**Assignment: Autonomous Battery-Operated Micro Aquatic Boat**

BITS IDs of all team members:

|  | **BITS ID** | **Name** | **Contribution** |
| --- | --- | --- | --- |
| 1 | 2025AE05517 | NIMISHAMBA S | 100% |
| 2 | 2025AE05518 | JITENDRA KR TIWARI | 100% |
| 3 | 2025AE05519 | DADDEKAR ROHIT TUKARAM VAISHALI | 100% |
| 4 | 2025AE05520 | RASMITA JENA | 100% |
| 5 | 2025AE05521 | C. AJAY KUMAR | 100% |

**Section 1: The Autonomous Aquatic Boat as an Intelligent Agent**

The autonomous agent is a micro aquatic boat to be used for disaster relief in Chennai. This would need a formal definition of agent, its purpose, its capabilities and contextual operation. This can be done via the standard PEAS framework by classifying the task environment that underlines the problem space and eventually helps design a rational agent.

**1.1 PEAS Specification**

The PEAS (Performance measure, Environment, Actuators, Sensors) framework is a structured definition for an intelligent agent's role and objectives and its methodology.For the autonomous aquatic boat, the PEAS components are specified as follows:

**Performance Measure**

The success of the agent is evaluated against a multi-objective performance measure that balances mission completion with operational efficiency and safety. This agent doesn’t count as a monolithic agent

The primary criteria for success are:

* **Completeness:** The agent must successfully traverse 100% of the designated roads (edges in the graph) exactly once. This is the **core requirement** of the problem statement.
* **Efficiency:** The agent must minimize battery consumption. Since the boat is battery-operated, every action has an energy cost. In this abstracted problem, efficiency is directly proportional to minimizing the total path length. As the problem requires traversing every edge once, all valid solutions have the same path length, making this criterion a constraint (cover all edges) rather than an optimization variable for path choice.
* **Timeliness:** The mission should be completed in the minimum possible time to deliver aid swiftly. This measure is closely related to path length and the agent's operational speed.
* **Mission Success:** The agent must accurately identify and log the geographic coordinates of locations where people are waving for help. This is a primary function of its deployment in a disaster zone.
* **Safety:** The agent must navigate the flooded areas without colliding with submerged or floating debris, other vessels, or structures. It must also operate in a manner that does not endanger the people it is trying to help. This is a critical real-world performance metric not explicitly modeled in the graph traversal problem but essential for a truly autonomous agent.

**Environment**

The agent operates within two distinct but related environments:

1. **The Physical Environment:** The flooded city of Chennai. This is a complex, real-world setting characterized by unpredictable water currents, floating debris, variable water depths, and changing weather conditions. It is the world the agent's sensors perceive and actuators affect.
2. **The Abstract Environment:** A graph representation, **G= (V, E),** where vertices V are landmarks and edges E are the flooded roads connecting them. This is a simplified, static model of the physical world upon which the planning algorithms will operate.

The agent's intelligence lies in its ability to use the abstract environment to formulate a plan (the path) and then execute it under severe challenging conditions

**Actuators**

Actuators are the mechanisms through which the agent enacts its decisions and interacts with the physical environment. For the aquatic boat, these include

* **Propulsion System:** Electric motors, propellers, and thrusters that generate force to move the boat through the water.
* **Steering System:** A rudder or differential thruster control to change the boat's direction and follow the planned path.
* **Communication Module:** A wireless transmitter (e.g., cellular or satellite) to report the completed path and the coordinates of identified distress locations back to the central command.
* **Signaling System:** Onboard lights or an acoustic device to acknowledge and respond to people waving for help, confirming that they have been seen.

**Sensors**

Sensors are the agent's means of perceiving its environment, providing the raw data necessary for localization, navigation, and mission-specific tasks.

* **Global Positioning System (GPS):** To determine the boat's absolute position (latitude and longitude) on the map, correlating its physical location with vertices in the abstract graph model.
* **Inertial Measurement Unit (IMU):** Comprising accelerometers and gyroscopes, the IMU provides data on the boat's orientation, pitch, roll, and velocity, which is crucial for stable navigation and path following.
* **Cameras (Visible and Infrared):** High-resolution cameras serve as the primary sensor for detecting people waving for help. Infrared capability would be essential for night operations or low-visibility conditions. These sensors are also critical for identifying surface-level obstacles.
* **Sonar/LiDAR:** Sonar (Sound Navigation and Ranging) is used for detecting submerged obstacles, while LiDAR (Light Detection and Ranging) can map surface-level debris and the immediate shoreline with high precision.
* **Battery Level Sensor:** To monitor the agent's primary resource—energy. This data is vital for ensuring the agent can complete its mission and return to base.

**1.2 Analysis of the Task Environment**

The agent's Task is classified along several key dimensions relative to the Environmental challenges encountered by the Agent. These are defined as assisting in deciding the Agent architecture relative to the complexity of the problem. This classification helps articulate the discrepancy between the simplified model used for planning and the complex reality of execution.

**1.2.1 Detailed Environment Analysis**

**Table 1: Task Environment Classification for the Autonomous Aquatic Boat**

| Property | Classification | Justification and Impact on Agent Design |
| --- | --- | --- |
| **Observability** | Partially Observable | Sensors have limited range. The agent cannot see the entire map at once. This necessitates an internal state representation, including a memory of traversed edges, to make rational decisions. |
| **Determinism** | Stochastic | Real-world effects like currents and debris make action outcomes uncertain. The planning algorithm operates on a deterministic graph model, but the execution system must be robust to stochasticity, likely requiring feedback control loops. |
| **Episodic/Sequential** | Sequential | The current action (edge choice) directly impacts future options and the ability to complete the mission. The agent requires planning and lookahead, not just reactive responses. |
| **Static/Dynamic** | Dynamic | The environment (e.g., obstacles, water levels) can change during the mission. The provided graph model is static, implying the need for a separate reactive layer or a re-planning capability in a full implementation. |
| **Discrete/Continuous** | Continuous (physical), Discrete (planning) | The boat's physical state is continuous, but the problem is abstracted to a discrete graph. The agent must translate the discrete plan into continuous motor commands for navigation. |
| **Single/Multi-agent** | Single-agent | The problem is defined for one agent. A real-world system would be multi-agent, requiring communication and coordination protocols to divide the workload efficiently. |
| **Known/Unknown** | Known (map), Unknown (hazards) | The graph topology is known, allowing for offline planning. The specific state of the edges (e.g., presence of debris or people) is unknown and must be discovered online through sensing. |

**1.2.2 Environment Complexity Analysis**

Observability Challenges:

* Limited Sensor Range: Cannot simultaneously observe entire Chennai network
* Urban Occlusion: Buildings and infrastructure block line-of-sight between areas
* Weather Interference: Monsoon conditions reduce sensor effectiveness
* Solution: Maintain internal map of visited edges and discovered emergency locations

Stochastic Elements:

* Water Current Variations: Unpredictable flow patterns near drainage systems
* Debris Movement: Floating obstacles creating dynamic navigation hazards
* Weather Changes: Sudden intensity variations affecting operational safety
* Solution: Robust control systems with error recovery and replanning capability

Dynamic Environment Factors:

* Water Level Changes: Rising/falling flood levels altering navigation routes
* Emergency Situations: New distress calls requiring immediate response
* Infrastructure Changes: Newly damaged areas or cleared routes
* Solution: Continuous environment monitoring with adaptive response protocols

**1.3 Problem Formulation for Chennai Relief Operations**

**1.3.1 Formal State Space Definition**

* State Representation: Each state s = (current\_location, visited\_edges\_set, battery\_level)
* current\_location ∈ {Nungambakkam, Guindy, Velachery, Tambaram, Perumbakkam, Purasawalkam}
* visited\_edges\_set ⊆ {all 9 road connections in Chennai network}
* battery\_level ∈ {0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%} (11 discrete levels)
* State Space Size: 6 × 2^9 × 11 = 33,792 possible states (manageable for exact algorithms)
* Initial State: s₀ = (Nungambakkam, ∅, 100%) - Starting from central command with full battery
* Goal States: Any state s = (location, E, battery) where |visited\_edges\_set| = 9 and battery ≥ 20%

**1.3.2 Problem Characteristics**

Graph-Theoretic Properties:

* Problem Type: Chinese Postman Problem variant (traverse each edge exactly once)
* Graph Connectivity: All vertices reachable with multiple path options ensuring solution existence
* Optimality: Multiple optimal solutions possible due to graph structure
* Complexity: Polynomial-time solvable for connected graphs

Computational Considerations:

* Search Space: Moderate complexity suitable for both informed and uninformed search
* Solution Quality: A\* guarantees optimal solution; DFS provides feasible solution
* Resource Requirements: Memory usage manageable for onboard computing systems
* Real-time Constraints: Path planning must complete within operational time limits

**1.4 Algorithm Selection Rationale**

**1.4.1 A\* Search Justification**

Advantages for Chennai Operations:

* Optimality Guarantee: Critical for emergency response where efficiency saves lives
* Heuristic Efficiency: **Reduces search space** in 6-location network through intelligent guidance
* Completeness: Guaranteed solution finding in connected Chennai graph
* Performance: Well-suited for moderate-sized problems with 9 edges

Chennai-Specific Benefits:

* **Minimizes battery consumption t**hrough optimal routing
* Reduces mission time in potentially dangerous flood conditions
* Provides predictable performance for mission planning

**Approach (A\* Approach)**

**Problem summary**

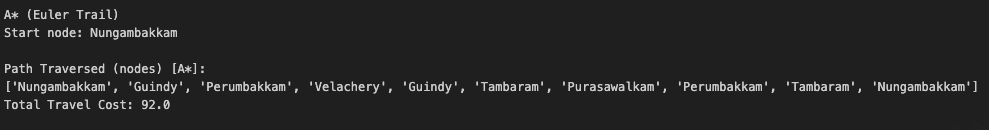
Find a route for an autonomous boat that traverses every lane (edge) exactly once (if possible). We model the flooded city as an undirected graph: vertices = landmarks, edges = lanes. The agent may revisit vertices but must not repeat edges.

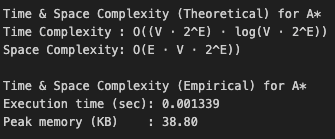
Goal: find a sequence of moves that covers all edges.

* Approach: A\* over (node, remaining\_edges) state
* State representation: - (current\_node, remaining\_edges\_set)
  + Actions: - Choose an unused edge incident to current\_node and traverse it; this moves to the neighbouring node and removes the edge from remaining\_edges\_set.
  + Goal test: - remaining\_edges\_set is empty.
  + Cost: - g(n) = total weight (or number) of edges traversed so far.
  + Heuristic (h): - h(n) = number\_of\_remaining\_edges. This is admissible when every edge costs at least 1, because each leftover edge requires at least one traversal.
    - Notes: - This A\* formulation is exact but expensive: state-space size is O(N \* 2^E) in the worst case (where E = number of edges). Practical for small graphs (E <= ~20).

**Algorithm (high level)**

* Initialize the priority queue with (f = h(start), g = 0, state = (start\_node, all\_edges), path = [start\_node], edges\_traversed = []).
* Pop the state with smallest f = g + h.
* If remaining\_edges\_set is empty, return path/edges\_traversed as the solution.
* Expand: for each unused edge (eid) from current\_node to neighbor, generate new state removing eid, update g, compute f, and push to queue.
* Use closed-set (visited dict) keyed by (node, remaining\_edges\_frozenset) to keep best g found for that state and prune worse entries.





**1.4.2 Depth-First Search Justification**

Advantages for Resource-Constrained Environment:

* Memory Efficiency: **Linear space complexity** crucial for embedded boat computers
* Implementation Simplicity: Robust operation in challenging Chennai flood conditions
* Completeness: Guaranteed solution in connected graph structure
* Fault Tolerance: Simple algorithms less prone to failure in harsh conditions

Chennai-Specific Benefits:

* Reliable operation despite potential sensor/communication failures
* **Lower computational overhead** preserving battery for navigation
* Straightforward debugging and maintenance in field conditions

## **Approach (DFS Approach):**

**Problem Summary**

Find a route for an autonomous boat that traverses every lane (edge) exactly once using depth-first search strategy. We model the flooded city as an undirected graph: vertices = landmarks, edges = lanes. The agent may revisit vertices but must not repeat edges.

Goal: find a sequence of moves that covers all edges through systematic depth-first exploration.

* Approach: DFS over (node, remaining\_edges) state
* State representation: (current\_node, remaining\_edges\_set) - identical to A\* but explored via stack (LIFO)

Actions:Choose an unused edge incident to current\_node and traverse it; this moves to

the neighboring node and removes the edge from remaining\_edges\_set

Goal test:remaining\_edges\_set is empty

Cost:g(n) = total weight of edges traversed so far

Search Strategy:LIFO stack-based exploration without heuristic guidance

Explores deepest available path first before backtracking

No f-value calculation: purely systematic depth-first traversal

Notes:

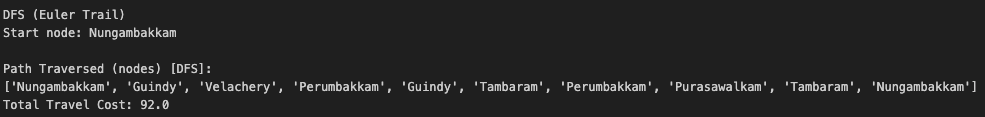
Memory efficient: O(bm) space complexity vs A's O(N \* 2^E)

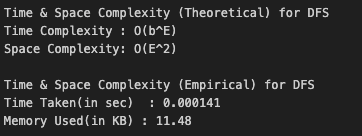
Not guaranteed optimal but finds complete solutions quickly for connected graphs

Excellent for Chennai's small network (6 vertices, 9 edges)

**Algorithm (High Level)**

* Initialize the stack with (state = (start\_node, all\_edges), path = [start\_node], edges\_traversed = [])
* Pop the most recently added state from stack (LIFO order)
* If remaining\_edges\_set is empty, return path/edges\_traversed as the solution
* Expand: for each unused edge (eid) from current\_node to neighbor, generate new state removing eid, update g, and push to stack
* Use visited set keyed by (node, remaining\_edges\_frozenset) to avoid cycles and redundant exploration





**Section 2: Heuristic and or fitness function:**

## Define the heuristic and/or fitness function for the given algorithms and the given problem

**State Representation**

n = (v, Et)

v: current vertex.

Et: set of traversed edges.

Goal State

Et = E

(All edges in the graph have been covered exactly once.)

# **2.1 A\* Algorithm**

**Cost Function g(n) — Distance Already Covered**

## g(n) = ∑ weight(e)

e∈Et

Represents the actual distance (or cost) traveled so far from the starting vertex to the current state.

**Heuristic Function h(n) — Estimated Remaining Cost**

## h(n) = ∑ weight(e)

e∈(E∖Et)

Represents the estimated cost of the remaining roads not yet traversed. This heuristic is admissible

because all remaining roads must be covered exactly once.

**Fitness Function f(n)**

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

Balances the distance already covered with the estimated distance left, guiding the search toward the shortest path.

# **2.2 Depth-First Search (DFS)**

**Cost Function g(n) — Distance Already Covered**

* Not used to guide DFS.
* We may track the cumulative distance along the current recursion path for **reporting** (solution cost) as

## g(n) = ∑ weight(e)

e∈Et

* but DFS **does not** use g(n)g(n)g(n) to decide which node to expand next.

**Heuristic Function h(n) —** DFS does not use heuristics. It explores one branch fully before backtracking.

**Fitness Function f(n)**

* Not applicable.
* DFS does **not** compute f(n)=g(n)+h(n). Node expansion order is purely **LIFO** (the most recently generated node is expanded next). Any “fitness” definition would be unused.
* The evaluation depends only on the distance already covered, with no forward-looking estimate.