Samudrapath Documentation/Idea

Objective: We aim to develop a versatile and fast algorithm for ship routing specifically for the Indian Ocean region that optimizes fuel consumption while considering other important objectives, such as weather changes, travel time, route safety (including the avoidance of regions such as pirate threat zones, ecological areas, and other countries boundaries).

Algorithmic details:

We are utilizing a version of the A* algorithm called theta star using which we are calculating three major paths: **Safest path** - a path which is considered safety as the topmost priority, **Fuel efficient path** - a path which utilizes the least fuel, **Shortest path** - a path which is the shortest possible path between source and destination.

Why theta star?

Theta* is an extension of the well-known A* algorithm that avoids the problem of line-of-sight shortcutting by considering straight paths that can cut through obstacles. This modification makes it more efficient for environments with complex navigational constraints, such as ship routing in the Indian Ocean, where various factors (weather, safety zones, obstacles) must be considered. Theta* is suitable for this problem as it balances optimality and efficiency, providing faster pathfinding while ensuring safety and efficiency in fuel consumption.

Parameters considerations:

Start and Goal: The starting and goal positions are provided as grid coordinates. These represent the ship's departure and destination points, respectively.

Binary Map: Represents the navigable area, where obstacles are marked, and areas with potential movement are indicated.

Wind Speed and Direction (Wind Speed Map, Wind Angle Map): Maps of wind speed and wind direction in radians across the grid. Wind impacts the ship's speed and fuel consumption.
Wave Height Map: Represents the height of waves at each grid cell, affecting the ship's resistance and speed.

Ship Speed: The constant or typical speed of the ship used for calculating fuel consumption.

Lat/Long Parameters (lat_min, lon_min, lat_res, lon_res): These are used to convert grid-based coordinates to geographical coordinates (latitude and longitude).

Pirate Risk Map: A risk map indicating pirate threat levels in different regions. This affects route selection and, potentially, fuel consumption or detours.

Ecological Zone: The Indian Ocean contains several ecological zones where ships are restricted. In our model we make sure that these routes are avoided by ships

Constants (a, b, n_h, n_s, n_e, csfoc): Constants used for fuel consumption and resistance calculations. These parameters are based on empirical models or ship characteristics (e.g., ship's hull efficiency, engine specifications, fuel conversion factors).

How is this algorithm working?

Initialization:

- The algorithm starts by initializing an open list (min-heap), `came_from` dictionary (for path reconstruction), `fuel_score` dictionary (to store the fuel cost of reaching a node), and `total_time` dictionary (for the time spent on each path).

Heuristic Function:

- A heuristic based on the expected fuel consumption is used to guide the search. This heuristic considers the minimal fuel needed to travel from the current point to the goal point, taking into account the ship's speed and the distance between the two locations.

Pathfinding Loop:

- The algorithm iteratively pops the most promising node (based on fuel cost and heuristic) from the open list and explores its neighboring nodes.
- For each neighbor, the algorithm calculates the fuel consumption and time needed to travel there, considering environmental factors like wind, waves, and pirate risks.
- The algorithm then updates the fuel cost and time for each neighbor and adds it to the open list if a better path is found.

Resistance Calculations:

- The ship's resistance is calculated by considering various factors like wave resistance, wind resistance, and the ship's hull resistance using methods like Holtrop-Mennen.
- These resistances influence the ship's speed and fuel consumption.

Fuel Consumption:

- The fuel consumption is estimated using the total resistance and actual speed of the ship. This is done for each movement to ensure the path with the least fuel consumption is selected.

R_tot = R_t + R_aw + R_aa # Total resistance (N)

p_b = (R_tot * ship_speed) / (n_e * n_h * n_s) # Power required (W)

fuel_consumption = p_b * c_sfoc # Fuel consumption

fuel_cost = (fuel_consumption * distance) / V_a

Parameters:

- V_a # Effective speed of the ship (m/s), considering environmental factors
  - Va=Vo-(1.08*h-0.126*q*h+2.77*10^(-3)F*math.cos(alpha))*(1-2.33*10^(-7)*D*Vo)
    - Vo = hydrostatic speed of the ship in km/h
    - F = wind speed in knots
    - D = displacement of the ship in metric tons
    - h = significant wave height in meters
    - q = relative angle between ship's heading and wave direction in degrees
    - alpha = relative angle between ship's heading and wind direction in degrees

- R_t # Resistance from calm water (N)
  - R_t = (1 + k) * R_f + R_wb + R_add + R_rough + R_bulb

    Where

    - k is the frictional resistance correction factor
    - R_f is the frictional resistance,
    - R_wb is the wave-making and wave-breaking resistance,
    - R_add is the additional resistance,
    - R_rough is the roughness allowance and still air resistance,
    - R_bulb is the resistance due to the bulbous bow.

- R_aw # Added resistance due to waves (N)
  - R_aw = R_awr + R_awm
    - R_awr: Added resistance due to wave reflections (N)
    - R_awm: Added resistance due to ship motions at sea (N)

- R_aa # Added resistance due to wind (N)
  - R_aa = 0.5 * rho_air * C_wind * A_trans * V_wind**2 - 0.5 * rho_air * C_wind_0 * A_trans * V_ship**2
    - rho_air: Mass density of air (kg/m^3)
    - C_wind: Wind resistance coefficient at the given wind speed -
    - C_wind_0: Wind resistance coefficient at zero wind speed -
    - A_trans: Transverse projected area above the waterline including superstructures (m^2)
    - V_ship: Measured ship's speed over the ground (m/s) -
    - V_wind: Relative wind velocity at the reference height (m/s)

- n_h # Hull efficiency (dimensionless)
- N_s # Propeller efficiency (dimensionless)
- n_e # Engine efficiency (dimensionless)
- c_sfoc # Specific fuel oil consumption (g/kWh)

Path Reconstruction:

- Once the goal node is reached, the algorithm reconstructs the path by backtracking through the `came_from` dictionary.

Why theta star fails for multiple objective problems?

- **Single Objective Focus**: Theta* is primarily designed for single-objective optimization, typically the shortest or least-cost path. When extended to multiple objectives (e.g., minimizing fuel, avoiding dangerous regions, optimizing travel time), it doesn't natively handle trade-offs between different objectives. This limitation leads to suboptimal solutions when multiple competing goals are involved.
- **Lack of Flexibility**: In real-world applications, objectives like safety (avoiding pirate regions or ecological zones), environmental factors (weather), and time constraints might conflict with each other. Theta* lacks a built-in mechanism for balancing these objectives dynamically, which is crucial for problems like ship routing.
- **Complex Constraints**: For a problem like ship routing, which involves diverse constraints (pirate risk, ecological areas, etc.), Theta* might find it challenging to adjust the pathfinding efficiently across all constraints simultaneously. The algorithm does not inherently support multi-objective optimization or constraint prioritization, limiting its effectiveness for such problems.

Risk Calculation

Total risk=risk_wind+risk_wave+risk_current

risk_wind=u10/u10max

Here u10 max is the maximum wind speed a ship can handle above 10m above ship surface

- risk_wave=total resistance of wave(formula given above)

Risk current based on resonance theory of a ship in waves, the ship is in the harmonic when

Rolling area is between **0.70<(tq/te)<1.3.** In this area a ship may have a large roll angle, threatening its safety

Risk_wave = {

tq/te :when it is in between 0 to 1

2-tq/te :when it is between 1 and 2

2 when :greater than 2

}

tq is natural rolling period

Inexperience user problem in weighted cost function optimization:

Initially, we considered using the *Theta algorithm** to optimize a weighted cost function based on user preferences. This approach aimed to find the most optimal path by combining multiple preferences (e.g., fuel consumption, travel time, and safety). However, one major issue we encountered was that the success of this approach heavily depended on the user-provided weights. For an inexperienced user, assigning appropriate weights to each preference could lead to poor path selection and suboptimal results.

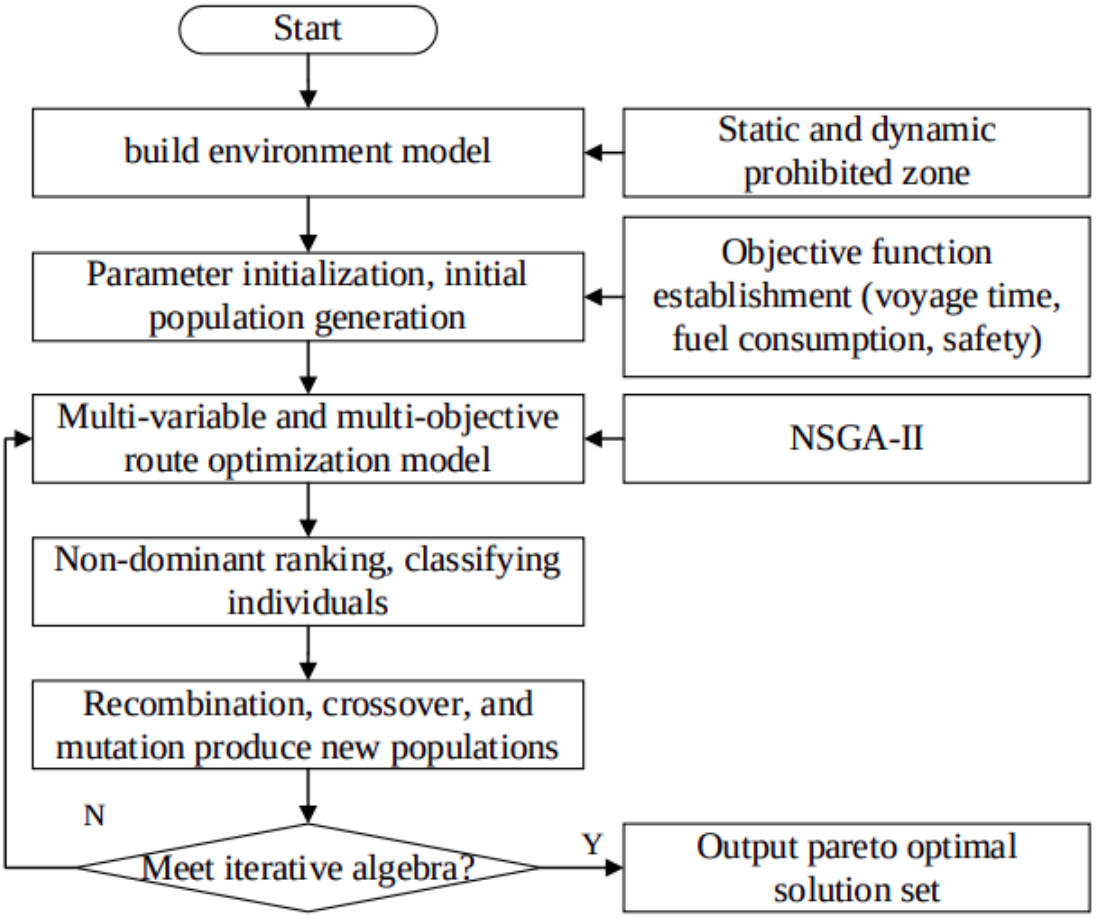
Genetic Algorithm (NSGA-II):

After thoroughly evaluating the **Theta*** algorithm and reviewing several research papers, we found that **Theta*** struggles to find optimal paths when optimizing for multiple objectives simultaneously. In our case, we needed to balance **fuel consumption**, **Passenger Safety and travel time**—three major conflicting objectives.

As a result, we decided to shift to a **Genetic Algorithm (GA)**, specifically the **NSGA-II** (Non-dominated Sorting Genetic Algorithm II), which is designed to handle multi-objective optimization problems more effectively.

We are using the calculated path from **Theta* algorithm** to generate the **initial population** for the Genetic Algorithm**.** After generating the initial population, we used different operators such as Selection, Crossover, and Mutation.

Using **NSGA-II** allowed us to explore and optimize multiple objectives without the need for a predefined weight system, making the algorithm less dependent on user input. This approach provided a more robust solution, especially for users with limited experience in defining preferences.



Parameters Used in Genetic Algorithm:

- **individual**: List of (lat, lon) tuples representing the route.
- **V0**: Hydrostatic speed of the ship (knots).
- **transform**: Affine transform of the raster.
- **wind_dir_data**: 2D numpy array of wind directions (degrees from north).
- **wind_speed_data**: 2D numpy array of wind speeds (knots).
- **wave_height_data**: 2D numpy array of wave heights (meters).
- **wave_dir_data**: 2D numpy array of wave directions (degrees from north). If None, it is handled in speed calculation.
- **D**: Actual displacement of the ship (tons). If None, a random value is generated per segment.
- **a1**: Weight for wind risk.
- **a2**: Weight for wave risk.
- **Fb**: Freeboard of the ship (meters).
- **a**: Fuel consumption coefficient.
- **b**: Fuel consumption exponent.
- **u10max**: Maximum wind speed (knots). If None, it will be calculated.

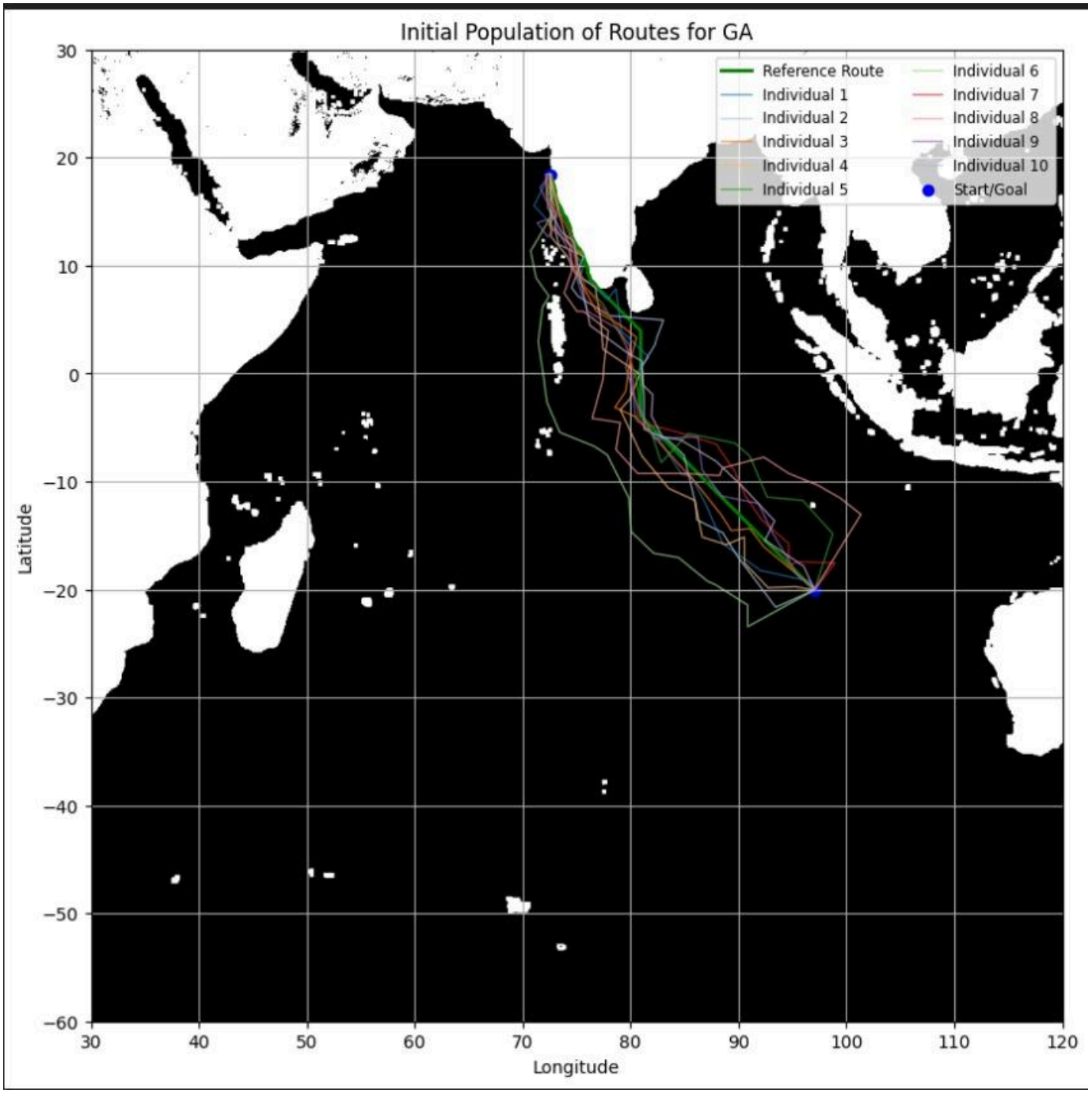
How is our Genetic Algorithm working?

Dynamicity of ship types

We are considering different types of ships, such as Passenger, Cargo, and Fishing ships, to calculate ship safety, fuel consumption, and travel time. We take into account various details for each ship, including their weights, frontal area, hull efficiency, propeller efficiency, and shaft engine efficiency, water displacement, and height of the ship.

Initial Population Creation

We are using a path found from theta* to create an initial population. We are taking random waypoints by taking some deviation from the path found by theta*, and then we are finding different paths from source to destination along these waypoints to create several paths as an initial population. By taking waypoints we are making sure that waypoints do not lay in land or ecological areas.



Crossover

Crossover is a method in which we select two parents based on their fitness. We combine some characteristics of each parent to create offsprings, offspring which are better than their parent, replace parents in a population, in this way we our population get better by better in each generation.

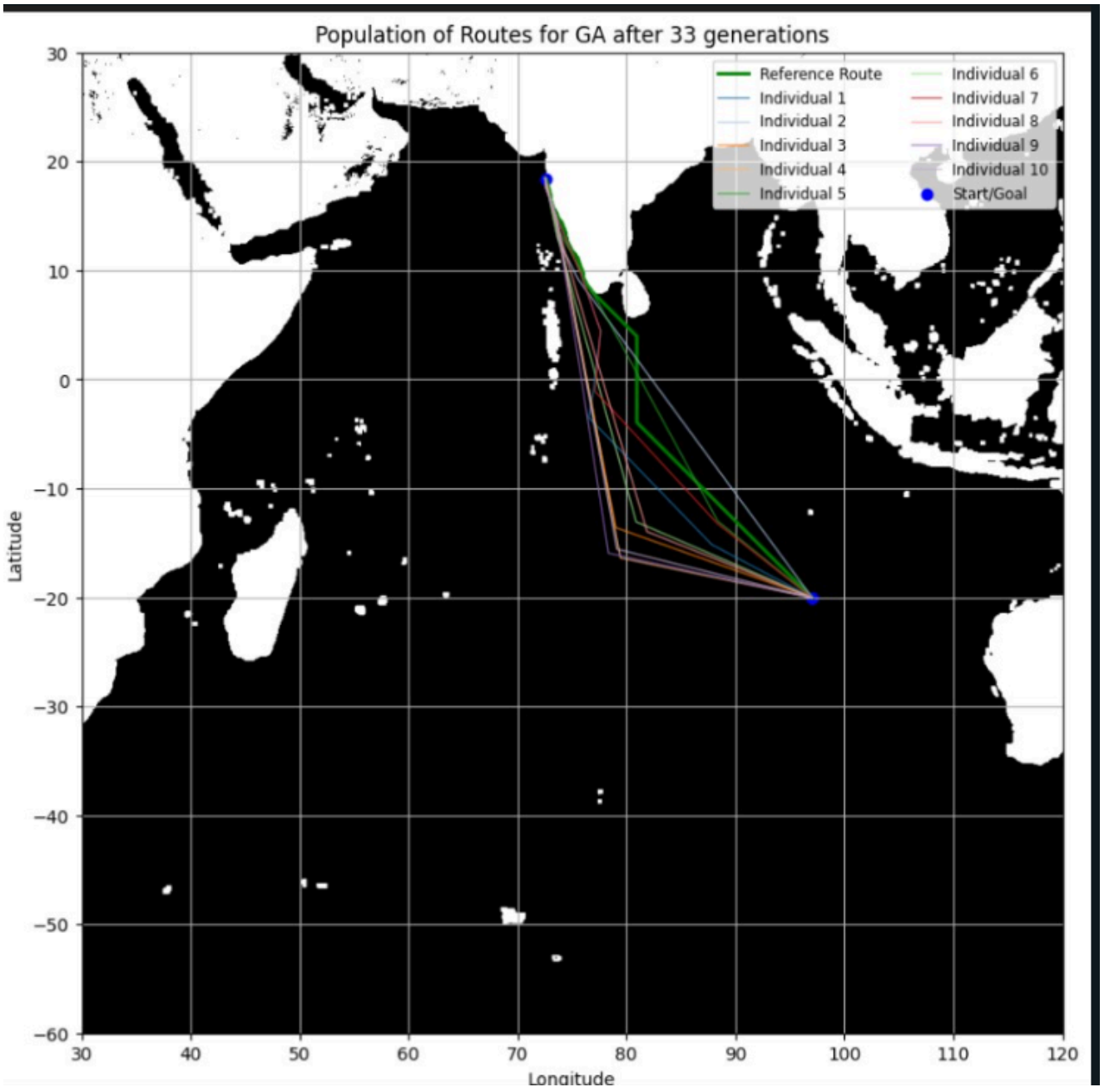
In our implementation, the function selects a random crossover point in the first parent's path. It then finds the closest corresponding point in the second parent's path using the `find_closest_point` function.

Before creating child paths, the operator checks if the crossover can be performed without creating invalid segments. It uses the `is_path_clear` function to verify that the new connections between path segments are obstacle-free.

If the crossover is valid, two children are created:

- The first child combines the first parent's initial segment with the second parent's terminal segment
- The second child does the opposite, taking the second parent's initial segment and the first parent's terminal segment

If the crossover creates an invalid path, the original parent paths are preserved unchanged. This ensures that only valid path combinations are generated during the evolutionary process.



As you can see after 33 generations our population has started converging toward a more optimized path.

Mutation on Waypoints

Just like in nature, mutation is a deviation of characteristics from both parents, which can be possibly good or even bad, but this allows the child to be different in some characteristics from both parents. This makes sure that the population does not converge in local optima.

In our implementation of each point in the path (excluding the start and end points), the function applies a Gaussian mutation. It generates random x and y offsets using a normal distribution with a standard deviation of sigma. These offsets are added to the coordinates of a randomly selected point in the path.

The mutation is only accepted if the new point maintains a clear path to both its neighboring points, which is checked using an `is_path_clear` function. This ensures the mutated path remains valid within some predefined space or obstacle map.

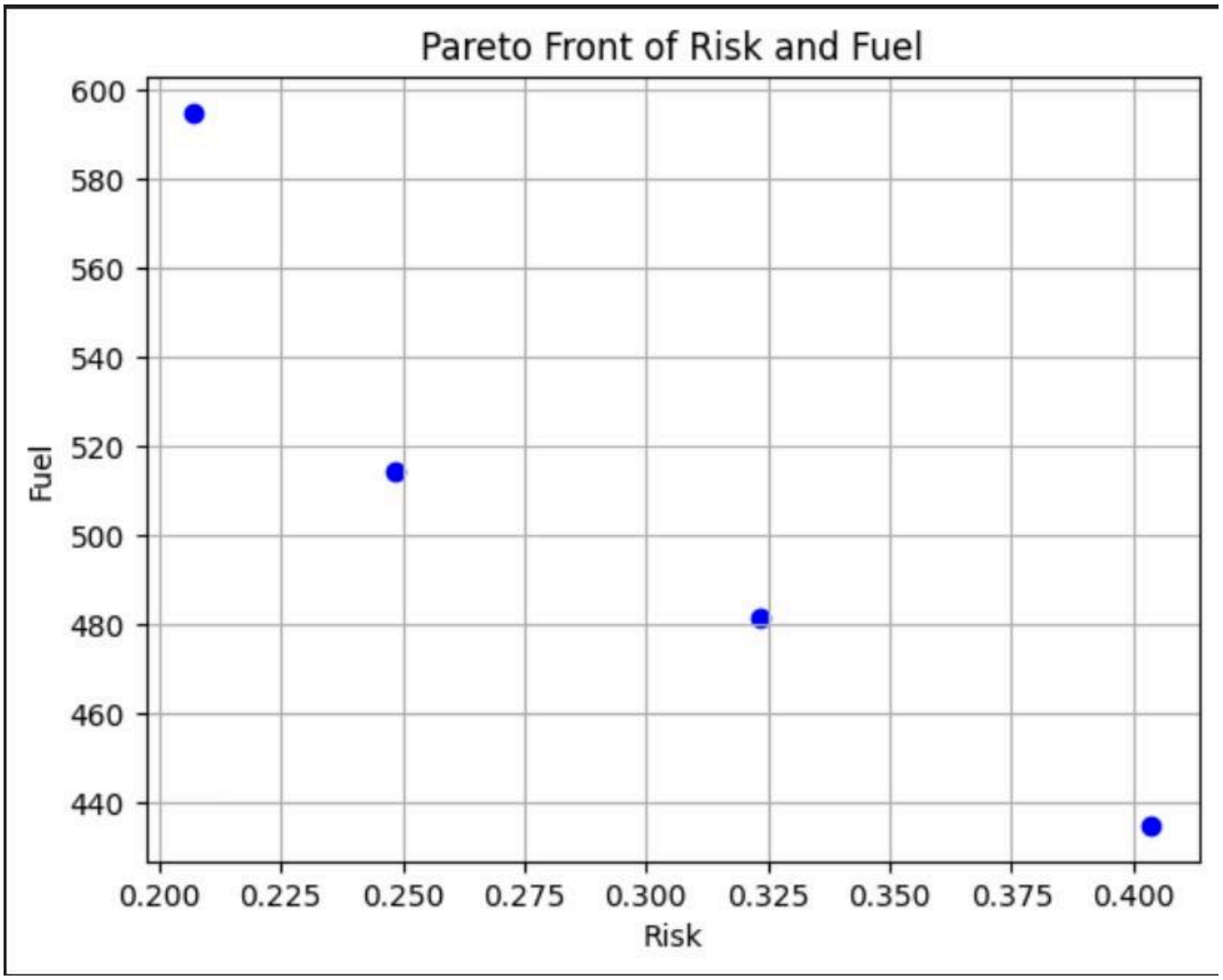
Additionally, there's a 10% chance of removing an intermediate point from the path if doing so creates a direct, clear line between its neighboring points. This allows for path simplification.

The function returns a mutated copy of the original path, preserving the original while creating a slightly modified version that could potentially improve the path's characteristics.

Pareto Front

Pareto front represents the set of non-dominated solutions in multi-objective optimization, where no solution can be improved in one objective without degrading performance in at least one other objective. It visualizes the best trade-off solutions, showing the optimal points where improving one objective necessarily comes at the cost of worsening another.
Process:

- Evaluate all solutions across multiple objectives
- Identify non-dominated solutions.
- Rank solutions into different non-domination levels.
- Calculate crowding distances within each front.
- Select the best solutions that represent optimal trade-offs between objectives.



Pareto Front obtained after 100 generations, producing 4 non-dominated solutions.

Dynamic Route Optimization by Incorporating Forecasted Data Every 24 Hours

Given the extended duration (usually more than a day) of maritime voyages, static route planning can become rapidly outdated due to evolving environmental conditions and may result in miscalculations of fuel, time, and risk factor.

To counter this we recalculate a new path from the forecasted data for the next 24 hours that we fetch the data by leveraging the Open Meteo API (https://open-meteo.com/), and we systematically refresh route calculations every 24 hours with the latest predictive weather data. This methodology enables:

- Precise fuel consumption modeling
- Accurate time estimation
- Comprehensive risk assessment
- Adaptive routing strategy

Advantages of Genetic Algorithm over Theta* Algorithm:

Multi-Objective Optimization:

- **GA** can handle multiple conflicting objectives simultaneously (e.g., minimizing fuel consumption, optimizing travel time, and avoiding dangerous regions).Theta* is primarily designed for single-objective optimization, such as finding the shortest or least-cost path.

Flexibility in Problem Representation:

- **GA** can be adapted to a wide variety of problem domains by adjusting the genetic representation (e.g., chromosomes) and fitness functions. **Theta*** requires a more rigid grid-based structure and is not as flexible in representing complex or dynamic environments.

Ability to Handle Complex Constraints:

- **GA** is well-suited to problems with complex constraints (e.g., avoiding pirate-infested areas, ecological zones, or geopolitical boundaries) and can dynamically adjust to varying conditions, such as changes in weather or environmental hazards. **Theta*** might struggle when there are numerous and varied constraints.

Exploration of Multiple Solutions:

- **GA** explores a population of solutions across generations, maintaining diversity and potentially finding better trade-offs among conflicting goals. **Theta*** only explores a single path at a time, which may result in getting stuck in local minima and not finding the best global solution.

Global Search Capability:

- **GA** has a global search approach, capable of exploring a larger search space in parallel and potentially finding better solutions in more complex, multi-dimensional spaces. **Theta*** tends to focus on local search, which can limit its ability to find optimal solutions when faced with complex, non-linear objective functions.

Adaptability to Dynamic Environments:

- **GA** can adapt to changes in the environment during the search process. For example, if new obstacles (e.g., pirates, storms) are detected, **GA** can evolve new solutions based on updated information. **Theta*** typically requires

recalculation from scratch if the environment changes, which is computationally expensive.

Robustness to Noise:

- **GA** can continue to perform well even in noisy or uncertain environments by exploring multiple solutions and not relying on a precise pathfinding mechanism. **Theta*** may not perform as well when faced with uncertain or incomplete information, as it strictly follows its heuristics.

Scalability:

- **GA** can handle very large and complex search spaces more effectively because of its population-based search. **Theta*** can struggle with scaling, especially when the search space grows exponentially, requiring more memory and time.

References:

A multi-objective routing optimization model for ship intelligent navigation
https://iopscience.iop.org/article/10.1088/1742-6596/1684/1/012115/pdf

Multicriteria Ship Route Planning Method Based using Genetic Algorithm
https://www.mdpi.com/2077-1312/9/4/357