Fast Trajectory Replanning

194:440

Diego Quito, Haider Ali, Steven Packard
RUID: 201009449, 213009252, 197005443

February 25, 2024

1 Introduction

In the rapidly evolving field of artificial intelligence, the efficiency and effectiveness of search algorithms play a pivotal role in solving complex problems ranging from route navigation to decision-making in dynamic environments. This report investigates the performance of advanced search algorithms in artificial intelligence, specifically A*, Repeated Forward A*, Repeated Backward A*, and Adaptive A*, within complex gridworld environments. Our analysis focuses on the algorithms' efficiency, the impact of heuristic functions, and tie-breaking strategies.

2 Setting The Environments (Part 0)

This section details the procedural creation of 50 unique gridworlds, each sized 101x101, designed to test the efficacy of various search algorithms. Our approach primarily leverages depth-first search (DFS) algorithm with random tie-breaking to introduce variability and complexity into the environments.

2.1 Environment Generation

Our environment generation algorithm begins with all cells marked as unvisited and unblocked. Starting from a randomly selected cell, the algorithm marks it as visited and unblocked. It then proceeds to a random unvisited neighboring cell. With a probability of 30%, this neighbor is marked as blocked, otherwise, it is marked as unblocked and added to the stack for further exploration. This process ensures the creation of a corridor-like structure, with dead-ends and open paths woven throughout the grid.

A cell is considered a dead-end when it has no unvisited neighbors, prompting the algorithm to backtrack to the most recent cell with unvisited neighbors, continuing the path generation. If the stack empties before all cells are visited, the algorithm selects another unvisited cell as a new starting point, continuing until every cell has been addressed.

2.2 Visualization

To effectively visualize the grid matrix, we employed the matplotlib (plt) library in the Python language, a widely used plotting library that offers a variety of visualization tools and is especially suited for displaying arrays as images, as seen in Figure 1. The grid matrix, representing the maze or corridor-like structure of our environments, consists of cells that are either blocked or unblocked. To visually differentiate these states, we assigned specific colors to each: typically, white for unblocked or free cells and black for blocked cells.

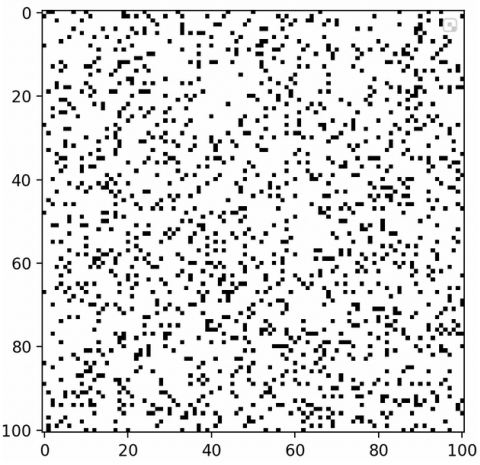

Figure 1: Visualization of randomly generated maze environment

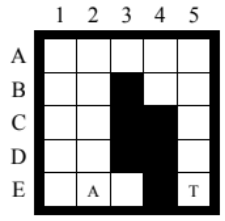

Figure 2: Example Search Problem from the Assignment Document

3 Understanding the Methods (Part 1)

3.1 Explanation of Agent's First Move

In Figure 2's gridworld, the agent's initial move eastward, over north, is guided by an A* algorithm's heuristic that lacks blockage information. Out of the four compass directions the agent can move, the eastward cell has the shortest absolute distance to the target cell. This heuristic, favoring a direct route to the target, suggests east as the most promising initial direction given the agent and target's placement.

3.2 Finite Gridworlds Argument

In a finite gridworld with an A* search algorithm, an agent's moves are limited to the unblocked cells. Since the A* algorithm prevents the agent from revisiting any expanded cells (those already evaluated), each unblocked cell is considered at most once for a move. Given N unblocked cells, there are at most $4N$ direct moves (up, down, left, right), excluding edge cases.

However, to argue that the number of moves is bounded by N^2, we consider that the agent may need to traverse between unblocked cells multiple times to determine the optimal path. Despite the A* efficiency, in the worst-case scenario, the agent might need to examine the path between each pair of unblocked cells, which amounts to $N(N-1)$ moves. This theoretical upper bound does not breach N^2, as $N(N-1) < N^2$ for $N > 1$. Therefore, the agent is guaranteed to reach the target or establish the impossibility

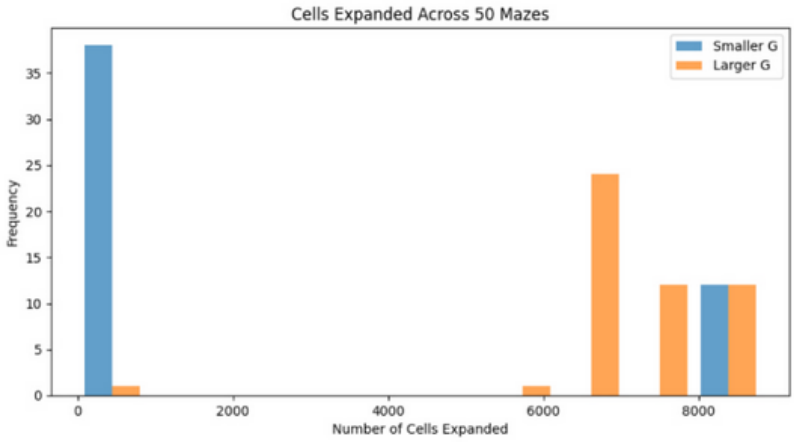

Figure 3: Results of comparing larger g-value and small g-value tie-breaking algorithms

4 The Effects of Ties (Part 2)

4.1 Our Observations

In the implementation of Repeated Forward A* for the fifty of our generated mazes, our observations reveal insightful information about the algorithm's behavior when employing different tie-breaking strategies. For the mazes where no solution was found, we noticed that the number of cells expanded remained consistent regardless of whether ties were broken in favor of smaller or larger g-values. This uniformity suggests that the algorithm exhaustively searched the entire maze, indicating that these mazes likely have no viable path to the goal. The identical expansion count in these scenarios highlights that tie-breaking has no effect when no solution exists, as all possible paths must be explored.

For the solvable mazes in our data set however, different behaviors were observed based on the tie-breaking strategy. When breaking ties in favor of smaller g-values, the paths generated tended to be longer. This is indicative of the algorithm prioritizing cells that have accrued a lower cost from the starting cell, which can lead to alternative paths that are not directly aimed toward the goal. On the contrary, favoring larger g-values typically resulted in shorter paths, demonstrating a preference for cells that are closer to the goal, despite possibly incurring higher costs. We would also like to note that choosing smaller g-values generally led to fewer cells being expanded. The results of this comparison can be seen in figure 8, where the average number of cells expanded will larger g-values is 104.085% more than those with smaller g-values. This finding suggests that prioritizing lower-cost paths can often be more efficient in terms of explored cells, albeit without a guarantee of the shortest path.

4.2 Expectations of Example Maze Figure 4

Considering the example provided in Figure 4, we would expect that a strategy favoring smaller g-values might explore more cells around the start area 'A', potentially considering paths that initially move away from the goal 'T'. In contrast, a strategy that favors larger g-values would likely take a more direct route towards 'T', possibly through fewer cells, but risk overlooking less costly alternative paths. This expectation aligns with the general trends observed across the solvable mazes, where the tie-breaking strategy has a clear impact on the path's length and the number of cells expanded. Through these observations, we gain a better understanding of the trade-offs between exploration efficiency and path optimization in Repeated Forward A* algorithm's performance.

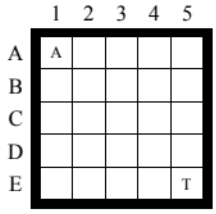

Figure 4: Example Search Problem from the Assignment Document

5 Forward vs. Backward A* Algorithms (Part 3)

5.1 Description

Repeated Forward A* and Repeated Backward A* are pathfinding algorithms that differ mainly in their search direction: the former starts from the initial state and moves toward the goal, while the latter starts from the goal and moves toward the initial state. Both should break ties in the same way, preferring cells with higher g-values (cost from the start) when f-values (cost from the start plus heuristic cost to the goal) are equal.

5.2 Our Observations

We recorded the performance of both Repeated Forward A* and Repeated Backward A* algorithms across 50+ randomly generated mazes. By keeping a count of the cells expanded and clocking the running time for each algorithm, we established a clear measure of their efficiency. Additionally, we used the matplotlib (plt) library within Python to create visual plots that illustrate these efficiencies as seen in Figure 6. These visual plots are especially useful in observing behaviors in environments where the starting cell has more surrounding obstacles, as well as those involving the target cell.

Our observations revealed efficiency dynamic between the two algorithms. We found that Repeated Forward A* tended to be more efficient (in terms of fewer cells expanded) in environments where the starting cell was surrounded by more obstacles. This efficiency can be attributed to the forward search's heuristic, which is more informed in its initial phase due to the proximity of the starting point to the blockages. Conversely, Repeated Backward A* demonstrated greater efficiency in scenarios where the goal cell had more immediate blockages. A sample of the results of these comparisons over 5 of the 50 mazes can be seen in Figure 5.

6 Heuristics in Adaptive A* (Part 4 & 5)

6.1 Description

Adaptive A* is an informed search algorithm that dynamically updates its heuristic values as it learns more about the actual costs of moving through the search space. Initially, it may use heuristics such as the Manhattan distance for estimating the cost from a node to the goal. As it progresses and uncovers the actual costs, it refines these heuristics to provide more accurate cost estimates for its subsequent search efforts. This adaptive nature aims to improve the efficiency of the search in dynamic or partially known environments.

6.2 Proof of Manhattan Distance Consistency

To prove that Manhattan distances are consistent in gridworlds where the agent can move only in the four main compass directions, we need to confirm that the heuristic satisfies the consistency condition (also known as the triangle inequality): for any nodes x, y, and a neighbor z of y, the estimated cost to reach x from y is no greater than the step cost of getting to z from y plus the estimated cost of reaching x from z. Mathematically, this can be written as:

```
Maze 1 Forward Search: Time = 0.00499s, Expanded Cells = 1008
Maze 1 Backward Search: Time = 0.00650s, Expanded Cells = 1285

Maze 2 Forward Search: Time = 0.00002s, Expanded Cells = 1
Maze 2 Backward Search: Time = 0.00001s, Expanded Cells = 1

Maze 3 Forward Search: Time = 0.05400s, Expanded Cells = 11972
Maze 3 Backward Search: Time = 0.00314s, Expanded Cells = 655

Maze 4 Forward Search: Time = 0.00592s, Expanded Cells = 1183
Maze 4 Backward Search: Time = 0.04620s, Expanded Cells = 10258

Maze 5 Forward Search: Time = 0.00455s, Expanded Cells = 904
Maze 5 Backward Search: Time = 0.00474s, Expanded Cells = 987
```

Figure 5: Results of Repeated Forward A* and Repeated Backward A* comparison across 5 mazes

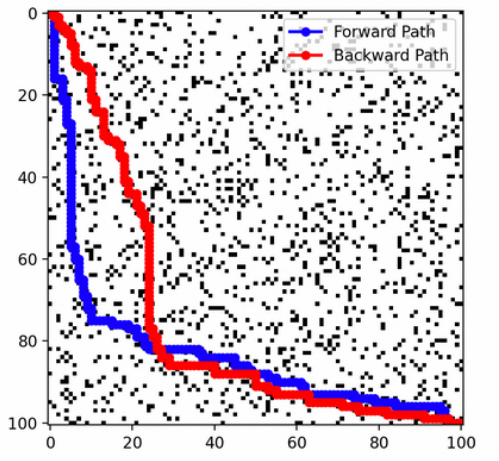

Figure 6: Forward vs. Backward A* Algorithms Representation

$$h(y, x) \leq c(y, z) + h(z, x)$$

where h is the heuristic function, and $c(y, z)$ is the cost function.

In gridworlds, the Manhattan distance is used as the heuristic, which is the sum of the absolute values of the differences in the x and y coordinates of two points. Since the only movements allowed are up, down, left, and right (each with a cost of 1), the Manhattan distance between any point y and its neighbor z will be exactly 1. Thus, for the consistency condition, we have:

$$|y_x - x_x| + |y_y - x_y| \leq 1 + |z_x - x_x| + |z_y - x_y|$$

This inequality holds because moving from y to z changes the Manhattan distance by exactly 1, which is the same as the step cost from y to z. Therefore, the Manhattan distances are consistent in such gridworlds.

6.3 Proof of Adaptive A* Consistency

For Adaptive A*, the heuristic values $h_{\text{new}}(s)$ are updated during the search to reflect the learned information about the actual costs from the states to the goal. The update mechanism ensures that if the heuristic values were initially consistent, they remain so even as action costs increase. This preserves the consistency as seen in the update formula, where $h_{\text{new}}(s) = g(\text{goal}) - g(s)$. The $g(\text{goal})$ represents the actual cost from the start state to the goal state as determined by the search, and $g(s)$ is the cost from the start state to any state s. Since $g(\text{goal})$ reflects the minimum cost to reach the goal under the current conditions, including any increases in action costs, subtracting $g(s)$ from $g(\text{goal})$ yields an estimate that accurately reflects the minimum required cost to get from s to the goal. This method makes sure that the updated $h_{\text{new}}(s)$ values do not overestimate the true cost to reach the goal from state s, adhering to the principle of consistency. The consistency of heuristic values is crucial for ensuring the A* algorithm is optimal, and Adaptive A*'s method of updating heuristics based on search experience maintains this property, even in environments where action costs may increase.

6.4 Implementation and Observation of Adaptive A*

To compare the performance of Repeated Forward A* and Adaptive A*, we ran comparisons across 50 randomly generated mazes. Both algorithms were compared on each maze, and three key metrics were measured: the length of the path found, the runtime of each algorithm, and the number of cells expanded during the search process. This approach allows for a numerical comparison between the two algorithms, enabling us to assess not just the efficiency and speed of each algorithm, but also their effectiveness in navigating complex maze configurations.

In the comparison between Repeated Forward A* and Adaptive A* with a focus on their runtime, it was observed that Adaptive A* consistently outperformed Repeated Forward A* in terms of speed. Both algorithms were implemented to break ties among cells with the same f-value by favoring cells with larger g-values, and any remaining ties were resolved randomly. Despite this common tie-breaking strategy, Adaptive A* demonstrated a notable advantage in runtime efficiency across various maze configurations. This advantage can be seen in Figure 7 which showcases the results of these comparisons over 5 of the 50 mazes.

The reason for this efficiency gain in Adaptive A* is due to its ability to dynamically update its heuristic based on the information learned from previous searches. Unlike Repeated Forward A*, which uses a static heuristic for navigating the maze, Adaptive A* adjusts the heuristic values of the nodes it has expanded, using the actual cost from the node to the goal obtained in previous searches. This approach allows Adaptive A* to have a more informed heuristic in subsequent searches, effectively reducing the search space and leading to faster travel towards the goal. This adaptive refinement of the heuristic value is the key factor that enables Adaptive A* to perform searches more quickly than Repeated Forward A*, as it uses the knowledge gained from past searches to optimize the pathfinding process in real-time.

7 Statistical Significance (Part 6)

To assess whether the performance differences between two search algorithms are systematic or due to sampling noise, a statistical hypothesis test can be effectively utilized. The process begins by creating two

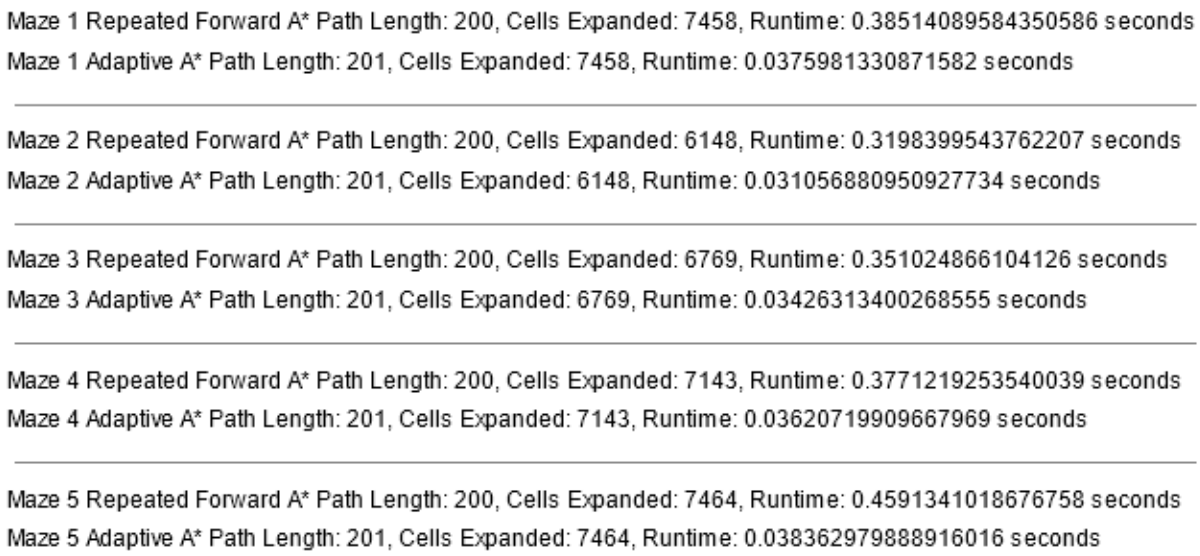

Figure 7: Results of Adaptive A* vs. Repeated Forward A* comparison across 5 mazes

hypotheses: the null hypothesis (H0) which represents that there is no significant difference between the algorithms' performances, while the alternative hypothesis (H1) suggests that a significant difference exists. A suitable test for this scenario is the t-test for independent samples, which is ideal if the performance metrics follow a normal distribution for both algorithms across selected comparisons (such as runtime).

The t-test compares the average performance of each algorithm to figure out if the observed difference is statistically significant. This involves calculating the average and standard deviation of the performance metrics for each algorithm, followed by the test statistic, which measures the difference between the two averages relative to the spread or variability of the scores. If this statistic exceeds a critical value, determined by the chosen significance level p, the null hypothesis is rejected, indicating a systematic difference in performance.

This method functions best where there are a large sample of test cases, such as the 50 we have generated earlier. This large sample ensures that the comparison accurately reflects the algorithms' capabilities and not variations resulting from chance alone. By executing this statistical test, we can confidently determine whether performance differences are due to inherent algorithmic efficiency or merely random variations in the selected test data.

8 Conclusion

In this project, we explored and analyzed the efficiency of pathfinding algorithms in unknown environments, focusing on the Repeated Forward A*, Repeated Backward A*, and Adaptive A* algorithms. Our observations helped bolster our understandings of the critical role of heuristics, particularly the consistency and admissibility of Manhattan distances, in optimizing search strategies. Additionally, our comparison between algorithms highlighted Adaptive A*'s superior ability to dynamically adjust heuristic values, significantly enhancing search efficiency. Our findings highlight the practicality of these algorithms in navigating complex environments and their implications for artificial intelligence.

9 References

- Python Software Foundation. (n.d.). *Python language reference*. Retrieved 2024, from `https://www.python.org`
- Hunter, J. D., Droettboom, M., & the Matplotlib development team. (n.d.). *Matplotlib: Visualization with Python*. Retrieved 2024, from `https://matplotlib.org`
- Cohen, P. R. (1995). *Empirical methods for artificial intelligence*. MIT Press.