
ARTIFICIAL INTELLIGENCE
FIND THE SHORTEST ROUTE USING THE A*
ALGORITHM ON A REAL-WORLD MAP
Explore Now

MEET OUR TEAM

Nguyen Tien Anh

(Leader)

23070798

Nguyen Trung Kien

(Member)

23070353

Nuyen Nhat Minh

(Member)

23070461

Do Phuong Thuy

(Member)

23070314

OBJECTIVE

To develop an interactive simulation system that visualizes the A search algorithm* using live geographical data from OpenStreetMap (OSM).

THE POWER OF A

- Combines the optimality of Dijkstra with the speed of Greedy Best-First Search.
- Uses the evaluation function: $f(n) = g(n) + h(n)$.
- $g(n)$: Actual cost from start to current node.
- $h(n)$: Heuristic estimate (straight-line distance) to the goal.

REAL-WORLD CONTEXT

1. Logistics & Fleet Management: Optimizing delivery routes to minimize fuel costs and travel time.
2. Autonomous Systems: Enabling self-driving cars and warehouse robots to navigate dynamic environments safely.
3. Games & Virtual Simulations: Guiding NPCs through complex, obstacle-filled maps with realistic movement.
4. Emergency Services: Routing ambulances and fire trucks via the fastest paths to save lives.

TECHNICAL FRAMEWORK & SYSTEM ARCHITECTURE

01

Core Technologies

- Leaflet.js: Open-source library for interactive map rendering and layer management.
- OpenStreetMap (OSM): Extracted real road data (Nodes & Ways) via the Overpass API.
- JavaScript (ES6+): Handles the graph logic, A* execution, and UI interactions.

02

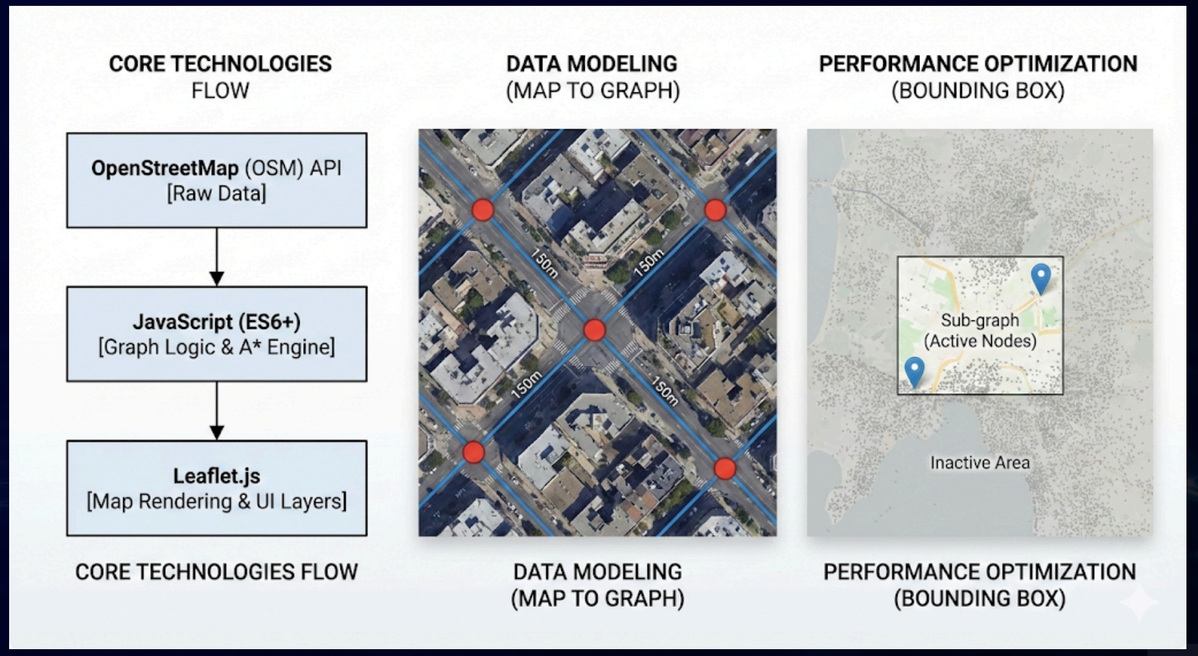
Data Modeling

- Nodes: Intersections or road points (ID, Lat, Lng).
- Edges: Road segments between nodes with weights calculated by geographic distance.

03

Performance Optimization

- Bounding Box (BBox): Filters the search space to a specific area between two user-clicked points.
- Sub-graph Extraction: Dramatically reduces the number of nodes processed, ensuring smooth browser performance.



ALGORITHM IMPLEMENTATION & VISUALIZATION

THE VISUALIZATION LOGIC

- Frontier (Yellow): Nodes currently in the priority queue (Open Set) awaiting expansion.
- Visited (Orange): Nodes already processed and expanded (Closed Set).
- Optimal Path (Blue): The final shortest route generated after reaching the goal.

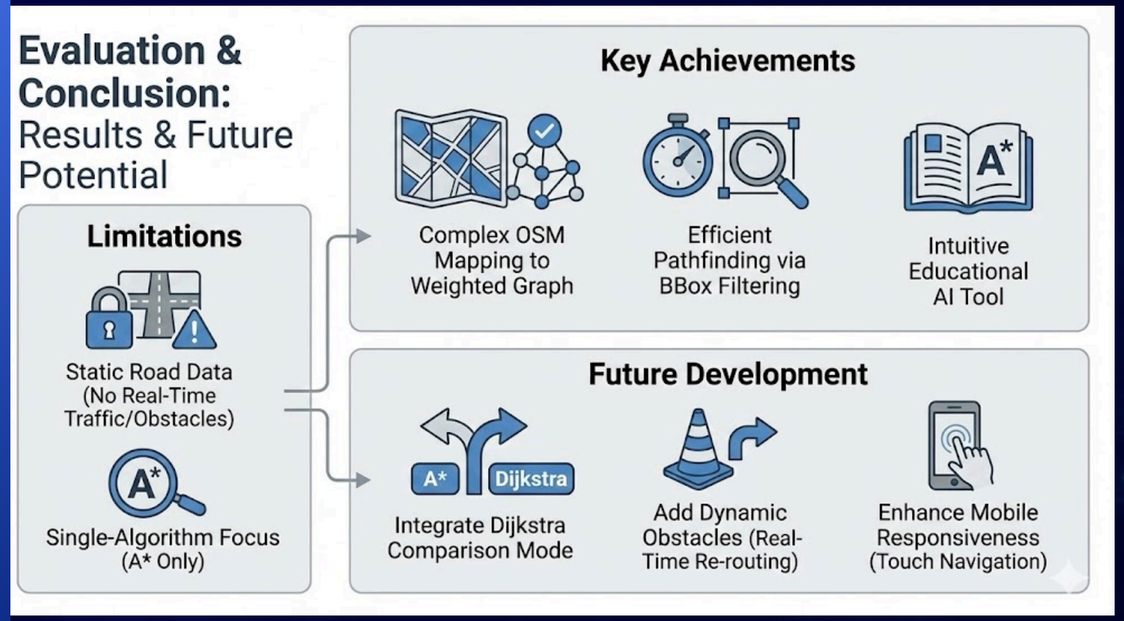
ANIMATION SYNCHRONIZATION

- runId Mechanism: A unique ID for each search session to cancel old animations if a user clicks new points rapidly.
- Sequential Rendering: Uses setTimeout with incremental delays to simulate the "wave-front" expansion of the algorithm.

INTERACTIVE UI

Users can select any two arbitrary points on the map, and the system snaps them to the Nearest Node in the graph.

RESULTS & FUTURE POTENTIAL



01

Key Achievements

- Successfully mapped a complex OSM dataset into a functional weighted graph.
- Achieved efficient pathfinding even on large-scale maps using BBox filtering.
- Created an intuitive educational tool for understanding AI pathfinding.

02

Limitations

- Currently limited to static road data (no real-time traffic or dynamic obstacles).
- Single-algorithm focus (A* only).

03

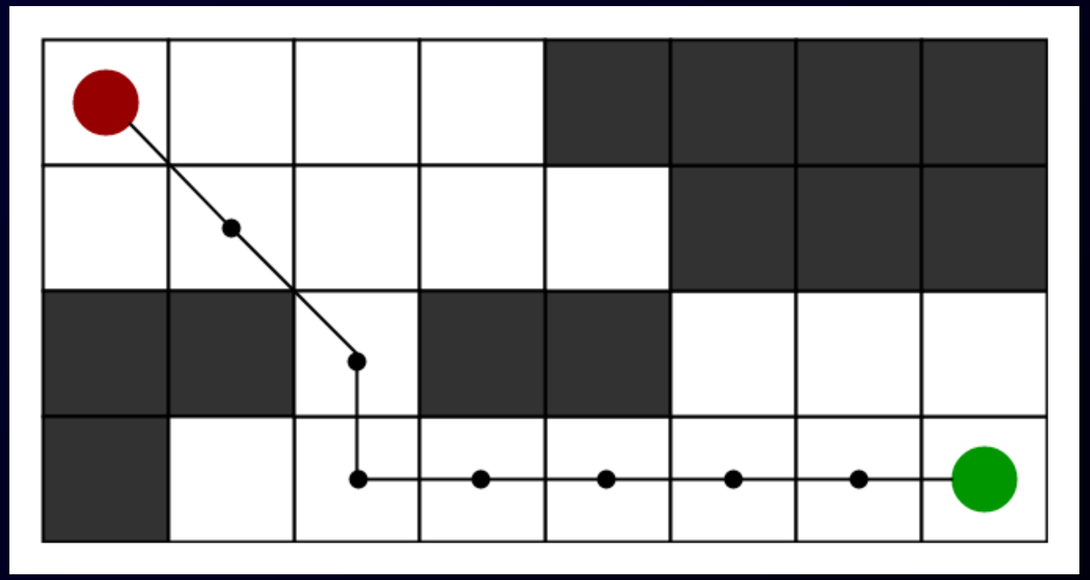
Future Development

- Integrate Dijkstra comparison mode to show efficiency differences.
- Add dynamic obstacles (construction, roadblocks) for real-time re-routing.
- Enhance Mobile Responsiveness for touch-based navigation.

A* ALGORITHM OVERVIEW

The A* algorithm is one of the most optimal and efficient shortest-path search algorithms, widely applied in:

- Mapping systems,
- Autonomous robotics,
- Game AI,
- Intelligent navigation systems.



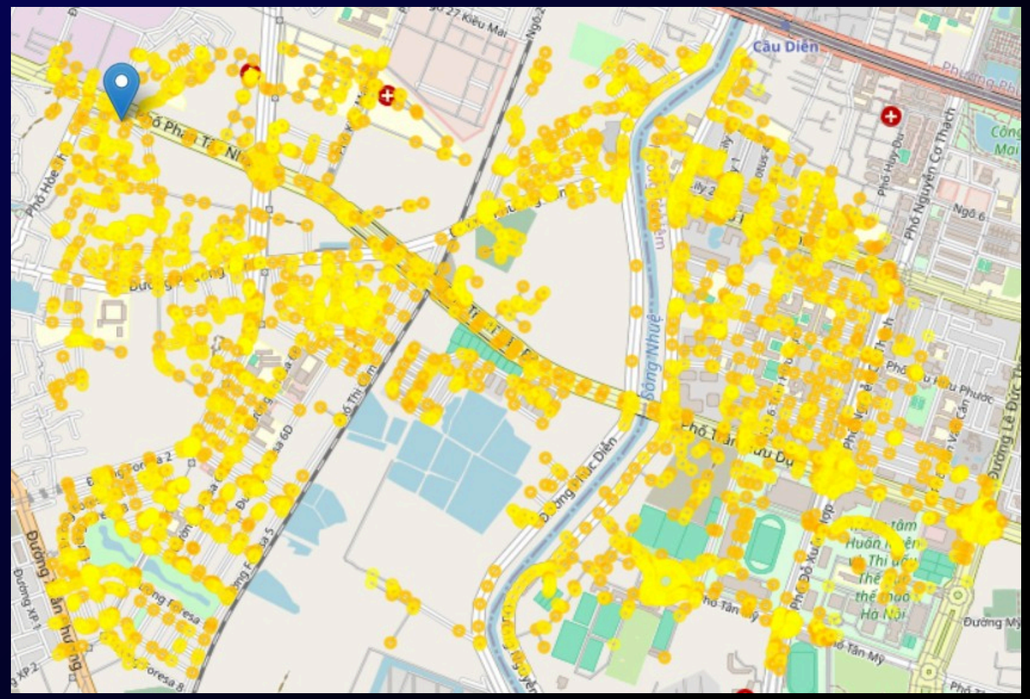
Core evaluation formula

$$f(n) = g(n) + h(n)$$

- g(n): Actual cost from the starting point to the current node
- h(n): Estimated (heuristic) cost from the current node to the goal

  → If h(n) is good (never wish too high), A* will find the shortest and fastest path.

A ALGORITHM OVERVIEW*



Reconstruct path using **cameFrom** map

(Track parent nodes to build path backwards from goal to start)

Visited nodes (expanded) → **Orange**

Frontier (open set) (awaiting expansion) → **yellow**

SYSTEM DESIGN

Main modules

- buildGraph(): Build graph from OSM data (nodes + bidirectional edges)
- aStarTrace(): Execute A* with tracing visited/frontier
- subGraphFromBBox(): Create subgraph limited to bounding box

Speed optimization with Bounding Box

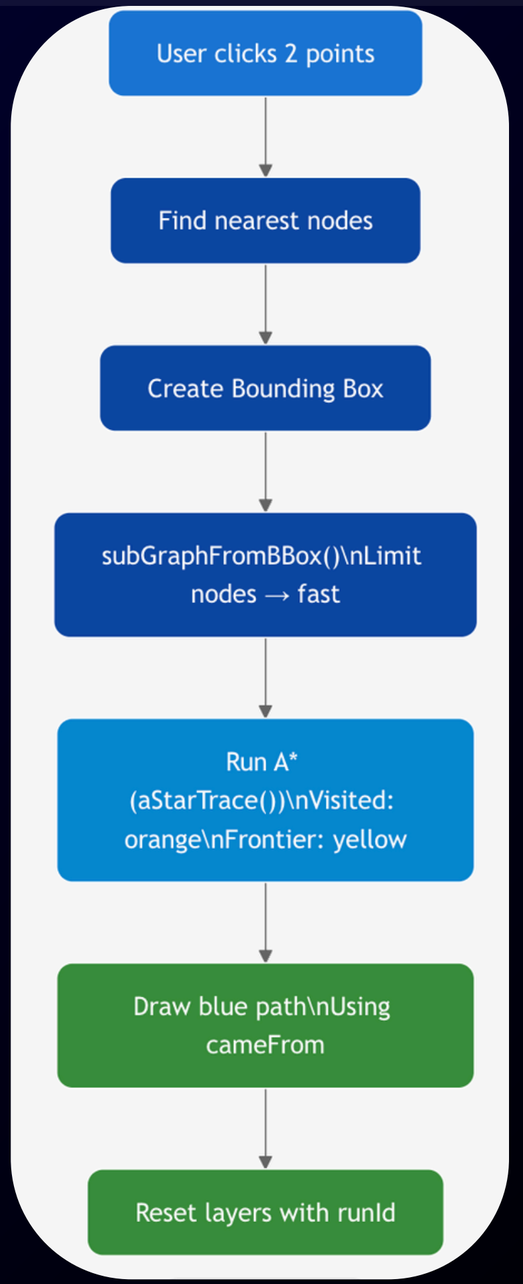
- Limit search area around the 2 clicked points
- Reduce nodes from thousands to hundreds → avoid lag

User interaction

- Click 2 points on the map → automatically find nearest nodes
- Trigger A* → display animation (visited orange, frontier yellow)
- Draw final path in blue after animation

Modular design

- Separate layers (visited, frontier, path, marker) → easy to clear & reset
- runId to cancel old animations when clicking quickly



ANIMATION & UI ARCHITECTURE

Layer Management (Leaflet.js)

- Separated *L.LayerGroup* for distinct visual elements:

visitedLayer (Orange):
Processed nodes

frontierLayer (Yellow):
Nodes in priority queue

pathLayer (Blue):
Final shortest path

- **Benefit**: Modular control, easy to clearLayers() on reset.

Asynchronous Animation:

- Uses *setTimeout* with incremental delay (i * 10ms) to simulate "wave-like" expansion.

```
visited.forEach((id, i) => {
  const n = subGraph.nodes.get(id);
  setTimeout(() => {
    if (runId !== currentRun) return;
    L.circleMarker([n.lat, n.lon], {
      radius: 3, color: "orange", opacity: 0.5
    }).addTo(visitedLayer);
  }, i * 10);
});
```

Concurrency Control (*runId*):

- Problem: Rapid user clicks cause animation conflicts.
- Solution: A unique *runId* increments on every click.
- Logic: *if (runId !== currentRun) return;*

→ Immediately cancels outdated animations.

EXPERIMENTAL RESULTS

Workflow Visualization:

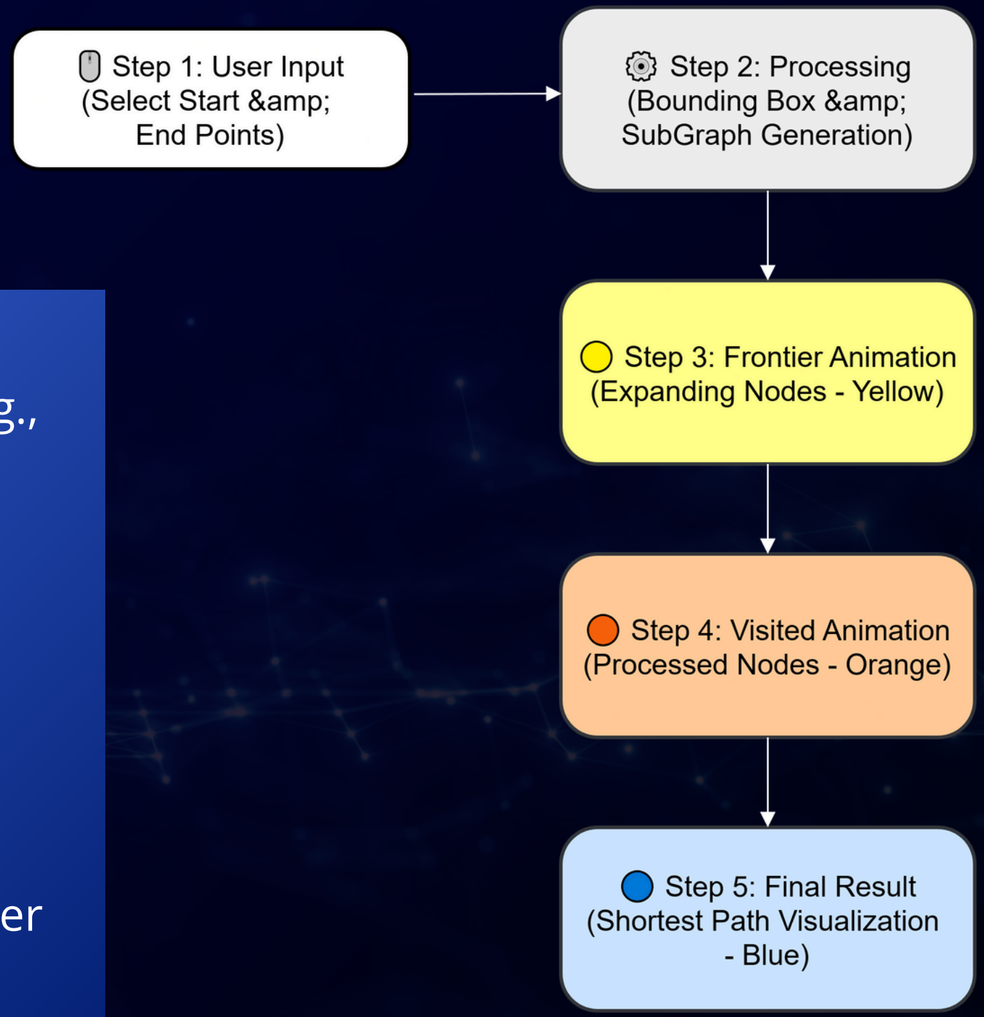
1. Input: User selects 2 points anywhere on the map (e.g., Nam Tu Liem district).
2. Processing:

- Calculates Bounding Box.
- Finds nearest graph nodes.

3. Execution:

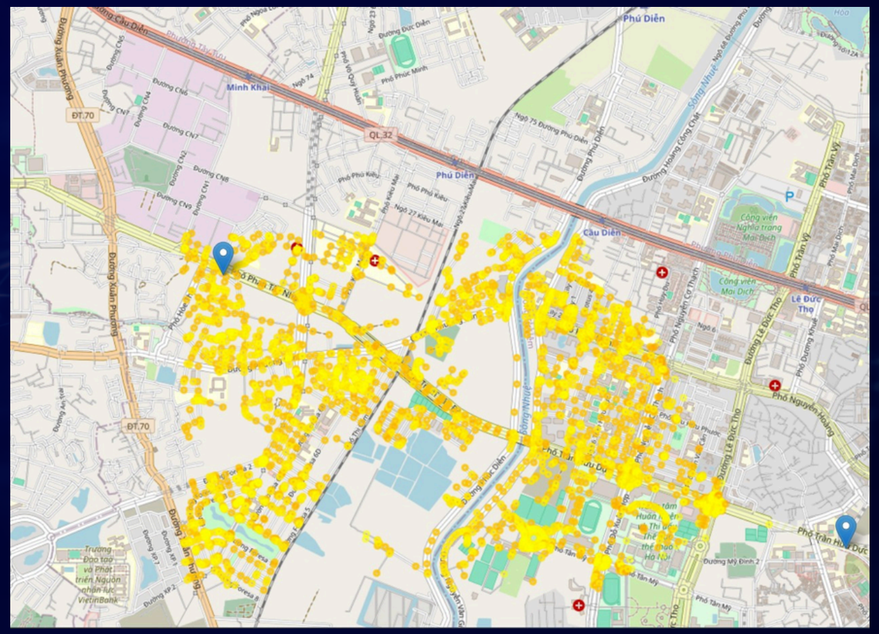
- Frontier (Yellow): Expands towards the target.
- Visited (Orange): Marks explored areas.

4. Output: The shortest path (Blue polyline) is drawn after search completion.

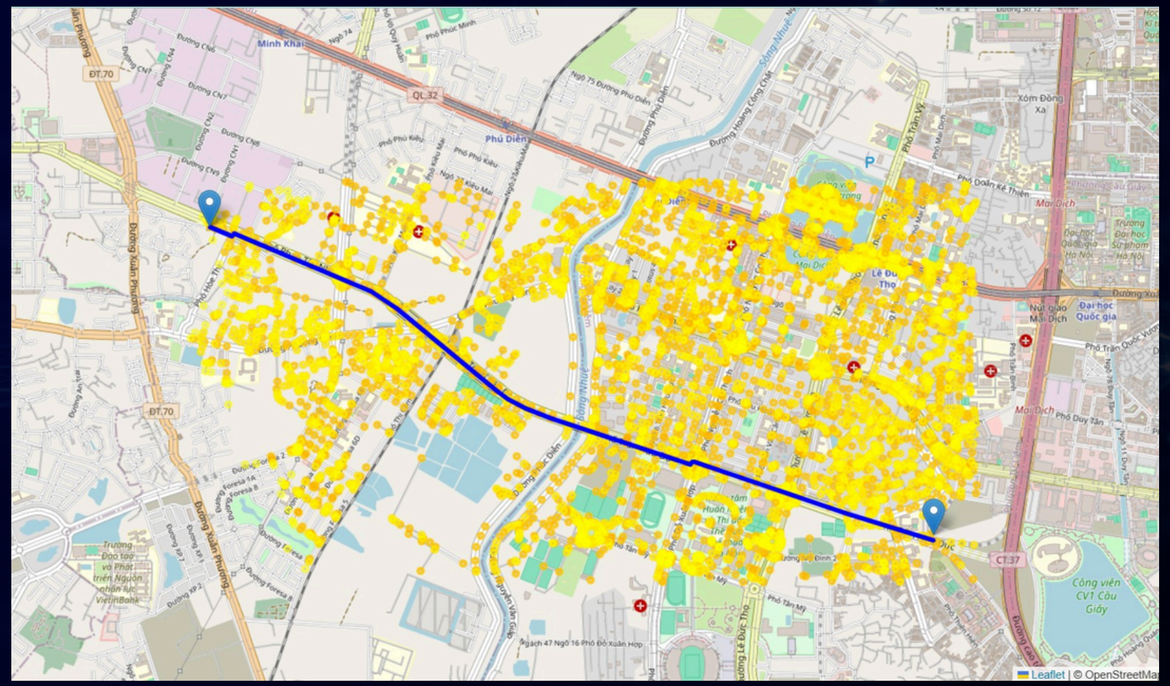


PERFORMANCE

- Smooth rendering on browser.
- Fast computation due to Subgraph extraction (Bounding Box).



EXPERIMENTAL RESULTS



EVALUATION & FUTURE WORK

Strengths:

- Educational: Visually demonstrates how A* works (f(n) = g(n) + h(n)) via color coding.
- Optimized: Efficient subgraph extraction prevents browser lag.
- Robust UI: Handles race conditions effectively.

Limitations:

- Single Algorithm: Lack of comparison with Dijkstra or BFS.
- Static Graph: Does not account for dynamic obstacles (traffic jams, construction).
- Platform: Not fully optimized for mobile touch interfaces.

Future Development:

- Integrate Dijkstra mode for performance comparison.
- Add Dynamic Obstacles (allow users to block roads).
- Implement Responsive Design for mobile devices.

THANK YOU