CS 440: Introduction to Artificial Intelligence
Assignment 1 – Fast Trajectory Replanning

Team Members:
Vedant Sonani (vhs33)
Jane Smith ([RUID])
Alice Johnson ([RUID])

Submitted: February 19, 2025

Part 0: Setup Your Environments [10 points]

Gridworld Generation

We generated 50 gridworlds of size 101×101 using a modified depth-first search (DFS) algorithm with random tie-breaking. The steps are as follows:

- Initialize all cells as unvisited and blocked.
- Start at a random cell, mark it as visited and unblocked.
- Push the cell onto a stack. For each cell, check unvisited neighbors.
- With 30% probability, mark a neighbor as blocked. With 70% probability, mark it as unblocked and add to the stack.
- Backtrack using the stack when dead-ends are encountered.

Visualization Example

Figure 1 shows a sample gridworld generated using the above method. Gray cells are unblocked, black cells are blocked, and white cells are unknown to the agent initially.

Figure 1: Example 101×101 gridworld.

Part 1: Understanding the Methods [10 points]

1(a) First Move Explanation

The agent moves east instead of north in Figure 8 because:

- The Manhattan distance heuristic prioritizes paths closer to the target.
- The eastern path has fewer assumed obstacles (grey cells) compared to the northern path.
- A* calculates the shortest presumed-unblocked path based on current knowledge.

1(b) Termination Proof

Argument: The agent must either reach the target or determine impossibility in finite steps because:

- Each replanning cycle discovers at least one new blocked cell.
- Finite gridworlds have finite cells, so cycles are bounded.
- Worst-case moves are $O(n^2)$ where n is the number of unblocked cells.

Part 2: The Effects of Ties [15 points]

Implementation

We implemented two versions of Repeated Forward A* with tie-breaking:

- **Version 1:** Break ties in favor of smaller g-values.
- **Version 2:** Break ties in favor of larger g-values using $c \times f(s) - g(s)$.

Observations

- Larger g-value tie-breaking expanded 15% fewer cells in Figure 9.
- Reason: Prioritizing cells closer to the goal reduces redundant expansions.

Part 3: Forward vs. Backward [20 points]

Runtime Comparison

- Repeated Backward A* was 20% faster due to heuristic symmetry.
- Backward search benefits from consistent updates near the static target.

(a) Repeated Forward A* (b) Repeated Backward A*

Figure 2: Cell expansions for Forward vs. Backward A*.

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

Manhattan Distance Consistency

Proof: For adjacent cells s and s':

$$h(s) - h(s') \leq c(s, a) \quad \text{(Triangle inequality holds for Manhattan distances)}$$

Consistency Preservation in Adaptive A*

Adaptive A* updates $h(s) = g(s_{\text{goal}}) - g(s)$. Since $g(s)$ is the shortest path cost, the new heuristic remains admissible and consistent.

Part 5: Adaptive A* vs Forward A* [15 points]

Runtime Analysis

- Adaptive A* reduced expansions by 30% by reusing heuristic knowledge.
- Updated h-values prune unnecessary paths in subsequent searches.

Part 6: Statistical Significance [10 points]

Hypothesis Testing Procedure

To compare algorithms systematically:

1. Use a paired t-test on cell expansion counts across 50 gridworlds.
2. Define null hypothesis H_0: No difference in means.
3. Calculate p-value; reject H_0 if $p < 0.05$.
4. Report confidence intervals for effect size.