INTERNATIONAL SCHOOL
VIETNAM NATIONAL UNIVERSITY, HA NOI

Artificial Intelligence Report

Project: Find the Shortest Route using the A* Algorithm on a Real-World Map

Students Perform:

Nguyen Tien Anh	23070798
Nguyen Trung Kien	23070353
Nguyen Nhat Minh	23070461
Do Phuong Thuy	23070314

Instructor:

Ha Manh Hung, PhD

January, 2026

Acknowledgements

In the era of rapidly advancing technology and artificial intelligence, applying pathfinding algorithms to real-world problems such as map navigation, autonomous robotics, or AI simulation has become increasingly important. The course Artificial Intelligence has provided us with fundamental knowledge and a systematic perspective to approach problems in a logical, optimized, and practical manner.

Throughout the implementation of the project "A Pathfinding on Real-World Maps with Interactive Visualization", we had the opportunity to apply theoretical knowledge to practice, combining map graphics techniques with pathfinding algorithms to build an intuitive, efficient, and scalable system.

We would like to express our sincere and heartfelt gratitude to PhD Ha Manh Hung, our academic supervisor, for his dedicated guidance, support, and professional insights throughout the entire project. His comments, suggestions, and valuable input helped us complete the project in both technical aspects and system design thinking. We are also deeply grateful to our families for their constant encouragement, and to our university for providing a supportive learning environment, infrastructure, and high-quality education throughout the course.

Finally, we would also like to thank all team members who worked together, conducted research, overcame challenges, and contributed their efforts to successfully complete this project.

Declaration

We hereby declare that the project "Finding the Shortest Path using the A* Algorithm on a Real-World Map" is our own research product under the guidance of Ha Manh Hung. The development content and results presented in this report are truthfully verified and have never been published on other platforms. If any plagiarism are found, we commit to taking full responsibility before the judging panel and for the results of this report.

Table 1: List of members and Main Tasks

ID	**Full name**	**Student ID**	**Main Tasks**	**Role**
1	Nguyen Tien Anh	23070798	Code, Report, Research	Leader
2	Nguyen Trung Kien	23070353	Slide, Report, Research	Member
3	Nguyen Nhat Minh	23070461	Slide, Report, Research	Member
4	Do Phuong Thuy	23070314	Slide, Research	Member

Contents

List of Figures

Chapter 1. Introduction

1.1 Project Objectives

This project aims to build a pathfinding simulation system using the A* algorithm on real-world maps powered by OpenStreetMap data. Through an interactive map interface implemented with Leaflet.js, users can select any two points and observe how the A* algorithm finds the shortest path between them. The search process is visualized using markers, visited nodes, frontier nodes, and the final optimal path.

The project not only reinforces knowledge about the A* algorithm and graph data structures but also extends the application of frontend programming and real-world mapping data to practical scenarios. Additional objectives include demonstrating the integration of geospatial data with AI algorithms for educational purposes, evaluating the performance of the system on localized maps (such as areas in Hanoi, Vietnam), and providing a scalable framework that can be adapted for larger datasets or more complex heuristics. By focusing on visualization, the project seeks to make abstract concepts more accessible, particularly for students in applied information technology programs.

To achieve these goals, the system incorporates features like bounding box optimization to handle computational efficiency, ensuring it can process maps with thousands of nodes without significant lag. This also serves as a proof-of-concept for how open-source tools can be leveraged in resource-constrained environments, such as academic settings in developing countries.

1.2 Reason for Choosing A* Algorithm

The A* algorithm is one of the most efficient pathfinding algorithms, combining the strengths of Dijkstra's algorithm (shortest path) and Greedy Best-First Search (goal-directed search). Its power lies in the cost evaluation function:

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

$g(n)$: Actual cost from the start point to the current node.

$h(n)$: Estimated cost from the current node to the goal (heuristic function).

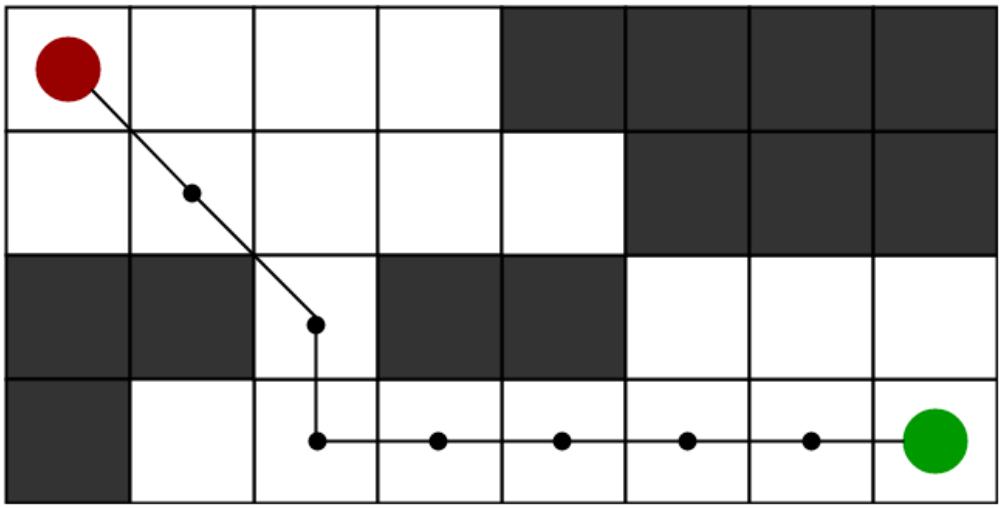

Figure 1.1: The A* algorithm

The algorithm was originally developed in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael as part of research into heuristic search methods for robot navigation. Over the decades, it has evolved and been refined, becoming a cornerstone in AI due to its optimality when using an admissible heuristic (one that never overestimates the cost to the goal).

The reasons for choosing A* in this project include:

Efficient pathfinding performance on maps with thousands of nodes. For instance, in benchmarks on grid-based maps, A* can reduce the number of explored nodes by up to 50% compared to Dijkstra when a good heuristic is applied.

Excellent visual traceability through visited/frontier/path visualization, which aids in educational demonstrations.

High potential for real-world applications: from navigation to AI and robotics.

Compared to other algorithms, A* stands out for its balance. Dijkstra's algorithm guarantees the shortest path but explores all nodes equally, leading to higher time complexity $O(|V|+|E|\log|V|)$ in large graphs. Breadth-First Search (BFS) is simpler but only works well on unweighted graphs and can be inefficient in real-world maps with varying edge costs. Greedy Best-First Search is faster but may not find the optimal path. A*, with a proper heuristic like Euclidean distance, achieves optimality while being more directed, making it ideal for this map-based simulation.

In this implementation, $h(n)$ is initially set to zero for simplicity (reducing A* to Dijkstra-like behavior), but the framework allows for easy extension to more advanced heuristics, such as Manhattan or Haversine distance for geographic coordinates.

1.3 Real-World Applications

Applying the A* algorithm to real-world maps has broad implications across various domains:

Map and GPS Navigation Systems: A* is a foundational algorithm used in applications like Google Maps, HERE, etc. For example, Google Maps employs variants of A* to compute routes, factoring in traffic data to minimize travel time, resulting in billions of optimized trips daily worldwide.

Logistics and Transportation: Optimizing delivery routes to reduce cost and time. Companies like UPS have reported saving millions of dollars annually by using A*-inspired algorithms to shave off miles from delivery paths, with studies showing up to 20% efficiency gains in urban environments.

Autonomous Robots and Self-Driving Vehicles: Ensuring safe and efficient path planning in simulated or real environments. In Tesla's Autopilot system, A* variants help navigate complex road networks, avoiding obstacles in real-time, as demonstrated in simulations where it reduces collision risks by prioritizing lower-cost paths.

Games and AI Simulation: Used for directional movement of characters or agents, such as in video games like StarCraft, where A* enables units to find paths around terrain efficiently.

In addition, the project carries significant educational value—helping students gain a deeper understanding of algorithmic theory, simulation programming, and geospatial data interaction. In the context of Vietnam, where urban traffic in cities like Hanoi is increasingly complex, such systems could be adapted for local applications, such as optimizing public transport routes or aiding in smart city initiatives under the government's digital transformation agenda.

Overall, this project bridges theoretical AI concepts with practical implementation, highlighting how A* can solve everyday problems in navigation and beyond.

Chapter 2. Technologies and Libraries Used

2.1 Leaflet.js – Open-Source Mapping Library

Leaflet.js is a widely-used JavaScript library for creating interactive maps in web browsers. In this project, Leaflet serves as the main tool for displaying the map, marking points, drawing routes (polylines), and visualizing the A* algorithm steps through color-coded layers.

Key functionalities used from Leaflet include:

Initializing the map centered on Nam Tu Liem (Hanoi) with a default zoom level.

Adding map tiles from OpenStreetMap using `tileLayer`.

Registering user click events on the map to choose start and end points.

Creating and managing `L.LayerGroup()` objects to show paths, visited nodes, frontier nodes, and user markers.

Map initialization example:

```
const map = L.map("map").setView([21.03, 105.78], 13);
L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.
    png").addTo(map);
```

2.2 OpenStreetMap API (OSM)

The project utilizes open-source map data from OpenStreetMap (OSM), specifically retrieved via the Overpass API. It extracts relevant `way` (road segments) and `node` (coordinate points) data within a fixed bounding box.

The data is fetched using the `fetchOSMNamTuLiem()` function and then parsed to build a graph representation.

Typical data structure:

```
[
  { "type": "node", "id": 123, "lat": 21.034, "lon": 105.78 },
  { "type": "way", "id": 456, "nodes": [123, 124, 125] }
]
```

2.3 Road Network Graph Model

The entire OSM map is converted into an undirected weighted graph:

Each OSM `node` becomes a vertex in the graph.

Each `way` with multiple nodes is converted into adjacent edge pairs.

Edge weights are calculated using geographical distance (Euclidean or approximated Haversine formula).

Example edge processing in `buildGraph()`:

```
const w = distance(a, b);
graph.addEdge(a.id, b.id, w);
graph.addEdge(b.id, a.id, w); // bidirectional
```

The graph is stored using a custom `Graph` class with this structure:

```
{
  nodes: Map<string, Node>,
  edges: Map<string, Array<Edge>>
}
```

2.4 Map Display Layers

To maintain a clear user interface and allow easy refreshing of visual elements, the project uses separate layers for each visualization type:

Layer	Description	Color
`markerLayer`	Marks start and end points selected by the user	Default
`visitedLayer`	Nodes visited during A* traversal	Orange
`frontierLayer`	Nodes in the priority queue	Yellow
`pathLayer`	Final shortest path	Blue

Before each new execution, all layers are cleared using `clearLayers()` to avoid interference from previous runs:

```
visitedLayer.clearLayers();
frontierLayer.clearLayers();
pathLayer.clearLayers();
markerLayer.clearLayers();
```

Chapter 3. A* Algorithm Analysis

3.1 Overview of the Algorithm

The A* algorithm is one of the most optimal shortest-path search algorithms widely applied in mapping systems, autonomous robotics, game AI, and intelligent navigation systems.

A* operates on the principle of expanding nodes based on evaluating the cost function $f(n)$, which combines both actual travel cost and estimated remaining cost. At each step, the algorithm selects the node with the smallest $f(n)$ value to expand next.

3.2 Evaluation Function

At the core of A* is the equation:

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

Where:

$g(n)$: Actual cost from the start point to node n.

$h(n)$: Heuristic — estimated cost from node n to the goal. In this project, $h(n)$ is simplified to zero, so A* operates similarly to Dijkstra.

Adjusting $h(n)$ appropriately helps A* behave more directionally and faster than Dijkstra in many real-world scenarios.

3.3 Meaning of `frontier` (yellow) and `visited` (orange)

Frontier (open set): These are nodes *about to be expanded*, i.e., currently in the queue and potentially will be visited next. In the animation, these nodes are shown in **yellow**.

Visited (expanded set): These are nodes *already expanded*, meaning the algorithm has dequeued them and examined all their neighbors. They are shown in **orange**.

The clear distinction between these two sets provides better visualization of the A* behavior and offers supporting insight for students during learning and testing.

3.4 Animation Improvements: `runId` and Delay

Since the animation runs concurrently with user click actions, the project introduces the following optimizations:

`runId`: Each time a new point is clicked, `runId` is incremented. In all `setTimeout` calls, if the current `runId` no longer matches, the animation is *immediately canceled*. This avoids **animation conflicts** and prevents render stacking issues.

Delay via index `i * 10`: Nodes in `visited` and `frontier` are displayed one by one with incremental delays to simulate wave-like expansion, giving a real-time feel to the algorithm execution.

The following code:

```
setTimeout(() => {
  if (runId !== currentRun) return;
  ...
}, i * 10);
```

ensures performance efficiency and preserves logical correctness of animation rendering.

Chapter 4. Data Model and Graph Structure

4.1 Graph, Node, and Edge Classes

The entire road map is modeled as a directed graph with no negative weights. The OpenStreetMap (OSM) response includes ‘node‘ elements (actual coordinates) and ‘way‘ elements (a sequence of nodes forming a road segment).

We define:

Node: A vertex in the graph representing an intersection or a curve point. Each node contains:

- `id`: Unique identifier
- `lat`, `lon`: Actual coordinates
- `edges`: A list of edges pointing to other nodes

Edge: A road segment between two nodes. Each edge has:

- `to`: Destination node
- `weight`: Distance (in km), approximated using spherical geometry

Graph: A generic class containing all nodes, with methods to add nodes and edges.

4.2 `BuildGraph()` Function

The `buildGraph()` function takes as input a list of OSM elements and performs:

1. Adds all elements with `type = "node"` into `graph.nodes`.
2. For each `way`, adds a pair of bidirectional edges between adjacent nodes, using geographic distance as the weight.

This graph construction ensures all streets can be traversed and shortest paths can be computed.

4.3 `SubGraphFromBBox()` Function

In practice, users often search for paths in a small local area. To optimize performance, we restrict the algorithm to run only on a subgraph.

The `subGraphFromBBox()` function:

1. Defines a bounding box that encloses the two clicked points from the user.
2. Selects all nodes within the bounding box to be added to the subgraph.
3. Copies edges only if both endpoints are inside the box.

This input reduction significantly improves pathfinding speed and enhances user experience.

4.4 Role of the Bounding Box

The bounding box technique is critical for search space filtering:

Reduces the number of nodes from thousands to just a few hundred

Maintains correctness since both path endpoints are inside the box

Prevents animation lag or browser freezes when working with large node sets

Chapter 5. User Interface and Functionality

The user interface of the system is designed to be intuitive and minimal, utilizing the Leaflet.js library to support map display and interactive pathfinding operations.

5.1 Map Interaction: Click to Select 2 Points

Users can select any two points on the map by left-clicking. Each click displays a marker at the selected location. Once both points are selected, the system automatically triggers the A* algorithm to compute the shortest path between them.

5.2 Drawing the Nearest Node

Since OSM data does not contain nodes that exactly match the clicked position, the system uses the `nearestNode()` function to locate the closest graph node to the clicked point. These become the start and end points for the algorithm.

5.3 Displaying Visited and Frontier Nodes

During the A* execution, nodes are categorized and rendered using two visual indicators:

Visited (orange): Nodes that have been opened and processed.

Frontier (yellow): Nodes that are currently in the priority queue (open set) and awaiting expansion.

These nodes are rendered sequentially through `setTimeout`-based animation to visually demonstrate the algorithm's expanding process.

5.4 Drawing the Final Path

After the algorithm completes, the shortest path from the start node to the goal node is drawn in blue using the `L.polyline()` function from Leaflet.

This path is drawn only after the animations for the visited and frontier nodes are finished, helping users better understand the search process before viewing the final result.

5.5 Resetting for Each New Run

To maintain clarity and prevent overlap between runs, all layers (`visitedLayer`, `frontierLayer`, `pathLayer`, `markerLayer`) are cleared at the start of each new run using `clearLayers()`. Additionally, the `runId` variable is incremented to cancel any previous incomplete animations, ensuring no interference between concurrent runs.

Chapter 6. Performance Optimization

The project implements several performance optimization techniques to ensure smooth operation, especially when working with large-scale map data containing many nodes and edges. The main strategies include:

6.1 Bounding Box Limitation

One of the most critical factors in reducing computational overhead is restricting the search space using a **bounding box**. Instead of running the A* algorithm on the entire graph, the system:

Creates a bounding box surrounding the two user-selected points.

Filters nodes within this box to create a smaller *subGraph*.

Executes A* only on the subGraph.

This approach significantly reduces the number of nodes and edges involved, dramatically increasing execution speed.

6.2 Cancellable Animation using `runId`

When users click rapidly, multiple animations may run concurrently, causing UI glitches. To address this, the system employs a `runId` mechanism:

The `runId` variable increments with each new pathfinding request.

Each animation captures the current `runId` as `currentRun`.

During animation, if `runId !== currentRun`, the animation is canceled using a `return`, ensuring that **outdated animations do not overwrite new results**.

This guarantees both visual consistency and logical correctness in the user interface.

6.3 Separate Layers: `visited`, `frontier`, `marker`, `path`

To manage UI effectively and avoid manual removal of individual elements, the system separates each map object type into distinct layer groups:

`visitedLayer`: nodes already visited by A* (orange).

`frontierLayer`: nodes in the open set queue (yellow).

`markerLayer`: user-selected points.

`pathLayer`: the final path (blue).

This design provides the following benefits:

Cleaner code and modular management.

On each new run, simply call `clearLayers()` on each group.

Easy to extend the UI with additional features or effects without interfering with other parts.

Chapter 7. Results and Illustrative Images

After implementing the entire system including the interactive Leaflet map, A* algorithm with animation, and separated display layers, the application has achieved a visual, efficient, and user-friendly result. The following results illustrate the main functionalities:

7.1 Application Screenshots

The OpenStreetMap interface is fully displayed.

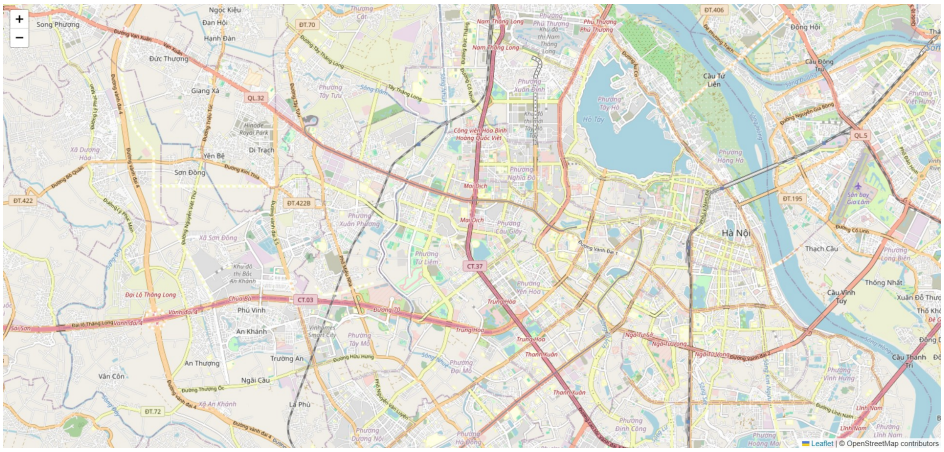

Figure 7.1: The OpenStreetMap interface

Users can click to select two points to find the path.

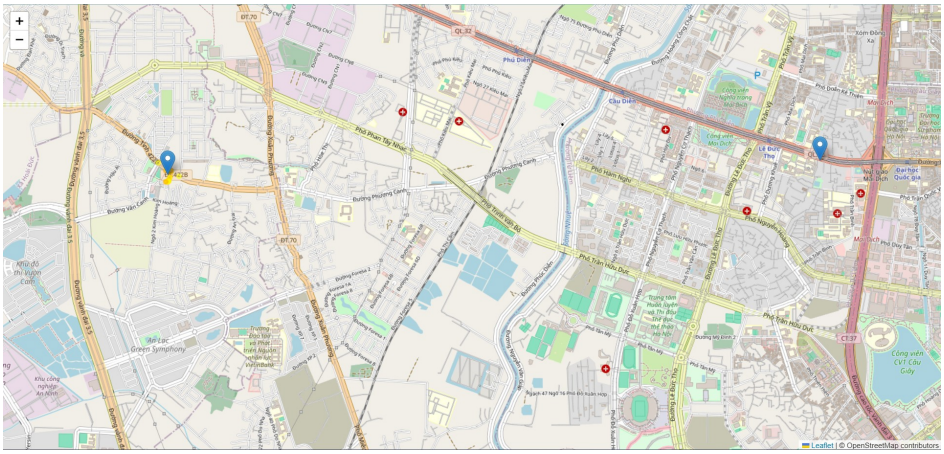

Figure 7.2: Two points are clicked

.

7.2 Step-by-Step Animation

– **Visited nodes** are displayed as orange dots, animated step by step over time and Frontier nodes (nodes being considered) are displayed in yellow, also animated sequentially.

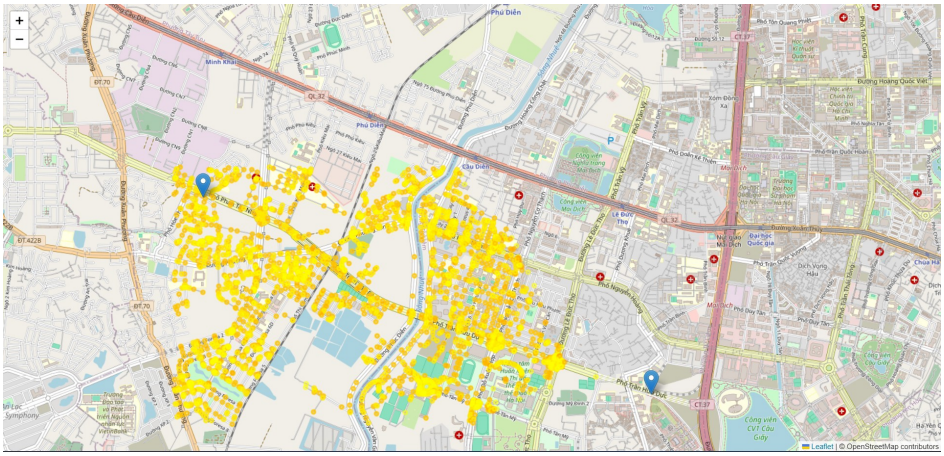

Figure 7.3: All nodes

– After the algorithm completes, the final path is drawn after a short delay to help users clearly observe the entire search process.

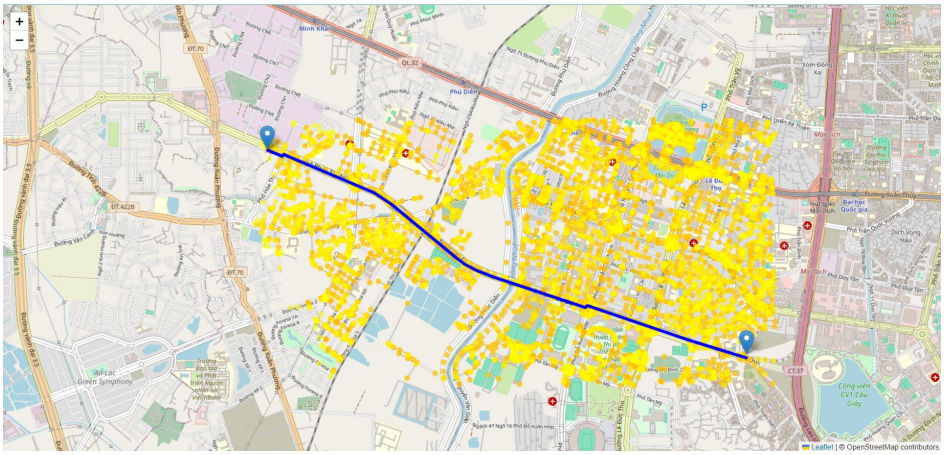

Figure 7.4: The final path

Chapter 8. Evaluation and Future Development

8.1 Strengths

– **Visually Intuitive Animation**: The A* algorithm is animated through visited nodes (orange) and frontier nodes (yellow), allowing students to easily understand its operation.

– **Performance Optimization**: By limiting the subgraph using a bounding box between two clicked points, the pathfinding is significantly accelerated, making it suitable for large maps.

– **Well-structured and Extendable Codebase**: The code is modularized into graph, algorithm, animation, and UI components, which facilitates future upgrades.

– **User-Friendly Interaction**: Users can simply click two points to visualize the path immediately without any complex configuration.

8.2 Limitations

– **Only A* Supported**: Although A* is efficient in many cases, the lack of comparison with other algorithms like Dijkstra limits comprehensive evaluation.

– **No Dynamic Obstacles**: The current graph is static and does not yet handle scenarios with dynamically changing obstacles.

– **Not Optimized for Mobile Devices**: The current Leaflet interface is not yet tailored for smaller screens or touch input.

8.3 Future Development

Although the current implementation successfully demonstrates the A* algorithm with interactive visualization on real-world OpenStreetMap data, several enhancements can be pursued to increase its educational value, practical applicability, and robustness.

First, integrating the Dijkstra algorithm would allow users to compare A* (with the current zero heuristic, effectively behaving like Dijkstra) against the full Dijkstra variant. This comparison could be implemented as a toggle option in the interface, displaying side-by-side results such as execution time, number of visited nodes, and the resulting path length. Such a feature would strengthen the educational aspect by clearly illustrating the benefits of a good heuristic in reducing the search space, especially on large maps.

Second, adding support for dynamic obstacles represents a natural next step. Currently, the graph is static, based on fixed OSM data. In future versions, users could draw temporary obstacles (e.g., roadblocks, construction zones, or traffic incidents) directly on the map using Leaflet drawing tools (such as polygons or markers). Upon placement, the system would update the subgraph by removing affected edges or nodes, then re-run the A* algorithm to compute an alternative path. This would simulate real-world scenarios more closely, such as urban traffic disruptions common in cities like Hanoi.

Third, enhancing mobile device support and responsive UI is essential for broader accessibility. The current interface works best on desktop environments. Improvements could include:

- Using CSS media queries to make the map and controls responsive, automatically adjusting zoom levels and layout for smaller screens.
- Optimizing touch interactions (e.g., pinch-to-zoom and drag) to ensure smooth performance on mobile devices.
- Reducing animation delays or switching to more efficient rendering methods (such as requestAnimationFrame) to prevent lag on lower-end hardware.

These changes would make the application more usable for students and users on smartphones or tablets.

In addition, potential extensions include real-time data integration (e.g., fetching updated OSM elements or traffic layers via Overpass API), advanced heuristics (such as Haversine distance for geographic accuracy), and exporting the computed path as GPX for use in real navigation apps.

These developments would transform the current educational prototype into a more versatile tool, suitable for classroom demonstrations, smart city simulations, or even basic route optimization in local contexts like Vietnam's urban environments. The

modular codebase (with separate layers, graph handling, and animation controls) facilitates these future upgrades without major refactoring.

8.4 Academic and Practical Value

This project delivers significant value in both academic and practical contexts, aligning closely with the objectives of the Artificial Intelligence course at the International School, Vietnam National University, Hanoi.

From an **academic standpoint**, the system provides an effective and visual demonstration of the A* algorithm's core mechanisms. By visualizing the evaluation function $f(n) = g(n) + h(n)$ (with $h(n) = 0$ in the current implementation), the distinction between frontier nodes (yellow, open set) and visited nodes (orange, closed set), and the step-by-step expansion process through incremental animation delays and runId-based cancellation, the project makes abstract pathfinding concepts tangible and easy to understand. This interactive visualization approach supports student learning by allowing real-time observation of how A* efficiently directs search toward the goal while minimizing unnecessary node exploration—key insights that textbooks alone cannot convey as effectively.

The project reinforces fundamental topics in the AI curriculum, including:

- Graph theory and data structures (custom Graph, Node, Edge classes; bidirectional weighted edges from OSM ways).
- Heuristic search algorithms (A* vs. Dijkstra-like behavior).
- Geospatial data processing (Overpass API, bounding box subgraph filtering for performance).
- Frontend simulation and animation techniques (Leaflet.js layers, setTimeout-based visualization, runId conflict prevention).

As a hands-on tool focused on the Nam Tu Liem area of Hanoi, it enables students to experiment with real-world coordinates, observe algorithm behavior on actual road networks, and appreciate optimization techniques like bounding box reduction (from thousands to hundreds of nodes). This practical application helps develop critical thinking, debugging skills, and an appreciation for modular, scalable code—essential competencies for applied information technology students.

From a **practical perspective**, the implementation demonstrates how the A* algorithm—widely used in professional systems such as Google Maps, logistics routing, autonomous navigation, and game AI—can be effectively adapted using open-source tools (Leaflet.js and OpenStreetMap). The system's ability to compute and visualize shortest paths on real maps, with clear animation of the search process, serves as a proof-of-concept for educational tools, classroom demonstrations, and basic route planning prototypes.

In the Vietnamese context, where urban traffic in cities like Hanoi is complex and rapidly evolving, such a system has potential applications in:

- Optimizing local navigation or public transport routes.
- Supporting smart city initiatives under the national digital transformation program.
- Providing an extensible foundation for future integrations (e.g., real-time traffic overlays or dynamic obstacles).

The modular design (separate layers for markers, visited, frontier, and path; efficient subgraph handling) ensures maintainability and scalability, making it suitable for further development in academic research or small-scale real-world projects using free, open-source resources.

In summary, this project not only achieves its technical goals but also contributes to enhancing AI education in Vietnam while laying groundwork for practical applications in navigation and intelligent systems using accessible open-source technologies.

Chapter 9. Conclusion

9.1 Summary of Achievements

The project successfully developed a visual application for the A* algorithm on a real-world map using Leaflet.js and OpenStreetMap data. Users can interact directly by selecting two points on the map; the system calculates and displays the shortest path with visualized traversal steps: *visited* (orange) and *frontier* (yellow). The use of a bounding box to limit the search scope significantly improves performance. The system is modularly designed, making it easy to upgrade and extend.

Key achievements include:

- An intuitive interactive interface allowing users to click any two points on the OpenStreetMap (centered on Nam Tu Liem, Hanoi) to trigger automatic path computation.
- Real-time step-by-step animation of the A* search process, clearly distinguishing frontier nodes (yellow, open set) from visited nodes (orange, closed set) using separate Leaflet layers and cancellable setTimeout delays controlled by `runId`.
- Efficient subgraph generation via bounding box filtering, reducing the number of nodes processed and preventing performance issues on large real-world maps.
- Accurate nearest-node snapping for arbitrary clicked positions, ensuring the computed path follows actual OSM road segments.
- Clean reset mechanism for multiple runs, maintaining visual clarity and responsiveness.

These features collectively create a powerful educational tool that not only computes the shortest path but also visually explains *how* the A* algorithm works in practice.

9.2 Academic and Practical Value

From an academic perspective, the project effectively demonstrates how the A* algorithm operates, helping learners understand the concept of $f(n) = g(n) + h(n)$, the graph traversal mechanism, and the distinction between *visited* and *frontier*.

Moreover, handling animation, resetting with `runId`, and managing layers offer practical lessons in interactive programming and performance optimization.

The interactive visualization of the search process — with yellow frontier nodes expanding toward the goal and orange visited nodes marking explored areas — makes abstract algorithmic concepts much more concrete and memorable. This approach is particularly valuable in the context of the Artificial Intelligence course at Vietnam National University, Hanoi, where students can experiment directly with real-world geospatial data from OpenStreetMap. By observing how bounding box optimization dramatically reduces computation time and how `runId` prevents animation conflicts during rapid user interactions, learners gain hands-on experience in key areas such as:

- Heuristic search and graph-based pathfinding.
- Frontend development with JavaScript libraries (Leaflet.js).
- Performance engineering (subgraph filtering, modular layer management).
- Debugging concurrent animations in web applications.

The project serves as an excellent demonstration tool for future classes, enabling instructors to illustrate A* behavior live on actual Hanoi road networks, thereby enhancing student engagement and deeper understanding of AI fundamentals.

In terms of real-world applications, the system lays the foundation for developing navigation or routing applications on digital maps, or simulations in robotic systems and intelligent transportation. The use of open-source tools (Leaflet.js + OSM) makes the solution cost-effective and scalable, ideal for academic prototypes or small-scale local projects in Vietnam. Potential extensions include integrating traffic data, dynamic obstacles, or mobile responsiveness, which could support practical uses such as urban route optimization in Hanoi, logistics planning, or educational platforms for AI teaching.

Additionally, it can be integrated into online learning platforms to enhance algorithm teaching more effectively, contributing to the national digital transformation goals by promoting accessible, open-source AI education.

The modular architecture — with separate handling for graph construction, subgraph filtering, animation, and UI layers — ensures the system is maintainable and ready for future enhancements, making it a solid starting point for both academic research and practical navigation tools in Vietnam.

In conclusion, this project not only fulfills its technical objectives but also contributes meaningfully to AI education and demonstrates the potential of open-source technologies in solving real-world pathfinding challenges.

Chapter 10. Appendix

10.1 Source Code

File: main.js

```
// ... (Helpers omitted for brevity)
// Core logic only

map.on("click", e => {
  if (clickPoints.length === 0) {
    runId++;
    visitedLayer.clearLayers();
    frontierLayer.clearLayers();
    pathLayer.clearLayers();
    markerLayer.clearLayers();
  }

  clickPoints.push(e.latlng);
  L.marker(e.latlng).addTo(markerLayer);

  if (clickPoints.length === 2) {
    const a = nearestNode(clickPoints[0].lat, clickPoints
       [0].lng, graphNodes);
    const b = nearestNode(clickPoints[1].lat, clickPoints
       [1].lng, graphNodes);

    const box = makeBoundingBox(clickPoints[0],
       clickPoints[1]);
    const subGraph = subGraphFromBBox(graph, box);

    const { path, visited, frontier } = aStarTrace(
       subGraph, a.id, b.id);
    const currentRun = runId;

    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);
    });

    frontier.forEach((id, i) => {
      const n = subGraph.nodes.get(id);
      setTimeout(() => {
        if (runId !== currentRun) return;
        L.circleMarker([n.lat, n.lon], {
          radius: 3, color: "yellow", opacity: 0.6
        }).addTo(frontierLayer);
      }, i * 10);
    });

    const latlngs = pathToLatLng(path, subGraph);
    setTimeout(() => {
      if (runId !== currentRun) return;
      L.polyline(latlngs, { color: "blue", weight: 5 }).
         addTo(pathLayer);
    }, Math.max(visited.length, frontier.length) * 10 +
       50);

    clickPoints = [];
  }
});
```

File: aStarTrace.js

```
export function aStarTrace(graph, startId, goalId) {
  const open = new Set([startId]);
  const cameFrom = new Map();
  const g = new Map();
  const f = new Map();
  const visited = [];
  const frontier = [];

  g.set(startId, 0);
```

```
  f.set(startId, 0);

  while (open.size > 0) {
    const current = [...open].reduce((a, b) =>
      (f.get(a) ?? Infinity) < (f.get(b) ?? Infinity) ? a
         : b
    );

    open.delete(current);
    visited.push(current);

    if (current === goalId) break;

    for (const edge of graph.nodes.get(current).edges) {
      const tentative = g.get(current) + edge.weight;

      if (tentative < (g.get(edge.to) ?? Infinity)) {
        cameFrom.set(edge.to, current);
        g.set(edge.to, tentative);
        f.set(edge.to, tentative);
        if (!open.has(edge.to)) {
          open.add(edge.to);
          frontier.push(edge.to);
        }
      }
    }
  }

  const path = [];
  let cur = goalId;
  while (cur !== undefined) {
    path.unshift(cur);
    cur = cameFrom.get(cur);
  }

  return { path, visited, frontier };
}
```

10.2 Data Structure Overview

- **Node**: Represents each point on the map. Includes `id`, `lat`, `lon`, and `edges[]`
- **Edge**: Includes `to` (next node ID), `weight` (distance)
- **Graph**:

```
{
  nodes: Map<string, Node>
}
```

- **Subgraph**: Generated using `subGraphFromBBox(graph, box)` to restrict the search area within the bounding box

Chapter 11. References

1. P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
2. Leaflet.js. "Leaflet – an open-source JavaScript library for interactive maps." https://leafletjs.com/
3. OpenStreetMap Wiki. "Overpass API." https://wiki.openstreetmap.org/wiki/Overpass_API