European Journal of Operational Research xxx (xxxx) xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Decision Support

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

Alaleh Maskooki[a,*], Kalyanmoy Deb[b], Markku Kallio[c]

[a] University of Turku, Vesilinnantie 5, Turku FI-20014, Finland
[b] Michigan State University, East Lansing, MI 48824, USA
[c] Aalto University School of Business, Ekonominaukio 1, Espoo FI-00076, Finland

ARTICLE INFO

Article history:
Received 29 July 2020
Accepted 13 May 2021
Available online xxx

Keywords:
Genetic algorithms
Moving-target traveling salesman problem
Dynamic network
Dynamic programming

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, Chvâtal, & Cook, 2006; Lawler, Lenstra, AHG, & Shmoys, 1985; Reinelt, 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, Feillet, 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, Kloster, Nilssen, & 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 intercepts 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, & Savelsbergh, 2020). We will show that, in the case of minimizing travel distance, waiting can be beneficial even within the time window. Several other studies (Choubey, 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

* Corresponding author.
E-mail addresses: alamas@utu.fi (A. Maskooki), kdeb@egr.msu.edu (K. Deb), markku.kallio@aalto.fi (M. Kallio).

https://doi.org/10.1016/j.ejor.2021.05.018

generalized to a variant in which the target locations can be defined more freely. Recently, Cambella, Naoum-Sawaya, and Ghaddar (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, Vaultier, 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, Vaultier, 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[1] 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 (Grob, 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 $\tilde{\alpha}$ with a known probability distribution; i.e., given a realization α of $\tilde{\alpha}$, with $\alpha \leq n$, 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 α (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 Governmnet, 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 27×32 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 (α) 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 α and z that reflects the perceived value of a single measurement among thousands of ships to be inspected annually. Therefore, increasing α 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 α since the schedule for large α 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.

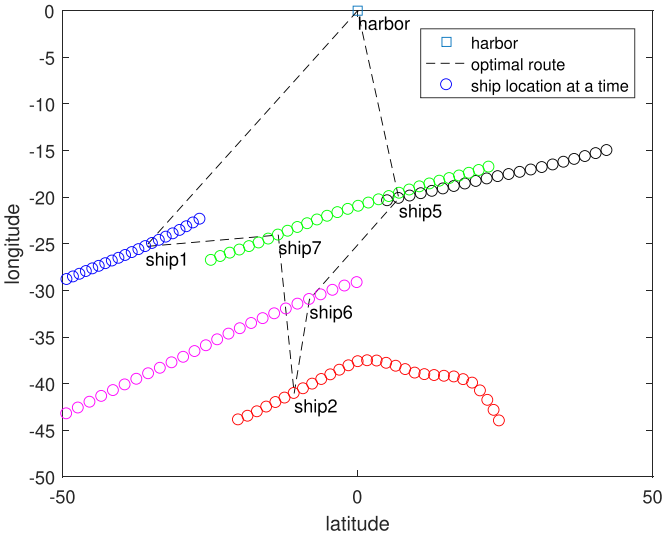


Fig. 1. Schematic representation of the bi-objective routing problem in a dynamic network. Circles show the locations of ships moving in the work area within the time horizon. The dashed line denotes the optimal route of the surveillance boat, starting at the harbor, visiting five ships in the sequence of optimal time slots determined by the dynamic ship-scheduling model, and returning to the harbor: (harbor,time0) → (ship1,time10) → (ship7,time23) → (ship2,time24) → (ship6,time26) → (ship5,time29) → (harbor,time30).

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 $[(k-1)w, kw)$ 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 to not 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_{jl} of, respectively, ship i in time slot k and ship j in time slot l, the distance from v_{ik} to v_{jl} is $d_{ik}^{jl} = \|v_{ik} - v_{jl}\|$ and the travel time to move from v_{ik} to v_{jl} with constant speed c is $t_{ik}^{jl} = d_{ik}^{jl}/c$. 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

possible travel from node v_{ik} to v_{jl}; it is defined only if $i \neq j$ (each ship $i \in N$ is to be visited, at most, once) and both node v_{ik} and node v_{jl} are present (i.e., $k \in S_i$ and $l \in S_j$). The arc (ik, jl) connects the two nodes in layers k and l; the layers are distinct with $k < l$, because no more than one processing is feasible in one time slot. Hence, nodes of the same layer are not connected.

For timing constraints of a tour, we provide a minor simplification of the model proposed by Maskooki and Nikulin (2020). Our formulation proves valuable for the genetic algorithm proposed below for solving the scheduling problem. Consider arc (ik, jl) of traveling from node v_{ik} to v_{jl}. Measurement for a ship j in location v_{jl} is feasible if the surveillance boat arrives at v_{jl} before the end of time slot l (before time wl). Hence, recalling that $s_k = (k-1)w$ is the starting time of time slot k, we find that traveling along arc (ik, jl) is not feasible if $s_k + p_i + t_{ik}^{jl} \geq wl$, so arc (ik, jl) is omitted from the network (in the preprocessing phase). For admissible arcs (ik, jl), we require $s_k + p_i + t_{ik}^{jl} < wl$. For $s_k \leq t < wk$, assume that t denotes the starting time of the measurement for ship i in time slot k. If $t + p_i + t_{ik}^{jl} < wl$, 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 $(t + p_i + t_{ik}^{jl})$ 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 $s_k = wk$ to point to the end of time slot k, for all k, so that the revised admissibility conditions $s_k + p_i + t_{ik}^{jl} < wl$ 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.[2]

Proceeding from the conditions above, we define the set of *admissible arcs* Ξ in the network as follows:

$$\Xi = \{(ik, jl) \mid i, j \in N \cup \{0\},\ i \neq j,\ k \in S_i,\ l \in S_j,\ k < l,\ s_k + p_i + t_{ik}^{jl} < wl\}. \tag{1}$$

For the two criteria, let α denote the number of measurements (visits to ships) and z denote the travel distance within a tour starting at the depot, visiting α ships, and returning to the depot. In the bi-criteria problem, α 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}, $x_{ik}^{\bullet\bullet} = \sum_{jl} x_{ik}^{jl}$ where the set of pairs of indices jl is defined by $(ik, jl) \in \Xi$.

Given endogenous variables α, z, and binary variables x_{ik}^{jl}, for $(ik, jl) \in \Xi$, the routing model satisfies the following constraints:

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

$$x_{\bullet\bullet}^{\bullet\bullet} = \alpha + 1, \tag{2}$$

where $x_{\bullet\bullet}^{\bullet\bullet}$ counts all entries for ships $i \in N \cup \{0\}$ but returning to the depot, $i = 0$, is not counted in α.

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

$$z = \sum_{(ik, jl) \in \Xi} d_{ik}^{jl} x_{ik}^{jl}. \tag{3}$$

[2] 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 following constraint ensures exactly one exit from depot node $i = 0$:

$$x_{0\bullet}^{\bullet\bullet} = 1. \tag{4}$$

For $i \in N$, if ship i is visited in time slot $k \in S_i$, then the immediate predecessor ship $j \neq i$, $j \in N \cup \{0\}$, is visited in time slot $l \in S_j$ with $l < k$, and the immediate successor ship $j \neq i$, $j \in N \cup \{0\}$, is visited in time slot $l \in S_j$ with $l > k$. This is ensured by the following flow-conservation constraints:

$$x_{\bullet\bullet}^{ik} = x_{ik}^{\bullet\bullet}, \qquad \forall\ i \in N,\ k \in S_i. \tag{5}$$

To ensure that each ship $i \in N$ is visited no more than once within the time horizon set, we need the following constraint:

$$x_{i\bullet}^{\bullet\bullet} \leq 1, \qquad \forall\ i \in N, \tag{6}$$

where the left-hand side counts all departures from ship $i \in N$.

We are now ready to state the bi-criteria moving-target TSP as the following vector-maximization problem of finding α, z, and binary variables x_{ik}^{jl}, for all $(ik, jl) \in \Xi$:

$$\text{v-max } \{(\alpha, -z) \mid (2) - (6)\}. \tag{7}$$

The Pareto frontier for problem 7 can be obtained by finding the minimum distance z given α and letting α vary over the set $\{1, \ldots, n\}$. Given that parameter α is known, the problem is to minimize the travel distance by finding binary variables x_{ik}^{jl}, for all $(ik, jl) \in \Xi$, as follows:

$$\min \{z \mid (2) - (6)\}, \tag{8}$$

which is an ILP problem after substitution of z from (3) into problem (8).

As mentioned earlier, in our case of minimizing travel distance, waiting can occur when ship i is visited in time slot k, followed by the immediate successor ship j in time slot l, and the surveillance boat may reach location v_{jl} from location v_{ik} before the beginning of time slot l with the given constant speed c. Such waiting time can be interpreted as traveling from v_{ik} at a sufficiently reduced speed to reach node v_{jl} precisely at time $(l-1)w$ instead of traveling at speed c from location v_{ik} to v_{jl}.

An optimal solution of the ILP problem (8) provides a Pareto-optimal result (α, z) with respect to the chosen value of α. Thus, to find multiple Pareto-optimal solutions, multiple ILP applications must be made by systematically changing α. However, in practice, large-scale problems are most common, and standard ILP solvers are not expected to produce a solution within the available time frame, due to the presence of increasingly large number of variables with increasing α. To tackle this issue efficiently, we propose a customized population-based optimization approach, in the genetic algorithm framework.

3. The customized genetic algorithm (*CGA*)

This section lays out the customized genetic algorithm we developed to estimate the efficient frontier for the bi-criteria vector-maximization problem (7). The estimation is based on the predicted ship trajectories as input parameters, which are available before the start of the planning horizon (see Section 2). Here, a chromosome is a tour (a Hamiltonian circuit) defined by an ordered sequence $I = \{(i_1, k_1), \ldots, (i_\alpha, k_\alpha)\}$; for $\nu = 1, 2, \ldots, \alpha$, ship $i = i_\nu$ is visited (measurement is conducted) in time slot $k = k_\nu \in S_i$ at node v_{ik}. A tour sets off from and ends at the harbor ($i = 0$ is not shown in the sequence I). The population P is a finite set of such tours. Two performance measurements are assigned for each member $I \in P$: $\alpha(I)$ is the number of ships to be visited, and $z(I)$ is the travel distance of tour I. For each tour I, the fitness function $V(I) = \lambda\alpha(I) - z(I)$ is defined as the regularized aggregation of

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_τ denote the population in iteration τ, for $\tau = 0, 1, 2, \ldots, \hat{\tau}$. The iteration's limit $\hat{\tau}$ and the population size $\pi = |P_\tau|$ 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*.

Step 0: Set iteration counter $\tau = 0$ **and** create an *initial population* P_0 at random, using Algorithm 2 (Subsection 3.2).
Step 1: Define *front sets* F_κ, $\kappa = 1, 2, \ldots$,for P_τ **and** assign *front number* $n_F(I)$ and *crowding distance* $\delta(I)$ for all $I \in P_\tau$(Subsection 3.3).
Step 2: **If** $\tau = \hat{\tau}$ **then** stop **else** copy $P_\tau$to an *auxiliary population* P'_τ.
Step 3: Choose two *parents* I_1 and I_2 at random from P_τ,using front number and crowding distance(Subsection 3.4).
Step 4: Apply the *crossover* operation among I_1 and $I_2$to create two offspring O_1 and O_2 (Subsection 3.5).
Step 5: **For** each of the offspring O_o, $o = 1, 2$,
  Step 5.1: For each ship appearing twice in O_o, remove the latter one.
  Step 5.2: Optionally, perform *replacement mutation* on O_o,using Algorithm 3 (Subsection 3.6.1).
  Step 5.3: Optionally, perform *insertion mutation* on O_o,using Algorithm 4 (Section 3.6.2).
  Step 5.4: *Recover feasibility* of O_o by using the DP_r algorithm(Subsection 3.7).
  Step 5.5: Append O_o to auxiliary population P'_τ.
Step 6: **If** $|P'_\tau| < 2\pi$ **then return** to *Step* 3 **else**
  Step 6.1: Apply survival selection to P'_τ to reduce its size to π for creating new population $P_{\tau+1}$ (Subsection 3.8).
  Step 6.2: Increment τ 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 using an operator DP_0, which is a minor modification from standard dynamic programming (Bellman, 1957). In each round with DP_0, 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 $k \in S_i$, let V_{ik} denote the value function at node v_{ik} and let σ_{ik} be the set of ships $j \in N$ along the sub-tour (chosen in backward recursion) from node v_{ik} to the end of the tour (at the depot). At the end of the time horizon, for $i = 0$ and $k \le m$ let $V_{0k} = 0$ and let σ_{0k} be empty. The algorithm's steps are presented in Algorithm 2.

Algorithm 2 Dynamic Programming for Initial Population Generation (DP_0).

Input: predicted trajectories during T, set of admissible arcs Ξ, weight λ, probability p^*
Output: I, $V(I)$

for all time slots $k \in \{m-1, \ldots, 1\}$ going backward in time, **do**
  for all ships $i \in N$ that are present in time slot k, **do**
    if arc $(ik, 0m) \in \Xi$ **then**
      Set the value function $V_{ik} = \lambda - d_{ik}^{0m}$, and set sub-tour $\sigma_{ik} = \{i\}$
    else
      Set $V_{ik} \leftarrow -\infty$
    Save the node v_{0m} (depot) as the successor node of v_{ik}
    for all ships $j \in N$ **and** time slots $l > k$, $l < m$, such that $i \notin \sigma_{jl}$, **do**
      if $(ik, jl) \in \Xi$ **then**
        Save the value $V \leftarrow \lambda - d_{ik}^{jl} + V_{jl}$
      else
        Set $V \leftarrow -\infty$
      Let a be a random draw from uniform distribution $U(0, 1)$
      if $V > V_{ik}$ and $a < p^*$ **then**
        Update the value function $V_{ik} \leftarrow V$ and save v_{jl} as the successor node of v_{ik}
        Update the set of sub-tour ships $\sigma_{ik} \leftarrow \{i\} \cup \sigma_{jl}$
(*Back to depot node* v_{00})Let $V_{00} \leftarrow -\infty$
for all ships $j \in N$ and time slots l, $0 < l < m$, at which j is present, **do**
  if $(00, jl) \in \Xi$ **then**
    Save the value $V \leftarrow -d_{ik}^{jl} + V_{jl}$
  else
    Set $V \leftarrow -\infty$
  if $V > V_{00}$ **then**
    Update the value function $V_{00} \leftarrow V$ and save node v_{jl} as the successor of depot node v_{00}
On the basis of the saved successor nodes, rebuild the tour I 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_{ik}, the algorithm DP_0 searches among *feasible nodes* v_{jl} (each being a location of ship j in time slot $l > k$ with admissible arcs $(ik, jl) \in \Xi$) such that $i \notin \sigma_{jl}$, 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_{jl}, 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 DP_0, given a randomly drawn parameter λ, a feasible tour I is produced with a sequence of ships to be visited. Said procedure violates the dynamic programming principles, and, consequently, the tour I produced by DP_0 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 λ and eligible successor nodes (from the set of feasible nodes) are adopted to diversify the solutions obtained from DP_0. 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, introduced 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_τ or an auxiliary population P'_τ), 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 $\alpha(I)$ and $z(I)$ given for all members $I \in P$. We subdivide P into non-overlapping subsets, front sets F_κ, for $\kappa = 1, 2, 3, \ldots$, such that $P = \cup_\kappa F_\kappa$. For $\kappa = 1$, F_1 is the set of non-dominated members of P, where dominance is defined in terms of the two criteria $\alpha(I)$ and $z(I)$ within P. For $\kappa > 1$, let $P^{\kappa-1}$ be the set obtained after removal of members of F_ξ for all $\xi < \kappa$ from P. If $P^{\kappa-1}$ is non-empty, then F_κ is the set of non-dominated members within $P^{\kappa-1}$.

Front number For all $I \in P$, if $I \in F_\kappa$, then κ is the *front number* of I and we state that $n_F(I) = \kappa$.

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_κ, the crowding distance $\delta(I)$ for $I \in F_k$ is a measure quantifying the density of other solutions (tours) surrounding I in F_k. Thus, crowding distance is determined by the front set for all $I \in F_\kappa$.

Crowding-distance computation For $I \in P$, to define the *crowding distance* $\delta(I)$, we let κ be such that $I \in F_\kappa \subseteq P$, and we define $\Delta\alpha$ and Δz thus:

$$\Delta\alpha = \min_{J\in F_\kappa}\{\alpha(J) \mid J \neq I,\ \alpha(J) \geq \alpha(I)\} - \max_{J\in F_\kappa}\{\alpha(J) \mid J \neq I,\ \alpha(J) \leq \alpha(I)\}, \tag{9}$$

$$\Delta z = \min_{J\in F_\kappa}\{z(J) \mid J \neq I,\ z(J) \geq z(I)\} - \max_{J\in F_\kappa}\{z(J) \mid J \neq I,\ z(J) \leq z(I)\}. \tag{10}$$

Then the crowding distance is

$$\delta(I) = \frac{\Delta\alpha}{r_\alpha} + \frac{\Delta z}{r_z}, \tag{11}$$

where $r_\alpha = \max_{J\in F_\kappa} \alpha(J) - \min_{J\in F_\kappa} \alpha(J)$ and $r_z = \max_{J\in F_\kappa} z(J) - \min_{J\in F_\kappa} z(J)$.

Crowding distances $\delta(I)$ are used to determine the tournament *winner* in the event of a tie (by using front numbers); also, $\delta(I)$ is used as a criterion (larger is better) for choosing members from the auxiliary population P'_τ for the subsequent population $P_{\tau+1}$. 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 $I, I' \in P_\tau$ for a tournament. Ensure that the parents are not identical ($I \neq I'$) and that their cardinalities are at least 2 ($|I| > 1$ and $|I'| > 1$).
- Choose the member with larger front number n_F.
- If I and I' 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 I' for the crossover operation. The first parent I is chosen as described above, while the second parent I' is chosen from the population P_τ 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 I'. Thus, I' solves the following problem:

$$\min_{I'\in P_\tau}\{|\alpha(I') - \alpha(I)| + 0.001z(I') \mid I' \neq I,\ |I'| > 1\}. \tag{12}$$

Here the goal is to mate two members I and I' 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 $I = \{(i_\nu, k_\nu)\}_{\nu=1}^{\alpha}$ and $I' = \{(i'_\nu, k'_\nu)\}_{\nu=1}^{\alpha'}$ with cardinalities $|I| = \alpha$ and $|I'| = \alpha'$. Crossover is performed between parents I and I' to generate two offspring. For choosing the crossing point ν for each parent, we discuss three alternative options (i–iii).

(i) Firstly, for $1 < \nu < \alpha$, let $(i,k) = (i_\nu, k_\nu)$ and $(j,l) = (i_{\nu+1}, k_{\nu+1})$ refer to two successive visits to nodes v_{ik} and v_{jl} in I. Similarly, for $1 < \nu' < \alpha'$, let $(i',k') = (i'_{\nu'}, k'_{\nu'})$ and $(j',l') = (i'_{\nu'+1}, k'_{\nu'+1})$ be two successive visits in I'. Given that ν and ν' are the chosen crossing points, let

$$t_{12} = l'w - (k-1)w - (p + t_{ik}^{j'l'}), \tag{13}$$

$$t_{21} = lw - (k'-1)w - (p + t_{i'k'}^{jl}). \tag{14}$$

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

$$\min_{1<\nu<\alpha} \min_{1<\nu'<\alpha'} \frac{\max(\alpha/2 - \nu, \alpha'/2 - \nu')}{\max(1, t_{12} + t_{21})}. \tag{15}$$

The selection rule (15) tends to keep the crossover points near the middle of tours I and I'. Furthermore, to improve the chance of getting feasible offspring after the crossover operation, that rule tends to maximize the extra time during crossing between I and I'. 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.[3] 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 ν and ν' 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 ν of parent I is chosen by a random draw from a uniform distribution in $[2, \alpha - 1]$ and crossing point ν' of parent I' is based on problem 15 with ν fixed within I.

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

$$\max_{1<\nu'<\alpha'} (1, t_{12} + t_{21}), \tag{16}$$

where t_{12} and t_{21} (in minutes) are defined as above (13,14). This determines the crossing point ν' of I', 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.

[3] If $t_{12} + t_{21}$ is small or negative, we have a safeguard level of 1 min. in the denominator of (15).

Generating offspring After the crossover points ν and ν' are found, two offspring

$$O = \{(i_1, k_1), \ldots, (i_\nu, k_\nu), (j'_{\nu'+1}, l'_{\nu'+1}), \ldots, (i'_{\alpha'}, k'_{\alpha'})\},$$
$$O' = \{(i'_1, k'_1), \ldots, (i'_{\nu'}, k'_{\nu'}), (j_{\nu+1}, l_{\nu+1}), \ldots, (i_\alpha, k_\alpha)\},$$

with cardinalities $\alpha(O) = \nu + \alpha' - \nu'$ and $\alpha(O') = \nu' + \alpha - \nu$, are generated. Thereafter, each of the offspring has options for two types of mutation (replacement and insertion), discussed in Section 3.6. 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 Ξ. Recovery from such infeasibility is handled by means of the DP_r algorithm, introduced in Section 3.7, thereby 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'_τ to be considered for inclusion in the subsequent population $P_{\tau+1}$.

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_{rs} to node v_{jl} 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_{jl} 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 $t_r \leftarrow \infty$
6: **for** all ships r not in the sequence I **and** all time slots s such that $k < s < l$, **do**
7: Let $t_i \leftarrow t^{rs}_{ik}$ and $t_j \leftarrow t^{jl}_{rs}$
8: **if** $(ik, rs), (rs, jl) \in \Xi$ **and** $t_i + t_j < t_r$ **then**
9: Save (r, s) as a candidate for insertion and let $t_r \leftarrow t_i + t_j$
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 $V(I) \leftarrow V(I) + \lambda - c(t_r - t^{jl}_{ik})$
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 $I = \{(i_\nu, k_\nu)\}_{\nu=1}^{\alpha}$ 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 Ξ. Let $N' = \{i_1, i_2, \ldots, i_\alpha\}$ 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 α ships while obeying the order of ships specified in N'. For notational convenience, for $\nu = 1, 2, \ldots, \alpha$, we relabel the ships i_ν in terms of ν, 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, regarding the set of admissible arcs Ξ (1), we define the subset $\Xi' \subseteq \Xi$ as follows:

$$\Xi' = \{(ik, jl) \in \Xi \mid 0 \leq i, j \leq \alpha \text{ and } (i < j \text{ or } j = 0)\}. \tag{17}$$

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

$$\max \quad \lambda x^{\bullet\bullet}_{\bullet\bullet} - \sum_{(ik,jl)\in\Xi'} d^{jl}_{ik} x^{jl}_{ik}, \tag{18}$$

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:

$$x^{\bullet\bullet}_{0\bullet} = 1. \tag{19}$$

For $i \in N'$, if ship i is visited in time slot k, $k \in S_i$, then the flow enters node v_{ik} from some node v_{jl} 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_{jl}

with $j > i$ or $j = 0$ at a later time slot $l > k$ with $l \in S_j$. This is ensured by the following flow-conservation constraints:

$$x^{ik}_{\bullet\bullet} = x^{\bullet\bullet}_{ik} \qquad \forall\ i \in N',\ k \in S_i. \tag{20}$$

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 I', a mutant of I, is determined by the binary variables x^{jl}_{ik}, 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 DP_r.

The DP_r algorithm is similar to DP_0; however, N' determines the number of ships, and the tour obeys the order of ships in N'. In addition, the probability p^* (of accepting successor nodes) is not considered further. Therefore, the DP_r algorithm becomes standard DP with backward recursion. The problem is to find an optimal tour of visiting some or all of the α ships in N' by maximizing the value function (18). If no feasible time slot is found for a ship $i \in N'$, that ship is omitted. We can simplify DP_r by excluding the possibility of omitting visits to some ships in N'. This procedure might be useful for enhancing members of the initial population obtained from DP_0.

3.8. The auxiliary population based on NSGA-II

After generating π offspring, we have an auxiliary population P'_τ of 2π feasible members consisting of π members of current population P_τ and π offspring. The population P'_τ is then sorted into κ front sets in accordance with NSGA-II (Deb, 2002), and a front number $n_F(I)$ is assigned to each tour $I \in F_\xi$, for $\xi \le \kappa$. Crowding distances (11) are determined for members of front set F_κ only.

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

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 DP_0 model via Algorithm 2. Although the Bellman optimality principle is violated in this case and the resulting solutions may not be optimal, DP_0 provides good starting solutions and is computationally quick to evaluate. Furthermore, as stated at the end of Section 3.2, for diversification of the solutions obtained after each run of DP_0, 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 DP_r for adjusting the time slots of a given sequence and, if necessary, omitting ships from the sequence to make the solution feasible. Using DP_r 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.[4] 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.[5] 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 α,

[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.

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 $[0, T]$.

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

Table 2
Summary statistics for the number of α levels, accuracy of the *CGA*, and speed-up with the *CGA* relative to ILP. T: time horizon (hours); $n_{\max}$: number of feasible α levels; n_{GA}: number of final α levels from the *CGA*; n^0_{GA}: number of initial α levels from the *CGA*; n_{lim}: number of α levels at which the ILP solver hits the 2 h time limit; $\Delta z^0 = (z^0_{GA} - z_{ILP})/z_{ILP}$: initial average relative difference in total travel distance, a positive value indicating better performance for ILP); $\Delta z = (z_{GA} - z_{ILP})/z_{ILP}$: final average relative difference; $\Delta HV^0 = (H_{ILP} - HV^0_{GA})/HV_{ILP}$: initial relative hypervolume difference, a positive value indicating better performance for ILP; $\Delta HV = (HV_{ILP} - HV_{GA})/HV_{ILP}$: final relative hypervolume difference; sec_{ILP}: total time (wallclock sec.) of the ILP solver; sec_{GA}: total time (CPU sec.) of the *CGA*; and speed-up: sec_{ILP}/sec_{GA}.

	Number of levels α				Differences				Solution time		
T	$n_{\max}$	n_{GA}	n^0_{GA}	n_{lim}	$\Delta\ z^0$	$\Delta\ z$	$\Delta\ HV^0$	$\Delta\ HV$	sec_{ILP}	sec_{GA}	Speed-up
4	15	15	13	0	0.006	0.002	0.0158	0.0015	93	663	<1
6	20	19	9	1	0.030	0.018	0.0449	0.0076	13,868	1659	8
8	25	25	10	3	0.053	0.020	0.0927	0.0150	39,592	2418	16
10	31	31	12	9	0.038	0.012	0.0514	0.0005	85,945	3438	25
12	36	36	17	14	0.028	0.016	0.0477	0.0023	122,215	5052	24
14	40	40	16	13	0.056	0.031	0.0455	0.0057	107,479	6105	18
16	48	46	20	16	0.064	0.042	0.0481	0.0031	141,579	7106	20

starting with $\alpha = 1$, we let α 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 α 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 the $T > 8$ 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 α.

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 $\hat{\tau} = 50$ for all experiments. For generating the initial population in iteration τ=0, we use the modified dynamic programming procedure DP_0 (introduced in Section 3.2). For each run of DP_0, the weighting parameter λ (the weight for the objective α 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 α levels. In the recursive steps of DP_0, a probability $p^* = 0.8$ is used to randomly accept feasible successor nodes (see Section 3.2), and thereby to increase the diversity in the initial population. The algorithm DP_0 repeats until $\pi = 50$ distinct members are generated for the initial population P_0. No replacement or insertion mutation is applied to the members (tours) produced by DP_0.

After this, at the beginning of each iteration τ, (i) the population P_τ is subdivided into front sets, (ii) the front numbers are assigned, and (iii) the crowding distances are calculated for all members of P_τ as is 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_τ 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 α, 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 α 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 α 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 $p_R = 0.2$, and the insertion mutation is applied with probability $p_I = 1$; however, we restrict the number of insertions to 0, 1, or 2 ships at most, each with equal probabil-

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_f with a large weight λ=1,000 for the objective α. If the final offspring solution is not a duplicate of an existing population member, it is appended to the auxiliary population P'_τ. Offspring are generated until P'_τ has $2\pi = 100$ members.

At the final stage of iteration τ, the set P'_τ 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'_τ 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 bi-criteria 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 α, including the maximum $\alpha = 15$, are generated already by DP_0 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 α 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 DP_0 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 α levels found by DP_0, are shown in Table 2 in the "differences" columns. After 40 iterations, all α 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 α 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 DP_0, 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 α 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 α 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 α 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 α 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 α levels are produced in F_1 (the missing levels are $\alpha = 3$ and 14). The complete set of α 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 α levels are present in F_1 after iteration 16 (the missing α 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.

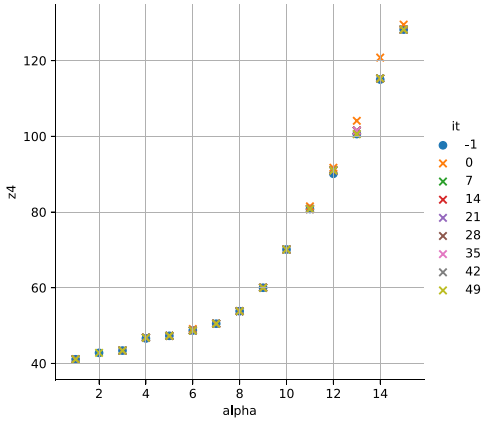
Finally, the $T = 16$ h case, representing the largest problem in our study, has an average error of 0.04 (Fig. 4(b)) at the final iteration (no. 50). At the maximum level, $\alpha = 48$, the relative error is -0.006; i.e., the solution produced by the *CGA* is better than the one obtained by the ILP solver in two hours. The complete set of α levels is not produced in F_1; however, after 50 iterations only two α 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 α levels, average relative error, hypervolume, and computation time. The number of feasible α levels ($n_{\max}$) varies from 15 to 48 across the T values. In our results, the number of α levels in the first front of the final 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 α levels in the first front of the initial population with the *CGA* (n^0_{GA}) is less than half of the full count n_{GA}, except for $T = 4$ h. The number of α 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.

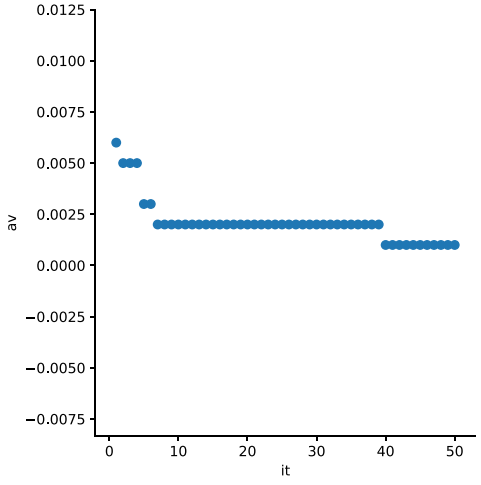
For the tour travel distance z_{GA}s produced by the *CGA* and the z_{ILP}s from the ILP, the average relative differences $(z_{GA} - z_{ILP})/z_{ILP}$ (over α levels) in early iterations are in the range 0.6 to 6.4% and decrease with further iterations to the final range of 0.2% to 4.2% of the ILP solutions.

Given a reference point $(\alpha, z) = (0, z_{\max})$, the hypervolume ($HV$) of a given frontier set P is the surface area (volume[6]) of

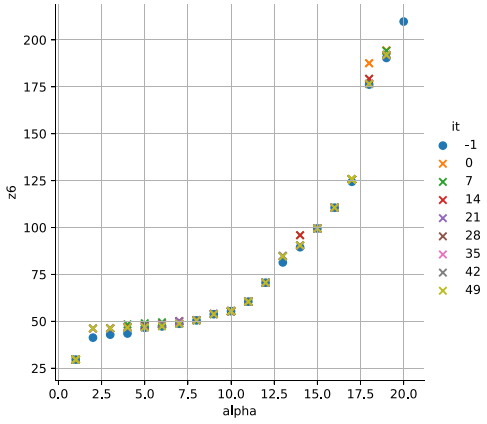
[6] Surface area here refers to points (α, z) with $\alpha \geq 0$ and $z_{max} \geq z \geq 0$ weakly dominated by P. Alternatively, if only values $\alpha = 1, 2, 3, \ldots$ are considered, then we



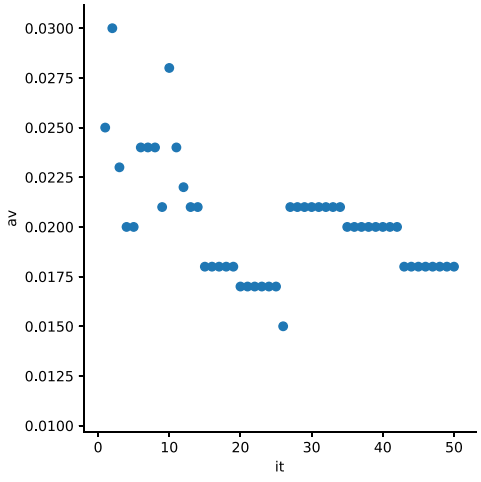
(a) Optimal frontier from CGA and ILP for $T = 4$ h.



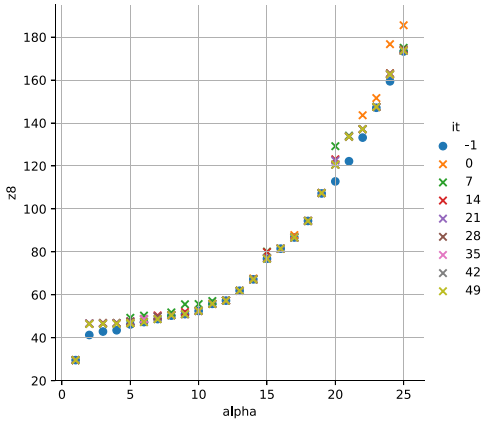
(b) Average relative error of CGA for $T = 4$ h.



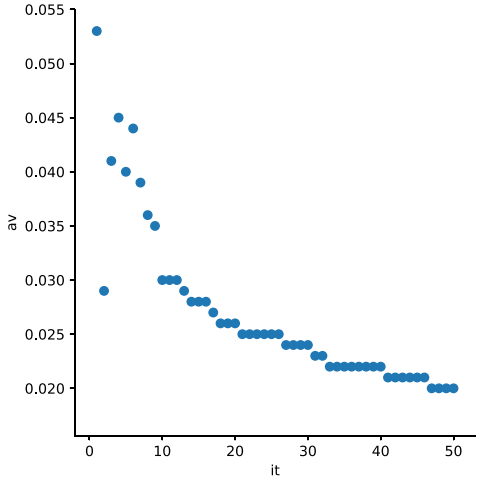
(c) Optimal frontier from CGA and ILP for $T = 6$ h.



(d) Average relative error of CGA for $T = 6$ h.

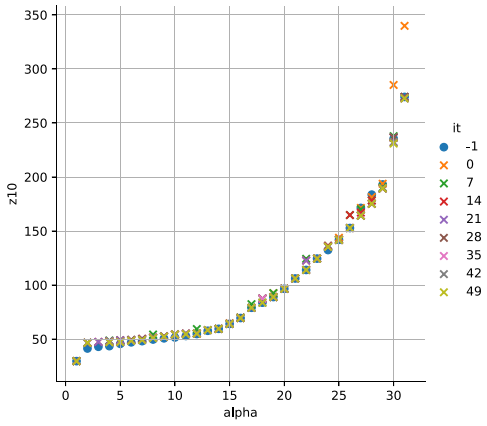


(e) Optimal frontier from CGA and ILP for $T = 8$ h.

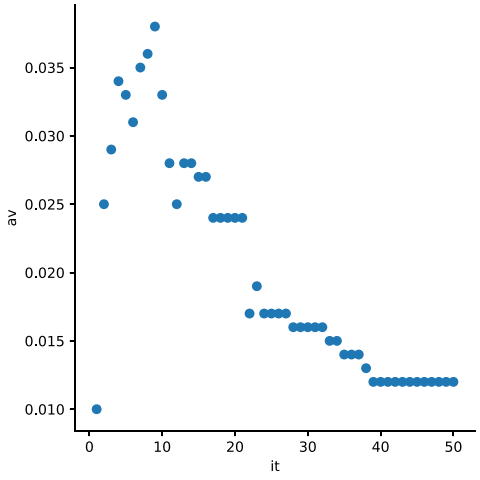


(f) Average relative error of CGA for $T = 8$.

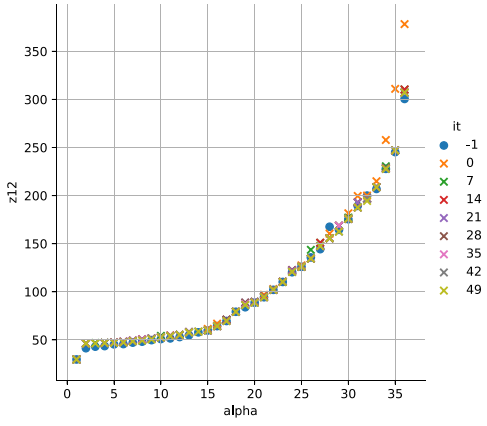
Fig. 2. The figures on the left show comparison of the *CGA*'s front set F_1 with the Pareto frontier obtained by the ILP model for the cases $T = 4$, 6, and 8 h. Iteration -1 indicates the ILP-model-produced optimal frontier, in blue dots. The right-hand figures show the average relative-error rate for the *CGA*'s front set F_1 over the iterations.



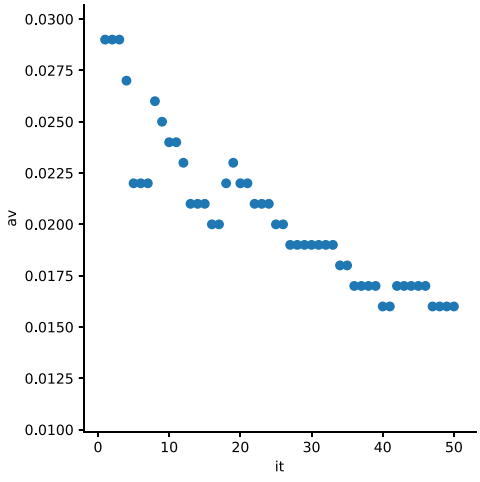
(a) Optimal frontier from CGA and ILP for $T = 10$.



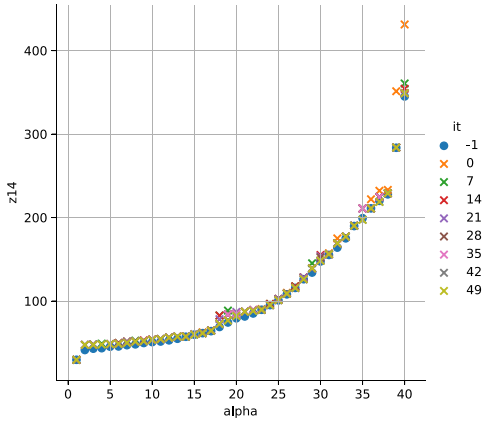
(b) Average relative error of CGA for $T = 10$ h.



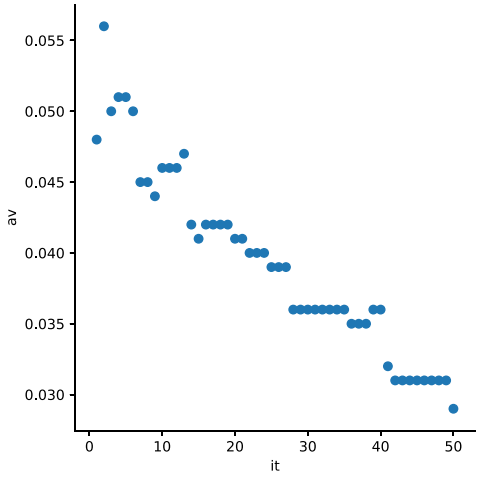
(c) Optimal frontier from CGA and ILP for $T = 12$ h.



(d) Average relative error of CGA for $T = 12$ h.

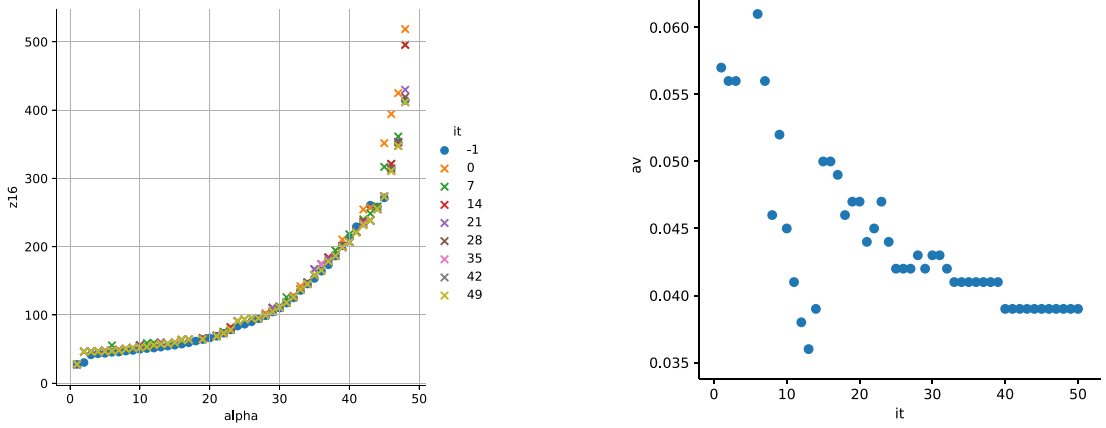


(e) Optimal frontier from CGA and ILP for $T = 14$ h.



(f) Average relative error of CGA for $T = 14$ h.

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.



(a) Optimal frontier from *CGA* and ILP for $T = 16$ h.

(b) Average relative error of *CGA* for $T = 16$ h.

Fig. 4. The left-hand figures show comparison of the *CGA*'s front set F_1 with the Pareto frontier obtained by the ILP model for the case $T = 16$ h. Iteration -1 indicates the ILP-produced optimal frontier, in blue dots. The right-hand figures show the average relative-error rate for the *CGA*'s front set F_1 plotted against iterations.

points (α, z) that (i) are dominated by some point in P and (ii) satisfy $\alpha \geq 0$ and $z \leq z_{\max}$ Deb (2002). For each T, we consider three frontier sets: the set of Pareto points P_{ILP} obtained from the ILP model, the set P^0_{GA} obtained from DP_0 for the initial population from the *CGA*, and P_{GA} obtained via the *CGA* for the final population; for each T separately, $z_{\max}$ is the largest travel distance observed in the three sets of Pareto frontiers. The respective hypervolumes are denoted by HV_{ILP}, HV^0_{GA}, and HV_{GA}. Table 2 shows that the relative difference $(H_{ILP} - HV^0_{GA})/HV_{ILP}$ initially is in the 1.6% to 9.3% range. At the final iteration, $(HV_{ILP} - HV_{GA})/HV_{ILP}$ 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_{ILP} in wallclock sec.), the solution time for the *CGA* (sec_{GA} in CPU sec.), and the speed-up sec_{ILP}/sec_{GA} 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 α 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_{ILP} in Table 2. Secondly, the solution time sec_{GA} for the *CGA* is given in CPU seconds, whereas solution time sec_{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.

The observation in all cases above attests that large α levels including the maximum often get produced by DP_0 in the front set F_1 of the initial population with a good accuracy (as indicated by the values of Δz^0 and ΔHV^0). Therefore, we find that DP_0 provides a fast and 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_1. 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 α 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 ($T \geq 8$ 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 work day, 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 α 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-

define the hypervolume by $\sum_{\alpha>0} h_\alpha$ where h_α is the length of the line segment of points (α, z) dominated weakly by some point in P. However, the numerical value of the hypervolume is the same between the two cases.

Table B.3
The ILP solution (wallclock) times in seconds for each case T and each level of α. Figures above 7,200 s indicate hitting the 2 h solver time limit.

α	$T=4$	$T=6$	$T=8$	$T=10$	$T=12$	$T=14$	$T=16$
1	3	6	13	19	26	33	53
2	3	48	1072	278	527	742	94
3	2	76	194	290	872	1554	2284
4	5	36	127	305	591	970	2277
5	3	41	136	514	426	712	1453
6	2	17	89	209	281	278	1346
7	3	18	51	121	197	295	1186
8	3	11	61	134	172	252	809
9	3	13	28	74	148	175	753
10	6	8	25	68	113	151	403
11	8	8	29	39	63	103	228
12	9	26	16	25	46	61	186
13	14	48	26	41	33	44	168
14	18	53	53	25	63	70	177
15	11	108	1167	47	32	81	125
16		88	480	91	63	45	120
17		86	718	1170	207	43	73
18		35,857	530	1033	2810	211	104
19		5977	3774	1126	3233	472	104
20		1888	522	3278	2526	438	74
21			6919	6547	2474	314	73
22			1962	5518	6230	282	147
23			21,755	7267	7219	337	427
24			15,997	7218	7221	471	670
25			16,268	7213	7219	550	631
26				7212	7219	1961	438
27				7214	7220	2909	545
28				7216	7221	7224	528
29				7217	7220	7224	723
30				7218	7220	7225	837
31				7218	7220	7225	3269
32					7220	7224	5444
33					7219	7225	7249
34					7220	7227	7249
35					7223	7225	7251
36					7221	7226	7252
37						7224	7250
38						7226	7254
39						7225	7251
40						7225	7249
41							7256
42							7248
43							7250
44							7249
45							7250
46							7250
47							7250
48							7250

vealed that the probabilistic dynamic program DP_0, 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 (Groba 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 unavoidable 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
m = number of time slots of length w
$k = 0, 1, 2, \ldots, m$, 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 $[0, T]$; $i = 0$ refers to the depot
v_{ik} = coordinate vector of the location of ship i in time slot k
$S_i = \{a_i, \ldots, b_i\}$, set of time slots for which ship i is in the work area
$d_{ik}^{jl} = \|v_{ik} - v_{jl}\|$ = distance (km) from v_{ik} to v_{jl}
c = speed (km/h) of the emissions-measurement boat
$t_{ik}^{jl} = d_{ik}^{jl}/c$ = time (h) taken to move from v_{ik} to v_{jl} 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

Applegate, D. L., Bixby, R. E., Chvâtal, V., & Cook, W. J. (2006). *The traveling salesman problem: A Computational study*. Princeton University Press.
MOSEK ApS (2019). MOSEK fusion API for.NET manual. version 9.0. https://docs.mosek.com/9.0/dotnetfusion/mip-optimizer.html.
Archetti, C., Feillet, D., Mor, A., & Speranza, M. G. (2020). Dynamic traveling salesman problem with stochastic release dates. *European Journal of Operational Research, 280*, 832–844.
Bellman, R. E. (1957). *Dynamic programming*. Princeton University Press.
Bertsimas, D. (1992). A vehicle routing problem with stochastic demand. *Operations Research, 40*, 574–585.
Bertsimas, D., & Ryzin, G. V. (1991). A stochastic and dynamic routing problem in the euclidean plane. *Operations Research, 39*(4), 601–615.
Beume, N., Fonseca, C. M., López-Ibáñez, M., Paquete, L., & Vahrenhold, J. (2009). On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation, 13*(5), 1075–1082.
Bourjolly, J., Gurtuna, O., & Lyngvic, A. (2006). On-orbit servicing: A time-dependent, moving-target traveling salesman problem. *International Transactions in Operational Research, 13*, 461–481.
Bullo, F., Frazzoli, E., Pavone, M., Savla, K., & Smith, S. L. (2011). Dynamic vehicle routing for robotic systems. *Proceedings of the IEEE, 99*(9), 1482–1504.
Cambella, C., Naoum-Sawaya, J., & Ghaddar, B. (2018). The vehicle routing problem with floating targets: Formulation and solution approaches. *Informs Journal on Computing, 30*(3), 554–569.
Choubey, N. (2013). Moving target travelling salesman problem using genetic algorithm. *International Journal of Computer Applications, 70*, 30–34.
Chowdhury, S., Marufuzzaman, M., Tunc, H., Bian, L., & Bullington, W. (2019). A modified ant colony optimization algorithm to solve a dynamic traveling salesman problem: A case study with drones for wildlife surveillance. *Journal of Computational Design and Engineering, 6*(3), 368–386.
Deb, K. (2002). *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: Wiley.
Deb, K., & Myburgh, C. (2017). A population-based fast algorithm for a billion-dimensional resource allocation problem with integer variables. *European Journal of Operational Research, 261*(2), 460–474.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*, 182–197.
Ehrgott, M., Fonseca, C. M., Gandibleux, X., Hao, J., & Sevaux, M. (2009). Evolutionary multi-criterion optimization. In *5th international conference, EMO 2009, Nantes, France, April 7–10, 2009 proceedings. lecture notes in computer science 5467*. Springer.
Eksioglu, B., Vural, A. V., & Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers and Industrial Engineering, 57*(4), 1472–1483.
Ministry of Economic Affairs and Employment (2020). Finland's long-term low greenhouse gas emission development strategy. United Nations Climate Change (unfccc.int) online publication. Accessed January 2021.
Finnish Governmnet (valtioneuvosto.fi) (2020). https://valtioneuvosto.fi/en/-/finland-seeks-an-ambitious-solution-to-reduce-greenhouse-gas-emissions-from-shipping. Accessed January 2021.
Flatberg, T., Hasle, G., Kloster, O., Nilssen, E. J., & Riise, A. (2007). Dynamic and stochastic vehicle routing in practice. In V. Zeimpekis, C. D. Tarantilis, G. M. Giaglis, & I. Minis (Eds.), *Dynamic fleet management. operations research/computer science interfaces series 38*. Boston, MA: Springer.
Fourer, R., Gay, D. M., & Kernighan, B. W. (2003). *AMPL: A modeling language for mathematical programming* (2nd ed.). Duxbury-Thomson.
Gao, S., Wang, Y., Cheng, J., Inazumi, Y., & Tang, Z. (2016). Ant colony optimization with clustering for solving the dynamic location routing problem. *Applied Mathematics and Computation, 285*, 149–173.
Goldberg, D. E. (1989). *Genetic algorithms for search, optimization, and machine learning*. Reading, MA: Addison-Wesley.
Grob, M. (2006). Routing of platforms in a maritime surface surveillance operation. *European Journal of Operational Research, 170*, 613–628.
Groba, C., Sartal, A., & Vázquez, X. (2015). Solving the dynamic travelling salesman problem using a genetic algorithm with trajectory prediction: An application to fish aggregating devices. *Computers and Operations Research, 56*, 22–32.
Groba, C., Sartal, A., & Vázquez, X. (2018). Integrating forecasting in metaheuristic methods to solve dynamic routing problems: Evidence from the logistic processes of tuna vessels. *Engineering Applications of Artificial Intelligence, 76*, 55–66.
Hammar, M., & Nilsson, B. J. (2002). Approximation results for kinetic variants of TSP. *Discrte & Computational Geometry, 27*, 635–651.
Hassoun, M., Shoval, S., Simchon, E., & Yedidsion, L. (2020). The single line moving target traveling salesman problem with release times. *Annals of Operations Research, 289*, 449–458.
Helvig, C. S., Robins, G., & Zelikovsky, A. (2003). The moving-target traveling salesman problem. *Journal of Algorithms, 49*, 153–174.
Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: MIT Press.
Jaillet, P. (1988). A priori solution of a travelling salesman problem in which a random subset of the customers are visited. *Operations Research, 36*, 929–936.
Jiang, Q., Sarker, R., & Abbass, H. (2005). Tracking moving targets and the non-stationary traveling salesman problem. *Complexity International, 11*, 171–179.
Klapp, M. A., Erera, A. L., & Toriello, A. (2018). The one-dimensional dynamic dispatch waves problem. *Transportation Science, 52*, 402–415.
Kryazhimskiy, A. V., & Savinov, V. B. (1993). The travelling-salesman problem with moving objects. *Journal of Computer and System Sciences International, 33*, 144–148.
Laporte, G., Louveaux, F., & Mercure, H. (1992). The vehicle routing problem with stochastic travel times. *Transportation Science, 26*(3), 161–170.
Lawler, E. L., Lenstra, J. K., AHG, R., & Shmoys, D. B. (1985). *The traveling salesman problem: A guided tour of combinatorial optimization*. New York: Wiley.
Li, C., Yang, M., & Kang, L. (2006). A new approach to solving dynamic traveling salesman problems. *Asia-Pacifif Conference on Simulated Evolution and Learning*, 236–243.
Marlow, D., Kilby, P., & Mercer, G. (2007). The travelling salesman problem in maritime surveillance-techniques, algorithms and analysis. *Proceedings of the International Congress on Modelling and Simulation*, 684–690.
Maskooki, A., & Nikulin, Y. (2020). Bi-objective routing in a dynamic network: An application to maritime logistics. *Control and Cybernetics, 49*(2), 211–232.
Maskooki, A., Virjonen, P., & Kallio, M. (2020). Assessing the prediction uncertainty in a route optimization model for autonomous maritime logistics. *International Transactions in Operational Research, 28*(4), 1765–1786.
Mavrovouniotis, M., Müller, F. M., & Yang, S. (2016). Ant colony optimization with local search for dynamic traveling salesman problems. *IEEE Transactions on Cybernetics, 99*, 1–14.
Montoya-Torres, J. R., Franco, J. L., Isaza, S. N., Jiménez, H. F., & Herazo–Padilla, N. (2015). A literature review on the vehicle routing problem with multiple depots. *Computers and Industrial Engineering, 79*, 115–129.
Moraes, R. S., & Freitas, E. P. (2019). Experimental analysis of heuristic solutions for the moving target traveling salesman problem applied to a moving targets monitoring system. *Expert Systems With Applications, 136*, 392–409.
Nguyen, T., Yang, S., & Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation, 6*, 1–24.
Pavone, M., & Frazzoli, E. (2010). Dynamic vehicle routing with stochastic time constraints. In *IEEE international conference on robotics and automation, Anchorage, AK. 1460–1467*.
Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research, 225*(1), 1–11.
Reinelt, G. (1994). *The traveling salesman: Computational solutions for TSP applications*. Berlin: Springer-Verlag.
Secomandi, N. (2003). Analysis of a rollout approach to sequencing problems with stochastic routing applications. *Journal of Heuristics, 9*, 321–352.
Smith, S. L., Pavone, M., Bullo, F., & Frazzoli, E. (2010). Dynamic vehicle routing with priority classes of stochastic demands. *SIAM Journal on Control and Optimization, 48*(5), 3224–3245.
Toriello, A., Haskell, W. B., & Poremba, M. (2014). A dynamic traveling salesman problem with stochastic arc costs. *Operations Research, 62*(5), 1107–1125.
Finnish Transport Infrastructure Agency (Väylävirasto) (2019). https://vayla.fi/. Accessed October 2019.
Viel, C., Vaultier, U., Wan, J., & Jaulin, L. (2019). Genetic algorithm-based multiple moving target reaching using a fleet of sailboats. *IET Cyber-Systems and Robotics, 1*(3), 93–100.
Virjonen, P., Nevalainen, P., Pahikkala, T., & Heikkonen, J. (2018). Ship movement prediction using k-NN method. In *2018 baltic geodetic congress (BGC geomatics), 304–309*.
Vu, D. M., Hewitt, M., Boland, N., & Savelsbergh, M. (2020). Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. *Transportation Science, 54*(3), 703–720.
Xu, H., Pu, P., & Duan, F. (2018). Dynamic vehicle routing problems with enhanced ant colony optimization. *Discrete Dynamics in Nature and Society, 2018*, 1–13.
Zhou, A., Kang, L., & Yan, Z. (2003). Solving dynamic TSP with evolutionary approach in real time. In *Proceedings of IEEE-CEC 2003: 2* (pp. 951–957).