Optimization and Analysis of the Frontier Selection Strategy

1. Analysis of the Original Strategy and its Limitations

Oscillation due to Rigid Priority

The original `determine_frontier_path` function used a **Lexicographical Sort** on the `max()` function to select the optimal frontier. This assigned **absolute highest priority** to `heading_projection`. Due to this rigid weighting, even minor superiority in `heading_projection` could cause the system to ignore a frontier with high overall value (e.g., better **proximity to the goal**). This led to **logic-induced oscillation** when the robot, e.g., at `seed = 243`, was positioned between similarly-rated frontiers, preventing exploration progress.

Inherent Flaw in Frontier Definition

The `detect_frontiers` function defines a frontier as a **free** cell adjacent to an **unknown** space. When the robot is approaching the final target (`goal_cell`), its sensors map the area, updating the status of `goal_cell` and its neighbors from 'unknown' to 'free'. This state change causes the frontier candidate pool (`frontiers` list) to become empty (`[]`). Consequently, the robot terminates prematurely as it finds "no valid frontier," even when the goal is reachable.

2. Detailed Explanation of the Improved Strategy

2.1 Improvement I: Adoption of a Weighted Sum Model

An initial attempt to prevent oscillation by applying a penalty to recently-visited points proved suboptimal, as the robot frequently changed its target instead of committing to one. This penalty concept was also contrary to the notion of `score_alignment`. We therefore shifted to a comprehensive **Weighted Scoring Function**, `get_weighted_score`.

```
def get_weighted_score(cell: Cell) -> float
best_frontier_cell = max( pool, key=lambda cell: get_weighted_score(cell) )
```

This model provides a Holistic evaluation mechanism, leading to a smoother and more stable decision-making process. A minor deficit in a frontier's heading_projection can be compensated by high scores in other dimensions, such as "proximity to goal" `score_goal_dist` or "appropriate distance" `score_robot_dist` . This mechanism ensures the robot consistently focuses on high-value targets, fundamentally suppressing oscillation by preventing unnecessary decision changes or reversals triggered by minor heading disturbances.

2.2 Improvement II: Introduction of Goal Compensation Logic

A new check mechanism was added specifically to handle the special scenario where the candidate pool (pool) is empty (i.e., detect_frontiers returns no valid frontier).

```
if not pool:    best_frontier_cell = goal_cell
else:   best_frontier_cell = max( pool, key=lambda cell: get_weighted_score(cell) )
```

Since the pool/frontiers list is empty only when the robot has completed the exploration of all reachable areas, setting `best_frontier_cell = goal_cell` is appropriate. This allows the system to directly invoke the A* search algorithm, planning the shortest path to the final goal, thereby preventing the robot from terminating prematurely at the finish line.

3. Conclusion: Current Limitations and Future Directions

The implementation of **Weighted Scoring** and **Goal Compensation Logic** has effectively resolved the oscillation and termination issues, significantly improving navigation robustness. However, the strategy retains two key areas for future optimization:

1. The current manually tuned weight in the weighted scoring function are performance-sensitive. Future work should adopt **Machine Learning** to systematically determine the optimal parameter set, achieving superior goal ranking and exploration efficiency.
2. The `detect_frontiers` function is flawed as it can erroneously identify **three-sided enclosed regions** as valid frontiers. This requires integration with **SLAM Map Optimization** and geometric filtering to ensure only frontiers leading to substantial unknown space are selected.