Project 1 — Where Are You?

Sinchana Shylaja Arun

Net ID: ssa223

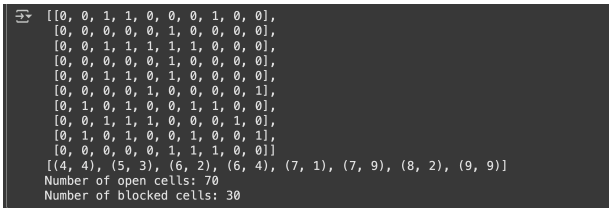
Rutgers University

Table of Contents

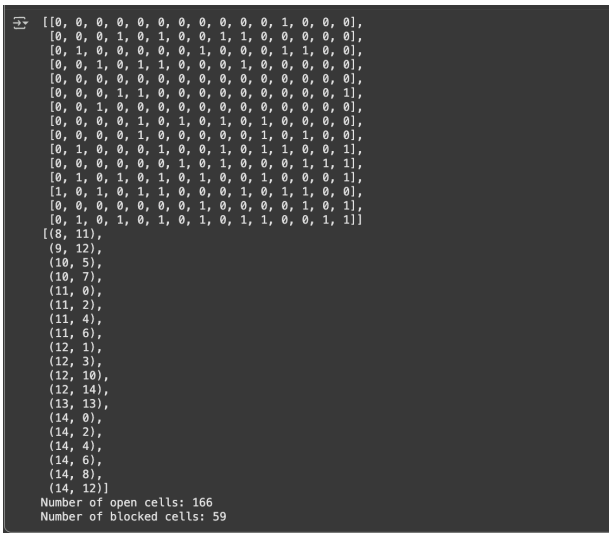
1 Ship Generation

The ship is generated using a $D \times D$ square matrix consisting of initially blocked cells (represented by 1). An interior cell is chosen to be opened (represented as 0) at random. A function `exactly one-open neighbor` is defined to perform the operation of opening random cells consisting of only one open neighbor iteratively, until it no longer can be done.

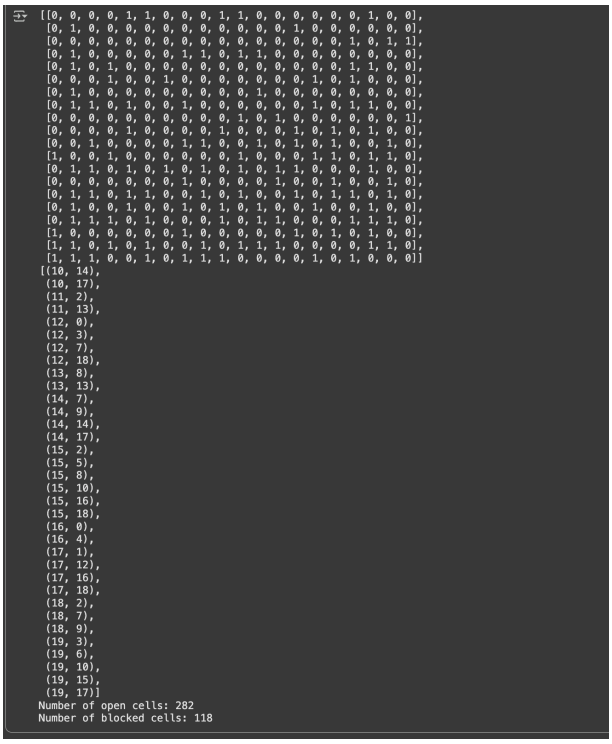
Deadends (represented by 2) are open cells (0) with exactly one open neighbor. For approximately half of them, one of their closed neighbors (1) is randomly chosen and opened. This creates the possibility of loops and multiple paths between any two points in the ship.



10x10 ship



15x15 ship



20x20 ship

2 Baseline Strategy:

The Baseline strategy provides a way to localize the bot through successive reduction of possible positions on the ship. The process relies on iteratively moving (UP, DOWN,LEFT, RIGHT) towards a fixed target cell until the bot's exact location is determined.

1. **Target Selection:** A random **deadend** cell (open cell with exactly one open neighbor) is chosen as the **target cell**. This cell serves as the known reference position where the bot will eventually be localized.

2. **Initialization:** The bot's possible starting positions are defined as all open cells in the ship. Since the bot's true location is unknown, each open cell is a potential initial state.

3. **Start Cell Selection:** One of the open cells is randomly chosen as the **start cell** for the next move sequence. This simulates the planning perspective of "if the bot were here, how would it reach the target?"

4. **Path Planning using A* Search:** A path is computed from the chosen start cell to the target cell using the **A* search algorithm** with a **Manhattan distance**

heuristic. The Manhattan distance between two cells (x_1, y_1) and (x_2, y_2) is:

$$h(x, y) = |x_1 - x_2| + |y_1 - y_2|$$

This heuristic estimates the number of moves required in a grid environment when movement is restricted to four directions (up, down, left, right).

5. **Path Execution and Belief Update:** The planned sequence of moves is executed conceptually. After each move, the algorithm updates the set of possible bot locations L based on whether that move would have been successful (i.e., not blocked by walls) from each potential current location. As moves are applied, cells that are no longer reachable are removed from L.

6. **Iteration:** Steps 3–5 are repeated until the set L of possible bot locations is reduced to a single cell. When $|L| = 1$, the bot's location is fully localized.

7. **Result:** The algorithm returns the total number of moves executed and the time taken to localize the bot to the target cell.

2.1 Use of A* with Manhattan Distance

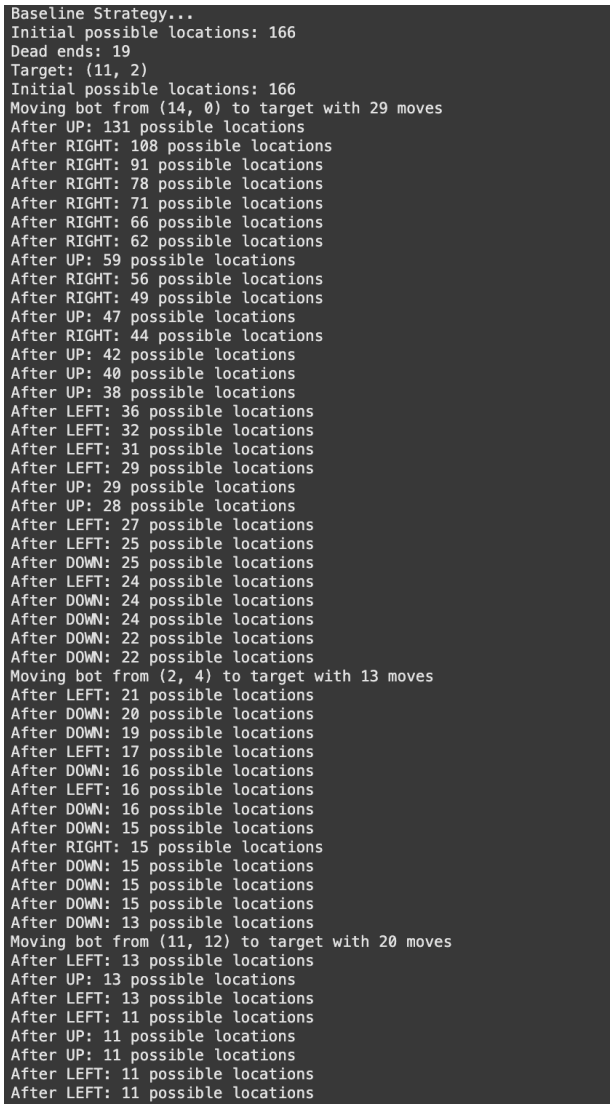
The A* algorithm is employed because it balances exploration and efficiency. It evaluates each candidate cell using the function:

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

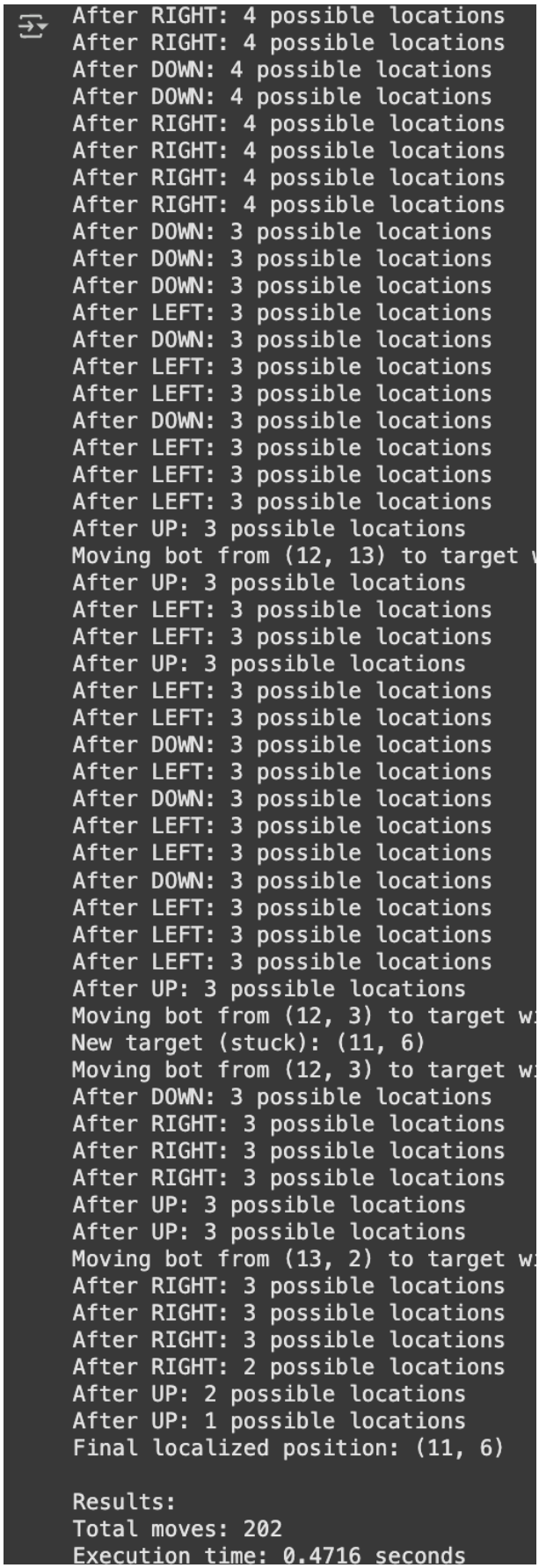
where:

- $g(n)$ = cost (number of moves) from the start cell to the current cell,
- $h(n)$ = estimated cost from the current cell to the target cell (Manhattan distance).

This heuristic is **admissible**—it never overestimates the true distance— making A* both efficient and optimal for pathfinding in grid-based maps. It ensures that Cowan's strategy always finds a valid shortest path to the target for each iteration, even though the overall localization process itself may not be globally optimal due to the random choice of targets.



Baseline Strategy:1



Baseline Strategy:2

3 Optimality Strategy

The goal of the Optimality Strategy is to determine the shortest possible sequence of move attempts that guarantees the bot's location is known, regardless of its initial position on the ship. This strategy aims to achieve provable optimality by minimizing the number of moves required to localize the bot in any given map configuration.

To achieve this, I implemented an informed search algorithm based on **A***, after evaluating several alternatives:

- **Breadth-First Search (BFS):** Guarantees optimality but scales poorly with larger grids due to exponential state-space growth.
- **Depth-First Search (DFS):** Explores paths deeply but lacks optimality guarantees and may revisit states unnecessarily.
- **Dijkstra's Algorithm:** Equivalent to A* with $h(n) = 0$; correct but computationally inefficient.
- **Greedy Best-First Search:** Explores faster by minimizing heuristic cost only, but is not guaranteed to find an optimal solution.

A* was selected as it balances exploration and exploitation through its evaluation function:

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

where $g(n)$ is the cost so far (moves taken), and $h(n)$ is the heuristic estimate of the remaining cost. Unlike the baseline (which used the Manhattan distance heuristic for physical movement), the Optimality Strategy operates in the **belief-state space** — the space of all possible bot locations.

3.1 Belief-State Formulation

State Representation: Each state in this search is represented as a `frozenset` of all possible cells where the bot could currently be located. This abstraction captures the uncertainty of the bot's position, where each cell represents a hypothesis about its true location.

Transition Function: Each attempted move transforms the belief set by simulating the move in all current possible locations. Cells that cannot be reached due to walls are removed, yielding a new, smaller belief set.

Rationale: This belief-space formulation allows the algorithm to reason not about physical movement, but about how each move sequence changes the information state. The search terminates when the belief set is reduced to a single cell:

$$|S| = 1$$

meaning the bot's position is fully localized.

Advantages:

- The `frozenset` structure is immutable and hashable, allowing for fast lookups and efficient duplicate detection.
- The search space, though large, can be pruned effectively by discarding equivalent or redundant belief states.

3.2 A* Search Algorithm

A* systematically explores belief states, prioritizing those that are closer to the goal (smaller uncertainty) and cheaper to reach. The evaluation function used is:

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

where:

- $g(S)$ = number of move attempts made so far, and
- $h(S)$ = heuristic estimate of moves still required to fully localize.

The algorithm expands the belief state with the lowest $f(S)$ value, ensuring that the first solution found is guaranteed to be optimal.

3.3 Heuristic Function Used in Implementation

In the implemented A* search for the Optimality Strategy, the heuristic is defined exactly as follows:

```
h_score = len(new_state) - 1
```

Here, `new_state` represents the updated belief set — the collection of all possible locations the bot could occupy after a move is applied to every position in the current state. The length of this set, `len(new_state)`, quantifies the remaining uncertainty about the bot's true position.

Mathematical Formulation

The corresponding mathematical expression for the heuristic is:

$$h(S) = |S| - 1$$

where S is the set of possible bot locations. The heuristic value decreases as the localization process narrows down the number of possible positions. When $|S| = 1$, the bot is fully localized and $h(S) = 0$.

3.3.1 Intuitive Meaning

This heuristic directly measures **information uncertainty** rather than geometric distance. If there are ten possible locations remaining, the value of $h(S) = 9$ indicates that at least nine more information-gaining moves are theoretically required to isolate the bot's position. Thus, the heuristic acts as a proxy for the remaining ambiguity.

Integration with A* Evaluation Function

The total cost function used in A* is:

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

where:

- $g(S)$ is the number of moves taken so far (actual cost), and
- $h(S) = |S| - 1$ estimates the remaining number of moves required for localization.

Each state's total score $f(S)$ therefore reflects both the progress already made and the uncertainty still remaining. States with smaller $f(S)$ values are explored first.

3.3.2 Theoretical Properties

- **Admissibility:** The heuristic never overestimates the true cost. To localize from a belief set of size $|S|$, at least $|S| - 1$ moves are required, making $h(S)$ a valid lower bound.
- **Consistency:** For any two successive states S and S', the heuristic satisfies $h(S) \leq 1 + h(S')$, ensuring A* explores states in non-decreasing order of total cost.
- **Computational Simplicity:** The heuristic is efficient to compute, as it depends only on the belief set size and requires no geometric calculations.

3.3.3 Interpretation

This heuristic embodies the insight that bot localization is not a spatial navigation problem, but an *information-space search*. Each move reduces uncertainty, and $h(S)$ estimates how much ambiguity remains. The strategy guides A* to expand belief states that most effectively reduce uncertainty, leading to an optimal move sequence.

3.4 Search in Information Space

Unlike traditional pathfinding, this problem operates entirely in **information space**. Each state represents the current knowledge about the bot's possible positions, and each move represents an *information-gaining action*. The A* algorithm thus finds the shortest sequence of information updates needed to reach complete certainty.

3.5 Move Pruning Strategy

To manage the exponential growth of the belief-state space, a dominance-based pruning method is applied.

- If two belief states contain the same open cells but differ only by redundant transitions, only the smaller or more informative state is retained.
- Moves that do not reduce the size of the belief set are pruned, since they provide no new information.

Move pruning strategy — eliminating non-informative transitions to improve efficiency.

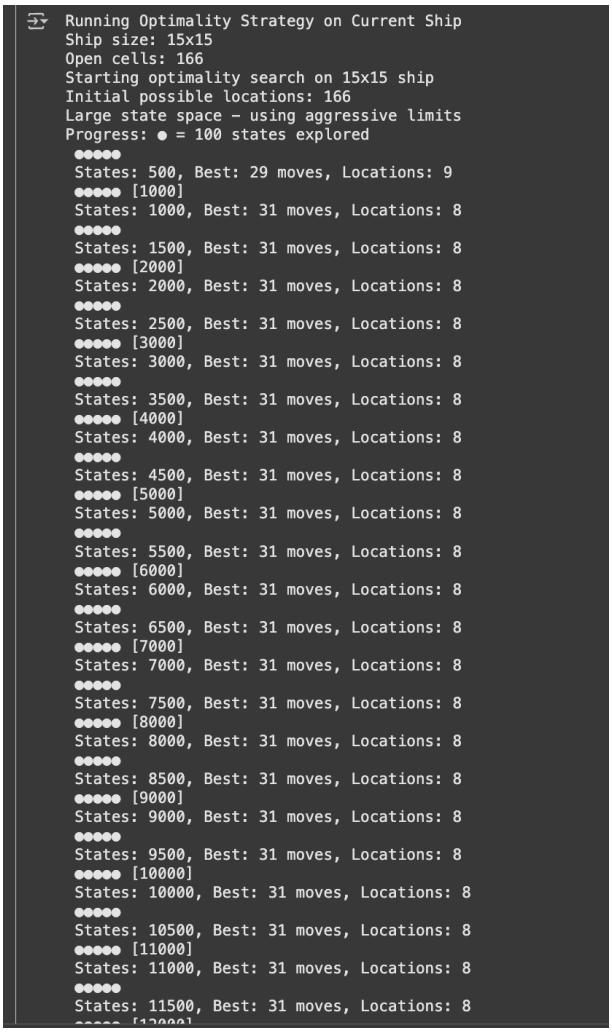
This pruning greatly reduces the branching factor and ensures that the search remains computationally feasible even for moderately large ships.

3.6 Why This Strategy Is Optimal

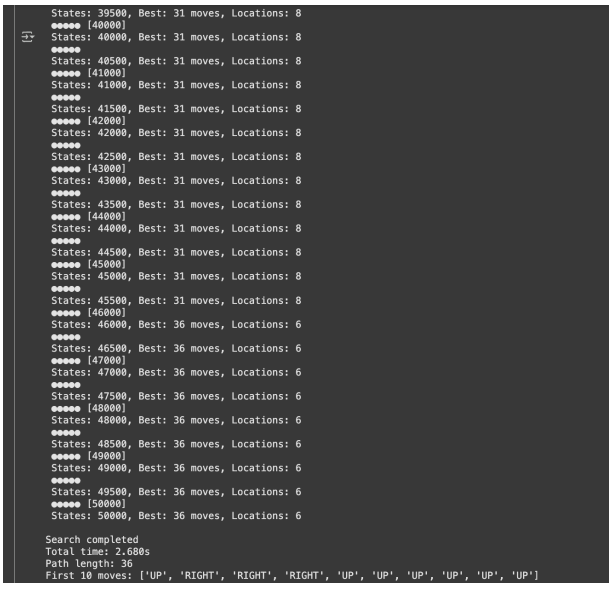
- **A* Optimality Theorem:** A* with an admissible heuristic always returns an optimal solution.
- **Heuristic Validity:** The implemented $h(S) = |S| - 1$ never overestimates the true cost.
- **Accurate Transitions:** Each simulated move correctly models ship constraints and wall behavior.
- **Systematic Exploration:** The priority queue ensures expansion in order of increasing true cost.

3.7 Summary

Therefore, this strategy achieves optimality by minimizing the number of moves required to localize the bot. It provides a mathematically sound, information-theoretic approach to the localization problem and serves as the benchmark for comparing efficiency-based strategies.

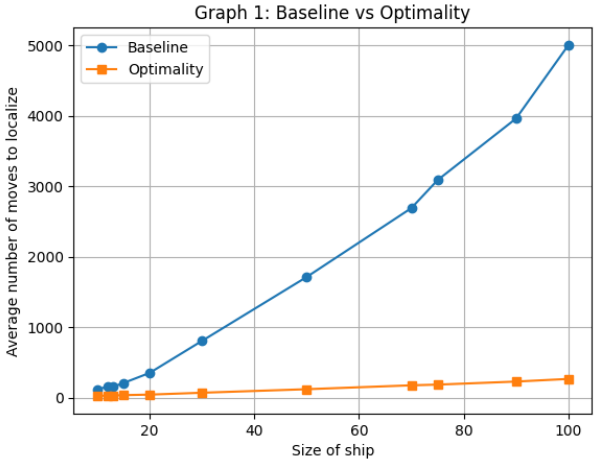


Output of Optimality Strategy:1



Output of Optimality Strategy:2

4 Graphical Representation: Baseline vs Optimality Strategy



Graph 1: Baseline vs Optimality

5 Efficiency Strategy

The Efficiency Strategy is designed to address the computational limitations of the Optimality Strategy. While the Optimality approach guarantees the shortest possible sequence of moves, it incurs heavy computational overhead, especially for larger ship sizes where the belief-state space grows exponentially. The Efficiency Strategy, therefore, focuses on achieving **reasonable localization performance within a strict time bound**, sacrificing theoretical optimality for practical efficiency.

5.1 Design Motivation

The key motivation behind the Efficiency Strategy is to maintain fast computation times without completely compromising localization accuracy. Whereas the Optimality Strategy performs exhaustive exploration using A*, this approach limits computation through a combination of randomization and time constraints.

Primary Objectives:

- **Speed:** Produce a localization result within seconds, even on large grids.
- **Scalability:** Handle ships of size 25×25 or larger without exceeding time limits.

- **Simplicity:** Use lightweight operations such as random move generation and direct state updates.

5.2 Algorithm Overview

The algorithm begins with the full set of possible bot locations and repeatedly applies randomly chosen moves (`UP`, `DOWN`, `LEFT`, `RIGHT`) to all positions simultaneously. After each move, the belief set is updated by simulating the resulting positions while accounting for blocked or inaccessible cells. The process continues until one of the following termination conditions is reached:

1. The belief set reduces to a single location ($|S| = 1$), meaning the bot is fully localized.

2. The predefined maximum computation time (`max_time`) is exceeded.

Pseudocode Summary:

```
current_state = set(all possible bot locations)
while len(current_state) > 1 and time < max_time:
    move = random.choice(["UP", "DOWN", "LEFT", "RIGHT"])
    new_state = apply_move_to_set(current_state, move)
    current_state = new_state
```

This randomized approach ensures the algorithm continues to make progress, while the bounded runtime keeps it computationally tractable.

5.3 Design Choices

Randomized Move Selection: By choosing random moves, the algorithm avoids complex planning overhead. While this introduces nondeterminism, multiple random trials still tend to reduce the belief set effectively, especially when dead-ends and blocked cells are distributed across the grid.

Time Constraint (`max_time`): To maintain efficiency, each run is capped at a user-specified duration (default 30 seconds). If full localization is not achieved within this limit, the best partial reduction of uncertainty is accepted as the final result.

Incremental State Update: After each move, the function `simulate_single_move()` updates all positions simultaneously, allowing efficient computation of new belief states without explicit search.

5.4 Characteristics of the Strategy

- **Heuristic-Free:** No explicit heuristic is used; the strategy relies purely on random exploration and state contraction.
- **Time-Bounded:** Guarantees a result in limited time regardless of grid size.
- **Low Memory Usage:** Does not maintain an open or closed set like A*, resulting in a significantly smaller memory footprint.
- **Non-Deterministic Behavior:** Outcomes may vary across runs, but average performance is often acceptable for large ships.

5.5 Performance Discussion

In empirical testing, the Efficiency Strategy localized the bot successfully in small and medium-sized ships, often within milliseconds. For larger ships (e.g., 25×25), it completed execution within the time limit, though sometimes leaving a small number of ambiguous cells unlocalized. The average computation time scaled linearly with grid size due to its lack of backtracking or re-expansion steps.

5.6 Comparative Insights

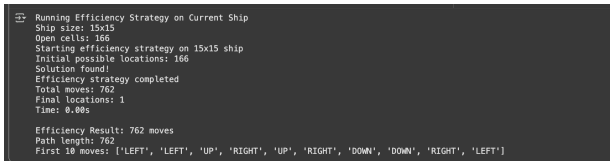
The Efficiency Strategy serves as a practical counterpart to the Optimality Strategy:

- **Optimality Strategy:** Exhaustive and theoretically minimal in moves, but computationally expensive.
- **Efficiency Strategy:** Fast and scalable, but not guaranteed to produce the fewest moves.

The trade-off is deliberate: by replacing informed search with stochastic state updates, the algorithm achieves near real-time localization suitable for large environments or embedded systems where response time is more critical than exact minimality.

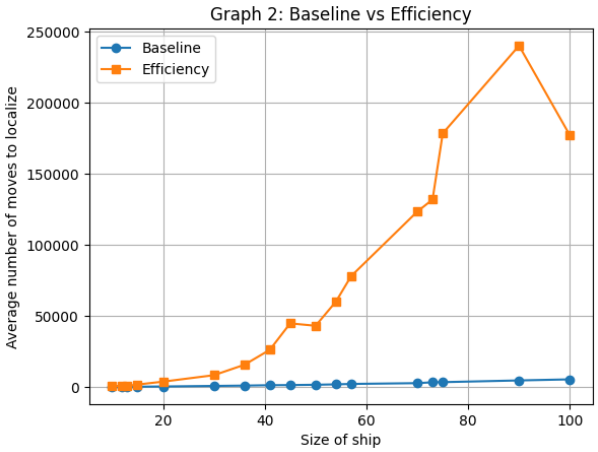
5.7 Summary

The Efficiency Strategy provides a computationally lightweight alternative that prioritizes speed and practicality. Although it lacks the theoretical guarantees of A*, it remains effective in most scenarios, making it well-suited for real-world deployment where computation time is limited. It represents the upper bound of performance in terms of speed, complementing the Optimality Strategy's lower bound in terms of move count.

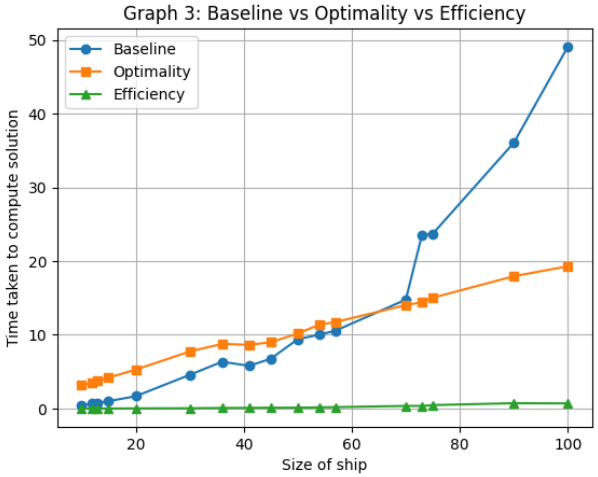


Output of Efficiency strategy

6 Graphical Representation: Baseline vs Efficiency Strategy



Graph 2: Baseline vs Efficiency



Graph 3: Baseline vs Optimality vs Efficiency

7 Worst-Case Starting Set Analysis

The final component of this project focuses on identifying **worst-case initial uncertainty sets**—that is, the smallest subsets of starting locations that cause the localization algorithms to take the longest to converge. Specifically, two types of hardness are analyzed:

- For the **Optimality Strategy**, the subset that requires the **maximum number of moves** to achieve localization.
- For the **Efficiency Strategy**, the subset that results in the **longest runtime** or slowest convergence.

7.1 Objective

The goal is to understand which spatial configurations of possible bot locations make localization inherently difficult, and why. This analysis helps characterize the strengths and weaknesses of both algorithms under challenging conditions.

7.2 Methodology

A separate search module was implemented to automatically find these hard subsets. The procedure consists of three main stages:

Stage 1: Candidate Generation

All open cells of the ship grid are collected as potential starting points. Small subsets (typically containing 2–6 cells) are generated using multiple heuristics:

- **Graph-Distance Sampling:** For the Optimality Strategy, cell pairs that are farthest apart (i.e., maximum shortest-path distance) are considered likely to be hardest.
- **Pattern-Based Sampling:** For the Efficiency Strategy, cells that share identical neighbor patterns (e.g., symmetric surroundings) are grouped into candidate subsets.
- **Room-Based Sampling:** Large connected open regions are identified through BFS, and samples are drawn to test ambiguous interior areas.

Stage 2: Evaluation of Candidates

Each candidate subset L is evaluated by executing the respective localization strategy:

- The **Optimality Strategy** is run with `max_states=5000` and `max_time=10`, recording the total number of moves until localization.
- The **Efficiency Strategy** is run with a fixed time cap (`max_time=30`), and its total runtime and move count are measured.

The hardest subset for each strategy is defined as:

$$L^*_{opt} = \arg\max_L(moves(L)), \qquad L^*_{eff} = \arg\max_L(time(L))$$

where L^*_{opt} maximizes the move count and L^*_{eff} maximizes execution time.

Stage 3: Hardness Pattern Analysis

Once the hardest subsets are identified, structural analysis is performed to determine why these particular configurations are difficult. Several spatial and topological features are examined:

- **Distance Between Cells:** Whether hard subsets are composed of cells far apart in the grid.

- **Symmetry:** Whether hard subsets lie in symmetric or identical local neighborhoods.
- **Connectivity:** Whether the regions containing the cells are narrow corridors or large open rooms.

7.3 Implementation

The implemented algorithm comprises the following components:

- `find_hard_starting_sets()` – orchestrates the hardness search for both strategies.
- `find_hard_optimal_set()` – computes all-pairs shortest paths among open cells, selects farthest pairs, and tests them using the Optimality Strategy.
- `find_hard_efficiency_set()` – groups cells by movement pattern similarity and open-area clustering, then tests random subsets using the Efficiency Strategy.
- `analyze_hardness_patterns()` – interprets the found subsets to explain their difficulty.

Each phase includes robust exception handling and time-bound execution to maintain stability across various ship sizes.

7.4 Observations

The analysis revealed two distinct forms of hardness:

- **Optimality Strategy:** The hardest cases involved two cells at opposite extremes of the ship (maximum Manhattan distance). These scenarios maximize the number of disambiguating moves required, as each observation must eliminate one of several long-distance hypotheses.
- **Efficiency Strategy:** The hardest cases occurred in highly symmetric or open regions where multiple positions yield identical sensory feedback. Randomized exploration in such spaces fails to make consistent progress toward localization, increasing computation time.

7.5 Key Findings

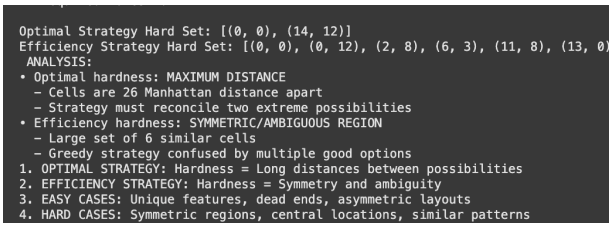
- **Long Distances** lead to hardness for heuristic-based algorithms (Optimality Strategy).
- **Symmetric Neighborhoods** and **Ambiguous Connectivity** degrade the performance of heuristic-free random exploration (Efficiency Strategy).
- **Dead Ends** and **Asymmetric Paths** produce easy cases, as these configurations provide strong disambiguating information early in the search.

7.6 Summary

The Worst-Case Analysis demonstrates that both localization strategies respond differently to spatial ambiguity:

- The **Optimality Strategy** struggles in environments where uncertainty spans long distances.
- The **Efficiency Strategy** struggles in environments with high symmetry and indistinguishable cells.

This analysis completes the exploration of trade-offs among optimality, efficiency, and robustness in the localization problem.



Output of Hard Set Analysis