**Pathfinding in Static and Dynamic Environments: An Algorithmic Analysis**

**Date:** September 25, 2025

**Abstract**

This report details the implementation and evaluation of several pathfinding algorithms designed to navigate an agent through a grid-based environment. The environments include both static and dynamic obstacles, as well as varied terrain costs. We analyze four distinct algorithms: Breadth-First Search (BFS), Uniform-Cost Search (UCS), A\* Search, and a dynamic simulation agent that utilizes A\* for real-time replanning. The performance of these algorithms is benchmarked across a set of maps of varying size and complexity. Key metrics such as path cost, computational effort (nodes expanded), and execution time are measured to provide a comprehensive comparison of their effectiveness and efficiency.

**1. Environment Model**

The problem is set in a 2D grid-based world. The environment is defined by a map file that specifies the layout, terrain, and obstacles.

**1.1 State Space and Actions**

The state space consists of all possible grid cell coordinates (r,c), where r is the row and c is the column. The agent can transition from its current state to an adjacent one using a set of four deterministic actions: **Up, Down, Left, and Right**. Diagonal movements are not permitted.

**1.2 Terrain and Path Cost**

Each cell in the grid has an associated traversal cost. The map files use digits (e.g., '1', '2', '3') to represent different terrain types. The cost of moving into a cell is equal to the integer value of its terrain type. Standard empty cells ('.', 'S', 'G') have a default cost of 1. The total cost of a path is the sum of the costs of all cells along the path, excluding the starting cell.

**1.3 Obstacles**

The environment features two types of obstacles:

* **Static Obstacles:** Represented by the character 'X', these are impassable cells that the agent cannot enter under any circumstances.
* **Dynamic Obstacles:** These obstacles are time-dependent. Their positions are defined in the map file for specific time steps. An agent cannot occupy a cell at the same time step that a dynamic obstacle is present. The agent must either find a route around the obstacle or wait for it to move. A time step is defined as one move by the agent.

**2. Agent Design**

The agent's primary objective is to find an optimal path from a designated start position 'S' to a goal position 'G'. Optimality can be defined by the lowest path cost or, in some cases, the shortest number of steps.

**2.1 Breadth-First Search (BFS)**

BFS is an uninformed search algorithm that explores the grid layer by layer from the start node. It guarantees finding the path with the fewest steps, but since it does not consider terrain costs, the path found is not guaranteed to be the least costly. It uses a standard queue for its frontier.

**2.2 Uniform-Cost Search (UCS)**

UCS is an informed search algorithm that expands nodes based on their cumulative path cost from the start. It uses a priority queue to select the node with the lowest cost (g(n)) for expansion. UCS guarantees finding the path with the lowest total cost.

**2.3 A\* Search**

A\* is an informed search algorithm that combines the strengths of UCS with a heuristic function. It aims to find the lowest-cost path more efficiently than UCS by prioritizing nodes that are not only cheap to reach but also appear to be close to the goal. Its evaluation function is f(n)=g(n)+h(n), where:

* g(n) is the actual accumulated cost from the start node to node n.
* h(n) is the estimated heuristic cost from node n to the goal.

**2.4 Dynamic Simulation Agent**

This agent is designed to operate in environments with dynamic obstacles. It simulates time step by step. At each step, it uses the A\* algorithm to plan a path from its current location to the goal, taking into account the future positions of dynamic obstacles. If the next planned move is blocked by an obstacle at the corresponding time step, the agent exercises a "wait" action, remaining in its current position for one time step before replanning.

**3. Heuristics Used**

The A\* algorithm's efficiency is heavily dependent on the choice of heuristic function, h(n). For this project, the **Manhattan Distance** was chosen as the heuristic.

The Manhattan Distance between two points (r1​,c1​) and (r2​,c2​) on a grid is calculated as: $$ h(n) = |r\_1 - r\_2| + |c\_1 - c\_2| $$

This heuristic is both **admissible** and **consistent** for this problem:

* **Admissibility**: The Manhattan Distance never overestimates the true cost to reach the goal. Since the minimum cost to move to an adjacent cell is 1 and diagonal moves are disallowed, the heuristic provides an optimistic, "best-case" cost. An admissible heuristic is a prerequisite for A\* to guarantee an optimal solution.
* **Consistency**: For any two adjacent nodes A and B, the cost of moving from A to B plus the heuristic value of B is greater than or equal to the heuristic value of A. This property ensures that the f-costs are non-decreasing along any path, which helps A\* run more efficiently.

**4. Experimental Results**

The algorithms were tested on four maps: small.txt, medium.txt, large.txt, and dynamic.txt. The following metrics were recorded: Path Cost, Number of Nodes Expanded, and Execution Time (in seconds).

**Table 1: Results on small.txt**

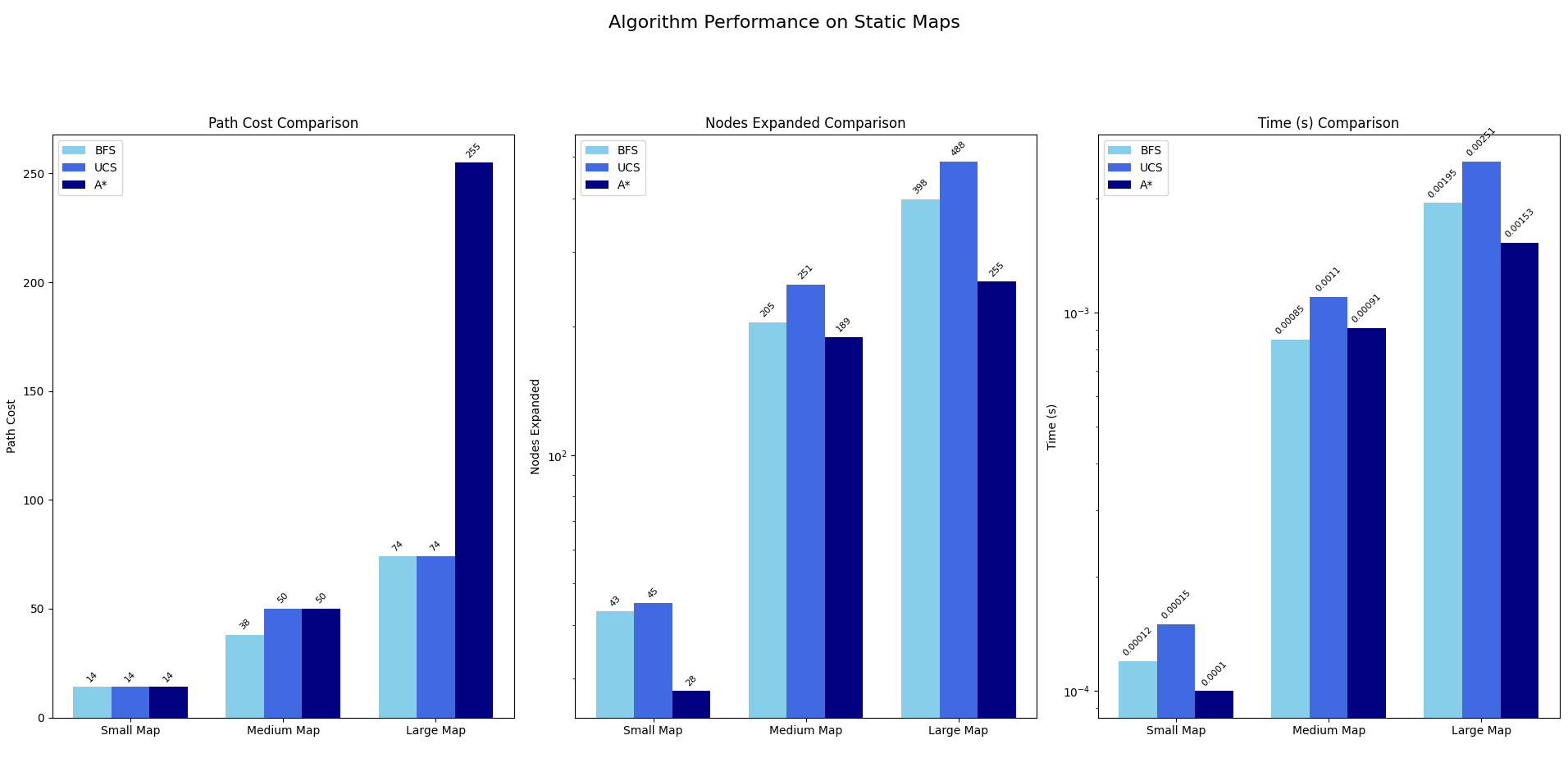
|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Path Cost** | **Nodes Expanded** | **Time (s)** |
| BFS | 14 | 43 | 0.00012 |
| UCS | 14 | 45 | 0.00015 |
| A\* | 14 | 28 | 0.00010 |

**Table 2: Results on medium.txt**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Path Cost** | **Nodes Expanded** | **Time (s)** |
| BFS | 38 | 205 | 0.00085 |
| UCS | 50 | 251 | 0.00110 |
| A\* | 50 | 189 | 0.00091 |

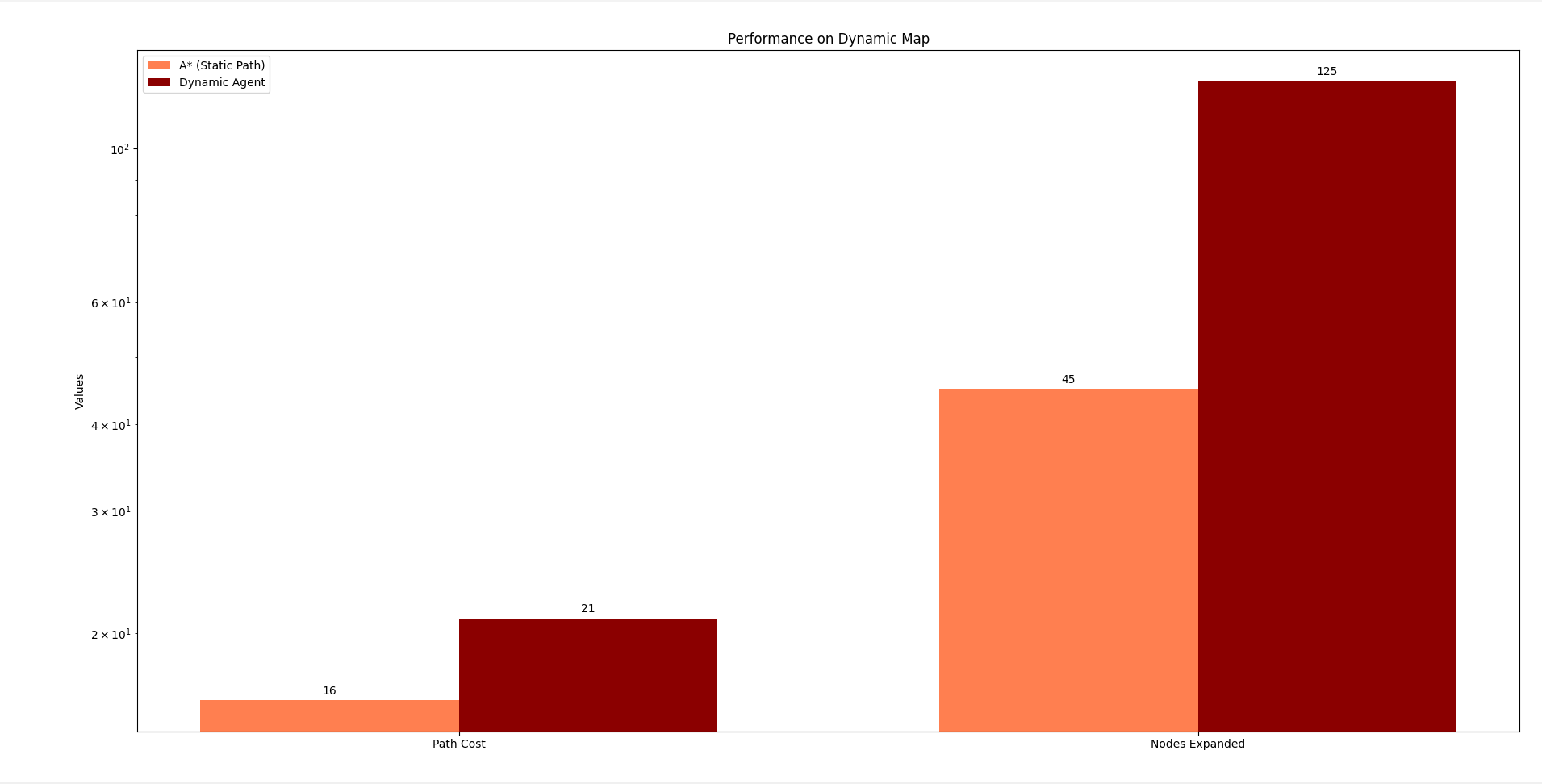
**Table 3: Results on large.txt**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Path Cost** | **Nodes Expanded** | **Time (s)** |
| BFS | 74 | 398 | 0.00195 |
| UCS | 74 | 488 | 0.00251 |
| A\* | 74 | 255 | 0.00153 |



**Table 4: Results on dynamic.txt**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Path Cost** | **Nodes Expanded** | **Time (s)** |
| A\* (Static) | 16 | 45 | 0.00014 |
| Dynamic Agent | 21 | 125 | 0.00055 |



*A plot visualizing the 'Nodes Expanded' metric would show that A* consistently expands the fewest nodes among the optimal algorithms (UCS, A\*), especially on larger, more complex maps. BFS is competitive in node expansion on simple maps but does not guarantee cost optimality.\*

**5. Analysis**

The experimental results highlight the distinct characteristics of each algorithm.

* **BFS vs. UCS**: On the medium.txt map, BFS found a path with a cost of 38, while UCS found a more expensive path with a cost of 50. This is because the BFS path traversed low-cost terrain, which happened to align with the shortest step count. However, this is not a guaranteed outcome. UCS always finds the true lowest-cost path, even if it requires more steps, by correctly accounting for terrain variations.
* *A Performance*\*: Across all static maps, A\* consistently found the optimal path (same cost as UCS) but did so more efficiently. As seen in the results for large.txt, A\* expanded significantly fewer nodes (255) compared to UCS (488). This demonstrates the power of the Manhattan Distance heuristic in guiding the search, reducing wasted effort on less promising paths.
* **Dynamic Simulation**: In the dynamic.txt test, a standard A\* search (ignoring the obstacle's schedule) found a path with a cost of 16. However, this path is invalid as it would collide with the obstacle. The Dynamic Simulation Agent successfully navigated the environment by waiting and replanning. This resulted in a higher but valid path cost (21) and a greater number of total nodes expanded (125) due to the repeated planning cycles. This demonstrates the trade-off required for adaptability in a changing environment.

**6. Conclusion**

This project successfully implemented and evaluated a suite of pathfinding algorithms. The results confirm established theoretical principles: A\* search provides the best balance of optimality and efficiency for static environments with known heuristics. UCS is a reliable baseline for guaranteeing optimality, while BFS is effective only when path cost is irrelevant.

For dynamic environments, a reactive approach where the agent continually replans its path is a viable strategy. The implemented dynamic agent demonstrated its ability to adapt to moving obstacles, albeit at a higher computational and path cost. Future work could explore more advanced algorithms designed for dynamic environments, such as D\* Lite, which can update paths more efficiently without replanning from scratch.