Introduction

Modern practical optimization problems are too often complex, nonlinear, large-dimensional, and sometimes dynamic making gradient-based and convex optimization methods too inefficient. In this paper, we present a formulation of a dynamic vessel-to-vessel service ship scheduling problem. In a span of several hours, the service ship must visit as many moving vessels as possible and complete the trip in as small a travel time as possible. Thus, the problem is bi-objective in nature and involves a time-dependent traveling salesman problem. We develop a level-wise customized evolutionary algorithm to find multiple trade-off solutions in a generative manner. Compared to a mixed-integer programming (MIP) algorithm, we demonstrate that our customized evolutionary algorithm achieves similar quality schedules in a fraction of the time required by the MIP solver.

DV2VRP

A service ship must leave from the harbor and simultaneously:
- Maximize the number of different target ships visited (α)
- Minimizing the total distance traveled (d)
and finally return to the harbor within a predefined time window \( T_w \).

Our approach is composed of three levels:
1. \( \alpha \)-level: Defining the subproblem and sequence length (\( \alpha \))
2. Upper level: Custom GA for optimizing routes given \( \alpha \)
3. Lower level: Optimizing schedules given a route by using dynamic programming

Designing routes for a given \( \alpha \) gets increasingly more complex as the number of ships through the working area increases. In our data-set there were 63 distinct ships passing through the working area.

\( \alpha \)-level

Each \( \alpha \)-level is a bi-level subproblem with sequences of length \( \alpha \). The upper level responsible for designing optimal routes and a lower level designing optimal schedules. To advance to the next \( \alpha \)-level we define a transition function to increase \( \alpha \).

The transition function selects a window of size \( n \) from an existing schedule, denoting which ships should be replaced. Then it generates a permutation of size \( k \) from target ships available within the associated time window. In the example above \( n = 1 \) and \( k = 2 \), thus we replace the subsequence [32] with a new subsequence [32, 30] resulting in \( \alpha \) increasing from 3 to 4.