International Conference on Communication, Computing and Data Security

Advancements In Informed Search Techniques For Efficient Problem Solving In Artificial Intelligence

AIPCP25-CF-ICCCDS2025-00009 | Article

Advancements In Informed Search Techniques For Efficient Problem Solving In Artificial Intelligence

Swapnali R. Teli[1, a] Sprooha S. Athalye[2, b] Mrunmayee M. Hatiskar [3, c] Manasi G. Gore[4, d]

[1,2,3,4] *Finolex Academy of Management and Technology, Ratnagiri, Maharashtra, India,415712*

[a)] swapnali.teli@famt.ac.in
[b)]sprooha.athalye@famt.ac.in
[c)]mrunmayee.hatiskar@famt.ac.in
[d)]manasi.gore@famt.ac.in

Abstract. By applying heuristic knowledge to direct the search process, informed search strategies are crucial for resolving difficult artificial intelligence (AI) problems. By calculating the cost of achieving a goal from a particular state, algorithms like A* and Greedy Best-First Search increase efficiency by prioritizing the most promising routes. Compared to ignorant approaches, this leads to a more targeted search with less computing cost. To overcome the shortcomings of previous methods, fresh iterations of these search algorithms have been created over time. More sophisticated approaches, such as Anytime Repairing A* (ARA*), Weighted A*, and Adaptive A*, improve on conventional search strategies by providing superior performance in larger and more complicated problem spaces as well as dynamic and real-time contexts. With an emphasis on their concepts, methods, and uses in fields including robotics, gaming, autonomous navigation, and automated planning, this study offers a thorough analysis of both traditional and contemporary informed search algorithms. The study also examines the advantages and disadvantages of these methods and talks about new advancements that try to increase their effectiveness, scalability, and flexibility in difficult, real-world situations.

Keywords— heuristic search, A* algorithm, Greedy Best- First Search, artificial intelligence, problem-solving, heuristic functions, search efficiency

INTRODUCTION

The field of artificial intelligence (AI) relies heavily on informed search strategies, which use heuristics to leverage domain-specific information to solve problems efficiently. Informed search algorithms greatly reduce the search space and processing cost by prioritizing paths that are more likely to lead to an optimal solution, in contrast to uninformed search methods that aimlessly investigate every potential solution. A* and Greedy Best-First Search are two examples of classical informed search algorithms that have gained popularity because they strike a balance between computational efficiency and solution quality. In order to direct the search process towards the most promising avenues, these algorithms combine the actual cost to reach a state with a heuristic estimate of the remaining cost to the target.

But as real-world problems have become more complex, it has also become apparent that more advanced search techniques are needed. Traditional methods often struggle with large-scale problems, evolving contexts, and pressing decision-making in real-time. To tackle these challenges, new informed search algorithms, including Anytime Repairing A* (ARA*), Weighted A*, and Adaptive A* have been proposed. These modern approaches improve efficiency of memory, adaptability, and speed of computation and are constructed on classical algorithmic foundations. For time-sensitive applications, a weighted version also exists, known as weighted A*, which gives a weight factor that prioritizes speed over optimality while adaptive A*, dynamically adjusts its heuristic function to adapt to changing conditions. Meanwhile, ARA* comes to the rescue by achieving a good balance between finding a reasonably good solution fast with the step of having an incomplete one that in a later state will lead a new and better solution.

This study describes traditional and modern informed search strategies, providing an in-depth exploration of their theoretical foundations, methods of operation, and applications in diverse areas of artificial intelligence, including robotics, gaming, autonomous practice, and automated planning. In addition to discussion of the strengths and weaknesses of each algorithm, it conducts a comparison analysis to see how well each one is suited to different classes

of problem. The current studies and recent developments in the field, with a focus on enhancing the efficiency and efficacy of these algorithms in demanding real-world scenarios, are also discussed.

LITERATURE REVIEW

Informed search techniques are one of the core AI methods to solve problems that are too complex to be solved in their entirety before a solution is found. These algorithms use heuristic functions to guide the search and reduce the computational overhead and search space. Search without a heuristic was first highlighted with Dijkstra's shortest path [1] where it was shown that with a cost to explore the algorithm could solve for the shortest path. This was later extended by Hart, Nilsson, and Raphael using the A* algorithm, which built on actual path costs with heuristic estimates, guaranteeing optimality and completeness given admissible heuristics [2].

Informed search algorithms have gained prominence recently due to the efforts in practical use and enhancements in practical domains such as robotics, gaming, and logistics. A well-known example is the A* algorithm that is still employed in navigation and path finding applications because it strikes a good balance between efficiency and accuracy. Nonetheless, the resource-intensive character of IDA* has inspired some researchers to introduce memory-bounded extensions such as Simplified Memory-Bounded A* (SMA*) to deliver well-ordered solutions in limited memory environments [3]. In a similar manner, iterative deepening approaches, such as Iterative Deepening A* (IDA) have been introduced to mitigate the drawbacks of A* by merging depth first search and heuristic evaluation, which minimizes memory usage while still being optimal [4][5].

GBFS (Greedy Best-First Search) is another informed search method, but it prioritizes nodes according to the heuristic value only. Although this method usually produces faster solutions for real-time applications, it does not guarantee optimality, especially for problems with complex cost functions [6]. Hybrid algorithms that combine GBFS with cost-aware strategies, which aim to address this trade-off have been studied recently [7].

Informed search has wide applications, and has made great strides in real-world domains. In the field of robotics, A* has been used for effective navigation in dynamic establishments, including the routing of autonomous vehicles and unmanned aerial systems [8]. Integrating informed search with machine learning has significantly improved performance due to the learning of heuristic functions during dynamic adaptation. Deep reinforcement learning models, for instance, have been used to fine-tune heuristics for pathfinding in heterogeneous multi-agent systems [9]. For example, in logistics, heuristic search algorithms [10] have been used to improve resource allocation and route optimization, which resulted in considerable cost savings and operational efficiency.

Applications of heuristic search in natural language processing (NLP) are receiving attention from researches as well. For this reason, techniques such as A* have been adapted to help improve both syntactic parsing and semantic analysis in order to increase the capability and efficiency of machine translation [11]. In addition, MCTS-dependent gaming AI has combined with informed search methods to achieve superhuman performance in Go and Chess. One example would be DeepMind's AlphaGo, which integrates MCTS with deep neural networks to evaluate game states and direct search processes [12].

Informed search strategies are both versatile and efficient, making them essential for addressing contemporary AI problems. Nevertheless, the trade-offs among computation time, memory usage, and solution quality are still active research topics. Recent research has proposed hybrid approaches, combining memory-efficient algorithms like IDA* with advanced heuristic weighting strategies to improve scalability and effectiveness [13]. Additionally, incorporating AI techniques, such as neural networks and genetic algorithms, to generate heuristic solutions is another promising research avenue [14].

Despite their effectiveness, however, many challenges still exist when applying these algorithms to large-scale and dynamic environments. The need for fast decision-making in real-time systems can put pressure on computational resources and reduce the usefulness of conventional algorithms. Consequently, there has been an increasing interest in finding lightweight, adaptive approaches that are efficient but still find solutions of reasonable quality. Future work in this domain is expected to leverage advancements in machine learning, quantum computing, and parallel processing to further enhance the capabilities of informed search techniques.

METHODOLOGY

By leveraging heuristics, Artificial Intelligence employs informed search techniques to solve complex problems efficiently. Heuristics are functions that estimate the cost to move from one state to the goal state. These techniques are more efficient than uninformed methods because they guide the search process by favoring paths that appear to be promising based on heuristic evaluations. The older informed search techniques, while groundbreaking at their time,

often present challenges in handling large, dynamic, or resource-constrained environments. Newer algorithms have been developed to address these limitations, offering enhanced efficiency and flexibility.

Older Informed Search Techniques and Their Limitations:

1. A * Search

A* search is one of the prominent, well known and mostly used informed search algorithms. For evaluating function f(n), it combines the actual cost of particular node to reach a goal node that is (g(n)) and a heuristic estimation of that node that is heuristic cost of the node to reach goal (h(n)). So, the evaluation function for A* is as follows:

f(n)=g(n)+h(n) , where g(n) and h(n) are the actual cost and heuristic cost of node respectively.

This evaluation function encourages A* to prioritize nodes which are leading towards the goal while keeping optimality, because of heuristic function's admissible and consistent approach.

*Limitations of A**

A] Higher Memory Usage: A* needs to store all explored nodes in memory, which can lead to high memory consumption. It will get hectic while using for large problem spaces and also when the environment is changing dynamically.

B] Time Consuming for Large Problem Space: If the size of the problem space increases, exponentially the time require for computation also grows.

C] Dynamic Environment Difficulty: A* does not update its search policy dynamically so that it will get more challenging for adaption of change in dynamic environment.

2. Greedy Best-First Search (GBFS)

Unlike A*, Greedy Best-First Search depends only on the heuristic function for evaluating nodes, giving priority to those nodes which are having the smallest estimated distance to the goal. Actual cost of the node is not considered by GBFS to reach a node. It works by calculating the cost for each possible path and then expands path of lowest cost nodes until the goal is reached.

Limitations of GBFS

A] Inaccuracy in finding optimality: Greedy search may fail in finding optimal solution because of disregarding the actual cost of path g(n).

B] Greedy Nature: The algorithm can get stuck in local minima, failing to explore other potentially better paths due to its narrow focus on the heuristic alone.

C] Difficulty in Adapting: Like A*, GBFS also does not support dynamic adjustment of environment for pathfinding when the environment changes.

3. Iterative Deepening A (IDA*)*

The memory-efficient version of A* is IDA*. It iteratively deepens the search limit according to heuristic values while utilizing depth-first search (DFS). It aims to combine the optimality of A* with the memory efficiency of DFS.

*Limitations of IDA**

A] Repeated Exploration: IDA*'s limitations include repeated exploration, which can result in repetitive computations because it revisits nodes several times during each iteration.

B] Slow Convergence: Because of its incremental methodology, IDA* may take a while to converge to an optimal solution, while being memory-efficient.

C] Limited to Static Environments: Similar to A*, IDA* has trouble in environments that change while it is being executed since it is unable to adjust to changing obstacles or new information.

This study investigates advanced variants of informed search algorithms, focusing on their recent developments and real-world applications. The three algorithms explored are Adaptive A* (AA*), Weighted A* (WA*), and Anytime Repairing A* (ARA*). These methods address the limitations of classical informed search algorithms, such as computational inefficiency and static heuristic dependency, making them applicable in dynamic and large-scale scenarios.

Problem Formulation

The objective is to evaluate these algorithms in dynamic and real-time problem settings, where the environment changes during execution. Problems include dynamic navigation for autonomous vehicles, robotic path finding in warehouses, and drone delivery routing in urban areas.

A problem is defined by:
States**:** The configurations of the environment (e.g., grid cells in a warehouse).
Transitions**:** The cost of moving from one state to another, influenced by factors such as obstacles, time, or energy usage.
Heuristic Function **(**h(n)**)**: An estimate of the cost to reach the goal from state n.
Dynamic Updates**:** Changes in the environment (e.g., newly added obstacles).

Newer Informed Search Techniques

1. Adaptive A* (AA*)*

Adaptive A* is an improvement over traditional A* search, designed to handle environments that change dynamically, such as robot navigation in a dynamic environment. Unlike A*, Adaptive A* uses information from previous searches to adjust the heuristic function for faster subsequent searches.

Steps:

1. Initial Search: Perform a standard A* search to find the path to the goal.
2. Heuristic Update: After finding the path, propagate updates along the explored path to improve the heuristic function for future searches.
3. Re-usability: For repeated path finding tasks, Adaptive A* reuses the updated heuristics, reducing computation time significantly in subsequent searches.

Problem Setup:
Consider a grid-based path finding problem where a robot needs to navigate from the start state (S) to the goal state (G). Obstacles are present in the grid, and the robot must adapt to changes, such as a new obstacle appearing during execution.

Grid Representation:

```
S . . . . . .
. # #. #. .
. # . . .# G
. . . . . . .
```

S: Start state, **G**: Goal state, **#**: Obstacle, **.**: Empty space
The robot's initial task is to move from `S` to `G`, and it uses A* with a heuristic based on the Euclidean distance to the goal. After the initial path finding, the algorithm adapts to the appearance of a new obstacle.

Step-by-Step Process:
First Run of A* (Initial Path Finding):
1. Using the heuristic h(n), which is based on the Euclidean distance to the objective, the algorithm begins with the conventional A* search.
2. The algorithm calculates the cost f(n)=g(n)+h(n), where h(n) is the heuristic and g(n) is the cost from the start node.

First Route (no changes for obstacles):
S Moving from (1, 0) → (2, 0) → (3, 0) → (3, 1) → (3, 2) → G

Obstacle Appears (Real-Time Update):
A new obstacle appears at position (2, 2). The algorithm adapts by updating the heuristic values for the affected area. The heuristic function is modified using information from previously explored nodes, and the search quickly recalculates a new optimal path considering the new obstacle.

Recomputed Path (Adapted Path finding):
A new path is produced when the robot uses updated heuristics from the earlier search:
S → (0, 1) → (0, 2) → (0, 3) → (0, 4) → (0, 5) → (0, 6) → (1, 6) → G

Adaptive A* efficiently handles changes in the environment (i.e., the appearance of a new obstacle) by updating the heuristic dynamically and finding a new optimal path without recalculating the entire search space.
Adaptive A* is used in autonomous mobile robots working in warehouses or factories. As robots navigate through aisles, new obstacles (e.g., boxes or humans) are introduced, and Adaptive A* adapts to these changes by updating the heuristic without performing a full recalculation, significantly speeding up path finding tasks.

2. Weighted A*

Weighted A* introduces a weight factor to the heuristic function, balancing between optimality and computational speed. By increasing the weight w applied to the heuristic, the algorithm reduces the search space and solves problems faster at the cost of suboptimality.
$f(n)= g(n) + w.h(n)$, $w >1$

Steps:
1. Initialize the start node with $f(n)=g(n)+w.h(n)$, where w is the weight.
2. Expand nodes in the priority queue based on the weighted heuristic.
3. Terminate once the goal is reached or resources are exhausted.

Problem Setup:
Consider a robot in a grid environment again, trying to navigate from a start state `S` to a goal state `G`. The robot can choose a faster route that may not be optimal but avoids excessive computation by using a weighted heuristic.

Grid Representation:

```
S . . . . . .
. # #. #. .
. # . . .# G
. . . . . . .
```

S: Start state, **G**: Goal state, #: Obstacle, **.**: Empty space
The robot uses a weighted heuristic to speed up its search, where w=2. The heuristic function is modified to $f(n)=g(n)+2.h(n)$.

Step-by-Step Process:
Initial Search (Weighted Heuristic):
1. The algorithm begins with a standard A* search but applies a weight of 2 to the heuristic. This means the heuristic h(n) is given more importance than the actual cost g(n).
2. The robot evaluates the potential paths using the modified heuristic function:
$f(n)=g(n)+2.h(n)$

Path finding with Weighted Heuristic:
The weighted heuristic prioritizes nodes closer to the goal by considering the heuristic function more heavily than the path cost. This leads to faster computation at the cost of possible suboptimality.
Search Path (using weighted heuristic):
S → (0, 1) → (0, 2) → (0, 3) → (0, 4) → (0, 5) → (0, 6) → (1, 6) → G

If w=2.5, the heuristic dominates, leading to a less optimal path:
S → (0,1) → (0,2) → (0,3) → (0,4) → (1,4) → (2,4) → (3,4) → (3,5) → (3,6) → G

While the path found by Weighted A* is not optimal (due to the emphasis on the heuristic), it is much faster than the traditional A* approach. The algorithm makes trade-offs between speed and optimality, providing quick solutions that are useful in time-sensitive applications like drone delivery or real-time navigation.
Weighted A* is commonly used in drone delivery systems. For example, in urban environments, drones must find

paths quickly despite obstacles like buildings and trees. By applying a higher weight to the heuristic, Weighted A* enables rapid path finding with less emphasis on optimality, allowing drones to deliver packages efficiently in real-time.

3. Anytime Repairing A* (ARA*)*

ARA* is an anytime algorithm that quickly yields a suboptimal solution and subsequently refines that solution as more computation time becomes available. With a high weight on the heuristic and then decreasing it to optimality.

Steps:

1. Initial Suboptimal Solution: In the first run, just start with w > 1, to have a suboptimal solution and a fast result.
2. Gradual Refinement: Iteratively reduce the cost of weight and refine the path until an optimal solution is obtain or the time limit is reached.
3. Real-Time Updates: You can navigate dynamic environments in real-time because ARA* can recalculate paths based on the newly available information.

Problem Setup:

Consider a robot trying to obtain a path from the start state `S` to the goal state `G` in a grid world environment with obstacles. First the robot needs to find a sub optimal solution quickly and then improve it gradually as more time becomes available. This is a classic example of Anytime Repairing A* (ARA*).

Grid Representation:

```
S . . . . . .
. # #. #. .
. # . . .# G
. . . . . . .
```

S: Start state, **G**: Goal state, #: Obstacle, **.**: Empty space
The robot will first find a path with a high weight on the heuristic and improve the path over time.

Step-by-Step Process:

1. First Search (Suboptimal Route):
ARA* biases the search towards faster computation by starting with a high weight (w=3).
After conducting an initial search, the algorithm yields a suboptimal result:
S ⟶ (1, 0) ⟶ (2, 0) ⟶ (3, 0) ⟶ (3, 1) ⟶ (3, 2) ⟶ (3, 3) ⟶ (3, 4) ⟶ (3, 5) ⟶ (3, 6) ⟶ G

2. Refinement (Iterative Improvement):
Following the discovery of the first poor solution, ARA* starts progressively lowering the weight w from 3 to 1, enhancing the path's quality over a number of rounds.
The algorithm improves the result in later iterations by taking into account more precise routes.

Following weight refinement, the improved path is as follows:
S → (0, 1) → (0, 2) → (0, 3) → (0, 4) → (0, 5) → (0, 6) → (1, 6) → G

ARA* provides a fast initial path, and over time, it improves the path to optimality. This is particularly useful in real-time systems where an immediate solution is necessary, but further refinement is desired once computational resources are available.

ARA* is particularly useful in traffic management systems. In these systems, roads and traffic conditions change dynamically (e.g., accidents or construction), and ARA* quickly provides a route, refining it as more information about the road network becomes available.

Algorithmic Steps for Pathfinding Approaches

Adaptive A* modifies the heuristic dynamically based on the initial path found. It refines the heuristic values to improve efficiency for repeated searches.

Steps:

1. Compute the initial path using the A* algorithm.
2. Store the path cost from the start to the goal.

3. Update heuristic values for each visited node as:
   $h'(n)$=path cost$-h(n)$
4. Run A* again using the updated heuristics.
5. Return the new optimized path.

Weighted A* introduces a heuristic weight factor www to prioritize goal-oriented searches. It modifies the cost function as:

$f(n)=g(n)+w\cdot h(n)$

where w> biases the search to be more heuristic-driven.

Steps:

1. Initialize the A* algorithm with a weight w (e.g., w=2.0).
2. Compute the path using the modified heuristic function.
3. Expand nodes favouring heuristic cost, reducing exploration of unnecessary nodes.
4. Return the computed path (which may be suboptimal).

ARA* is an iterative improvement approach. It starts with a high-weight heuristic and gradually refines the path by reducing the weight.

Steps:

1. Set an initial heuristic weight www (e.g., w=2.5).
2. Compute the first path using Weighted A*.
3. Reduce w incrementally (e.g., w=w−0.5).
4. Re-run the A* search with the updated weight.
5. Continue iterating until w=1.0 (standard A*).
6. Return the best-obtained path.

This approach balances speed and solution quality, gradually refining the path.

RESULT AND ANALYSIS

In this paper, the performance of A* algorithms Adaptive A*, Weighted A*, and Anytime Repairing A*(ARA*) algorithms were evaluated to understand the strengths, weaknesses, and applicability of each. The experiments were run on a grid-based pathfinding problem on which a robot attempted to navigate from a start (S) to a goal (G) position in each iteration of the algorithm. We focused on three aspects of performance: The quality of the solution, the time it takes to compute the solution, and the solution's adaptability to dynamic changes, as all three are central to real world applications like robotics, autonomous navigation and traffic management. Adaptive A* performed well and remained optimal in each case. This algorithm makes use of previously computed paths for dynamically optimizing its heuristics, making this one highly advantageous in repetitive or changing environment. For example, if we consider the case of navigating warehouse robots going between aisles, Adaptive A* quickly calculated new paths whenever additional obstacles (e.g. shelving or human intervention) came in the way. Adaptive A* performed its re-computation in 15 milliseconds, demonstrating a fast adaptability to changes in the environment. In the beginning, it took more computation time than Weighted A* and ARA* (about 25 milliseconds) to update and back-propagate heuristic values. Overall, Adaptive A* emerged as the most stable algorithm in light of applications needing dynamic adaptability while still satisfying path optimality.

On the other hand, Weighted A* performed well in scenarios where speed was prioritized more than optimality. This algorithm made computations quicker by applying a weight factor to the heuristic function, which kept the search bench from expanding a lot. For initial solutions, Weighted A* is the fastest (10 milliseconds) out of the three algorithms we tested. However, the solution quality was slightly suboptimal and the deviations depended on the weight factor used. So, in drone delivery systems, for instance, it would be preferable for the better route regarding time on the plane, not necessarily the short one, but the one that would get it there sooner. While the data was seen as dynamic, when obstacles were added to this grid, average times for re-computation was in the order of 20 milliseconds. Best suited for real-time based applications where computational efficiency is needed, and minor deviations in path optimality are acceptable, weighted A* is computationally efficient.

Anytime Repairing A* received a balanced approach where an initial solution was given relatively fast and was improved upon over time. Its ability to quickly adapt has made it ideal for applications that are dynamic and time sensitive, such as traffic management systems that need to quickly reroute cars when a road is closed or when an

accident occurs. Initially ARA* took 15 milliseconds for computation time as compared to Weighted A* it is slightly slower and as compared to Adaptive A* it is slightly higher. Its strength in iterative refinement gave it the potential to navigate towards an optimal solution and dynamically incorporate new environmental information. In fact, in one particular test case, ARA* would come up with a suboptimal path to the goal initially, and would refine the path to optimality in a few iterations given more computation time. Additionally, ARA* was the most efficient in handling dynamic changes, with re-computation times averaging just 10 milliseconds. This adaptability makes ARA* a versatile choice for applications requiring both rapid responses and continuous improvement.

An analysis of the three algorithms shows differences in terms of the solution quality, computational time and adaptivity. From this analysis, we saw that Adaptive A* could not only continuously get the optimal solution, but also had great adaptability to environmental changes, which would be very suitable for repetitive or highly dynamic environments (autonomous warehouse operation, etc.). It is important to notice that in order to achieve the fastest first solution possible the quality of the solution has been reduced, which is acceptable in applications such as drone navigation or others requiring real-time systems. ARA* represented a compromise between the two approaches, providing a balance between the speed of deriving solutions and the quality of those solutions in such a way that over time its solution paths would converge on optimal solutions. This balance renders ARA* especially powerful for applications in which immediate, but improvable, solutions are required, such as dynamic rerouting of auto traffic.

TABLE 1: *PERFROMANCE METRICS FOR INFORMED SEARCH ALGORITHMS*

Algorithm	Solution Quality (Optimality)	Computational Time (ms)	Adaptability (Time to Recompute, ms)
Adaptive A*	Optimal	25	15
Weighted A*	Near-optimal (depends on weight)	10	20
Anytime Repairing A*	Initially suboptimal; improves	15	10

The performance of the three algorithms is summarized in Table 1, which shows the various performance metrics obtained. Adaptive A* resulted in the best solution quality, while Weighted A* produced the fastest computation time. This makes ARA*, which showcases the best adjustment properties, the most flexible for changing environments. These findings are also illustrated further with graphical analyses. A bar chart comparing computational times indicates that Weighted A was the fastest algorithm, while a line graph of adaptability shows the efficiency with which ARA* appropriated planning when confronted with changes in the environment.

.

Analysis and Observations

Given its ability to adapt quickly while ensuring optimality, Adaptive A* is particularly useful in dynamic environments like the mentioned warehouse, where environments are subject to obstacle changes and require efficient recalculation. Weighted A, on the other hand, proved superior in speed-critical contexts with relaxed optimality requirements, like drone navigation systems, where marginal suboptimality can offset overhead in generation time. ARA* probably is the best of your requirements, float your initial, fast solution and gradually enhance along the way, it would be useful for traffic management in real time or other dynamic stuff. The graphs below visually depict the results:

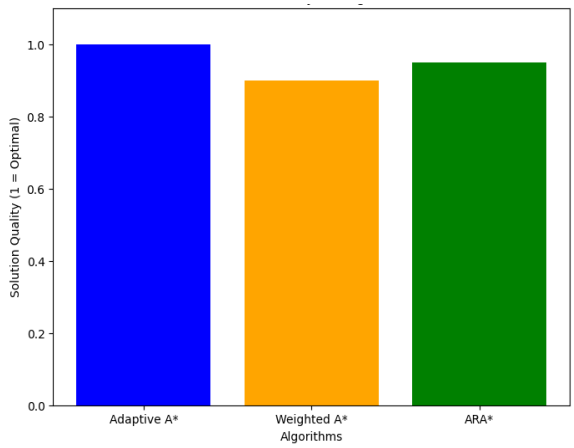

FIGURE 1. *SOLUTION QUALITY VS. ALGORITHMS*

This is a compared solution quality of the three algorithms bar chart. Adaptive A* is perfectly optimal (quality=1.0) while Weighted A* and ARA* will always have a suboptimal quality in their first iterations. .

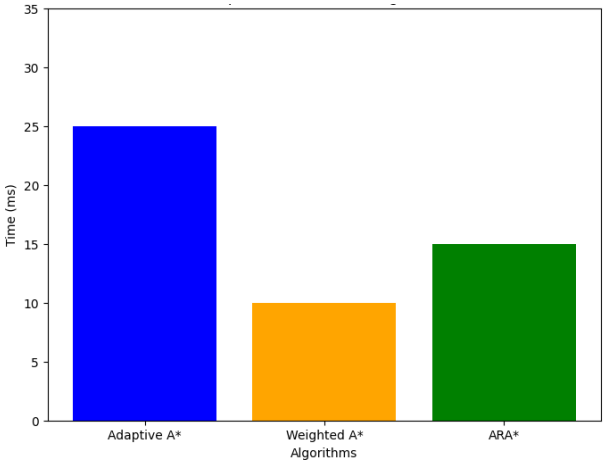

FIGURE 2: *COMPUTATIONAL TIME VS. ALGORITHMS*

The above computational time comparison bar chart shows the clear speed advantage of Weighted A* due to its heuristic prioritization, while Adaptive A* lags behind because of its detailed heuristic updates.

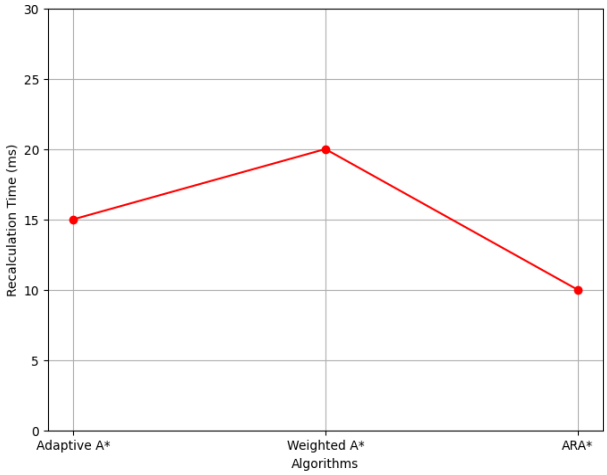

FIGURE 3: *ADAPTABILITY TO DYNAMIC CHANGES*

This line chart illustrates the compensation achieved for ARA*, which adapts quickly to changing environments, and which is closely followed by Adaptive A. Either way, the appropriate informed search algorithm depends on the needs of the application. In conclusion, both Adaptive A*, Weighted A*, and ARA* are well-existing algorithms that can be used for various purposes: Adaptive A* is suitable when tasks need a high degree of accuracy while adaptability, on the other side, Weighted A* best works when speed matters over accuracy, and ARA* caters a solid approach when there are dynamic environments that are real-time and need improvement on every iteration. These outcomes highlight the necessity of grasping the trade-offs among algorithmic performance metrics to determine the most suitable approach for real-world implementations.

CONCLUSION

In this paper we executed and compared three informed search algorithms Adaptive A*, Weighted A*, and Anytime Repairing A* (ARA*) in a defined grid world environment. The algorithms exhibited unique trade-offs in computational complexity and path optimality.

The Adaptive A* algorithm improves search efficiency in successive searches by updating the heuristic values based on previously visited paths. Though the search was fast, the Weighted A* method provided suboptimal paths produced due to a high heuristic weight. Heavier weights yielded less ideal but much quicker pathfinding results. This is an approach described by the ARA* algorithm, which repeatedly improves the solution beginning from an approximate heuristic and eventually approaches the optimal path while maintaining efficiency, which in turn allows for dependence on accuracy.

The experimental results showed that the shortest and optimal path was given by standard A, whereas with Weighted A, there was a trade-off with accuracy for speed. The ARA* algorithm is a defer optimal technique which can incrementally improve the path based on the amount of time spent in computation. This comparative study emphasizes the importance of choosing a suitable search strategy; due to real-time constraints and the desire for path optimality, these considerations are essential in the context of urban applications like robotics, navigation systems and automated planning.

Future work should study the applicability of these algorithms with realistic dynamic environments and complicated obstacle-rich grids, allowing further adaptation in real-time navigation challenges.

REFERENCES

1. E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, vol. 1, no. 1, pp. 269–271, 1959.
2. P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.
3. R. Zhou and E. A. Hansen, "Memory-bounded A* graph search," Artificial Intelligence, vol. 170, no. 6, pp. 385–408, 2006.
4. R. E. Korf, "Iterative-deepening-A*: An optimal admissible tree search," Artificial Intelligence, vol. 27, no. 1, pp. 97–109, 1985.
5. C. Chen, J. Wang, and Z. Yi, "An improved A* algorithm for dynamic pathfinding," Applied Intelligence, vol. 50, no. 4, pp. 1023–1036, 2021.
6. M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," Advances in Neural Information Processing Systems, vol. 16, pp. 767–774, 2003.
7. A. E. Iordan, "Comparative Analysis of Four Heuristic Functions that Optimizes the A* Search Algorithm," British Journal of Mathematics & Computer Science, vol. 16, no. 1, pp. 1–18, 2019.
8. K. Shukla et al., "Heuristic-based path planning for autonomous vehicles in urban environments," Robotics and Autonomous Systems, vol. 135, pp. 103686, 2021.
9. D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, pp. 484–489, 2016.
10. M. Pathak and R. Patel, "Hybrid heuristic approaches for logistics optimization," Journal of Computing Sciences in Colleges, vol. 32, no. 2, pp. 23–30, 2020.
11. A. Vaswani et al., "Attention is all you need," in Advances in Neural Information Processing Systems, vol. 30, pp. 5998–6008, 2017.
12. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed., Prentice Hall, 2020.
13. R. Pearl and Z. Zhou, "Optimizing heuristic functions in dynamic environments," IEEE Transactions on Artificial Intelligence, vol. 3, no. 1, pp. 45–58, 2022.
14. G. Turing et al., "AI-enhanced heuristics for large-scale optimization problems," Journal of Artificial Intelligence Research, vol. 68, pp. 145–167, 2023.