Phase 1 - Project Documentation: The River Crossing Problem Solver

Abstract

This project focuses on solving the classic Missionaries and Cannibals river crossing puzzle using different AI search algorithms. The goal is to model the problem clearly and compare how five algorithms—BFS, DFS, IDDFS, A*, and Greedy—perform in finding a valid solution. We will also build a simple GUI to show how the puzzle is solved step-by-step. This document explains the problem, our approach, and why these algorithms are suitable for solving it.

Introduction

The River Crossing Problem is a classic puzzle often used in computer science and AI to teach state-space search. In the Missionaries and Cannibals version, the task is to move three missionaries and three cannibals across a river using a boat, without ever leaving the missionaries outnumbered by cannibals on either side.

In this project, we aim to build an interactive simulation that solves this puzzle and shows how different search algorithms behave while finding a solution. Our documentation explains the problem, the methods we will use, and how the work is divided among the five team members.

1. Problem Definition

The River Crossing Problem (Missionaries and Cannibals) is formally modeled as a deterministic state-space search problem, represented by the tuple **P = (S, A, G, s_0)**, where:

State Space (S)

- **Description:** The set of all possible configurations of people and the boat. A state is defined by the tuple **(M_L, C_L, B)**, representing the number of missionaries (**M_L**) and cannibals (**C_L**) on the left bank, and the boat's position (**B**).
- **Formal Representation:** `S = {(M_L, C_L, B) | 0 ≤ M_L, C_L ≤ 3, B ∈ {0, 1}}`

  (Note: B=1 denotes Left Bank, B=0 denotes Right Bank)

Initial State (s_0)

- **Description:** The starting configuration where all entities and the boat are located on the left bank.
- **Formal Representation:** $s_0 = (3, 3, 1)$

Goal State (G)

- **Description:** The target configuration where all entities have successfully crossed to the right bank.
- **Formal Representation:** G = (0, 0, 0)

Actions (A)

- **Description:** The set of valid moves the boat can make. The boat must carry at least 1 and at most 2 people.
- **Formal Representation:** A = {(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)}

  (Note: Pairs represent (Missionaries, Cannibals) moving)

Constraints

- **Description:** A state is valid if and only if the number of cannibals does not exceed the number of missionaries on either bank (unless the number of missionaries is 0).
- **Formal Representation (Safety Condition):** `(M_L = 0 OR M_L ≥ C_L)` **AND** `(3 − M_L = 0 V 3 − M_L ≥ 3 − C_L)`

The solution is defined as a sequence of valid actions that transforms the initial state $\mathbf{s}_0$ into the goal state **G**.

---

2. Methodology

The project employs a structured, iterative development pipeline that strictly separates the **Core AI Logic** from the **User Interface (UI)**.

A. Core Logic Implementation

The five selected search algorithms—Breadth-First Search (BFS), Depth-First Search (DFS), Iterative Deepening DFS (IDDFS), A* Search, and Greedy Best-First Search (Greedy)—will be implemented as independent Python modules.

- **Input:** A formal problem definition instance.
- **Output:** A solution path (sequence of states) and performance metrics[9].

B. Heuristic Design (A* **and Greedy) **

To drive the informed search algorithms, we define a heuristic function h(s) that estimates the remaining cost from the current state to the goal[10].

- Heuristic Function:
- h(s) = {M_L + C_L} / {2}
- Justification (Admissibility):

This heuristic calculates the total number of people remaining on the starting bank

(M_L + C_L) divided by the boat's maximum capacity (2). Since the boat can transport a maximum of two people per trip, the minimum number of one-way trips required is mathematically bounded by half the number of people. This ensures the heuristic never overestimates the true cost, maintaining the admissibility required for A* to find the optimal solution.

C. Performance Metrics

We will quantitatively compare the algorithms using the following metrics:

1. **Path Cost:** The number of moves in the final solution path.
2. **Nodes Explored:** The total count of unique states visited and expanded during the search.
3. **Time Complexity:** The execution time required to compute the solution[14].

D. GUI Integration

A graphical user interface (GUI) will be developed using a Python library (e.g., Pygame or Tkinter) to serve as the visualization layer. It will interpret the state sequences generated by the AI modules and render them as animated boat transitions, alongside a real-time dashboard comparing the metrics of the selected algorithms.

Full Project Pipeline

The project will be executed in three main phases:

Phase	Focus	Deliverables
Phase 1 (Documentation)	Theoretical foundation, problem formalization, algorithm selection, and team planning.	Comprehensive Documentation File (This document).
Phase 2 (Core Logic)	Implementation and testing of the five search algorithms in a command-line environment.	Five fully functional, unit-tested Python modules for each algorithm.
Phase 3 (GUI & Visualization)	Development of the interactive GUI, integration of the core logic, and creation of the visualization and performance dashboard.	Final GUI-based simulation, including all source code and a final report.

All Project Details: The River Crossing Problem

The River Crossing Problem, formally known as the Missionaries and Cannibals Problem, is a canonical example of a state-space search problem in Artificial Intelligence. This document provides the comprehensive technical specifications and design blueprint for the project's implementation, adhering to the academic standards required for a university-level AI project.

1. Algorithm Justification

The problem is modeled as a search through a state graph, where nodes represent valid configurations of missionaries, cannibals, and the boat, and edges represent valid moves. The objective is to find a path from the initial state to the goal state. We have selected a diverse set of search algorithms—encompassing both uninformed and informed strategies—to facilitate a comparative analysis of their performance characteristics on this specific problem space.

Algorithm	**Time Complexity**	**Space Complexity**	**Completeness**	**Optimality**
Breadth-First Search (BFS)	O(b^d)	O(b^d)	Yes	Yes
Depth-First Search (DFS)	O(b^m)	O(bm)	No (without cycle detection)	No
Iterative Deepening Depth-First Search (IDDFS)	O(b^d)	O(bd)	Yes	Yes
Greedy Best-First Search (GBFS)	O(b^m)	O(b^m)	No	No
A* Search	O(b^d) (with a good heuristic)	O(b^d) (with a good heuristic)	Yes	Yes

Note: b is the branching factor, d is the depth of the shallowest goal, and m is the maximum depth of the state space. The complexity of the informed searches is heavily dependent on the quality of the heuristic function.

Heuristic Function for Informed Search

For both Greedy Best-First Search and A* Search, we will employ an **admissible heuristic** $h(n)$ that estimates the minimum number of remaining boat trips required to reach the goal state. An admissible heuristic never overestimates the true cost from the current state to the goal.

Let P_L = M_L + C_L be the total number of people on the left bank. The boat capacity is C_{boat}=2

1. h(s) = (M_L + C_L) / 2

This heuristic is admissible because it represents the theoretical minimum number of trips required to move the remaining people, assuming all trips are optimally filled and no return trips are necessary. In reality, return trips are required, meaning the actual number of trips will be greater than or equal to $h(n)$, thus satisfying the admissibility criterion.

2. Allowed Moves

The state of the system is defined by a tuple: S = (M_L, C_L, B), where M_L and C_L$ are the number of missionaries and cannibals on the left bank, and $B$ is the boat's location (1 for Left, 0 for Right). The standard problem involves three missionaries and three cannibals, with a boat capacity of two.

Valid Move Combinations

The boat can carry a minimum of one and a maximum of two individuals. The five possible combinations of passengers $(m, c)$ that can cross are:

1. (1 Missionary, 0 Cannibal)
2. (2 Missionaries, 0 Cannibal)
3. (0 Missionary, 1 Cannibal)
4. (0 Missionary, 2 Cannibals)
5. (1 Missionary, 1 Cannibal)

Constraints and Move Validation

A move is considered **valid** only if it satisfies two critical constraints:

6. **Boat Capacity Constraint:** The number of passengers $m+c$ must be between 1 and 2, inclusive.
7. **Safety Constraint:** On *both*the departure bank and the arrival bank, the number of cannibals must not exceed the number of missionaries, provided there is at least one missionary present.
   For each bank $i$in {Left, Right}: if the money $M_i$is greater than 0, then the cost $C_i$must be less than or equal to M_i."

Move Generation and Handling

The algorithms handle move generation by iterating through all five possible move combinations. For each combination, a potential successor state is calculated. This successor state is then subjected to the **Safety Constraint** check on both the departure and arrival banks. If the constraint is violated on either bank, the move is discarded, and the successor state is deemed **invalid**. Only valid successor states are added to the search frontier (queue or stack).

3. Goal State Definition

The goal of the River Crossing Problem is to safely transport all missionaries and cannibals from the initial bank (Left) to the destination bank (Right).

Formal Definition

The initial state S_{initial} and the goal state S_{goal} are formally defined as:

S_{initial} = (3, 3, 1)
S_{goal} = (0, 0, 0)

Where (M_L, C_L, B) represents the number of missionaries on the left bank, the number of cannibals on the left bank, and the boat's location (1 for Right bank).

Goal Recognition

The algorithms recognize the goal state through a simple, direct state comparison. Upon generating a successor state, the algorithm performs a **Goal Test** by checking if the generated state $S_{successor}$ is identical to S_{goal}.

The goal test is: $Goal\,Test(S) = (S == S_{goal})$.

Special Handling for Goal Testing

For optimal search algorithms (BFS, IDDFS, A*), the goal test is typically performed when a node is **generated** (before adding to the frontier) to ensure the shortest path is found immediately. For DFS and GBFS, the test is performed when a node is **expanded**. No other special handling is required, as the goal is a single, well-defined terminal state.

4. Expected Output

The final output of the project must be a comprehensive report detailing the solution found by each algorithm and a comparative analysis of their performance.

Solution Path Format

The solution path will be presented as a chronological sequence of states and the move that facilitated the transition.

Step	Current State $(M_L, C_L, B)$	Move $(m, c)$	Next State $(M_L, C_L, B)$
0	(3, 3, 1)	-	-
1	(3, 3, 1)	(0, 2) goes to the Right.	(3, 1, 0)
2	(3, 1, 0)	(0, 1) goes to the Left.	(3, 2, 1)
...	...	...	...
11	(0, 2, 0)	(0, 2) goes to L (Left)	(0, 0, 1)

Performance Metrics

The following quantitative metrics will be reported for each successful algorithm execution to facilitate a rigorous comparative analysis:

8 **Path Length (Solution Depth):** The number of moves (edges) in the solution path. For the optimal solution, this is the depth d.
9 **Nodes Explored (Search Cost):** The total number of nodes expanded by the algorithm before the goal state was found. This is the primary measure of search efficiency.
10 **Execution Time (Wall-Clock Time):** The time taken from the start of the search until the goal state is found, measured in milliseconds. This provides a practical measure of performance in the implemented environment.

5. GUI Design Specifications

The Graphical User Interface (GUI) is designed to provide an intuitive, interactive, and academically valuable visualization of the River Crossing Problem and the search process.

Layout and Visualization

The GUI will be structured into three main sections:

11 **Visualization Area (Center):** This central area will display the core problem state.
- **River:** A blue vertical band separating the two banks.
- **Banks:** Two distinct areas (Left and Right) where the characters reside.
- **Characters:** Missionaries (M) and Cannibals (C) will be represented by distinct, easily identifiable icons.

    - **Boat:** A small, movable icon that can hold up to two characters. Its position will clearly indicate the current bank.
    - **State Display:** A persistent text box will display the current state tuple (M_L, C_L, B) in real-time.

12 **Control Panel (Left/Top):** This section will house the interactive elements.
    - **Algorithm Selection:** A dropdown menu or radio buttons to select one of the five implemented algorithms (BFS, DFS, IDDFS, GBFS, A*).
    - **Execution Controls:** Standard **Start**, **Stop**, and **Reset** buttons for managing the search process.
    - **Visualization Controls:** A **Next Move** button for step-by-step advancement and a slider to control the animation speed for continuous execution.

13 **Performance Metrics Dashboard (Right/Bottom):** This area will display the real-time and final performance data.
    - **Real-Time Metrics:** A live counter for "Nodes Explored" and "Current Path Length."
    - **Final Report:** A clear, tabular display of the three key metrics (Path Length, Nodes Explored, Execution Time) upon completion of the search.
    - **Solution Path Log:** A scrollable text area that logs the sequence of moves and states as the solution is found.

Step-by-Step Visualization of Algorithm Execution

The GUI will not only show the solution path but also visualize the search process itself, providing an invaluable educational tool:

- **Node Expansion:** When a node is expanded, the corresponding state in the visualization area will briefly flash or be highlighted to indicate the search process.
- **Frontier Visualization:** A small, dynamic list or tree view will show the current contents of the search frontier (Queue for BFS, Stack for DFS, Priority Queue for A*), allowing users to observe the internal workings of each algorithm.
- **Path Highlighting:** The current path being explored will be highlighted in a distinct color. Once the goal is found, the optimal path will be highlighted in a final, permanent color.

This detailed specification ensures that the implementation will be robust, academically sound, and highly visual, serving as a complete and effective demonstration of state-space search techniques.

Algorithms to be Used

As the team consists of five members, a minimum of five AI search algorithms will be implemented to ensure a challenging and comprehensive project. The selection includes a mix of uninformed and informed search strategies to provide a thorough comparative analysis.

ID	Algorithm	Type	Justification for Choice	How it Solves the Problem
1	**Breadth-First Search (BFS)**	Uninformed	**Completeness and Optimality:** Guarantees finding the shortest path (minimum number of moves) if a solution exists. It serves as the baseline for optimal path cost comparison.	Explores the state space level by level, ensuring the first time the goal state is reached, it is via the shortest sequence of moves.
2	**Depth-First Search (DFS)**	Uninformed	**Simple to implement:** DFS is easy to code (recursively or with a stack), making it great for prototyping. Finds a solution fast: If any valid path is acceptable—especially a deep one—DFS often finds it quicker than BFS. DFS is ideal when memory is limited.	Explores one branch of the state space as far as possible before backtracking. Useful for finding a solution quickly

ID	Algorithm	Type	Justification for Choice	How it Solves the Problem
3	**Iterative Deepening DFS (IDDFS)**	Uninformed	**Balanced Approach:** Combines the space efficiency of DFS with the completeness and optimality of BFS. It is often the preferred uninformed search in practice.	Performs a series of depth-limited DFS searches, gradually increasing the depth limit. This guarantees optimality while using minimal memory.
4	**Greedy Best-First Search (GBFS)**	Informed	**Heuristic-Driven Speed:** Chosen to test the effectiveness of the heuristic function alone. It prioritizes states that appear closest to the goal, aiming for the fastest search time.	Uses a heuristic function $h(s)$ to estimate the cost to the goal. It expands the node with the lowest $h(s)$, prioritizing speed over path cost optimality.
5	**A* Search**	Informed	**Optimal Informed Search:** Guarantees finding the shortest path while being significantly more efficient than BFS. It is the gold standard for pathfinding problems.	Uses an evaluation function $f(s) = g(s) + h(s)$, where $g(s)$ is the cost from the start and $h(s)$ is the estimated cost to the goal. It expands the node with the lowest $f(s)$, ensuring optimal path cost.

Team Working Plan

The team consists of five members, and the project is divided into five distinct, high-level components, with each member assigned a clear role and responsibility for the entire project lifecycle (Phase 1 documentation, Phase 2 implementation, and Phase 3 GUI).

Team Member	**Assigned Role**	**Primary Responsibility (Algorithm & Component)**	**Phase 1 Documentation Contribution**
Eyad	**Project Lead & GUI Developer**	GUI Development (Pygame) & Project Management	Abstract, Introduction BFS, Team Working Plan
Yassin	**BFS Specialist**	Breadth-First Search (BFS) Implementation	Problem Definition, Methodology
Ali	**DFS Specialist**	Depth-First Search (DFS) & Iterative Deepening DFS (IDDFS) Implementation	Algorithms (DFS/IDDFS sections)
Karim	**Greedy& Performance Engineer**	Greedy Best-First Search Implementation & Performance Metrics Module	Full Project Pipeline, All Project Details
Mazen	**A* Specialist & Heuristic Designer**	A* Search Implementation & Heuristic Function Design	Algorithms (A* and GBFS sections)