Rohan Dash rd890 Maharsh Desai Md1593 Rick Hsu rh758

Understanding A* and Heuristic Search
February 26, 2025

Part 1: Understanding the Methods

1.1 (a) Why does the agent move east first?

Question: Explain in your report why the first move of the agent for the example search problem from Figure 8 is to the east rather than the north, given that the agent does not initially know which cells are blocked.
Answer: There are three possible directions that the agent can move: North, East, and West. Since the agent does not initially know which cells are blocked, it assumes all unknown cells are unblocked and selects the move that minimizes its estimated cost to the goal. The Manhattan distances to the goal for each possible move are calculated as follows:

Start: (E, 2) → (5, 2)
Goal: (E, 5) → (5, 5)
Manhattan Distance Formula: $h(s) = |xs - xt| + |ys - yt|$

Current: $|2 - 5| + |5 - 5| = 3$
North: $|2 - 5| + |4 - 5| = 4$
East: $|3 - 5| + |5 - 5| = 2$
West: $|1 - 5| + |5 - 5| = 4$

Since East has the lowest h(s) value, it is chosen as the next position to move to rather than any other possible direction. A* selects moves based on the smallest estimated total cost $f(s) = g(s) + h(s)$, making east the optimal choice.

1.2 (b) Proof that the agent terminates in finite time

Question: This project argues that the agent is guaranteed to reach the target if it is not separated from it by blocked cells. Give a convincing argument that the agent infinite gridworlds indeed either reaches the target or discovers that this is impossible in finite time. Prove that the number of moves of the agent until it reaches the target or discovers that this is impossible is bounded from above by the number of unblocked cells squared.
Answer: In a finite gridworld, the agent is guaranteed to either reach the target or determine that no path exists within a finite number of moves. This follows from the completeness of A*, which systematically explores paths in increasing order of estimated cost until the goal is reached or all possible routes have been examined. Since the environment consists of a bounded N × N grid, with at most U unblocked cells, the agent's search space is finite. In the worst case, the agent may have to recompute its path multiple times due to newly discovered blocked cells. Each A* search explores at most U cells, and since the agent may need to replan up to U times, the total number of moves is bounded by $O(U^2)$. This ensures that the agent terminates in finite time, either by successfully reaching the target or exhausting all available paths, confirming that the goal is unreachable.

PART 2: THE EFFECTS OF TIES IN REPEATED FORWARD A*

TIE-BREAKING STRATEGIES

In A* search, when multiple cells have the same f-value (f = g + h), a strategy is needed to decide which cell to expand next. Two common strategies are:

1. Favor smaller g-values: Expand cells with smaller g-values first, preferring those closer to the start.
2. Favor larger g-values: Expand cells with larger g-values first, preferring those closer to the goal.

EXPECTED IMPACT

Theoretical Analysis

When A* encounters multiple cells with the same f-value, the choice of which cell to expand can significantly impact search efficiency:

- Favoring smaller g-values tends to explore breadth-first around the start position. This can lead to expanding more cells that are far from the goal.
- Favoring larger g-values tends to explore depth-first toward the goal, which can be more efficient because:
  - It focuses the search toward the goal.
  - It finds a path to the goal more quickly.
  - It reduces the number of cells expanded.

For the example in Figure 9 from the assignment, when favoring larger g-values, A* will prioritize cells that are closer to the goal among those with the same f-value. This results in a more direct path toward the goal, potentially finding a solution with fewer cell expansions.

IMPLEMENTATION DETAILS

Our implementation uses a dictionary-based priority queue, where:

- States are organized by their f-values.
- For states with the same f-value, selection is based on g-values according to the chosen strategy.
- The number of expanded cells is tracked to measure efficiency.

EXPERIMENTAL RESULTS

After running experiments on multiple maze environments, we observed:

1. Fewer expanded cells when favoring larger g-values.
2. Faster runtime when favoring larger g-values.
3. Similar path lengths for both strategies (since both find optimal paths).

The detailed results and visualizations can be found in the following files:

- Tie_breaking_results.txt
- tie_breaking_comparison.png

EXPLANATION OF OBSERVATIONS

Favoring larger g-values is generally more efficient because it guides the search more directly toward the goal. When multiple paths have the same estimated total cost (f-value), choosing the one that has made more progress toward the goal (larger g-value) tends to find a solution faster. In the context of Repeated Forward A*, this efficiency is even more critical since the algorithm performs multiple A* searches as the agent discovers new information about the environment. Each search must be as efficient as possible to minimize the overall computational cost.
This behavior aligns with the intuition that when faced with multiple equally promising directions, it's better to continue in the direction that has made the most progress toward the goal rather than exploring alternatives that are closer to where you started

Part 3 - Forward vs Backward A Analysis

Implementation

We implemented both Repeated Forward A* and Repeated Backward A* algorithms for the Fast Trajectory Replanning problem. Both implementations break ties among cells with the same f-value in favor of cells with larger g-values.

Experimental Results

Our experiments on 20 randomly generated mazes of size 101×101 with approximately 30% blocked cells showed significant performance differences between the two algorithms:

1. Expanded Cells:
   - Forward A*: Average of 2,121.40 cells expanded
   - Backward A*: Average of 11,544.95 cells expanded
   - Ratio (Forward/Backward): 0.18x

Forward A* expanded only 18% as many cells as Backward A*, making it substantially more efficient in terms of search space exploration.

2. Path Length:
   - Forward A*: Average path length of 164.60
   - Backward A*: Average path length of 155.30

While Backward A* found slightly shorter paths on average, Forward A* was more computationally efficient.

3. Runtime:
   - Forward A*: Average runtime of 0.0259 seconds
   - Backward A*: Average runtime of 0.1366 seconds
   - Ratio: Forward A was approximately 5.3x faster*

Runtime correlates with the number of expanded cells, confirming Forward A*'s efficiency advantage.

Detailed Analysis

Looking at the individual maze results, we observe:

1. Consistency of Performance Advantage:
   - Forward A* consistently expanded fewer cells than Backward A* across all 20 mazes.
   - Even in the worst case for Forward A*, it still expanded 15% fewer cells than Backward A*.

2. Correlation with Maze Complexity:
   - The advantage of Forward A* appears more pronounced in complex mazes.
   - In mazes where both algorithms expanded many cells (e.g., Maze 15: 5,304 vs. 58,208 cells), the ratio was smaller (0.09x), showing a greater advantage for Forward A*.

- In simpler mazes (e.g., Maze 9: 257 vs. 320 cells), the ratio was closer to 1 (0.80x).

Explanation of Observations

The dramatic difference in performance can be explained by several factors:

1. Fog of War Effect:
   - Forward A* starts from the agent's position, where it has more information about the Environment.
   - Backward A* starts from the target, where it has no initial knowledge of obstacles.
   - Backward A* wastes more time exploring invalid paths due to missing information.
2. Information Gain During Search:
   - Forward A* builds paths from known (agent's surroundings) to unknown.
     - More informed early decisions
     - Focused exploration in promising directions
     - Less need for recomputation when obstacles are discovered
   - Backward A* works from unknown (target) to known (agent), leading to inefficiencies.
3. Heuristic Effectiveness:
   - The Manhattan distance heuristic may be more accurate when estimating distance from the agent than from the target.
4. Path Quality Trade-off:
   - Backward A* sometimes finds slightly shorter paths, but at a huge computational cost.
   - The minor 5.6% improvement in path length does not justify the 5.3x slower performance.

Conclusion

For repeated A search in a fog of war scenario*, Forward A* significantly outperforms Backward A* in terms of computational efficiency:

- 5.3x faster
- 82% fewer cells expanded
- Only 5.6% longer paths (on average)

Thus, in applications where agents navigate partially observable environments, searching from the agent to the goal is preferable, especially when:

- Computational resources are limited
- Real-time performance is important

This result: when navigating an unfamiliar environment, it is more efficient to plan a path starting from where you have the most information (your current surroundings) rather than working backward from a destination with limited knowledge

Part 4 - Heuristics in the Adaptive A* [20 points]:

1) The project argues that “the Manhattan distances are consistent in gridworlds in which the agent can move only in the four main compass directions.” Prove that this is indeed the case.

Using the definition of a heuristic is consistent if h(s) <= c(s,a) + h(s’). h(s) is the heuristic estimate of the cost from going to s to the target. c(s,a) is the cost of moving from s to s’ and h(s’) is the heuristic estimate from s’ to the target.

Theorems

Because the state s is at coordinates (x,y) and the target is another set of (x',y') where x' is xgoal and y' is ygoal. Then h(s) = |x-x'| + |y-y'|

Because of this, the agent can move in 4 cardinals, with each move of 1 unit, where c(s,a) = 1

Proof
If we let s=(x,y), and s' be any neighboring state reached in any of the 4 directions, then moving east or west changes x by 1 unit, as h(s') = |(x +/- 1) - xgoal| + |y-ygoal|
When moving north or south, y coordinate changes by 1 as h(s') = |x-xgoal| + |(y+/- 1) - ygoal|

Since moving one step in any direction either gets you reduces or equal total cost to the target, by at most one, so h(s') >= h(s) -1 and h(s) <= h(s')+1
Since c(s,a) = 1, h(s) <= c(s,a) + h(s') , we satisfy the condition.

Conclusion: As manhattan distances satisfy the consistency condition and prove that the cost increase always be 1 or 0 in its travel to the target, Adaptive A* modifies this bc it updates h(s) by recognizing which are not going to break the consistency rule even if h(s) increases. This keeps the Manhattan distance valid and consistent through searching. In Adaptive A* it always finds the optimal path, and never revisits states unnecessarily This makes it efficient in its search. Overall, the Manhattan distance can only move in the main 4 compass directions, even when utilizing A*

2) Furthermore, it is argued that "The h-values $h_{new}(s)$... are not only admissible but also consistent." Prove that Adaptive A* leaves initially consistent h-values consistent even if action costs can increase.

The problem asks us to prove that the $h_{new}(s)$ in Adaptive A* will be consistent even if action costs go up.

Theorems and Definitions
h(s) is consistent when h(s) <= c(s,a) + h(s') ; h(s) is the heuristic estimate of the cost from going to s to the target. c(s,a) is the cost of moving from s to s' and h(s') is the heuristic estimate from s' to the target.

Proof
$h_{new}(s) = g(s_{goal}) - g(s)$, where g(s) is the shortest path from start to s found in curr search and $g(s_{goal})$ is the actual shortest path cost to the target. This update ensures that at state s is either as accurate as the previous search or better.

$h_{new}(s) = g(s_{goal}) - g(s)$
$h_{new}(s') = g(s_{goal}) - g(s')$

Because the agent moves from s to s' with cost c(s,a): $g(s) <= g(s) + c(s,a)$
$g(s_{goal}) - g(s) <= c(s,a) + g(s_{goal}) - g(s')$
$h_{new}(s) <= c(s,a) + h_{new}(s')$

This holds to the previous statement that even if the action cost increases b/w searches the new search accounts for the increased cost. It will never decrease h- values as Adaptive A* ensures that future searches expand fewer state by making the heuristic more efficient

Part 5 - Forward A vs. Adaptive A Analysis*

Implementation

We implemented both Repeated Forward A* and Adaptive A* for the Fast Trajectory Replanning problem. Both implementations break ties among cells with the same f-value in favor of cells with larger g-values. Adaptive A* dynamically updates its heuristic values based on previously computed paths to improve efficiency in future searches.

Experimental Results

We ran experiments on 50 randomly generated mazes of size 101×101 with approximately 30% blocked cells. The results show how Adaptive A* compares to Forward A* in terms of efficiency:

1. **Expanded Cells:**
    - Forward A*: Average of 7,916.52 cells expanded
    - Adaptive A*: Average of 8,042.46 cells expanded
    - Ratio (Adaptive/Forward): 1.016x (1.6% more expanded cells)

Contrary to expectations, Adaptive A expanded slightly more nodes than Forward A*. This suggests that the heuristic updates may not be significantly reducing search effort in subsequent iterations for mazes of size 101x101.

2. **Path Length:**
    - Forward A*: Average path length of 161.58
    - Adaptive A*: Average path length of 170.00
    - Difference: Adaptive A* paths were 5.21% longer

The Adaptive A* paths were slightly longer, which contradicts the typical assumption that improved heuristics should guide the search toward more efficient routes, but was not unexpected, as the larger the mazes become, the better adaptive A* will run.

3. **Runtime:**
    - Forward A*: Average runtime of 0.100888 seconds
    - Adaptive A*: Average runtime of 0.113215 seconds
    - Ratio (Adaptive/Forward): 1.12x (12.22% slower)

Adaptive A* was slower than Forward A* despite its intent to reduce search effort. This suggests that the overhead of heuristic updates was not offset by a reduction in node expansions.

Detailed Analysis

1. **Consistency of Performance**
    - Adaptive A was slower than Forward A* in most test cases.
    - The additional heuristic updates did not consistently reduce node expansions.
    - In some mazes, Adaptive A expanded fewer nodes, but in others, it expanded significantly more, making its performance inconsistent.
2. **Impact of Maze Complexity**
    - The performance gap increased in larger and more complex mazes.
    - In some cases, Adaptive A expanded more than 10% additional nodes compared to Forward A*.
    - In simpler mazes, the difference was negligible, suggesting that Adaptive A* is less beneficial in randomly structured environments.

Explanation of Observations

1. **Unexpected Increase in Expanded Nodes**
    - Adaptive A* updates heuristic values dynamically, intending to reduce future search efforts.
    - However, if the heuristic updates do not significantly improve search efficiency, the additional computational overhead may lead to more expanded nodes instead of fewer.
    - In these experiments, the heuristic updates may not have been effective enough to improve the overall search efficiency.
2. **Overhead of Heuristic Updates**
    - Each time Adaptive A* completes a search, it modifies the heuristic values for future searches.
    - This additional computation does not pay off unless the environment structure allows for significant reuse of the updated heuristics.
    - In these mazes, the adaptive updates did not meaningfully reduce search effort, leading to increased runtime.
3. **Trade-off Between Path Length and Computation**
    - Theoretically, Adaptive A should reduce node expansions and improve path efficiency.
    - However, in this case, it found longer paths while still expanding more nodes.
    - This suggests that the updated heuristics were not effectively guiding the search toward better paths.

Conclusion

For repeated A* search in a fog-of-war scenario, Adaptive A underperformed compared to Forward A*:

- Expanded 1.6% more nodes on average.
- Increased path lengths by 5.21%.
- Ran 12.22% slower despite being designed to improve efficiency.

These results indicate that Adaptive A may not be beneficial in highly random environments, where the heuristic updates do not sufficiently improve search efficiency. Forward A* remains more computationally efficient in this setting of 101x101 grids.

When to prefer Forward A*:

- When computational resources are limited.
- When search time is critical.
- When navigating environments where heuristic improvements have little impact.

When Adaptive A would be more efficient:

- When solving highly structured environments where updated heuristics can be reused effectively.
- When paths reuse previous computations rather than requiring new searches from different starting locations.

Key Takeaways

- Forward A was overall more efficient than Adaptive A* in this set of tests.
- The heuristic updates in Adaptive A* did not significantly reduce node expansions.
- The additional overhead of updating heuristics resulted in slower runtimes.
- In structured environments, Adaptive A may perform better, but in random environments, Forward A remains the better choice.

Part 6 - Statistical Significance: Hypothesis Testing for Forward A vs. Adaptive A**

To determine whether the observed differences in performance metrics between Forward A* and Adaptive A* are statistically significant or due to random variation, we will perform a paired t-test and compute the p-value.

1. **Defining the Hypothesis**
    - **Null Hypothesis (H_0)**: There is no significant difference in the performance of Forward A* and Adaptive A*. The observed differences in expanded nodes, runtime, or path length are due to random variation.
    - **Alternative Hypothesis (H_1)**: There is a statistically significant difference in the performance of the two algorithms, meaning the observed differences are not due to random variation.

    This test will determine whether one algorithm systematically outperforms the other.

2. **Choosing the Statistical Test: Paired t-test**

    Since each maze was tested with both Forward A* and Adaptive A*, the data is paired. The correct test is a paired t-test, which measures whether the mean difference between paired samples is statistically significant.

    We will perform the test on three different performance metrics:

    1. Expanded Nodes (computational efficiency)

2. Path Length (solution quality)
3. Runtime (execution speed)

The paired t-test is appropriate if the differences are normally distributed; otherwise, we might need to use a non-parametric test like the Wilcoxon signed-rank test (not required here since we are focusing on t-tests).

3. Steps to Perform the Paired t-test

Step 1: Collect Data

- X_i = Performance of Forward A* on maze i (expanded nodes, runtime, or path length).
- Y_i = Performance of Adaptive A* on maze i.
- Compute the difference: $D_i = Y_i - X_i$

This represents how much Adaptive A* differs from Forward A* on each maze.

Step 2: Compute the Mean and Standard Deviation of Differences

1. Mean difference: $Đ = (\Sigma D_i)/n$ ← Đ means D-bar

   where n = 50 (the number of mazes tested).

2. Standard deviation of differences: $S_D = (\Sigma(D-Đ)^2 / (n - 1))^(1/2)$

Step 3: Compute the t-statistic

The t-statistic measures how significant the difference is: $t = Đ / (S_D / n^(1/2))$

Where:

- D = Mean difference
- S_D= Standard deviation of differences
- n = 50 (number of test cases)

A larger |t| value means a stronger difference between the two algorithms.

Step 4: Compute the p-value

The p-value tells us how likely it is that the observed difference happened by chance.

- If $p < 0.05$, we reject H_0 and conclude that there is a statistically significant difference between Forward A* and Adaptive A*.
- If $p \geq 0.05$, we fail to reject H_0, meaning the difference could be due to random noise, and we do not have enough evidence to claim one algorithm is systematically better.

We compare our computed t-value to the critical t-value from a t-distribution table at 49 degrees of freedom (n-1)to determine if the result is statistically significant.

4. Interpretation of Results

1. Expanded Nodes:
    - If $p < 0.05$, Adaptive A* expands significantly more (or fewer) nodes than Forward A*.
    - If $p \geq 0.05$, the difference in expanded nodes is not statistically significant.
2. Path Length:
    - If $p < 0.05$, one algorithm finds significantly longer or shorter paths.
    - If $p \geq 0.05$, the path length difference is likely due to random noise.
3. Runtime:
    - If $p < 0.05$, there is a significant runtime difference.
    - If $p \geq 0.05$, the runtime difference is not significant.

If all p-values are high (above 0.05), the differences in performance between Forward A* and Adaptive A* are likely due to random variation in the mazes rather than a systematic difference.

Conclusion

To determine whether Forward A* or Adaptive A* is truly better, we perform a paired t-test on the expanded nodes, path length, and runtime. The p-value will tell us whether the observed differences are statistically significant.

If $p < 0.05$, we conclude that one algorithm systematically outperforms the other. Otherwise, any performance differences are likely due to random variation in the test cases rather than a fundamental advantage of one algorithm over the other.