Maskooki, Alaleh; Deb, Kalyanmoy; Kallio, Markku

A Customized Genetic Algorithm for Bi-objective Routing in a Dynamic Network

Published in:
European Journal of Operational Research

DOI:
10.1016/j.ejor.2021.05.018

Published: 01/03/2022

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
CC BY-NC-ND

Please cite the original version:
https://doi.org/10.1016/j.ejor.2021.05.018
Decision Support

A customized genetic algorithm for bi-objective routing in a dynamic network

Alaleh Maskooki, Kalyanmoy Deb, Markku Kallio

University of Turku, VESILINNANTEE 5, Turku FI-20014, Finland
Michigan State University, East Lansing, MI 48824, USA
Aalto University School of Business, Ekonominaukio 1, Espoo FI-00076, Finland

Abstract

The article presents a proposed customized genetic algorithm (CGA) to find the Pareto frontier for a bi-objective integer linear programming (ILP) model of routing in a dynamic network, where the number of nodes and edge weights vary over time. Utilizing a hybrid method, the CGA combines a genetic algorithm with dynamic programming (DP); it is a fast alternative to an ILP solver for finding efficient solutions, particularly for large dimensions. A non-dominated sorting genetic algorithm (NSGA-II) is used as a base multi-objective evolutionary algorithm. Real data are used for target trajectories, from a case study of application of a surveillance boat to measure greenhouse-gas emissions of ships on the Baltic sea. The CGA’s performance is evaluated in comparison to ILP solutions in terms of accuracy and computation efficiency. Results over multiple runs indicate convergence to the efficient frontier, with a considerable computation speed-up relative to the ILP solver. The study stays as a model for hybridizing evolutionary optimization and DP methods together in solving complex real-world problems.

1. Introduction

The traveling salesman problem (TSP) has become a widely studied classical routing problem (Applegate, Bixby, Chvatal, & Cook, 2006; Lawler, Lenstra, AHG, & Shmoys, 1985; Reintel, 1994) because of the frequent occurrence of its variants both in theory and in practical applications. Dynamic TSP (DTSP) is a class of problems with time-varying characteristics; it reflects the adaptation of the TSP to many real-world applications of routing problems. The dynamic features in the DTSP may refer, for instance, to target demand (Bertsimas, 1992; Smith, Pavone, Bullo, & Frazzoli, 2010), time windows for visits (Pavone & Frazzoli, 2010), delivery of goods to a distribution system (Archetti, Fiebig, Mor, & Speranza, 2020; Klapp, Erera, & Toriello, 2018), target locations (Bertsimas & Ryzin, 1991; Hammar & Nilsson, 2002; Helvig, Robins, & Zelikovsky, 2003), and edge costs (Laporte, Louveaux, & Mercure, 1992; Secomandi, 2003; Toriello, Haskell, & Poremba, 2014). Prior work includes surveys of dynamic vehicle-routing problems (Flatberg, Hasle, Klaster, Nilsson, & Riise, 2007; Pillac, Gendreau, Guéret, & Medaglia, 2013).

This article pertains to the Moving-Target TSP (MT-TSP), which goes back to the problem introduced in 1993 by Kryazhimskiy and Savinov (1993). Their work was followed by that of Helvig, Robins and Zelikovsky (2003), who developed an exact algorithm based on dynamic programming for a number of MT-TSPs where the targets move with constant velocities along straight lines toward or away from the origin. Their objective was to find the fastest tour, starting and ending at the origin, that intersects all targets. They showed for this particular MT-TSP that waiting does not occur in an optimal solution of minimizing travel time. However, waiting may be optimal in models with time windows when the vehicle arrives at a location before that location’s time window opens (Vu, Hewitt, Boland, & Savelberg, 2020). We will show that, in the case of minimizing travel distance, waiting can be beneficial even within the time window. Several other studies (Chouhey, 2013; Hammar & Nilsson, 2002; Hassoun, Shoval, Simchon, & Yedidsion, 2020; Helvig, Robins & Zelikovsky, 2003; Jiang, Sarker, & Abbass, 2005; Moraes & Freitas, 2019) have considered the particular case of MT-TSP defined by Helvig, Robins and Zelikovsky (2003), which requires intercepting all moving targets under the assumption that targets move linearly at constant velocities. However, in recent literature the definition of the MT-TSP is...
generalized to a variant in which the target locations can be defined more freely. Recently, Cambella, Naoum-Sawaya, and Ghadar (2018) addressed a dynamic vehicle-routing problem in which the pick-up locations of the targets are non-stationary. They proposed a mixed-integer second-order cone program formulation for the problem, along with valid inequalities for strengthening the continuous relaxation.

Although there is a vast body of work on the DTSP, the MT-TSP variant is less frequently addressed in the literature. Nevertheless, several studies of the MT-TSP do exist, mostly ones of real applications with problem-specific assumptions about the intrinsic features of the target dynamics. For instance, for the on-orbit servicing model by Bourjolly, Gurtuna, and Lyngvic (2006), the trajectories of satellites (targets) are determined by orbital mechanics; an exhaustive search suffices for solving the problems since the number of targets is small. Regarding methods based on genetic algorithms (GAs), Groba, Sartal, and Vázquez (2015) proposed a GA for the retrieval of fish-aggregating devices attached to floating buoys whose movements are predicted by Newton’s motion equation, to assess target dynamics. They extended their model for the multiple traveling salesman problem with moving targets to be applied for multiple tuna vessels (Groba, Sartal, & Vázquez, 2018). Viel, Vautlier, Wan, and Jaulin (2019) extended the GA (Groba, Sartal & Vázquez, 2015) to take into account wind direction and wind speed for purposes of a fleet of sailboats picking up buoys on the sea. Other work Li, Yang, and Kang (2006); Zhou, Kang, and Yan (2003) considers a DTSP wherein the number of targets in addition to their locations changes with time. The time discretization is applied and evolutionary algorithms are proposed for finding a sequence of solutions to a static TSP within each time slot, which makes the class of problem different from the one discussed in this paper. Several ant colony optimization methods exist for solving an ordinary single-objective dynamic vehicle-routing problem (Gao, Wang, Cheng, Inazumi, & Tang, 2016; Xu, Pu, & Duan, 2018) by using standard crossover and mutation operators, which can be too generic for solving large-scale problems. Moreover, handling a multi-objective moving-target version of these problems is computationally expensive. In all MT-TSP cases cited above, a single criterion is optimized subject to a requirement of visiting all targets in a given set. Research on evolutionary multi-criterion optimization (EMO) methods has progressed significantly since the early 1990s in designing efficient algorithmic methods and measuring the quality of approximations, as well as in hybridization with other strands of optimization (Ehrgott, Fonseca, Gandibleux, Hao, & Sevaux, 2009). The EMO methods are flexible for customization and have strong adaptation capabilities for solving complex problems, such as DTSPs. Their operators can be modified with knowledge from past environments (Chowdhury, Marufuzzaman, Tunc, Bian, & Bullington, 2019; Mavrovouniotis, Müller, & Yang, 2016; Viel, Vautlier, Wan & Jaulin, 2019). They can be infused with problem-specific local search methods in a hybrid manner. Furthermore, their population approach allows facilitating implicit parallel search in exploring multiple good search regions (Goldberg, 1989; Holland, 1975). A survey of evolutionary dynamic optimization is provided by Nguyen, Yang, and Branke (2012).

The problem addressed in our case study arises from a real-world application of a surveillance boat measuring greenhouse-gas (GHG) emissions of ships navigating in a specific area of the Baltic sea, referred to as the work area. We introduce a general case of the MT-TSP with the following additional features: (i) The number of targets changes over time. (ii) Targets have time windows within which they can be visited; this time starts when the ship enters the work area and ends when it leaves the area. (iii) The trajectories of the moving targets (ships) and their varying velocities can be defined arbitrarily. (iv) The total number of targets α (to be visited) is endogenous since visiting all targets is not possible given the time windows of targets, time horizon, limited work area, and surveillance-boat speed. (v) The problem is bi-objective; the number of measurements α is chosen by the decision-maker on the basis of the Pareto frontier, which shows the shortest possible travel distance at each level of α. Features (i) to (v) distinguish our problem from the published works on the MT-TSP discussed above. Case studies of maritime surveillance are reported elsewhere also (Groba, 2006; Marlow, Kilby, & Mercer, 2007). In the surveillance routing problem, a patrol aircraft is equipped with close-range sensors, and the positions of ships are discovered as the route is flown. The speed and the number of ships in the work area vary over time in a manner unknown beforehand. The course and speed of a ship is determined only when it is located within the radar detection range of the surveillance aircraft. The aim is to find a tour with the objective of detecting the maximum number of ships such that it optimizes the cost of servicing, such as travel time or other quality metrics. The problem settings in the two studies mentioned above are quite different from ours. Using simulated scenarios for ships, the authors proposed on-line search heuristics adapted for the dynamic environment under various assumptions, some of which differ from those in our case. Were predictions of ship trajectories available, our approach could provide an efficient routing solution addressing part of the problem considered in those two studies. Bullo, Frazzoli, Pavone, Savla, and Smith (2011) provided a survey of adaptive algorithms that enable real-time task allocation and dynamic vehicle routing, motivated by application for unmanned aerial vehicles with random target locations and demands. Our GA approach does not employ an on-line algorithm; our goal is to find efficient routes in a dynamic network when given a priori knowledge of the target trajectories, by means of predictions. Prior work offers classification of the literature on vehicle-routing problems (Eksioglu, Vural, & Reisman, 2009).

It is worth noting also that Jaillet (1988) considered a DTSP with n targets where the number of targets to be visited is a random variable α with a known probability distribution; i.e., given a realization α of α, with α ≤ n, not all n targets need be visited. The problem is to find a complete tour (intercepting all n targets) determining the order of visits in each sub-tour (of α targets) such that the expected travel distance of sub-tours is minimized. Instead, we consider a case of endogenous α.

The primary contribution of the research presented in this paper is to propose a customized genetic algorithm (CGA) for solving the extended MT-TSP case (characterized by items i–v above) in a computationally fast manner, so that the approach can be of practical use. Our CGA is a new hybrid evolutionary optimization method combining operators of a genetic algorithm with concepts adopted from dynamic programming (DP) (Bellman, 1957) to approximate the efficient frontier under two criteria. Two customized operators are designed for mutating the offspring after the GA’s crossover operation. We propose two variants of DP-based approaches (operators), one for generating initial feasible tours and the other for recovery from possible infeasibility in offspring tours. To sort non-dominated solutions, NSGA-II (Deb, Pratap, Agarwal, & Meyarivan, 2002) for evolutionary multi-criterion optimization is used as a core algorithm. We also formulate an exact integer linear programming (ILP) model, which is a simplification of the model introduced in prior work (Maskooki & Nikulin, 2020). The simplified model allows use of fast DP-based techniques. In real-world applications, exact methods often become prohibitively costly because of the huge number of binary variables. The CGA finds effi-

---

1 The time-ordered set of locations of a dynamic system.
cient alternatives for the number of targets $\alpha$ (to be visited), as well as the time and location of each visit. For our case study, target trajectories are predicted via a $k$-nearest neighbor method (Virjonen, Nevalainen, Pahikkala, & Heikkonen, 2018) over a 16-hour time horizon. In another contribution, we show that the DP-based approach generates fast and high-quality solutions with potential to be used stand-alone or in connection with other heuristics for solving large-size vehicle-routing problems.

The discussion proceeds as follows. In Section 2, the bi-criteria MT-TSP problem is defined and the ILP model for finding the efficient frontier is introduced. Section 3 presents the steps of the CGA for finding the efficient frontier. Then, with Section 4, we describe results from the CGA for a case study using real data sets, and we compare the results with those obtained from ILP solutions. Section 5 summarizes our conclusions.

2. A model for routing in a dynamic network

Reduction of GHG emissions from ships is a key topic for the Marine Environment Protection Committee of the International Maritime Organization (Finnish Governme, 2020). Regulations require Finland to reduce the GHG emissions of its sectors involved in the effort by a minimum of 39% from 2005 levels by 2030 (Ministry of Economic Affairs & Employment, 2020). The environmental measurements considered in this article are intended to meet the requirements of GHG regulations for Finnish marine traffic. Our study worked with the company in charge of developing an application for a surveillance boat’s measurement of SO$_2$, CO$_2$, and NO$_x$ emissions from ships. The boat can perform mobile measurements when it is in the vicinity of ships; the exhaust-gas trail is measured from a distance of 200–300 m from the target ship in a process that takes approximately 2–3 minutes. Throughout the process, the target ship and the surveillance boat stand almost still. The speed of the ships (targets) ranges from 8 to 25 knots, with an average of 13 knots. The preferred speed of the surveillance boat is 20 to 30 knots (46.3 km/h); however, it is allowed to increase its speed for short periods. The work area is approximately $27 \times 32$ nautical miles. Marine traffic is predicted daily, up to 16 hours in advance.

The goal for our task is to optimize the route (tour) of the surveillance boat by maximizing the number of ships subject to measurement ($\alpha$) and minimizing the total travel distance ($z$) within a working day of one to two shifts (8–16 hours). The two objectives are defined in collaboration with the company in question, with the choice of objectives being supported by two observations. Firstly, there is a trade-off preference between $\alpha$ and $z$ that reflects the perceived value of a single measurement among thousands of ships to be inspected annually. Therefore, increasing $\alpha$ by one further ship is not justified if the cost is too high. Secondly, the route optimization is based on predicted locations of ships. Implementation of such an off-line-calculated optimal tour in practice with the actual ship locations may result in a risk of traveling extra distances, which grows with increasing $\alpha$ since the schedule for large $\alpha$ becomes time-wise tight. For a comprehensive treatment of risk assessment related to the prediction uncertainty that may affect the optimal solutions obtained from the route-optimization model discussed in Section 2, see Maskooki, Virjonen, and Kallio (2020).

We formulate the problem as a network flow model over a layered graph, where each layer corresponds to a (discrete) time slot. The nodes correspond to locations of ships, and edges to feasible transitions of the surveillance boat from one node (location) to another, both depending on the time slots of transition. One unit of flow passing through the network is defined as a tour of the boat that starts at the harbor (depot), visits a number of ships, and returns to the harbor. In all feasible tours, each ship is visited once at most. Fig. 1 shows a schematic illustration of a Pareto-optimal itinerary found by solving the dynamic ship-scheduling problem described above.

Consider a time horizon $[0, T]$ over $T$ working hours of the surveillance boat. The prediction of ship trajectories is carried out before the start of the planning horizon for the full duration, $T$ hours. Regarding how the predicted locations are calculated, we explain the method briefly in Section 4.1. Assume that a set of $n$ ships $N = \{1, 2, \ldots, n\}$ is predicted to appear in the work area during $[0, T]$. Let index $i$ refer to the ships and $i = 0$ refer to the depot. A standard approach for modeling the problem as a combinatorial time-dependent TSP is time discretization. We subdivide the time horizon into $m$ time slots (intervals) of equal length $w$ such that $T = mw$. For $k = 1, \ldots, m$, time slot $k$ is the interval $[k−1,w)$ with starting time $s_k = (k−1)w$. Let $k = 0$ refer to the initial time $0$ and $S_0 = 0$. In the model, the location of each ship $i \in N$ is fixed at the predicted location over each time slot $k$ given by the coordinate vector $v_{ik}$. The length of time slots $w$ is chosen to be short enough not to allow more than one ship being processed in a single time slot. The length $w$ is chosen to take into account the average speed of ships. A shorter length indicates more accurate locations in the optimal plan but also more time slots within the given time horizon and, as a result, a more complex model to solve. The choice of $w$ is based on a trade-off between accuracy and computational complexity.

For $i \in N$, let $S_i$ denote the set of time slots for which ship $i$ is in the work area. The depot $i = 0$ is present in all time slots $k$; hence, $S_0 = \{0, 1, \ldots, m\}$, and the surveillance boat can set off from and return to the depot in any time slot to complete the tour. Let $p_i$ denote the service time that ship $i$ requires for performing emissions measurement. In line with current practice, we assume $p_i = p$, for all $i \in N$; for the depot, we have $p_0 = 0$. Given locations $v_{ik}$ and $v_{jk}$ of, respectively, ship $i$ in time slot $k$ and ship $j$ in time slot $l$, the distance from $v_{ik}$ to $v_{jk}$ is $d_{ik} = \|v_{ik} - v_{jk}\|$ and the travel time from $v_{ik}$ to $v_{jl}$ with constant speed $c$ is $t_{ik}^l = d_{ik}/c$. A notation list is provided in Appendix A. For the given time slots $k = 0, 1, 2, \ldots, m$ and ships $i \in N \cup \{0\}$, we define nodes $v_{ik}$ only if $i$ is present in time slot $k$ (i.e., $k \in S_i$). A directed arc $(ik, jl)$ indicates Fig. 1. Schematic representation of the bi-objective routing problem in a dynamic network. Circles show the locations of ships moving in the work area within the time horizon. The dashed line denotes the optimal route of the surveillance boat, starting at the harbor, visiting five ships in the sequence of optimal time slots determined by the dynamic ship-scheduling model, and returning to the harbor: (harbor) $\rightarrow$ (ship1, time10) $\rightarrow$ (ship7, time23) $\rightarrow$ (ship2, time24) $\rightarrow$ (ship6, time26) $\rightarrow$ (ship5, time29) $\rightarrow$ (harbor, time30).
possible travel from node \( v_{ik} \) to \( v_{jl} \); it is defined only if \( i \neq j \) (each

\( s_k = (k-1)w \)

is the starting time of time slot \( k \); we find that traveling along arc

\( (ik, jl) \)

is not feasible if \( s_k + p_i + t_{il}^j \geq w \), so arc \( (ik, jl) \) is omitted from the network (in the preprocessing phase). For admissible arcs

\( (ik, jl) \)

, we require \( s_k + p_i + t_{il}^j \leq w \). For \( s_k \leq t < w \), assume that

\( t \)

is the starting time of the measurement for ship \( i \) in time slot \( k \). If \( t + p_i + t_{il}^j \leq w \), then the service for ship \( j \) starts in time.

For \( t = s_k \), this is implied by admissibility. For \( t > s_k \), the time of arrival

\( t + p_i + t_{il}^j \)

at location \( v_{jl} \) may be delayed beyond time slot \( l \); however, the delay is no more than \( t - s_k < w \); the delay is always shorter than one time slot. To avoid such delay, we may redefine

\( s_k = w \)

to the point of the end of time slot \( k \), for all \( k \), so that the revised admissibility conditions \( s_k + p_i + t_{il}^j \leq w \) always guarantee in-time service for all ships along the tour. We abandoned this option, however, because it is unnecessarily restrictive and leads to loss of efficiency. Instead, we rely on the hypothesis that small delays (of less than one time slot) can always be avoided via occasional minor adjustment in the speed of the surveillance boat during the process of implementing an itinerary.\(^1\)

Proceeding from the conditions above, we define the set of admissible arcs \( \Xi \) in the network as follows:

\[
\Xi = \{(ik, jl) \mid i, j \in N \cup \{0\}, i \neq j, k \in S_i, l \in S_j, k < l, s_k + p_i + t_{il}^j < w\}.
\]

(1)

For the two criteria, let \( \alpha \) denote the number of measurements (visits to ships) and \( z \) denote the travel distance within a tour starting at the depot, visiting \( \alpha \) ships, and returning to the depot. In the bi-criteria problem, \( \alpha \) is to be maximized and \( z \) to be minimized. For expressing the routing model, we define binary variables based on admissible arcs in (1) as follows. For all

\( (ik, jl) \in \Xi \)

, the binary variable \( x_{ik}^{jl} \) takes the value 1 if and only if the arc \( (ik, jl) \) is included in the tour. For summation over binary variables, it is convenient to use dot notation: a dot replacing an index denotes summation over the index it replaces. For example, for binary variables \( x_{ik}^{jl} \), \( x_{ik}^{jl} = \sum_{jl} x_{ik}^{jl} \) where the set of pairs of indices \( jl \) is defined by

\( (ik, jl) \in \Xi \).

Given endogenous variables \( \alpha \), \( z \), and binary variables \( x_{ik}^{jl} \) for

\( (ik, jl) \in \Xi \), the routing model satisfies the following constraints:

For the first objective, \( \alpha \) (total number of visits), the goal constraint is as follows:

\[
x_{\alpha}^* = \alpha + 1.
\]

(2)

where \( x_{\alpha}^* \) counts all entries for ships \( i \in N \cup \{0\} \) but returning to the depot, \( i = 0 \), is not counted in \( \alpha \).

For the second objective, \( z \) (total travel distance), we have

\[
z = \sum_{(ik, jl) \in \Xi} d_{ik}^{jl} x_{ik}^{jl}.
\]

(3)

\(^1\) We carried out a test of the validity of this hypothesis and achieved a positive result. For brevity, we have omitted the test results from this report.
the two values; i.e., $V(I)$ is a linear value function for the vector-
maximization problem (7) with a weighting parameter $\lambda$ > 0. For
the proposed algorithm, let $P_t$ denote the population in iteration
$\tau$, for $\tau = 0, 1, 2, \ldots, \tau$. The iteration’s limit $\tau$ and the population
size $\pi = |P_t|$ are exogenously given parameters.

3.1. The steps in the CGA

The algorithmic steps of the proposed technique are presented by
Algorithm 1; the details of the operators in italics are discussed thereafter.

**Algorithm 1** Steps in the CGA.

**Step0:** Set iteration counter $\tau = 0$ and create an initial population
$P_0$ at random, using Algorithm 2 (Subsection 3.2).

**Step1:** Define front sets $P_k$, $k = 1, 2, \ldots$, for $P_k$ and assign front
number $n_k(I)$ and crowding
distance $\delta(I)$ for all $I \in P_\tau$ (Subsection 3.3).

**Step2:** If $\tau = \tau$ then stop else copy $P_t$ to an auxiliary population
$P_\tau^\prime$.

**Step3:** Choose two parents $I_1$ and $I_2$ at random from $P_\tau$, using
front number and crowding
distance (Subsection 3.4).

**Step4:** Apply the crossover operation among $I_1$ and $I_2$ to create
two offspring $O_1$ and $O_2$
(Subsection 3.5).

**Step5:** For each of the offspring $O_\sigma$, $\sigma = 1, 2,$

**Step5.1:** For each ship appearing twice in $O_\sigma$, remove the latter one.

**Step5.2:** Optionally, perform replacement mutation on
$O_\sigma$, using Algorithm 3
(Subsection 3.6.1).

**Step5.3:** Optionally, perform insertion mutation on $O_\sigma$, using
Algorithm 4
(Subsection 3.6.2).

**Step5.4:** Recover feasibility of $O_\sigma$ by using the DP algo-
rim (Subsection 3.7).

**Step5.5:** Append $O_\sigma$ to auxiliary population $P_\tau^\prime$.

**Step6:** If $|P_\tau^\prime| < 2\pi$ then return to Step 3 else

**Step6.1:** Apply survival selection to $P_\tau^\prime$ to reduce its size to
$\pi$ for creating new population
$P_{\tau+1}$ (Subsection 3.8).

**Step6.2:** Increment $\tau$ by 1, and return to Step 1.

3.2. Creating an initial population

Often the initial population for a GA is generated either randomly
or by a computationally fast problem-specific heuristic
method aimed at starting the algorithm’s iterations with a good set
of solutions. We generate the initial population for the CGA by us-
ing an operator $D_{\pi}$, which is a minor modification from standard
dynamic programming (Bellman, 1957). In each round with $D_{\pi}$, we
aim to maximize the regularized aggregation $V(I) = \lambda \alpha(I) - z(I)$ of
tour $I$ with two objectives, $\alpha(I)$ (to be maximized) and $z(I)$ (to be
minimized), using a randomly drawn weighting parameter $\lambda > 0$.

In a standard DP backward recursion over time slots $k = m, \ldots, 1, 0,$ for all $I$ such that $k \in S_t$, let $V_{ik}$ denote the value function
at node $v_k$ and let $\sigma_k$ be the set of ships $j \in N$ along the sub-
tour (chosen in backward recursion) from node $v_k$ to the end of
the tour (at the depot). At the end of the time horizon, for $i = 0$ and
$k < m$ let $V_{ik} = 0$ and let $\sigma_k$ be empty. The algorithm’s steps
are presented in Algorithm 2.

**Algorithm 2** Dynamic Programming for Initial Population Genera-
tion ($D_{\pi}$).

**Input:** predicted trajectories during $T$, set of admissible arcs $\Xi$,
weight $\lambda$, probability $p^*$

**Output:** $I$, $V(I)$

1. for all time slots $k \in \{m - 1, \ldots, 1\}$ going backward in time, do
2. for all ships $j \in N$ that are present in time slot $k$, do
3. if arc $(ik, 0m) \in \Xi$ then
4. Set the value function $V_{ik} = \lambda - d_{ik}^o$, and set sub-tour
5. else
6. Set $V_{ik} \leftarrow -\infty$
7. Save the node $v_{0m}$ (depot) as the successor node of $v_{ik}$
8. for all ships $j \in N$ and time slots $l > k$, $l < m$, such that
9. if $i \not\in \sigma_j$, do
10. if $(ik, jl) \in \Xi$ then
11. Save the value $V \leftarrow \lambda - d_{ik}^j + V_{jl}$
12. else
13. Set $V \leftarrow -\infty$
14. Let $a$ be a random draw from uniform distribution
15. if $V > V_{ik}$ and $a < p^*$ then
16. Update the value function $V_{ik} \leftarrow V$ and save $v_{ik}$ as
17. the successor node of $v_{ik}$
18. Update the set of sub-tour ships $\sigma_{ik} \leftarrow \{i\} \cup \sigma_j$
19. (Back to depot node $v_{00}$) Let $V_{00} \leftarrow -\infty$
20. for all ships $j \in N$ and time slots $l$, $0 < l < m$, at which $j$ is
21. present, do
22. if $(00, jl) \in \Xi$ then
23. Save the value $V \leftarrow -d_{ik}^j + V_{jl}$
24. else
25. Set $V \leftarrow -\infty$
26. if $V > V_{00}$ then
27. Update the value function $V_{00} \leftarrow V$ and save node $v_{ik}$ as
28. the successor of depot node $v_{00}$
29. On the basis of the saved successor nodes, rebuild the tour $I$
30. and assign $V(I) \leftarrow V_{00}$

Standard DP steps are modified in Algorithm 2 for generating
the initial population as follows. At each node $v_k$, the algorithm
$D_{\pi}$ searches among feasible nodes $v_{ij}$ (each being a location of
ship $j$ in time slot $l > k$ with admissible arcs $(ik, jl) \in \Xi$) such that
$i \not\in \sigma_j$, avoiding multiple visits to ship $i$. Unlike in standard
DP, the node providing the maximal value for $V_{ik}$ is not necessarily
chosen; instead, a probability $p^*$ is assigned to each feasible node
$v_{ij}$ for eligibility to participate in maximizing $V_{ik}$. In the numerical
tests shown in Section 4 we use $p^* = 0.8$. The backward recursion
yields a tour $I$ with the aggregate value $V(I) = V_{00}$. In each run of
$D_{\pi}$, given a randomly drawn parameter $\lambda$, a feasible tour $I$ is pro-
duced with a sequence of ships to be visited. Said procedure viol-
ates the dynamic programming principles, and, consequently, the
tour $I$ produced by $D_{\pi}$ may not be optimal under the objective
$V(I)$. Nevertheless, such solutions can be good enough to create
fast favorable members for the initial population.

Random selection of weights $\lambda$ and eligible successor nodes
(from the set of feasible nodes) are adopted to diversify the solu-
tions obtained from $D_{\pi}$. Another option for achieving heterogeneity
is to apply replacement mutation (introduced in Section 3.6.1)
for tour $I$. Furthermore, to enhance the chance of getting large-
cardinality $|I| = \alpha(I)$ for the tour, an insertion mutation, intro-
duced in Section 3.6.2, may be used.
3.3. Front sets, front numbers, and crowding distance

The proposed CGA is based on the non-dominated sorting approach proposed for evolutionary multi-objective optimization (Deb, 2002). For selection of efficient members from a population \( P \) (the current population \( P_t \) or an auxiliary population \( P'_t \)), \( P \) is subdivided into front sets on the basis of Pareto domination, and a front number is assigned to each member of \( P \) as follows.

Front-set determination Let \( P \) denote a population with \( \alpha(l) \) and \( z(l) \) given for all members \( l \in P \). We subdivide \( P \) into non-overlapping subsets, front sets \( F_k \), for \( k = 1, 2, 3, \ldots \), such that \( P = \bigcup k F_k \). For \( k = 1, F_1 \) is the set of non-dominated members of \( P \), where dominance is defined in terms of the two criteria \( \alpha(l) \) and \( z(l) \) within \( P \). For \( k > 1 \), let \( P_k \) be the set obtained after removal of members of \( F_k \) for all \( \xi < k \) from \( P \). If \( P_k \) is non-empty, then \( F_k \) is the set of non-dominated members within \( P_k \).

Front number For all \( l \in P \), if \( l \in F_k \), then \( k \) is the front number of \( l \) and we state that \( n_F(l) = k \).

For some competitive selections of population members, the crowding distance (Deb, 2002) is used as a tie-breaker. For a population \( P \) with front sets \( F_k \), the crowding distance \( \delta(l) \) for \( l \in F_k \) is a measure quantifying the density of other solutions (tours) surrounding \( l \) in \( F_k \). Thus, crowding distance is determined by the front set for all \( l \in F_k \).

Crowding-distance computation For \( l \in P \), to define the crowding distance \( \delta(l) \), we let \( k \) be such that \( l \in F_k \subseteq P \), and we define \( \Delta \alpha \) and \( \Delta \Delta \) thus:

\[
\Delta \alpha = \min_{j \in F_k} \alpha(j) \quad | J \neq I, \quad \alpha(j) \geq \alpha(l) \} - \max_{j \in F_k} \alpha(j) \quad| J \neq I, \quad \alpha(j) \leq \alpha(l).
\]

\[
\Delta \Delta = \min_{j \in F_k} \{ z(j) \quad | J \neq I, \quad z(j) \geq z(l) \} - \max_{j \in F_k} \{ z(j) \quad | J \neq I, \quad z(j) \leq z(l) \}.
\]

The crowding distance is

\[
\delta(l) = \frac{\Delta \alpha}{2} + \frac{\Delta \Delta}{2},
\]

where \( r_{\alpha} = \max_{j \in F_k} \alpha(j) - \min_{j \in F_k} \alpha(j) \) and \( r_z = \max_{j \in F_k} z(j) - \min_{j \in F_k} z(j) \).

Crowding distances \( \delta(l) \) are used to determine the tournament winner in the event of a tie (by using front numbers); also, \( \delta(l) \) is used as a criterion [larger is better] for choosing members from the auxiliary population \( P' \) for the subsequent population \( P_{t+1} \). Using such a survival selection ensures preservation of diversity in the evolving populations, for finally arriving at a well-distributed set of near-Pareto-optimal solutions (Deb, 2002).

3.4. Choosing parents by tournament

Two independent tournaments To find two parents for a crossover, we run two tournaments. In each tournament, a parent is chosen as follows:

- Randomly draw two members \( l, l' \in P_t \) for a tournament. Ensure that the parents are not identical (\( l \neq l' \)) and that their cardinalities are at least 2 (\( |l| > 1 \) and \( |l'| > 1 \)).
- Choose the member with larger front number \( n_F \).
- If \( l \) and \( l' \) have the same front number, choose the one with the larger crowding distance.

Mating restriction We introduce, as an option, a mating restriction for choosing the second parent \( l' \) for the crossover operation. The first parent \( l \) is chosen as described above, while the second parent \( l' \) is chosen from the population \( P_t \) so as to minimize the absolute difference in cardinality (the number of ships to be visited) between the two parents. If there is a tie, the choice is based on minimizing the total travel distance of \( l' \). Thus, \( l' \) solves the following problem:

\[
\min_{l' \in P_t} |\alpha(l') - \alpha(l)| + 0.001z(l') \quad | l' \neq I, \quad |l'| > 1 \}.
\]

Here the goal is to mate two members \( l \) and \( l' \) having almost the same number of ships in their tour. This will increase the chance of creating two feasible offspring after application of the crossover operator.

3.5. Crossover-point selection and offspring

Consider tournaments resulting in parents \( l = (i, k)_{t=1}^w \) and \( l' = (i', k')_{t=1}^w \) with cardinalities \( |l| = \alpha \) and \( |l'| = \alpha' \). Crossover is performed between parents \( l \) and \( l' \) to generate two offspring. For choosing the crossing point \( v \) for each parent, we discuss three alternative options (i–iii).

(i) Firstly, for \( 1 < v < \alpha \), let \( (i, k) = (i_v, k_v) \) and \((j, l) = (i_{v+1}, k_{v+1}) \) refer to two successive visits to nodes \( v_k \) and \( v_l \) in \( l \). Similarly, for \( 1 < v' < \alpha' \), let \((i', k') = (i'_{v'}, k'_{v'}) \) and \((j', l') = (i'_{v'+1}, k'_{v'+1}) \) be two successive visits in \( l' \). Given that \( v \) and \( v' \) are the chosen crossing points, let

\[
t_{12} = l'w - (k-1)w - (p + t_i^{l'}) \quad (13)
\]

\[
t_{21} = lw - (k'-1)w - (p + t_i^{k'}) \quad (14)
\]

Then the value \( t_{12} \) (in minutes) indicates the additional time during crossing from \( l \) after processing of ship \( l \) and \( p \) to \( l' \) before subsequent processing of ship \( j' \) at \( t_{v_l} \). A similar definition holds for \( t_{21} \). Crossover points \( v \) and \( v' \) are chosen as an optimal solution to the following problem:

\[
\min_{1 < v < \alpha \quad 1 < v' < \alpha'} \max(\alpha / 2 - v, \alpha'/2 - v') \quad \min(1, t_{12} + t_{21}) \quad (15)
\]

The selection rule (15) tends to keep the crossover points near the middle of tours \( l \) and \( l' \). Furthermore, to improve the chance of getting feasible offspring after the crossover operation, that rule tends to maximize the extra time during crossing between \( l \) and \( l' \). Here, we neglect the issue of the feasibility of the offspring tours: for instance, in case the conditions \( t_{12} \geq 0 \) and \( t_{21} \geq 0 \) may be violated.1 Feasibility of offspring will be recovered in Step 4.4 of the CGA via a DP approach; see Section 3.7.

(ii) For the second option, note that with the crossover operator described above, crossing points \( v \) and \( v' \) are chosen on the basis of the optimal solution to a min-max problem (15). To reduce computations, we suggest a modified crossover, wherein the crossing point \( v \) of parent \( l \) is chosen by a random draw from a uniform distribution in \([2, \alpha -1] \) and crossing point \( v' \) of parent \( l' \) is based on problem 15 with \( v' \) fixed within \( l \).

(iii) In the third option, we choose the crossing point \( v \) of parent \( l \) randomly as in option ii and simplify the crossing rule for \( l' \) by

\[
\max_{1 < v < \alpha'} (1, t_{12} + t_{21}) \quad (16)
\]

where \( t_{12} \) and \( t_{21} \) (in minutes) are defined as above (13,14). This determines the crossing point \( v' \) of \( l' \), such that the total waiting time at the crossing points (for the two offspring) is maximized. The goal is to increase the likelihood of obtaining feasible offspring.

\[1 \text{ if } t_{12} + t_{21} \text{ is small or negative, we have a safeguard level of } 1 \text{ in the denominator of (15).} \]
Generating offspring After the crossover points \( v \) and \( v' \) are found, two offspring
\[
O = \{ (i_1, k_1), \ldots, (i_v, k_v), (j_{v+1}', l_{v+1}'), \ldots, (j_{v'}, k_{v'}) \},
\]
and
\[
O' = \{ (i_1', k_1'), \ldots, (i_{v'}, k_{v'}'), (j_{v+1}', l_{v+1}'), \ldots, (i_l, k_l) \},
\]
are generated. Each offspring cross-points \( t \) and \( t' \) may not be in the admissible set \( \mathcal{S} \). Recovery from such infeasibility is performed as described in Section 3.6.2. Irrespective of mutations, offspring may be infeasible. We attempt to repair the sequence in the following ways. First, some ships may appear twice in a sequence of visits; in such a case, the second visit is omitted. Second, the crossing-point selection may leave insufficient time to travel from one ship to the next for the relevant crossover point; i.e., the new crossing arcs might not be in the admissible set \( \mathcal{S} \). Recovery from such infeasibility is performed by means of the DR algorithm, introduced in Section 3.7, whereby making the approach hybrid between population-based and point-based. After possible mutations and recovery of feasibility, the two offspring are appended to the auxiliary population \( P_k^r \) to be considered for inclusion in the subsequent population \( P_{k+1}^r \).

3.6. The mutation operator

For a tour \( I \) (for instance, offspring \( O \) or \( O' \)), we propose two mutation operators, replacement and insertion, as follows.

3.6.1. Replacement mutation

Random replacement is a simple operator intended to increase diversity among the offspring. The following steps are applied where replacement is performed for the offspring \( I \):

- Replacement is done independently for each of the offspring with the chosen probability \( p_r \) without checks for feasibility after replacement.
- Recovery of feasibility is performed as described in Section 3.7.

3.6.2. Insertion mutation

The insertion operator is aimed at increasing the cardinality of each offspring sequence \( I \). Insertion can take place at one or several points, each time with insertion of an additional ship to be visited. Insertion is done independently for each of the offspring, with a chosen probability \( p_i \). To avoid bias of creating long sequences only, we restrict the number of insertions to a randomly chosen number from the set \([0, 1, 2] \). The steps of the insertion operation are provided in Algorithm 4.

Algorithm 3 Replacement Mutation.

**Input:** predicted trajectories, sequence \( I \)

**Output:** modified sequence \( I \) and \( V(I) \)

1: On the basis of a uniform distribution, choose randomly \((i, k) \in I\), indicating a visit to ship \( i \) in time slot \( k \) at node \( v_{ik} \).
2: Find the ship \( j \in N \) nearest the location \( v_{ik} \) in time slot \( k \) such that \( j \) is not scheduled in \( I \).
3: Replace \((i, k) \) with \((j, k) \) in tour \( I \) and update \( V(I) \).
4: Return the modified \( I \) and \( V(I) \).

A brief explanation of the idea behind the insertion algorithm is in order. Let \((i, k) \in I \) (ship \( i \) being visited in time slot \( k \)), and let \((j, l) \in I \) denote the immediate successor of \((i, k) \). The algorithm finds all feasible sub-tours from node \( v_{ik} \) through an additional admissible node \( v_{jk} \) to node \( v_{ik} \) such that ship \( r \) is not in the sequence \( I \) and time slot \( s \) satisfies \( k < s < l \). All such potential added visits to nodes \( v_{rs} \) are recorded as candidates for insertion, and, if some exist, the one providing minimal travel time from location \( v_{ik} \) to \( v_{ji} \) via \( v_{rs} \) is chosen.

Algorithm 4 Insertion Mutation.

**Input:** predicted trajectories, sequence \( I \)

**Output:** extended sequence \( I \) and \( V(I) \)

1: Initialize with the depot node \((i, k) = (0, 0) \) and let \( a \) be a randomly drawn integer from \([0, 1, 2] \).
2: If \( a \geq 1 \) then
3: repeat
4: Let \((j, l) \in I \) be the immediate successor of \((i, k) \).
5: Let \( t_j \leftarrow \infty \).
6: For all ships \( r \) not in the sequence \( I \) and all time slots \( s \) such that \( k < s < l \), do
7: Let \( t'_j \leftarrow t'_j + t_r \) and \( t_l \leftarrow t'_l \).
8: If \((ik, rs), (rs, jl) \in \mathcal{S} \) and \( t_j + t_l < t_r \) then
9: Save \((r, s) \) as a candidate for insertion and let \( t_r \leftarrow t_j + t_l \).
10: If \( t_r < \infty \) (there is a candidate for insertion) then
11: Update \( I \) by inserting \((r, s) \) between \((i, k) \) and \((j, l) \), and let \( V(I) \leftarrow V(I) + \lambda - c(r, t_r) \).
12: Update \((i, k) \) to be the next node in the updated sequence \( I \).
13: Until the end node \((0, m) \) (the depot) is reached or \( a \) ships are inserted
14: Return extended sequence \( I \) with the updated value function \( V(I) \).

3.7. Recovering feasibility

Consider tour \( I \) to be the offspring after possible mutations. Suppose the tour \( I = \{(i_1, k_1), \ldots, (i_{v'}, k_{v'})\} \) visits each ship \( i \in N \) no more than once but may be still be infeasible because not all arcs along the tour \( I \) are in the admissible set \( \mathcal{S} \). Let \( N' = \{i_1, i_2, \ldots, i_{v'}\} \) be the ordered set of ships to be visited in \( I \). We now develop an optimization model to find a feasible tour by adjusting the time slots in \( I \) and possibly omitting one or more ships from the sequence if so required. The task is to find optimal time slots for visiting some or all of the \( \alpha \) ships while obeying the order of ships specified in \( N' \). For notational convenience, for \( \nu = 1, 2, \ldots, \alpha \), we relabel the ships \( i_\nu \) in terms of \( \nu \), so that the sequence of ships after relabeling is given by \( N' = \{1, 2, \ldots, \alpha \} \). The depot is again denoted as \( i = 0 \). After relabeling of the ships, regardless of the set of admissible arcs \( \mathcal{S}(1) \), we define the subset \( \mathcal{S}' \subseteq \mathcal{S} \) as follows:

\[
\mathcal{S}' = \{(ik, jl) \in \mathcal{S} \mid 0 \leq i < j < \alpha \text{ and } (i < j < 0)\}.
\]

(17)

Hence, \( \mathcal{S}' \) pertains only to ships in \( i \in N' \) and \( i = 0 \) (the depot) and the condition \((i < j < 0)\) for arc \((ik, jl)\) obeys the ordering given by \( N' \). The routing problem for recovering feasibility employs binary variables \( x^0_{ik} \) with \((ik, jl) \in \mathcal{S}' \) so that \( x^0_{ik} = 1 \) if and only if arc \((ik, jl) \in \mathcal{S}' \) is included in the tour. Again, for summation over binary variables we use dot notation. For example, \( x^0_{ik} = (\sum_{(ik, jl) \in \mathcal{S}} x^0_{ik} \) where the set of pairs of indices is defined by \((ik, jl) \in \mathcal{S}' \). A parameter \( \lambda > 0 \) is chosen such that it is large enough to maintain maximizing \( \alpha \) as the primary objective. The problem is to find binary variables \( x^0_{ik} \) for all \((ik, jl) \in \mathcal{S}' \) to

\[
\max \lambda X^0_{ik} - \sum_{(ik, jl) \in \mathcal{S}} d_{ik} X^0_{ik}.
\]

(18)

The first term above yields the value \( \lambda (\alpha + 1) \) and the second term determines the travel distance \( z \). The following constraint ensures exactly one exit from the depot node:

\[
x^0_{ik} = 1.
\]

(19)

For \( i \in N' \), if ship \( i \) is visited in time slot \( k \) \( i \in S_j \), then the flow enters node \( v_{ik} \) from some node \( v_{ij} \) with \( 0 \leq j < i \) at an earlier time slot \( l < k \) with \( l \in S_j \), and it exits the node \( v_{ik} \) for some node \( v_{ij} \)
with \( j > i \) or \( j = 0 \) at a later time slot \( l > k \) with \( l \in S_l \). This is ensured by the following flow-conservation constraints:

\[
x_{ik}^k = x_{ik}^{\ast} \quad \forall \ i \in N^l, \ k \in S_l.
\] (20)

The problem 18–20 is a network flow problem for which the binary variables can be relaxed to continuous variables in \([0, 1]\) and any optimal basic solution for the resulting linear programming problem is binary-valued. The tour denoted by \( l' \), a mutant of \( l \), is determined by the binary variables \( x_{ik}^l \), whose optimal value is 1.

**Dynamic programming procedure for recovering feasibility**

Using a standard linear-programming solver for the problem 18–20 for all infeasible tours in each iteration of the CGA increases the computation time significantly. To speed up the execution, we solve 18–20 by means of a pure DP procedure referred to as \( D_{\text{DP}} \).

The \( D_{\text{DP}} \) algorithm is similar to \( D_{\text{CGA}} \); however, \( N^t \) determines the number of ships, and the tour obeys the order of ships in \( N^t \). In addition, the probability \( p^t \) (of accepting successor nodes) is not considered further. Therefore, the \( D_{\text{DP}} \) algorithm becomes standard DP with backward recursion. The problem is to find an optimal tour of visiting some or all of the \( \alpha \) ships in \( N^t \) by maximizing the value function (18). If no feasible time slot is found for a ship \( i \in N^t \), that ship is omitted. We can simplify \( D_{\text{DP}} \) by excluding the possibility of omitting visits to some ships in \( N^t \). This procedure might be useful for enhancing members of the initial population obtained from \( D_{\text{CGA}} \).

3.8. The auxiliary population based on NSGA-II

After generating \( \pi \) offspring, we have an auxiliary population \( P' \) of \( 2m \) feasible members consisting of \( \pi \) members of current population \( P \) and \( \pi \) offspring. The population \( P' \) is then sorted into \( \kappa \) front sets in accordance with NSGA-II (Deb, 2002), and a front number \( n_{\kappa}(I) \) is assigned to each tour \( I \in F_{\kappa} \), for \( \kappa \leq \kappa \). Crowding distances (11) are determined for members of front set \( F_{\kappa} \) only.

For generating the subsequent population \( P_{\kappa+1} \) for the next iteration \( \tau + 1 \), let \( \kappa \geq 1 \) be the smallest integer such that \( \left| \cup_{\kappa=1}^\kappa F_{\kappa} \right| \geq \pi \). If \( \left| \cup_{\kappa=1}^\kappa F_{\kappa} \right| = \pi \), then \( P_{\kappa+1} = \cup_{\kappa=1}^\kappa F_{\kappa} \); otherwise \( P_{\kappa+1} \) is \( \cup_{\kappa=1}^\kappa F_{\kappa} \) plus the \( \pi - \left| \cup_{\kappa=1}^\kappa F_{\kappa} \right| \) members of \( F_{\kappa} \) with the largest crowding distances, appended to \( P_{\kappa+1} \).

3.9. The hybrid customized GA

As discussed above, to make our overall CGA computationally efficient, we introduced DP-based methods (to replace an ILP solver) within the CGA operators, as follows:

- To generate the initial population, we solve the \( D_{\text{CGA}} \) model via Algorithm 2. Although the Bellman optimality principle is violated in this case and the resulting solutions may not be optimal, \( D_{\text{CGA}} \) provides good starting solutions and is computationally quick to evaluate. Furthermore, as started at the end of Section 3.2, for diversification of the solutions obtained after each run of \( D_{\text{CGA}} \), when a potentially improved candidate is found during intermediate steps of the recursion, it is adopted with a given probability.

- For recovering feasibility after crossover and mutation, the ILP model (18–20), introduced in Section 3.7, is solved by means of the dynamic programming procedure of \( D_{\text{DP}} \) for adjusting the time slots of a given sequence and, if necessary, omitting ships from the sequence to make the solution feasible. Using \( D_{\text{DP}} \) leads to the optimal solutions of the ILP model (18–20).

We will show that quick dynamic programming methods within our proposed CGA operators introduce potentially good properties in the evolving population members.

4. Results

This section presents an implementation of the customized genetic algorithm introduced in Section 3, using real data sets. We compare the performance of the CGA and an ILP model (8) for obtaining the Pareto solutions for the bi-criteria MT-TSP defined in problem 7.

4.1. Data for numerical tests

Automatic Identification System (AIS) data are fetched from the open interface of the Finnish Transport Infrastructure Agency (Finnish Transport Infrastructure Agency, 2019) for ships (equipped with a class-A AIS transmitter) in a region of the Baltic sea with busy marine traffic. A k-nearest neighbor method, described by Virjonen, Nevalainen, Pahikkala and Heikkonen (2018), is used for prediction of ships’ trajectories. The Maritime Mobile Service Identity (MMSI) number of the ship, the location (latitude and longitude), and the time are employed for prediction. The prediction model is trained with historical data from May to June 2018. The trained model is used to predict ships’ trajectories over the 16 h time horizon (7am–11pm UTC+3) for all ships passing through the work area on July 1, 2018. Each predicted trajectory is interpolated with \( w = 5 \)-minute time spacing.

The estimated data are used separately for seven time horizons, of \( T = 4, 6, 8, 10, 12, 14, \) and 16 hours. They all start at 7am, which indicates the starting time \( t = 0 \) of the planning period. These data sets provide a range of problems from small to large size for experimenting and evaluating the performance of the CGA and ILP models. The dimensions of the seven ILP problems (8), in terms of the number of binary variables and constraints, are shown in Table 1.

Interestingly, the 16 h problem involves nearly a million binary variables, which makes this study instance one of the very large-dimensional problems solved with optimization algorithms, including genetic algorithms (Deb & Myburgh, 2017).

The speed of the surveillance boat is assumed to be 25 knots (46.3 km/h) on average, and the processing time for measurement is \( p = 3 \) minutes for each ship. For all cases of \( T = 4, \ldots, 16 \), the length of the time slots is \( w = 5 \) minutes.

4.2. Estimating Pareto-optimal frontiers via ILP

For each of the seven cases of \( T = 4, \ldots, 16 \) h, Pareto-optimal frontiers can be estimated using the ILP model (8) as described in the latter portion of Section 2. However, the number of binary variables shown in Table 1 ranges from tens of thousands to almost a million. Thus, solving the problems with common ILP solvers, particularly for a horizon \( T \) large enough for meeting the needs that arise in practice, becomes a challenge.

For test runs, we used the MOSEK solver (MOSEK ApS, 2019), with default settings except that the relative gap tolerance was set to 0.01 with a two-hour time limit for obtaining each Pareto solution separately. All implementations are done in an AMPL environment (Fourer, Gay, & Kernighan, 2003) on a standard HP Z230 workstation with 32 GB of RAM. To find the maximum feasible \( \alpha \),

---

4 The prediction model (Virjonen et al., 2018), the route-optimization model (Maskooki & Nikulin, 2020) and the risk assessment for the route-optimization model (Maskooki et al., 2020) are addressed in publications related to the research project explained in this paper. Our research collaborators provided the best available data for predictions of ship trajectories with regard to the work area of our case study.

5 In a practical setting, the surveillance-boat operator is expected to have, at most, 2 h to find a reasonably good schedule, before launching the trip from the harbor; therefore, we limit our solver time to two hours.
starting with $\alpha = 1$, we let $\alpha$ increase by 1 each time until the solver declares infeasibility for the ILP problem.

The smallest problems with a four-hour time horizon could be solved rapidly; for a six- and eight-hour horizon with some of the largest $\alpha$ levels, the solution time (wallclock time) exceeded 4 h, which already is prohibitive for practical use. For instance, with $T = 6$ h and $\alpha = 18$, even 10 hours was insufficient for confirmed optimality. The percentage of problems for which optimality was not confirmed by the solver within two hours ranges from 29% to 39% over $T > 8$ h time horizons. For each $T$, the number of such cases is shown in Table 2. Table B.3 (in Appendix B) shows the ILP solver’s solution time for each horizon $T$ and for each feasible level of $\alpha$.

### 4.3. Implementing the CGA

The proposed CGA is executed for all seven cases of time horizon, $T = 4, \ldots, 16$ h. Before discussing the results (in Section 4.4), we summarize the implementation details for the CGA.

The population size is set to $\pi = 50$ and the iteration limit to $\tau = 50$ for all experiments. For generating the initial population in iteration $\tau=0$, we use the modified dynamic programming procedure $DH_0$ (introduced in Section 3.2). For each run of $DH_0$, the weighting parameter $\lambda$ (the weight for the objective $\alpha$ in Algorithm 2) is set to $-\log(u)/0.03$, where $u$ is a random number drawn from a uniform distribution $U(0,1)$. Random drawing of weights yields a wide range of $\alpha$ levels. In the recursive steps of $DH_0$, a probability $p^r = 0.8$ is used to randomly accept feasible successor nodes (see Section 3.2), and thereby to increase the diversity in the initial population. The algorithm $DH_0$ repeats until $\pi = 50$ distinct members are generated for the initial population $R_0$. No replacement or insertion mutation is applied to the members (tours) produced by $DH_0$.

After this, at the beginning of each iteration $\tau$, (i) the population $P_\tau$ is subdivided into front sets, (ii) the front numbers are assigned, and (iii) the crowding distances are calculated for all members of $P_\tau$ as described in Section 3.3.

Parents for crossover are selected through two tournaments defined in Section 3.4, where two non-identical parents are selected at random from the population $P_\tau$ and the better is chosen as the winner. We also tested the mating restriction operator in which two parents with equal cardinality are allowed to participate in the recombination operation, as per rule (12). For $T = 8$ h, the mating-restriction rule improved convergence and average error. However, for $T > 8$ h, the outcome was unfavorable when this restriction was employed; for instance, for the maximum level of $\alpha$, the two-tournaments approach yields the optimal solution for $T = 10$ h and a near-optimal solution for $T = 12$ h. While the mating-restriction rule led to feasible solutions in each case, they were far from optimal. Therefore, in our tuned CGA, we use two independent tournaments for choosing two parents $I$ and $I'$ for crossover.

For crossing-point selection, we consider three rules, mentioned in Sections 3.5 (i–iii). Rule (i), using problem 15, is symmetric for parents $I$ and $I'$. The rule is designed to choose crossing points near the center of each sequence $I$ and $I'$. The crossing point in rule (ii) is chosen randomly for $I$, after which the rule in problem 15 is applied for $I'$. Comparisons of rules (i) and (ii) reveal that rule (ii) leads to a greater diversity of the population and produces a better coverage of $\alpha$ levels among the solutions. Therefore, rule (ii) is chosen for crossing-point selection. Also, we use rule (iii) which applies problem 16 to $I'$ in the aim of improving the chance of generating feasible offspring. Experimental results with rule (iii) do not show much difference in solution accuracy relative to rule (ii), but the latter generally leads to a better distribution of $\alpha$ levels and faster convergence to the Pareto-optimal frontier.

After crossover, each offspring solution is checked for possible duplication of visits in the sequence; if duplicates are detected, the latter visit is omitted. Other types of infeasibilities are also checked later as explained in Section 3.7.

With regard to the mutation phase, see Section 3.6.1. For each of the offspring, a single replacement mutation is performed with probability $p_m = 0.2$, and the insertion mutation is applied with probability $p_r = 1$; however, we restrict the number of insertions to 0, 1, or 2 ships at most, each with equal probabil-

### Table 1

Dimensions (after preprocessing) of ILP problems (8) for finding Pareto-optimal itineraries. $T$ is the time horizon (h), $n$ is the total number of ships in the work area during $[0, T]$, and $m$ is the number of time slots during $[0, T]$.

<table>
<thead>
<tr>
<th></th>
<th>$T = 4$</th>
<th>$T = 6$</th>
<th>$T = 8$</th>
<th>$T = 10$</th>
<th>$T = 12$</th>
<th>$T = 14$</th>
<th>$T = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>22</td>
<td>28</td>
<td>33</td>
<td>40</td>
<td>42</td>
<td>52</td>
<td>63</td>
</tr>
<tr>
<td>$m$</td>
<td>48</td>
<td>72</td>
<td>96</td>
<td>120</td>
<td>144</td>
<td>168</td>
<td>192</td>
</tr>
<tr>
<td>Constraints</td>
<td>263</td>
<td>469</td>
<td>705</td>
<td>889</td>
<td>1,053</td>
<td>1,153</td>
<td>1,338</td>
</tr>
<tr>
<td>Binary vars.</td>
<td>27,432</td>
<td>96,606</td>
<td>239,338</td>
<td>395,722</td>
<td>576,449</td>
<td>710,694</td>
<td>964,276</td>
</tr>
</tbody>
</table>

### Table 2

Summary statistics for the number of $\alpha$ levels, accuracy of the CGA, and speed-up with the CGA relative to ILP: $T$: time horizon (hours); $n_{max}$: number of feasible $\alpha$ levels; $n_{CGA}$: number of final $\alpha$ levels from the CGA; $n_{CGA}$: number of initial $\alpha$ levels from the CGA; $n_{init}$: number of $\alpha$ levels at which the ILP solver hits the 2 h time limit; $\Delta z^p = (z_{\alpha} - z_{CGA})/z_{CGA}$: initial average relative difference in total travel distance, a positive value indicating better performance for ILP; $\Delta z = (z_{\alpha} - z_{CGA})/z_{CGA}$: final average relative difference; $\Delta HV^p = (HV_{ILP} - HV_{CGA})/HV_{CGA}$: initial relative hypervolume difference, a positive value indicating better performance for ILP; $\Delta HV = (HV_{ILP} - HV_{CGA})/HV_{CGA}$: final relative hypervolume difference; $sec_{ILP}$: total time (wallclock sec.) of the ILP solver; $sec_{CGA}$: total time (CPU sec.) of the CGA; and speed-up: $sec_{ILP}/sec_{CGA}$.

<table>
<thead>
<tr>
<th>Number of levels $\alpha$</th>
<th>$T$</th>
<th>$n_{max}$</th>
<th>$n_{CGA}$</th>
<th>$n_{init}$</th>
<th>$\Delta z^p$</th>
<th>$\Delta z$</th>
<th>$\Delta HV^p$</th>
<th>$\Delta HV$</th>
<th>$sec_{ILP}$</th>
<th>$sec_{CGA}$</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
<td>15</td>
<td>13</td>
<td>0</td>
<td>0.006</td>
<td>0.002</td>
<td>0.0158</td>
<td>0.0015</td>
<td>93</td>
<td>663</td>
<td>&lt;1</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>19</td>
<td>9</td>
<td>1</td>
<td>0.030</td>
<td>0.018</td>
<td>0.0449</td>
<td>0.0076</td>
<td>13,868</td>
<td>1659</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td>25</td>
<td>10</td>
<td>3</td>
<td>0.053</td>
<td>0.020</td>
<td>0.0927</td>
<td>0.0150</td>
<td>35,592</td>
<td>2418</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>31</td>
<td>31</td>
<td>12</td>
<td>9</td>
<td>0.038</td>
<td>0.012</td>
<td>0.0514</td>
<td>0.0005</td>
<td>85,945</td>
<td>3438</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>36</td>
<td>36</td>
<td>17</td>
<td>14</td>
<td>0.028</td>
<td>0.016</td>
<td>0.0477</td>
<td>0.0023</td>
<td>122,215</td>
<td>5052</td>
<td>24</td>
</tr>
<tr>
<td>14</td>
<td>40</td>
<td>40</td>
<td>16</td>
<td>13</td>
<td>0.056</td>
<td>0.031</td>
<td>0.0455</td>
<td>0.0057</td>
<td>107,479</td>
<td>6105</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>48</td>
<td>46</td>
<td>20</td>
<td>16</td>
<td>0.064</td>
<td>0.042</td>
<td>0.0481</td>
<td>0.0031</td>
<td>141,579</td>
<td>7106</td>
<td>20</td>
</tr>
</tbody>
</table>
ity, 1/3. Hence, there is a probability of 1/3 of omitting any insertion.

After mutations, to recover feasibility of the offspring, we solve problem 18–20 by using DP $\mathcal{P}_\ell$ with a large weight $\lambda=1,000$ for the objective $\alpha$. If the final offspring solution is not a duplicate of an existing population member, it is appended to the auxiliary population $P'$. Offspring are generated until $P'$ has $2\pi = 100$ members.

At the final stage of iteration $\tau$, the set $P'$ is subdivided into front sets and the members are ranked as described in Section 3.8. Crowding distances are calculated for members of the last front set of interest. By means of NSGA-II, $\pi = 50$ distinct members of $P'$ are copied into the subsequent population $P_{\tau+1}$ on the basis of front number and/or crowding distance.

### 4.4. Comparing the tuned CGA and ILP

In this section, we discuss how the tuned CGA described in Section 4.3 fares in comparison with the ILP model outlined in Section 4.2 for estimating the Pareto-optimal frontier of the bicriteria problem (7). Performance is compared between the two methods in terms of Pareto-efficiency and computation time.

The accuracy of the final efficient frontier obtained by the CGA is evaluated via two proximity metrics, average relative error in travel distance and relative difference in hypervolume (Beume, Fonseca, López- Ibáñez, Paquete, & Vahrenhold, 2009) – a metric that computes the collective dominated region by the obtained Pareto set, on the basis of the solution obtained by the ILP model. Furthermore, the initial frontier from the CGA is compared with the efficient frontiers of the ILP model in light of both measurements.

For the challenging ILP problem instances, an optimal solution may not be confirmed by the solver within the 2 h time limit. Therefore, the relative error is not necessarily non-negative. In other cases too, (where optimality is confirmed), negative errors can arise, if the CGA solution falls within the 0.01 gap tolerance.

Comparisons of the front set $F_1$ obtained from the CGA and the Pareto frontier obtained by the ILP model for $T = 4, \ldots, 16$ h are shown in Figs. 2–4. We now turn to the performance of the CGA in detail for each time-horizon case separately.

For the $T = 4$ h case, we observe that 13 of the 15 possible levels of $\alpha$, including the maximum $\alpha = 15$, are generated already by $\mathcal{P}_4$ in the front set $F_1$ of the initial population (Fig. 2(a)). The average relative error in the initial front set $F_1$ is 0.06 (Fig. 2(b)). After 10 iterations, the average relative error falls to 0.002 and all 15 levels of $\alpha$ are covered by front set $F_1$. However, for such a small problem, the ILP solver is fast enough to find the Pareto-optimal solutions while the CGA with its population-based approach performs more slowly (see Table 2).

For the $T = 6$ h case, the CGA does not find a dominant solution for the maximum level, $\alpha = 20$; however, large levels, including 15, …, 19, are generated by $\mathcal{P}_6$ in the initial population (Fig. 2(c)) with good accuracy. The average errors in terms of travel distance and the hypervolume metric, alongside the number of $\alpha$ levels found by $\mathcal{P}_6$, are shown in Table 2 in the “differences” columns. After 40 iterations, all $\alpha$ levels (except the largest) are in $F_1$ and Pareto points are estimated quite accurately, with an average relative error of 0.02 (Fig. 2(d)). For the CGA, the speed-up values are already rather high. Speed-up values are shown in Table 2 under “solution time.” Fluctuation in the average error visible in Figs. 2–4 is explained by new $\alpha$ levels appearing in $F_1$ over the iterations. For example, in the $T = 6$ h case, the levels $\alpha = 13$ and 14 are not produced in the initial population by $\mathcal{P}_6$, but both are generated at iteration 1 with average error 0.03, whereas the average error in the initial iteration was 0.025.

For the case $T = 8$ h, the CGA generates a complete range of 25 levels for $\alpha$ in the front set $F_1$ at iteration 10 with an average relative error of 0.03 (see Fig. 2(e) and Fig. 2(f)). Hence, after iteration 10, the relative error monotonically decreases, and it reaches 0.02 at iteration 50. For the maximum level $\alpha = 25$, the relative error is 0.02 at iteration 50. The largest error appears in the (less important) case $\alpha = 2$, for which the relative error is 0.12. For $T = 8$ h, the CGA is at least 16 times faster than ILP in obtaining an entire Pareto-frontier estimate.

For the case $T = 10$ h, the 2 h time limit is hit by the ILP solver for all cases with $\alpha > 22$ (see Table B.3 in Appendix B). For such $\alpha$ levels, optimality is not confirmed for ILP problems. Consequently, as can be seen from Fig. 3(a), for some cases with $\alpha \geq 26$, the relative error is negative (the solution obtained by the CGA is better than the ILP solution). This holds, for instance, in the extreme case of $\alpha = 31$, for which the relative error is −0.005. At iteration 24, the CGA produces 30 of the 31 possible $\alpha$ levels in front set $F_1$ ($\alpha = 3$ is missing in $F_1$, although it exists in other front sets) with an average error of 0.017 (Fig. 3(b)). In the final iteration, all 31 possible levels $\alpha$ are produced in $F_1$ and the average error is 0.012. As for the computation time, the CGA produces the greatest speed-up, at 25 times the speed of the ILP solver (see Table 2). Accordingly, we identify a clear advantage of using the proposed CGA instead of ILP for large-dimension versions of the problem.

For the case $T = 12$ h, the solver hits the time limit for all cases with $\alpha \geq 23$ and the CGA obtains a negative relative error for some of the levels $\alpha \geq 26$. After 19 iterations, 34 out of the 36 possible $\alpha$ levels are produced in $F_1$ (the missing levels are $\alpha = 3$ and 14). The complete set of $\alpha$ levels is generated after 42 iterations, with an average relative error of 0.017 (see Fig. 3(c-d)).

For the case $T = 14$ h (Fig. 3(e)), 38 out of 40 possible $\alpha$ levels are present in $F_1$ after iteration 16 (the missing $\alpha$ levels are 4 and 20). Complete coverage is reached at iteration 39. The average relative error is 0.03 in iteration 50, the final iteration (Fig. 3(f)). In this case, CGA computations are at least 18 times faster than the ILP solver (see Table 2). The relative error at the largest level, $\alpha = 40$, is 0.01.

Finally, the $T = 16$ h case, representing the largest problem in our study, has an average error of 0.04 (Fig. 4(b)) at the final iteration (no. 50). At the maximum level, $\alpha = 48$, the relative error is −0.006; i.e., the solution produced by the CGA is better than the one obtained by the ILP solver in two hours. The complete set of $\alpha$ levels is not produced in $F_1$; however, after 50 iterations only two $\alpha$ levels are missing ($\alpha = 18$ and $\alpha = 20$) (Fig. 4(a)). The speed-up value for the CGA is 20 times in comparison with the ILP technique (Table 2).

Table 2 summarizes the results for all horizons $T$ in terms of the number of $\alpha$ levels, average relative error, hypervolume, and computation time. The number of feasible $\alpha$ levels ($n_{GA}$) varies from 15 to 48 across the $T$ values. In our results, the number of $\alpha$ levels in the first front of the initial population produced by the CGA ($n_{GA}$) is equal to $n_{max}$, the largest possible, in all cases except $T = 6$ h (one level missing) and $T = 16$ h (two levels missing). For all $T$ values, the number of $\alpha$ levels in the first front of the initial population with the CGA ($n_{GA}$) is less than half of the full count $n_{GA}$, except for $T = 4$ h. The number of $\alpha$ levels for which the ILP solver hits the time limit ($n_{lim}$) increases steadily with $T$, reaching 16 levels (out of 48) for $T = 16$ h.

For the tour travel distance $z_{GA}$ produced by the CGA and the $z_{ILP}$’s from the ILP, the average relative differences ($z_{GA} - z_{ILP}$)/$z_{ILP}$ (over $\alpha$ levels) in early iterations are in the range 0.6 to 6.4% and decrease with further iterations to the final range of 0.2% to 4.2% of the ILP solutions.

Given a reference point ($\alpha, z$) = (0, $z_{max}$), the hypervolume (HV) of a given frontier set $P$ is the surface area ($\text{volume}^6$) of
Fig. 2. The figures on the left show comparison of the CGA’s front set $F_1$ with the Pareto frontier obtained by the ILP model for the cases $T = 4, 6,$ and $8$ h. Iteration -1 indicates the ILP-model-produced optimal frontier, in blue dots. The right-hand figures show the average relative-error rate for the CGA’s front set $F_1$ over the iterations.
Fig. 3. The figures on the left plot the CGA’s front set $F_1$ against the Pareto frontier obtained by the ILP model for the cases $T = 10, 12,$ and $14$ h. Iteration -1 indicates the optimal frontier yielded by the ILP model, in blue dots. The figures on the right show the average relative-error rate for the CGA’s front set $F_1$ over the iterations.
points \((\alpha, z)\) that (i) are dominated by some point in \(P\) and (ii) satisfy \(\alpha \geq 0\) and \(z \leq z_{\text{max}}\) Deb (2002). For each \(T\), we consider three frontier sets: the set of Pareto points \(P_{\text{ILP}}\) obtained from the ILP model, the set \(P_{\text{CGA}}^0\) obtained from \(D\ell_p\) for the initial population from the CGA, and \(P_{\text{CGA}}\) obtained via the CGA for the final population; for each \(T\) separately, \(z_{\text{max}}\) is the largest travel distance observed in the three sets of Pareto frontiers. The respective hypervolumes are denoted by \(HV_{\text{ILP}}, HV_{\text{CGA}}^0\) and \(HV_{\text{CGA}}\). Table 2 shows that the relative difference \((HV_{\text{ILP}} - HV_{\text{CGA}}^0)/HV_{\text{ILP}}\) initially is in the 1.6% to 9.3% range. At the final iteration, \((HV_{\text{ILP}} - HV_{\text{CGA}})/HV_{\text{ILP}}\) lies within 0.05% to 1.5% of the ILP solutions.

The values under “Solution time” in Table 2 show the total solution time of the ILP solver (sec) in wallclock time. The solution time for the CGA \((sec_{\text{CGA}}\) in CPU sec), and the speed-up \(sec_{\text{ILP}}/sec_{\text{CGA}}\) of the CGA over ILP solutions. The speed-ups are indicative for the following reasons. Firstly, a large share of ILP problems related to different \(\alpha\) levels proves unsolvable within a two-hour time limit set for the MOSEK solver; for all such cases, we count only two hours for solution time \(sec_{\text{ILP}}\) in Table 2. Secondly, the solution time \(sec_{\text{CGA}}\) for the CGA is given in CPU seconds, whereas solution time \(sec_{\text{ILP}}\) for ILP is in wallclock seconds, so may be smaller than the CPU-time value on account of use of multiple processors. Therefore, we interpret the speed-up figures in Table 2 as underestimates for the proposed CGA method in comparison to the ILP method.

In the cases where high-quality starting point for converging to a complete efficient frontier in later iterations. Convergence in terms of accuracy often starts after a nearly complete set of Pareto points is produced in \(F_t\). This occurs in the moderate-size problems with \(T = 6\) and 8 h after 11–13 iterations and in larger problems with \(T = 10, 12,\) and 14 h after 17–23 iterations. After 50 iterations, an acceptable accuracy level is achieved in all cases, and large \(\alpha\) levels are obtained with high efficiency. Importantly, the speed-up of the CGA relative to the ILP is remarkably large for moderate to large-size problems; it is between 16 and 25 times faster for larger problems \((T \approx 8\) h). The CGA is able to achieve an efficient solution with a negligible sacrifice in total travel distance but with a much smaller computation time. From the results in Table 2, in a sample workday, the total (CPU) time for determining the entire Pareto frontier via CGA is under 30 minutes for the \(T = 6\) h time horizon and increases with \(T\) to about two hours for the \(T = 16\) h horizon; the corresponding computing times for the ILP model are 4–39 hours for the same problem set. For example, in the case of the largest-size problem, after a 2 h run, the ILP solver would have solved 1 to 6 out of 48 different \(\alpha\) levels for the available ships, while the CGA is able to find a solution covering 46 of the 48 available ships in the same amount of computation time. A fast computation of multiple near-optimal solutions, trading-off a wide range of distance traveled and ships covered by using the proposed hybrid evolutionary-DP approach, makes it pragmatic and is the main highlight of this study. This study also stays as one of the largest practical problems solved by a genetic algorithm approach.

5. Conclusions

We have presented an efficient hybrid evolutionary algorithm designed to find the efficient frontier of a bi-criteria moving-target TSP. Variants of dynamic-programming-based approaches have been employed for generating the initial population and for repairs to address infeasible members of the offspring set. The custom mutation operators, insertion and replacement, along with other local optimizations, have led to a new hybrid customized GA (proposed CGA), for solving the bi-criteria MT-TSP.

Experimental evaluations with large-scale data sets from a case-study problem have confirmed that the CGA is capable of generating a wide range of high-quality Pareto solutions trading off the two conflicting objectives. The CGA has been observed to be significantly faster in solving large-sized cases than an ILP solver. We have tested the CGA on problems with extremely large dimensions, ranging from tens of thousands of binary variables to nearly a million, and demonstrated that it can solve such problems significantly more rapidly than the ILP solver, with negligible average relative error. Importantly, some Pareto solutions of interest (with large sequences of targets to be visited) obtained via the CGA are more efficient than the solution found by the ILP solver within a two-hour time limit. In addition, the experimental results have re-
Table B.3

The ILP solution (wallclock) times in seconds for each case $T$ and each level of $\alpha$. Figures above 7,200 s indicate hitting the 2 h solver time limit.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$T = 4$</th>
<th>$T = 6$</th>
<th>$T = 8$</th>
<th>$T = 10$</th>
<th>$T = 12$</th>
<th>$T = 14$</th>
<th>$T = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

---

revealed that the probabilistic dynamic program $D_k^P$ proposed for generating the initial population, can provide high-quality starting solutions, and thereby clearly indicate its potential as a standalone or in combination with other heuristics for solving large-sized problems for which DP-based techniques are under consideration.

Future work could examine whether the results might be improved through inclusion of a priori information based on the features of the traffic pattern, such as clusters of ships along the predicted trajectories, and testing variants of the operators introduced could yield further benefits. One possible extension to our work that is especially interesting with regard to practice is to consider multiple surveillance boats and more than one depot (Gröba et al., 2018; Montoya-Torres, Franco, Isaza, Jiménez, & Herazo-Padilla, 2015). In this case, a home depot for each surveillance boat could be specified, and joint constraints must be established for all boats such that each ship gets visited no more than once. In continuation to this study, we plan to examine the effect of uncertainty in the ship movement, service time, and other un-

Avoidable circumstances on the MT-TSP problem for arriving at robust solutions.

Acknowledgments

The first author is financially supported by the University of Turku Graduate School’s MATTI doctoral program. The authors would also like to thank the anonymous referees for their constructive comments and suggestions, which have helped improve the quality of this manuscript.

Appendix A. List of Notations

$T = $ time horizon (hours)

$w = $ time discretization interval

$k = 0, 1, 2, \ldots, m$, time-slot index

$n = $ total number of ships to appear in the work area during $[0, T]$

$i = 1, 2, \ldots, n$, ships to appear in the work area during $[0, T]$

$\nu_k = $ coordinate vector of the location of ship $i$ in time slot $k$

$\nu_i = (\nu_{ik}, \ldots, \nu_{ij})$, set of time slots for which ship $i$ is in the work area

$d_{ik} = \|\nu_k - \nu_i\| = $ distance (km) from $\nu_k$ to $\nu_i$

$c = $ speed (km/h) of the emissions-measurement boat

$\nu_{ik} = d_{ik}/c = $ time (h) taken to move from $\nu_k$ to $\nu_{ij}$ with constant speed $c$

$p = $ service time (h) of ships performing the emissions measurement

Appendix B. Computation time for solving the ILP model

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2021.05.018.

References


