CS440: Introduction to Artificial Intelligence
Fall 2024
Assignment 1

Rahul Sankaralingam Sahil Sharma Dennis Hemans

October 1, 2024

Part 1 - Understanding the Methods

(a) In Figure 7, the agent's first move is to the east instead of north because of the heuristic values from an A* search, which determines the distances travelled to each cell. In brief, the agent does not know of the blocked cells in the grid, so it assumes the shortest "unblocked" path to the goal. By utilizing Manhattan distances, the lowest cost from the origin points towards the cell with the lower h-value; in this case, the agent points east. By going in this direction, the agent assumes that it can reach the goal in the shortest "unblocked" method (a total of 3 cell moves to the right).

(b) By using A* in gridworlds, the agent dynamically updates its knowledge on blockages and finds path efficiency, which allows the agent to constantly readjust its path. The agent will explore each adjacent cell to try to obtain the goal, based on if the cell is blocked or not. It will either reach the target or completely exhaust all possible paths to the target—either directly or indirectly blocked—in a finite number of steps. Therefore, in a finite grid, the agent is guaranteed to either obtain a possible path to the goal or determine that there are no possible paths to the goal due to obstructions. Furthermore, in the worst-case scenario for A*, the agent may need to explore every single unblocked cell. In addition, it would go towards unexplored areas as well as revisiting initial areas. Assuming there are a total of n unblocked cells, and a number of path movements in this backtracking scenario is also represented by n, the maximum moves that the agent can take towards the goal can be represented as the product of total unblocked cells and path movements that were back-traced: n^2. This value represents the upper bound of steps where the agent completely exhausts all cells and paths in a gridworld.

Part 2 - The Effects of Ties

Introduction

In this part of the assignment, we explored the impact of different tie-breaking strategies on the performance of Repeated Forward A*. The goal was to understand how these strategies influence the number of expanded nodes and runtime during pathfinding in a gridworld environment.

Tie-Breaking Strategies Compared:

- Smaller g-values: The algorithm prefers nodes with smaller g-values when nodes have the same f-value. This results in prioritizing nodes that are closer to the start.
- Larger g-values: The algorithm prefers nodes with larger g-values when nodes have the same f-value. This results in prioritizing nodes that are farther from the start.

Methodology

We generated 50 gridworlds and ran the Repeated Forward A* algorithm on each of them, using both tie-breaking strategies. For each gridworld, we recorded the following metrics:

- Number of expanded nodes.
- Runtime.

The results were saved in a CSV file, and the average values of these metrics were calculated to compare the two strategies.

Results

The average results obtained are summarized as follows:

Metric	**Smaller g**	**Larger g**
Average Expanded Nodes	3,469.94	2,675.74
Average Runtime (seconds)	0.009645	0.007403

Observations

1. Fewer Expanded Nodes with Larger g-values:
   - When breaking ties in favor of larger g-values, the average number of expanded nodes was significantly lower compared to breaking ties in favor of smaller g-values.
   - Reason: Prioritizing nodes with larger g-values keeps the search focused on nodes that are closer to the goal. This strategy avoids unnecessary detours and redundant exploration of nodes closer to the start.
2. Lower Runtime with Larger g-values:
   - The runtime of Repeated Forward A* was also lower when using larger g-values as the tie-breaking strategy.
   - Reason: With fewer nodes being expanded, the search completes faster, resulting in reduced runtime.

Conclusion

The results indicate that breaking ties in favor of larger g-values is more efficient in terms of both the number of expanded nodes and runtime. This behavior is expected because prioritizing nodes closer to the goal helps the algorithm stay focused on reaching the target, minimizing unnecessary exploration.

Part 3 - Repeated Forward A* VS Repeated Backward A*

Introduction

In this part of the assignment, we compare the performance of two algorithms, Repeated Forward A* and Repeated Backward A*, in grid-based pathfinding. The goal is to analyze and understand how these two versions of A* differ in terms of:

- Number of expanded nodes.
- Runtime.

Methodology

The algorithms were tested on 50 gridworlds of size 101x101. Each gridworld was generated using a randomized depth-first search (DFS) algorithm. For each gridworld, both Repeated Forward A* and Repeated Backward A* were executed, and the following metrics were recorded:

- Number of expanded nodes.
- Runtime (in seconds).

The results were stored in a CSV file and visualized using bar charts to observe the differences between the two algorithms.

Results

The average results obtained from running both algorithms on all 50 gridworlds are summarized as follows:

Metric	Repeated Forward A*	Repeated Backward A*
Average Expanded Nodes	3,469.94	5,437.98
Average Runtime (seconds)	0.009323	0.013919

Observations

1. Repeated Forward A* Expands Fewer Nodes:
   - On average, Repeated Forward A* expanded significantly fewer nodes (3,469.94) compared to Repeated Backward A* (5,437.98).
   - Reason: The backward search starts from the goal and searches towards the agent's starting position. In many grid-based environments, this results in expanding more nodes because the search space is less constrained and tends to cover more area.
2. Lower Runtime for Repeated Forward A*:
   - The runtime of Repeated Forward A* was also consistently lower than Repeated Backward A*.
   - Reason: Since Repeated Forward A* expands fewer nodes, it results in less computational overhead, leading to lower runtime.

Impact of Gridworld Structure

The structure of the gridworlds can heavily influence the performance of both algorithms. In gridworlds with narrow paths or lots of obstacles near the goal, Repeated Backward A* tends to explore many more nodes.

Conclusion

The results clearly indicate that Repeated Forward A* is more efficient than Repeated Backward A* in terms of both the number of expanded nodes and runtime. This is because starting the search from the agent's position allows Repeated Forward A* to better focus on the path towards the goal, minimizing unnecessary exploration.

On the other hand, Repeated Backward A* tends to expand a larger number of nodes, resulting in higher runtime. This suggests that Repeated Forward A* is generally more suitable for grid-based pathfinding tasks when compared to Repeated Backward A*.

Part 4 - Evaluating Adaptive A* Performance

Introduction

In this part, we looked at the performance of pathfinding tasks within grids by using Adaptive A*. The goal is to see how the adaptive environment of the heuristic values affects the efficiency of searching for path lengths and nodes expanded. This helps to authenticate consistently updated heuristics over different terrains.

Proof

If a heuristic value "h" is consistent, it is consistent if every node, say "n", and its successor, say "n'", are generated by an action, "a". The cost from goal "g" to "n" is:

$$h(n) \leq c(n, a, n') + h(n')$$

Here, c(n,a,n') represents the cost from n to n', through action "a". In the gridworlds, since the uniform cost is 1 per move, the condition for consistency is simplified as:

$$|x_{\text{goal}} - x_n| + |y_{\text{goal}} - y_n| \leq 1 + |x_{\text{goal}} - x_{n'}| + |y_{\text{goal}} - y_{n'}|$$

Since n' is always directly adjacent to n, the Manhattan distance is going to differ by 1 every time, so therefore our heuristics are consistent. In Adaptive A*, after reaching the goal, the heuristic value "h" for every single expanded node becomes updated as g(goal)-g(n), which is now the actual cost to the goal. Assuming that our initial heuristics are consistent, we can say for each node "n": h_new (n) = g(goal)-g(n) Since the g-values never reduces, and the goal distance is always getting reduced by 1 as each step is taken towards the target, this means that the update heuristics satisfy the triangle inequality (as they have the exact cost to goal node from each node), therefore showing consistency. Using the provided part4.py, we ran the Adaptive A* through multiple gridworlds, and outputted "yes" for almost all gridworlds (through the "is_consistent" function).

Conclusion

In brief, both theoretical and code analyses portray that Manhattan distances are a consistent heuristic measure for gridworlds (that are restricted to up, down, left, and right movements). Even as the action costs change, the heuristics remain consistent, which altogether underpins the strength and sturdiness of the Adaptive A* algorithm in efficient pathfinding strategies within stable heuristics frameworks.

Part 5 - Comparison of Repeated Forward A* and Adaptive A*

Introduction
In this part of the assignment, we compare Repeated Forward A* and Adaptive A* to understand how updating heuristic values in Adaptive A* influences the algorithm's performance. The comparison is based on two metrics:

- Number of expanded nodes.
- Runtime.

Methodology
Gridworlds: Both algorithms were tested on the same set of 50 gridworlds, each of size 101x101.

Metrics Recorded: For each gridworld, we measured:

- Number of expanded nodes.
- Runtime (in seconds).

Algorithm Overview:

- Repeated Forward A*:
  - A new A* search is performed every time the agent encounters a blocked cell along the current path.
- Adaptive A*:
  - Uses heuristic values that are updated based on the cost of previous searches, allowing the heuristic to improve and guide the search more effectively in subsequent searches.

Results
The average results obtained from running both algorithms on all 50 gridworlds are summarized as follows:

Metric	**Repeated Forward A***	**Adaptive A***
Average Expanded Nodes	3009.98	3006.32
Average Runtime (seconds)	0.010730	0.010822

Observations

1. Number of Expanded Nodes:
   - Repeated Forward A*: 3009.98 nodes
   - Adaptive A*: 3006.32 nodes
   - The difference in the number of expanded nodes is minimal, with Adaptive A* expanding slightly fewer nodes (3 nodes less on average). This suggests that the heuristic updates in Adaptive A* did not significantly impact the node expansions for this set of gridworlds.
2. Runtime:

- Repeated Forward A*: 0.010730 seconds
- Adaptive A*: 0.010822 seconds
- The runtime of Adaptive A* is slightly higher than Repeated Forward A* despite expanding fewer nodes. This is likely due to the overhead of updating heuristic values during each search. The difference is, however, marginal (around 0.000092 seconds).

Behavioral Differences

Although Adaptive A* explores slightly fewer nodes, the time taken to update heuristic values in Adaptive A* offsets this advantage, leading to similar runtimes.

In complex gridworlds with more obstacles or a convoluted path to the goal, Adaptive A* may show more significant improvements due to its ability to refine heuristic values over subsequent searches.

Explanation of Results

- Adaptive A's Slight Reduction in Expanded Nodes*:
  - Adaptive A* updates its heuristic values based on previous search results, allowing it to better estimate distances in future searches. This results in slightly fewer nodes being explored.
  - The small difference suggests that the initial heuristic values were already well-informed or that the specific gridworlds did not offer much room for heuristic refinement.
- Minimal Difference in Runtime:
  - The additional time taken by Adaptive A* to update heuristic values negates the small advantage it gains in reducing the number of expanded nodes.
  - This results in a runtime that is comparable to Repeated Forward A*, with a very slight increase in execution time for Adaptive A*.

Conclusion

The results show that Adaptive A* performs almost identically to Repeated Forward A* in terms of both expanded nodes and runtime. This suggests that in the given set of gridworlds, the overhead of heuristic updates in Adaptive A* did not lead to a significant performance gain. In more complex scenarios or with different gridworld structures, the advantage of Adaptive A* might become more pronounced.

In general, while Adaptive A* has the potential to reduce the number of expanded nodes in more challenging environments, the results suggest that it performs similarly to Repeated Forward A* when the gridworlds are relatively straightforward.