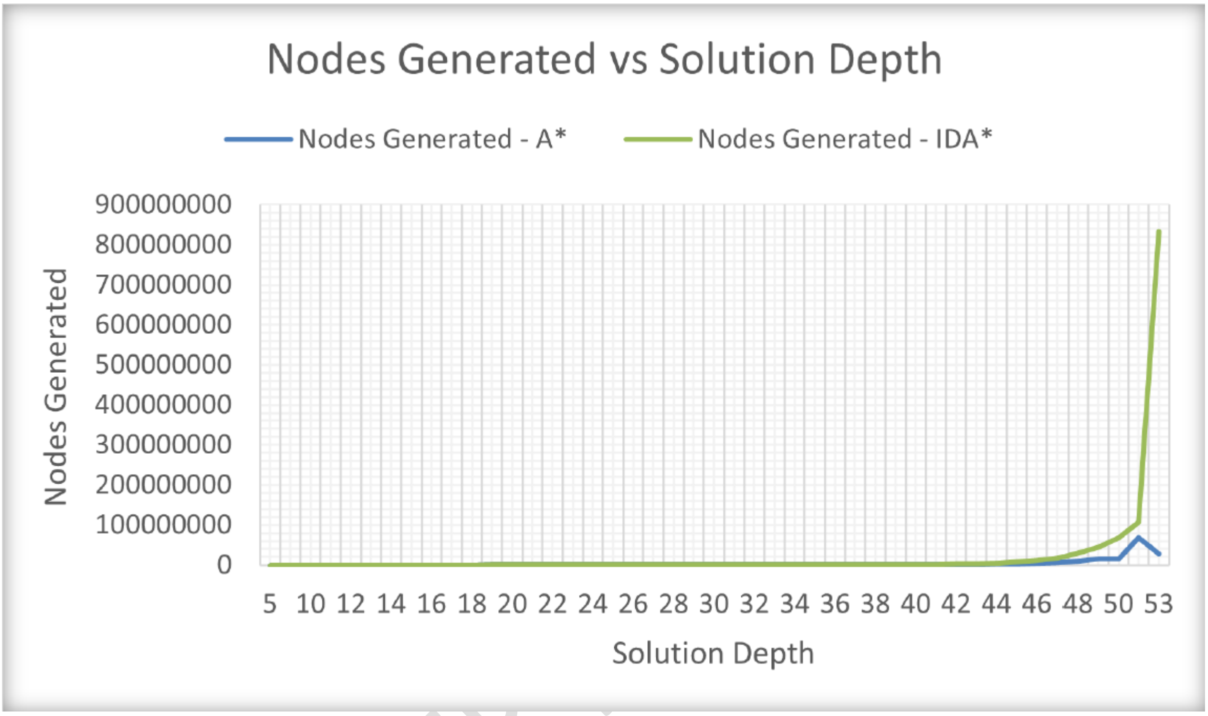

Fig 2:Average number of nodes generated

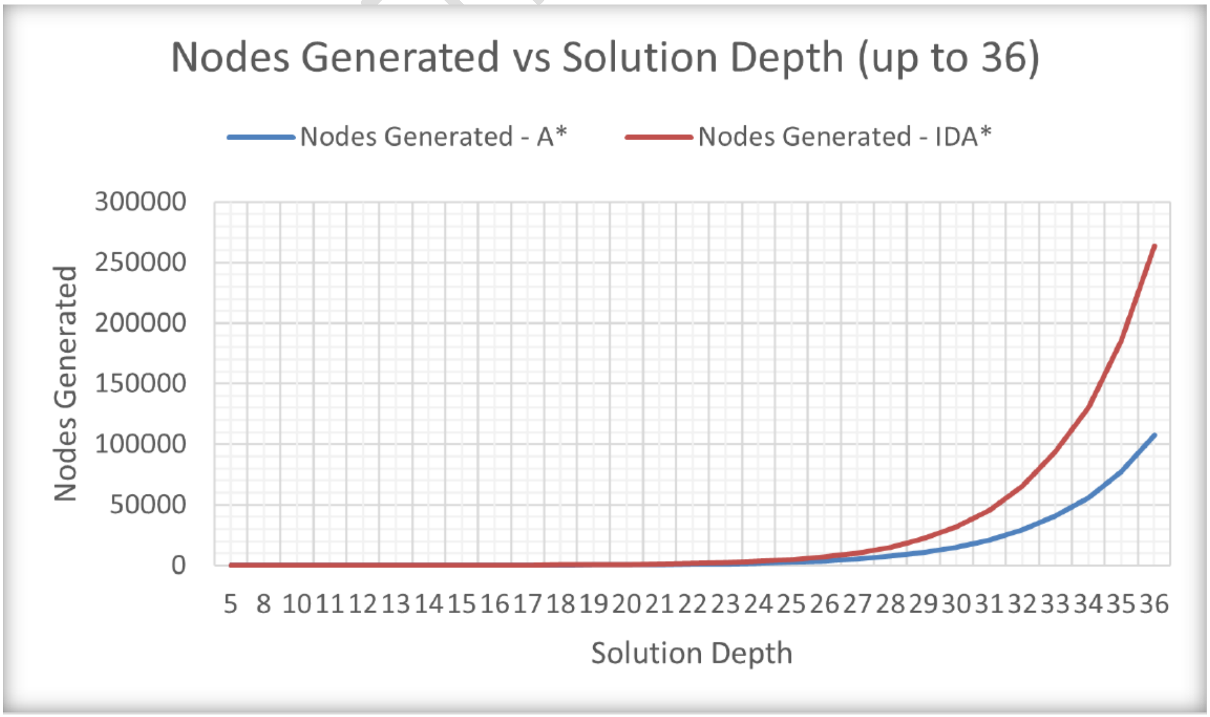

Fig 3:Average number of nodes generated up to solutiondepth 36

Fig 3 illustrates the trend in the number of nodes generatedby A* and IDA* across varying solution depths up to 36. Asthe solution depth increases, both algorithms naturally generatemore nodes due to the expanded search space. However,IDA* displays a steeper growth curve compared to A*. Thisis attributed to IDA*'s repeated re-expansion of the samestates during each iterative deepening cycle, particularly as thedepth threshold increases. In contrast, A* maintains a moremoderate and predictable growth due to its heuristic-guidedexploration and memory usage, which prevents revisitingalready expanded nodes. The figure highlights A*'s scalabilityand efficiency in managing node generation even as problemcomplexity increases. This reinforces the Manhattan distanceheuristic's effectiveness in steering the search process towardoptimal paths without exploring unnecessary branches.

Nodes Expanded

The number of nodes expanded provides a direct indicationof the processing load, as each expansion requires the algorithmto evaluate successors and update data structures (Fig4). A* expanded 61.6% fewer nodes than IDA*, demonstratingnot only that it generated fewer nodes, but also that it was moreselective in which nodes were expanded. IDA*'s repeated nodeexpansions, due to the absence of memory structures such asopen and closed lists, caused greater computational overhead.

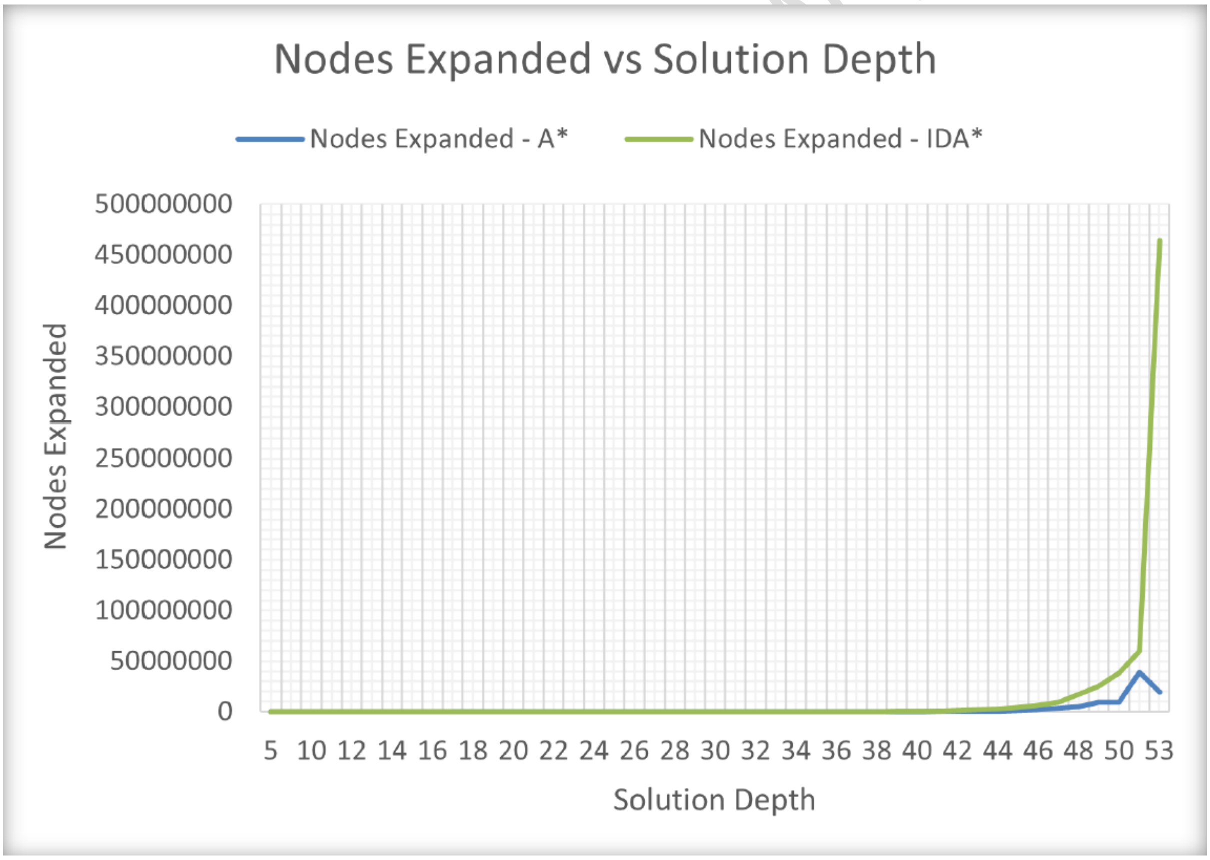

Fig 4:Average number of nodes expanded

Fig 5 illustrates the number of nodes expanded by both A*and IDA* algorithms up to solution depth 36. Similar to nodegeneration trends, node expansion also increases with depthfor both algorithms. However, IDA* has an irregularly highgrowth rate, especially after depth 25. This is again becauseIDA* lacks memory structures such as open and closed lists,causing the algorithm to repeatedly expand nodes it hasalready processed in previous iterations. A* demonstrates amore stable and lower growth rate in node expansion dueto its informed approach and its ability to avoid redundantprocessing. The Manhattan distance heuristic plays a criticalrole here by helping A* prioritize nodes closer to the goal and thus, reduce unnecessary expansions. This graph also supportsthe fact that A* is computationally more efficient and scalablefor deeper search instances.

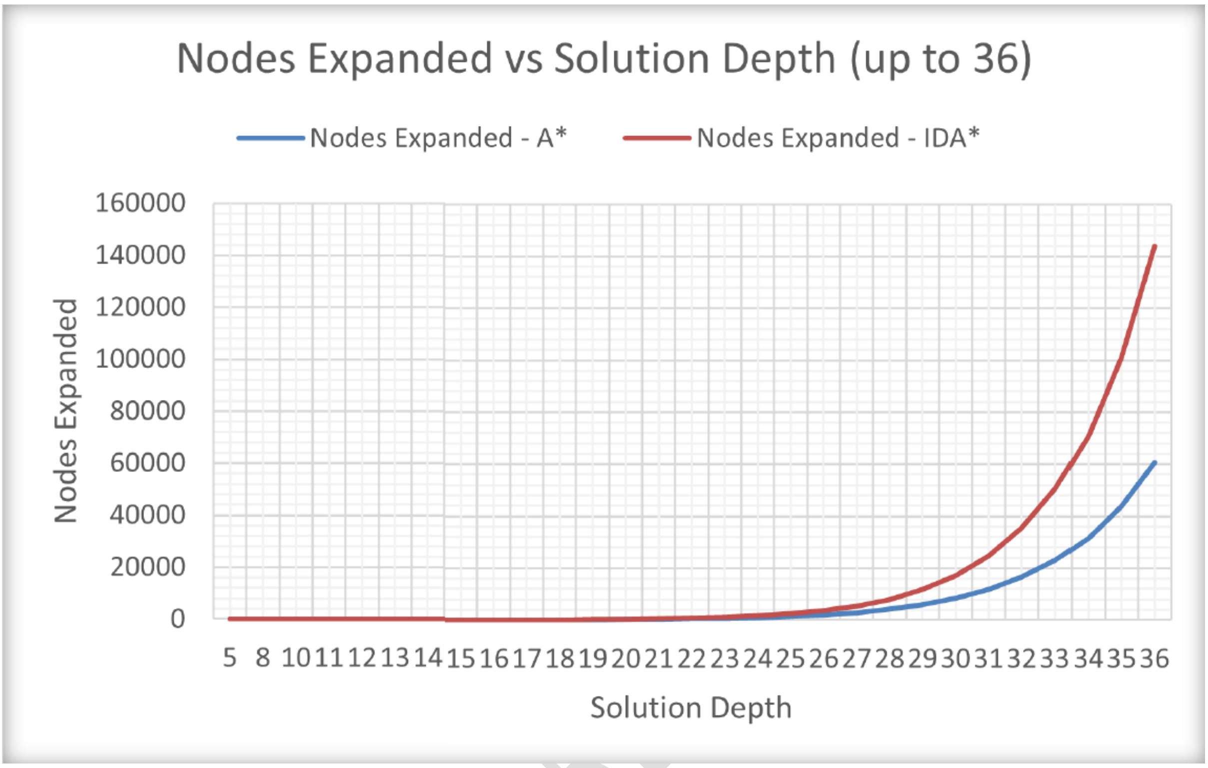

Fig 5:Average number of nodes expanded up to solutiondepth 36

Effective Branching Factor

The effective branching factor (EBF) measures how manychild nodes are explored on average at each level of the searchtree. As depicted in Fig. 6, A* showed a lower average EBF of 1.7254 compared to1.8261 for IDA*, representing a 5.51% reduction. While thenumerical difference appears small, it translates into significantcomputational savings at higher solution depths due to theexponential nature of search trees. Furthermore, A*'s EBFdecreased slightly with increased depth, indicating that itbecame more focused as the search progressed, an advantagegiven by the Manhattan distance heuristic.

CPU Time:

CPU time was measured to assess the real-world efficiencyof each algorithm. A* consistently outperformed IDA* acrossall depths, solving puzzle instances in approximately 51.46%less time. This difference became more pronounced withincreased solution depth (Fig. 7). The results confirm that A*'s guidedsearch using the Manhattan distance heuristic significantlyreduces execution time by avoiding unnecessary reprocessingof nodes. IDA*'s CPU time grew steeply with depth due toits exhaustive re-expansion strategy.

Fig 8 shows how A* and IDA* algorithms consume moreCPU time with growing solution depth up to 36. Both algorithmsexperience longer execution times at higher depths dueto increased search effort. However, IDA*'s runtime growsat a much faster rate than A*, particularly beyond depth25. This is simply because IDA*'s repeated reprocessingof nodes across multiple depth-limited iterations. On theother hand, A* demonstrates a relatively gradual increasein execution time, attributed to its informed search strategypowered by the Manhattan distance heuristic, which facilitatesthe algorithm’s faster convergence to the goal by exploringpromising directions first. The widening performance gap athigher depths emphasizes A*’s superior time efficiency andsuitability for time-sensitive applications, specifically thoseinvolving moderately difficult puzzle spaces such as the 11-puzzle.

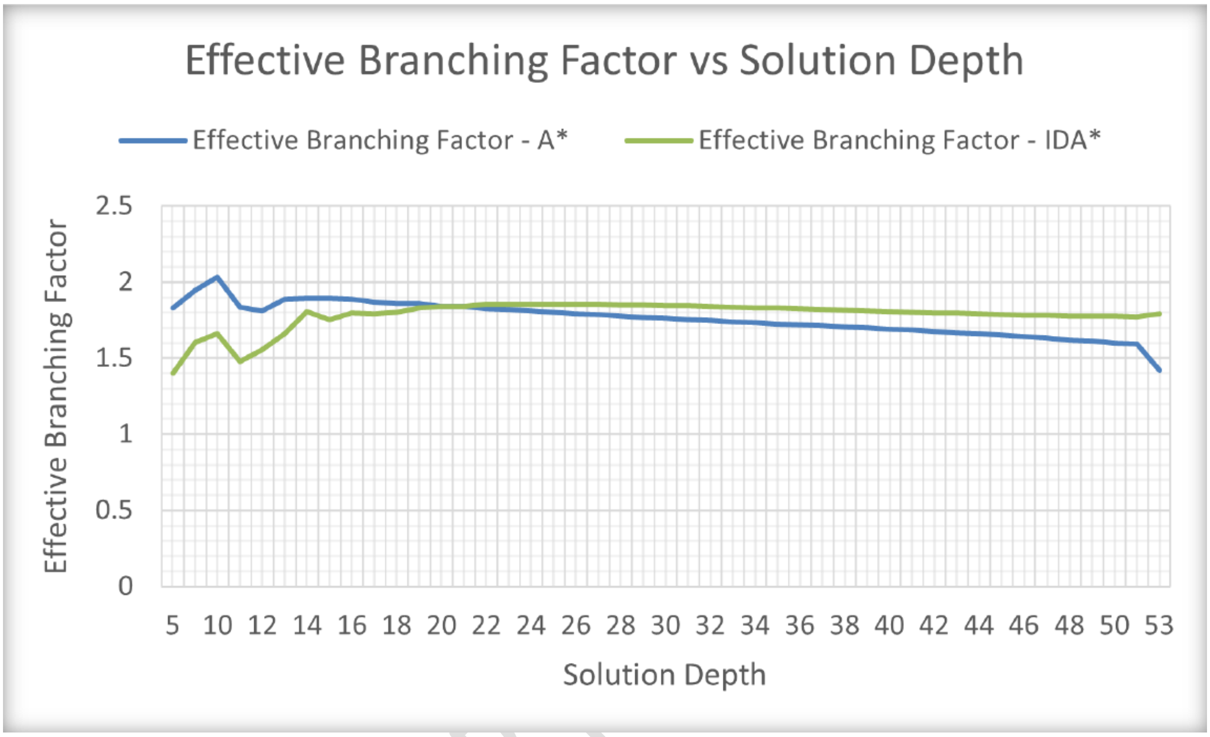

Fig 6: Effective branching factor against solution depth

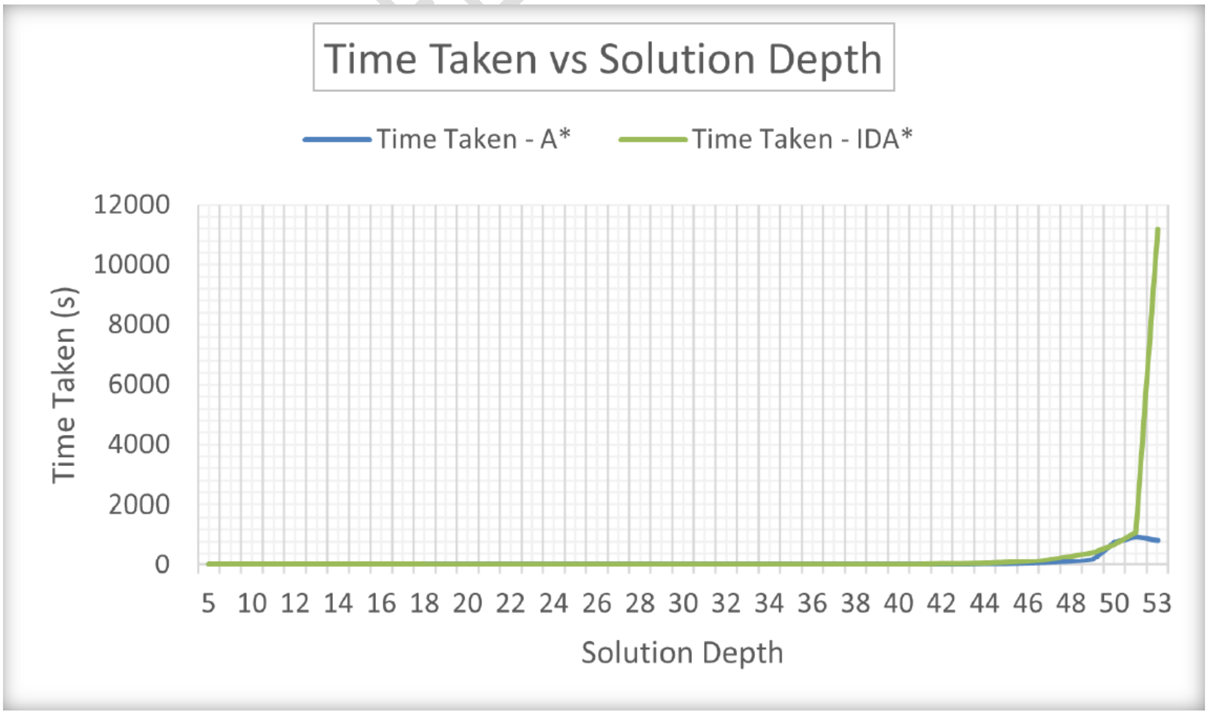

Fig 7: Average CPU time

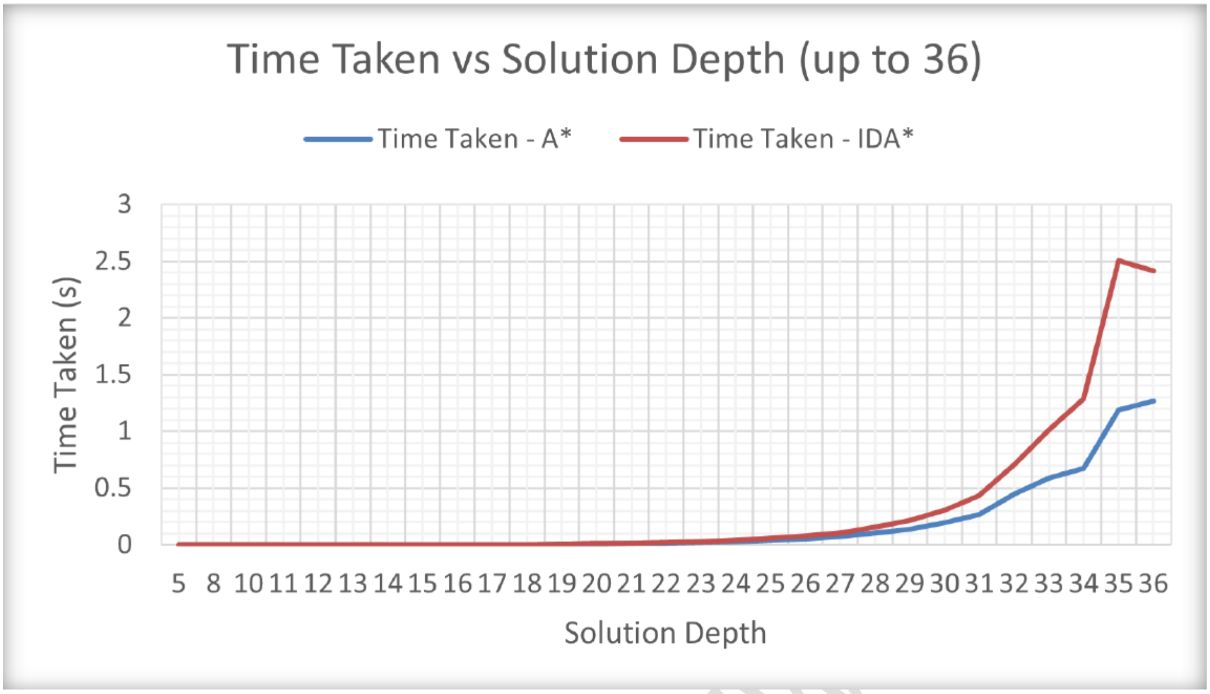

Fig 8: Average CPU time up to solution depth 36

Solution Depth

Fig 9 shows the distribution of solution depths amongall generated instances. The results indicate that most puzzleconfigurations required moderate depths to solve, with an average solution depth of 36 moves. This reinforces the 11-puzzle as a balanced test domain for evaluating algorithmperformance. The consistent optimal depth achieved by bothalgorithms also validates the effectiveness of the Manhattandistance heuristic in guiding both A* and IDA* toward optimalsolutions.

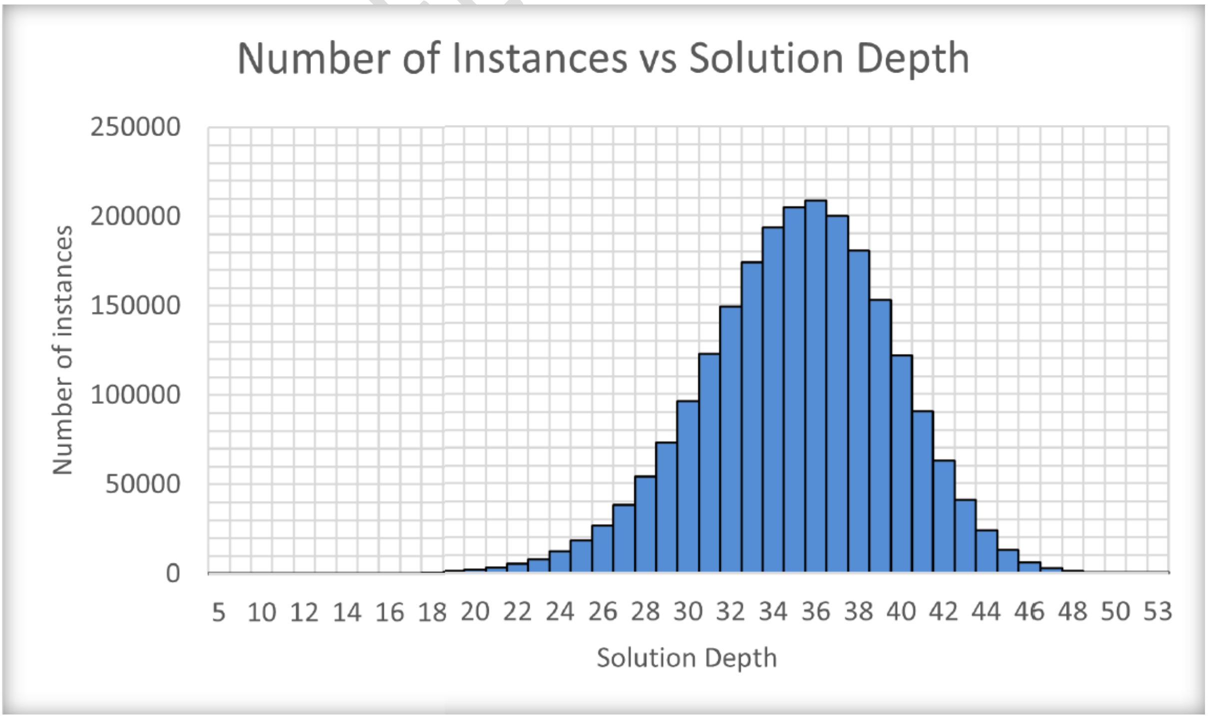

Fig 9: Number of instances of each solution depth

Summary of Performance Metrics

A detailed comparison of performance reductions for IDA*versus A* using the Manhattan distance heuristic is presentedin Table I. The table highlights A*'s significant efficiency interms of nodes generated, nodes expanded, effective branchingfactor, and CPU time.

Table 1: Comparison of percentage reduction in metrics ofA* compared to IDA*

Percentage Reduction of Search Algorithms	
Metric	**Percentage Reduction (A* vs IDA*)**
Nodes Generated	62.86%
Nodes Expanded	61.60%
Effective Branching Factor (EBF)	5.51%
CPU Time	51.46%

Discussion

The comparative analysis clearly demonstrates that A*outperforms IDA* across all major metrics when solving the11-puzzle with the Manhattan distance heuristic. The use ofan evaluation function $f(n) = g(n) + h(n)$ allows A* toexplore fewer paths and converge on the goal more efficiently,both in terms of processing time and search space traversal. Incontrast, IDA* while memory-efficient, suffers from repeatednode expansion and slower convergence due to its lack ofmemory and repeated iterations. These findings are consistentwith previous studies conducted on the 8-puzzle and 15-puzzle domains. The performance trends observed in this studysuggest that results from standard puzzle sizes generalize wellto mid-complexity domains such as the 11-puzzle. Therefore,the 11-puzzle can serve as a robust benchmark for evaluatingheuristic search algorithms in future research.

Conclusion

This study performed a comprehensive performance comparisonof two popular heuristic search algorithms, A* andIterative Deepening A* (IDA*) on the 11-puzzle problemusing Manhattan distance heuristic. The primary objective wasto analyse and compare the performance of these algorithms interms of their computational efficiency, scalability, and searcheffectiveness in a mid-complexity puzzle environment. Byutilizing a custom-built Python framework, solvable 11-puzzleinstances were generated and tested under identical conditions,ensuring fairness and reproducibility in the evaluation process.

The results clearly demonstrate that the A* algorithm significantlyoutperforms IDA* across all major performancemetrics. A* consistently generated and expanded fewer nodes,maintained a lower effective branching factor, and completedsearches in considerably less CPU timewith A* reducing the node generation by 62.86%, node expansion by 61.60%,EBF by 5.51%, and CPU time by 51.46%. This superior performancecan be attributed to A*'s informed search strategy,which leverages the Manhattan distance heuristic to focusexploration on the most promising paths, thereby avoidingredundant computations. In contrast, IDA*'s memory-efficientstructure comes at the cost of increased computational overheaddue to repeated node re-expansions across multipleiterations.Despite these limitations, IDA* remains a valuable algorithmin memory-constrained environments where spacecomplexity is a primary concern. Its ability to solve problemswithout maintaining large open and closed lists makes itsuitable for embedded systems or low-memory applications,even if it sacrifices execution speed.

More broadly, this research confirms earlier work validatingthe use of Manhattan distance heuristic in tile-based puzzlesolving. It also validates that the performance trends observedin smaller-scale problems like the 8-puzzle hold true for mid-scale puzzles such as the 11-puzzle. The11-puzzle thus proves to be a meaningful benchmark forevaluating heuristic search behaviour in more complex statespaces.

The findings are of practical value concerning time versusmemory trade-offs for heuristic search and present the basisfor future research. Future studies may explore testing withother heuristics such as Linear Conflict or pattern databaseheuristics, the construction of hybrid algorithms which benefitfrom the strengths of A* and IDA*, or the extension of thesemethods to real-time systems and constrained environments.

In conclusion, the A* algorithm, when paired with theManhattan distance heuristic, remains a robust and scalablesolution for solving pathfinding problems in AI. Its balanceof efficiency and optimality makes it a preferred choice inapplications where speed and accuracy are critical.

References

[1] S. D. T. Jananji and D. D. A. Gamini, “Measuring heuristic accuracyon the performance of search algorithms in solving 8-puzzle problems,”Current Scientia, vol. 27, no. 1, pp. 9–17, 2024.

[2] A. E. Iordan, “A comparative study of the a* heuristic search algorithmused to solve efficiently a puzzle game,” in IOP Conference Series:Materials Science and Engineering, vol. 294. IOP Publishing, 2018,p. 012049.

[3] C. T. Setyobudhi, “Comparison of a* algorithm and greedy best searchin searching fifteen puzzle solution,” International Journal of Computerand Information Technology, vol. 11, no. 3, 2022.

[4] D. O. Hasan et al., “The fifteen puzzle: A new approach throughhybridizing three heuristic methods,” arXiv preprint arXiv:2301.12345,2023.

[5] H. Dinh and H. Dinh, “Inconsistency and accuracy of heuristics with a*search,” University of Massachusetts Amherst, Tech. Rep., 2013.

[6] W. Hidayat, F. Susanthi, and D. R. Wijaya, “Comparative study ofinformed and uninformed search algorithms to solve eight puzzleproblem,” Journal of Computer Science, vol. 17, no. 2, pp. 145–151,2021.

[7] R. Jain and M. Patel, “Investigating the impact of different searchstrategies on 8-puzzle problem solving – a case study,” InternationalJournal of Advanced Research in Computer Science, vol. 14, no. 1, pp.112–117, 2023.

[8] D. Nayak, “Analysis and implementation of admissible heuristics in 8-puzzle,” 2014.

[9] A. Felner, R. Korf, and S. Hanan, “Additive pattern database heuristics,”Journal of Artificial Intelligence Research, vol. 22, pp. 279–318, 2004.

[10] D. Tolpinet al., “Rational deployment of multiple heuristics in ida*,” inProceedings of the Sixth Annual Symposium on Combinatorial Search(SoCS), 2014.

[11] Z. Bu and R. E. Korf, “A*+bfhs: A hybrid heuristic search algorithm,”arXiv preprint arXiv:2103.12701, 2021.

[12] Z. Bu and R. E. Korf, “A*+ida*: A simple hybrid search algorithm,”in Proceedings of the 28th International Joint Conference on ArtificialIntelligence (IJCAI-19), Macao, China, 2019, pp. 1180–1186.

[13] U. Zahaviet al., “Predicting the performance of ida* using conditionaldistributions,” arXiv preprint arXiv:1401.3493, 2014.

[14] R. K. P. Clausecker and F. Schintke, “A measure of quality for ida*heuristics,” Zuse Institute Berlin, Tech. Rep., 2021.