

# Data-Driven Hull-Fouling Cleaning Schedule Optimization to Reduce Carbon Footprint of Vessels

Samuel Ward  

School of Mathematical Sciences,  
University of Southampton,  
SO17 1BJ Southampton, UK

Marah-Lisanne Thormann  

School of Mathematical Sciences,  
University of Southampton,  
SO17 1BJ Southampton, UK

Julian Wharton  

National Centre for Advanced Tribology at Southampton (nCATS),  
University of Southampton,  
SO17 1BJ Southampton, UK

Alain Zemkoho  

School of Mathematical Sciences,  
University of Southampton,  
SO17 1BJ Southampton, UK

---

## Abstract

---

In response to climate change, the International Maritime Organization has introduced regulatory frameworks to reduce greenhouse gas emissions from international shipping. Compliance with these regulations is increasingly expected from individual shipping companies, compelling vessel operators to lower the CO<sub>2</sub> emissions of their fleets while maintaining economic viability. An important step towards achieving this is performing regular hull and propeller cleaning; however, this entails significant costs. As a result, assessing whether ship performance has declined sufficiently to warrant cleaning from an environmental and economic standpoint is a critical task to ensure both long-term viability and regulatory compliance. In this paper, we address this challenge by proposing a novel data-driven dynamic programming approach to optimize vessel cleaning schedules by balancing both environmental and economic considerations. In numerical experiments, we demonstrate the usefulness of our proposed methodology based on real-world sensor data from ten tramp trading vessels. The results confirm that over a four-year period, fuel consumption can be reduced by up to 5%, even when accounting for the costs of one or two additional cleaning events.

**Keywords:** Biofouling; Cleaning Schedule Optimization; Dynamic Programming; Fuel Consumption; Machine Learning

**Mathematics Subject Classification (2020):** 90B06; 90B25; 90C27; 90C39

---

**Funding Acknowledgement:** The work presented in this paper was funded by both the SMMI Research Collaboration Stimulus Fund 2022/23 and an EPSRC Impact Acceleration Account grant with reference 9060958. Samuel Ward's PhD is jointly funded by DAS Ltd and the School of Mathematical Sciences at the University of Southampton. Marah-Lisanne Thormann receives a PhD Studentship from the School of Mathematical Sciences at the University of Southampton. Alain Zemkoho is supported by the EPSRC with grant reference EP/X040909/1.

**Contents**

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Ship Performance Modeling . . . . .	3
1.2	Hull Fouling and its Impacts . . . . .	4
1.3	Hull Cleaning Events and FOC Optimization . . . . .	5
1.4	Main Contributions & Paper Outline . . . . .	6
<b>2</b>	<b>Fuel Predictions with Explainable Machine Learning</b>	<b>6</b>
2.1	Preliminaries on Regression Models . . . . .	6
2.2	Shapley Additive Explanations . . . . .	7
<b>3</b>	<b>Data-Driven Hull Fouling Cleaning Schedule Optimization</b>	<b>8</b>
<b>4</b>	<b>Empirical Data Sources</b>	<b>11</b>
4.1	Cleaning Reports . . . . .	11
4.2	Ship Sensor Data . . . . .	12
4.3	Weather and Marine Data . . . . .	14
<b>5</b>	<b>Practical Application &amp; Numerical Studies</b>	<b>14</b>
5.1	Selection of Fuel Prediction Function . . . . .	15
5.2	Empirical Results of Cleaning Schedule Optimization . . . . .	18
<b>6</b>	<b>Conclusions</b>	<b>20</b>
	<b>References</b>	<b>20</b>
<b>A</b>	<b>Appendix</b>	<b>26</b>
A.1	Theorem A.1 . . . . .	26
A.2	Theorem A.2 . . . . .	27
A.3	Proof of Theorem 3.1 . . . . .	28
A.4	Linear Regression . . . . .	28
A.5	Least Absolute Shrinkage and Selection Operator . . . . .	29
A.6	Multilayer Perceptron . . . . .	29
A.7	Support Vector Regression . . . . .	29
A.8	k-Nearest Neighbors . . . . .	30
A.9	Random Forest . . . . .	30
A.10	Extremely Randomized Trees . . . . .	30
A.11	Extreme Gradient Boosting . . . . .	31
A.12	K-Fold Cross Validation . . . . .	31
A.13	Further Regression Scores . . . . .	32
A.14	Bootstrap Confidence Intervals of Performance Metrics . . . . .	32
A.15	Python implementation . . . . .	34

## 1. Introduction

“Global warming is not a prediction. It is happening” [Hansen (2012)]. A key driver of this global climate change is the accumulation of CO<sub>2</sub> in the atmosphere [Nakamura and Ishida (2025)]. As one of the biggest contributors to global CO<sub>2</sub> emissions, the transportation industry is therefore required to reduce its environmental impact [Din et al. (2023); Chapman (2007, pp. 354–355)]. Within this sector, international shipping plays a central role, since approximately 90% of global goods are transported by sea [Parviainen et al. (2018, p. 50); Chapman (2007, p. 362)]. In 2012, global shipping emissions amounted to about 938 million tonnes of CO<sub>2</sub>, and 961 million tonnes of CO<sub>2e</sub> that combines CO<sub>2</sub>, CH<sub>4</sub> and N<sub>2</sub>O [Huang et al. (2022, p. 1)]. These emissions represented roughly 2.2% of global anthropogenic Green House Gases (GHGs) [Huang et al. (2022, p. 1)]. In response, the International Maritime Organization (IMO) has set targets to reduce GHG emissions by at least 20% by 2030, 70% by 2040, and to achieve net-zero by 2050 [IMO (2023b, p. 6)]. These objectives are reinforced by the EU’s Monitoring, Reporting, and Verification (MRV) regulation, which mandates reductions in maritime CO<sub>2</sub> emissions and introduces penalties for non-compliance [EU (2016, p. 3)].

Alongside these more stringent regulations, the IMO has increasingly held individual shipping companies accountable to meet the industry’s GHG emission targets. As a result, ship operators are now required to report performance ratings including the Carbon Intensity Indicator (CII), which ranks vessels on a scale from A to E based on their energy efficiency [IMO (2016)]. Under this regulatory framework, lower vessel ratings may result in financial penalties and fewer charter opportunities. A key component of the CII is the vessel-specific fuel efficiency, which can be improved through lower operating speeds, advanced engine technology, and optimized hull design [Chapman (2007, p. 362)]. In particular, the condition of the hull is crucial, as it begins to deteriorate once vessels operate in water due to the buildup of hull and propeller fouling, which increases hydrodynamic drag [Demirel et al. (2019, pp. 1–2)]. At present, the main strategy to mitigate this deterioration is to regularly perform costly hull inspection, cleaning, and repainting [Granhag et al. (2023)]. Consequently, shipping companies face the following trade-off: cleaning frequently to reduce environmental impact versus conducting maintenance only when the associated costs are offset by fuel efficiency gains. To reconcile both considerations, one of the major open issues is determining the optimal cleaning timing, as the hull’s condition cannot be assessed without costly in-water inspections. This paper addresses this challenge and proposes a data-driven optimization algorithm based on dynamic programming, which identifies the optimal cleaning schedule by approximating the impact of hull fouling on fuel efficiency.

### 1.1. Ship Performance Modeling

Optimizing the energy efficiency of maritime transportation has long been a complex challenge, which is typically addressed through measures related to ship design and operation [Lang et al. (2022, p. 1)]. While numerous indicators exist for operational measures, those derived from fuel consumption and speed remain the most prevalent [Soner et al. (2019, p. 545)]. Specifically, the Fuel Oil Consumption (FOC) is directly integrated into the CII and represents more than a quarter of a vessel’s operating costs [Gkerekos et al. (2019, p. 1)]. As GHG emissions are also directly proportional to this consumption [Coraddu et al. (2017, p. 351)], FOC is adopted as the primary metric for evaluating operational costs and energy efficiency within our data-driven optimization approach introduced in Section 3.

Given that FOC serves as a key metric in this study, its accurate prediction under varying operational conditions is essential, which requires a mathematical model capable of incorporating multiple variables and their interdependencies. In the literature, modeling of FOC is an active research area, with numerous recent studies employing Machine Learning (ML)-based approaches for prediction [Agand et al. (2023); Zhou et al. (2022); Moreira et al. (2021); Yan et al. (2020); Soner et al. (2019); Gkerekos et al. (2019); Coraddu et al. (2017)]. For example, Gkerekos et al. (2019) conducted a comparative study employing a diverse set of ML algorithms for FOC modeling. Across multiple performance metrics, Support Vector Machine (SVM) and Extremely Randomized Trees (ET) achieved the highest predictive accuracy. Conversely, Agand et al. (2023) reported eXtreme Gradient Boosting (XGB) as the top-performing model in their comparison, while Yan et al. (2020)

identified **Random Forest (RF)** as the best performer. These examples underscore that no consensus has been reached on an overall best modeling approach, primarily due to the heterogeneity of datasets, variable selection, and performance metrics across existing studies. Accordingly, this paper provides a brief overview of commonly used **ML** models in Subsection 2.1, and later compares their performance in Subsection 5.1 to identify the most suitable approach for predicting **FOC** on the given empirical datasets.

Once an appropriate mathematical framework for predicting **FOC** is established, the influence of the underlying variables can be assessed. For linear models, this is typically achieved by interpreting regression coefficients, whereas for most of the other **ML**-based approaches, interpretability remains considerably more challenging. These so-called black-box algorithms are difficult to interpret and often lack transparency for domain experts due to their reliance on numerous complex non-linear transformations [Loyola-González (2019, p. 154097)]. One way to mitigate this black-box characteristics is through tools from **eXplainable Artificial Intelligence (XAI)**, which have been developed in recent years and are gaining increasing attention in the academic literature. In particular, a widely adopted technique is the use of **SHapley Additive exPlanations (SHAP)** values, which have been applied by several researchers to enhance the interpretability of **FOC** models [Handayani et al. (2023); Ma et al. (2023); Wang et al. (2023); Molnar (2022, ch. 9.6)]. In Subsection 2.2, this approach is explained in more detail, and in Subsection 5.1 it is applied to verify whether the independent variables exert the expected influence on **FOC** predictions. The main novelty of this aspect of our study lies in computing and evaluating **SHAP** values for input variables approximating the effects of hull and propeller fouling on **FOC**.

## 1.2. Hull Fouling and its Impacts

Before discussing suitable approaches for approximating and detecting hull fouling, we define the phenomenon as accumulation of living organisms on the submerged surface of a vessel. It begins with a thin biofilm of bacteria that roughens the hull and propellers. Later in the process, larger macroalgae, plants and hard foulers (i.e., barnacles and mussels) will attach to the surface [AlSawaftah et al. (2022, p. 3)]. From an environmental perspective, hull fouling has two major negative impacts. The first relates to invasive species carried within the ship's fouling, which can outcompete local species or transmit diseases between normally disconnected habitats [Lewis and Coutts (2009); Floerl and Inglis (2003, p. 116)]. For instance, Darwin (1851) had already noted the spread of barnacles transported by trading ships more than a century ago. Similarly, a review by Molnar et al. (2008, p. 485) concluded that invasive marine species are a major threat to biodiversity, with profound ecological and economic consequences. In line with these findings, the **IMO** has identified this aspect of biofouling "... as one of the greatest threats to the ecological and the economic well being of the planet ..." [IMO (2023a)].

In addition to the spread of invasive species, a second negative environmental impact of hull fouling originates from increased frictional resistance between the ship's hull and the surrounding water, which can raise **FOC** by approximately 5–17% [Adland et al. (2018); de Haas et al. (2023); Bakka et al. (2022); Nikolaidis and Themelis (2022); Coraddu, Lim, Oneto, Pazouki, Norman and Murphy (2019); Coraddu, Oneto, Baldi, Cipollini, Atlar and Savio (2019); Uzun et al. (2019); Demirel et al. (2017)]. Depending on the fuel type, this may correspond to a substantial increase in emissions of sulfur, particles, and carbon dioxide to the atmosphere [Olivera et al. (2022, p. 3)]. This highlights the direct relevance of hull fouling to climate change and its conflict with the **IMO**'s **GHG** reduction targets. Consequently, from an environmental perspective, maintenance activities such as hull inspections and cleanings should be carried out as frequently as practicable. However, since most of this maintenance work is very expensive [Gupta et al. (2022, p. 1)], the actual degree of fouling on a vessel often remains unknown in practice. In particular, smaller shipping companies might neither have the budget for specialized equipment such as underwater cleaning robots nor to frequently hire professional divers to perform in-water inspections.

To address the financial constraints of hull inspections, an emerging research direction focuses on the use of inexpensive underwater cameras to enable reliable fouling assessment whenever vessels are in port. For instance, Bloomfield et al. (2021) and First et al. (2021) already demonstrated how images captured by these cameras can be combined with automated image classification techniques

to quantify fouling. Although this approach appears promising, its integration into our **FOC** model was not feasible because the vessel datasets available for this study do not contain sufficient imaging data. To still be able to account for the hull fouling effect in our predictive models, we decided to approximate it by counting the **Days Since Cleaning (DSC)**, which was suggested by Bakka et al. (2022) and Laurie et al. (2021). A major advantage of this approach is the inexpensive derivation of the corresponding variable based on vessel-specific cleaning event records in the provided datasets. This **DSC** variable can also be supplemented by the hours a vessel spends at anchor, or in low-speed and high-speed transit, which can easily be computed as well. For these types of operational states, it is well established that they result in differing rates of biofouling build up [de Haas et al. (2023); Lewis and Coutts (2009, p. 350)], especially in the case of self-polishing paints [Bressy et al. (2009)]. Overall, we therefore construct a vector of biofouling measures rather than relying on a single variable, which would likely only partially capture the targeted latent effect. To the best of our knowledge, this is a novel approach. Once the **FOC** prediction model is estimated, the influence of hull fouling can then be assessed by simulating a reset of this vector to zero.

### 1.3. Hull Cleaning Events and FOC Optimization

To compute the variable representing the **DSC**, a cleaning event must be performed and recorded. Over time, various hull-cleaning methods and devices, applied either on land or in water, have been introduced, as summarized in the comprehensive review by Song and Cui (2020). In principle, all these cleaning options could be incorporated as separate explanatory variables. However, since this study is limited to the cleaning records available in the datasets, we differentiate between two types of events. The first type is dry-dock cleaning, where the vessel is removed from the water for extensive maintenance on land. During this process, the hull is typically treated with sandblasting and high-pressure water to remove fouling [Hua et al. (2018, p. 249)]. Afterwards, a fresh anti-fouling paint may be applied. Traditionally, copper-based coatings dominated the market, but environmental concerns [Townsins (2003, pp. 9–10)] have driven a shift toward silicone-based alternatives [Srinivasan and Swain (2007)] or non-anti-fouling paints, which both foul quicker but offer less initial resistance.

The second cleaning option considered in this study is in-water maintenance, where divers or specialized machines scrub and water-jet the hull as needed. While this approach provides operational flexibility, its effectiveness is comparatively lower [Tamburri et al. (2020)]. Regardless of effectiveness, both cleaning methods impose costs on the operator. Dry-dock cleaning is particularly expensive, not only due to service charges but also because the vessel loses several days of operation that could otherwise generate revenue. For example, Schultz et al. (2011) estimated the average cost of a cleaning for a US Naval vessel at approximately \$26,808. Other studies report costs ranging from \$5,000 to \$50,000, depending on vessel size [Wu et al. (2022)].

Given the available cleaning methods, the question remains when to perform them. In practice, many ship operators adopt a policy of re-coating the hull every three to five years or when a significant performance drop is observed [Song and Cui (2020, p. 416); Hua et al. (2018, p. 255)]. While this provides a reasonable heuristic, it is far from optimal. From a mathematical perspective, the procedure could be substantially improved through a simple sensitivity analysis that varies an input variable to assess how strongly predicted fuel consumption changes. If a variable is identified as critical for reducing **FOC**, a more advanced optimization framework can then be developed. For example, Huotari et al. (2021) demonstrate an approach where voyage speeds are varied to minimize fuel consumption using convex optimization combined with Dijkstra's algorithm. Similarly, Fagerholt et al. (2010) aim to reduce fuel emissions by optimizing vessel speed along shipping routes. A different optimization approach is presented by Goren Huber et al. (2017), who utilize a linear regression model to predict the log-transformed fuel consumption based on several input variables, including features related to propeller polishing, hull cleaning and dry dock events. This data-driven model then serves as the core component of a mixed-integer linear program that aims to minimize operational costs by determining optimal values for the mentioned maintenance-related variables. Inspired by these speed- and maintenance-based methodologies, we propose a novel optimization framework based on dynamic programming that determines the optimal time points for performing cleanings to mitigate the negative effects of hull fouling.

#### 1.4. Main Contributions & Paper Outline

This paper presents the following contributions, which, to the best of our knowledge, are each new to the literature:

1. We introduce a well-posed mathematical formulation for the problem of choosing an optimal hull cleaning schedule, which is presented in Section 3 as (**CLEANING-2**). Based on this formulation, small to medium ship operating companies can form data-driven plans for their hull cleaning schedules under consideration of an environmental and economic impact. This reduces the dependence on expensive underwater surveys.
2. We show that (**CLEANING-2**) satisfies the optimal substructure property. Based on this result, we design a novel dynamic programming algorithm to efficiently compute solutions of (**CLEANING-2**). In particular, the algorithm includes a data-driven function to predict fuel consumption, which can be represented by **ML** models. In our numerical analysis, we demonstrate the computational efficiency of our approach, where the cleaning schedules of vessels are determined over a period of four years (approx. 30,000 observations) with up to two minutes of computation time. This represents a significant improvement compared to a brute force method, which would take several hours.
3. We present how **SHAP** values can be utilized to analyze and verify the influence of different biofouling metrics on **FOC** predictions made by a range of **ML** models.
4. We prepare and present a large dataset of around 800,000 ship sensor observations across ten commercial tramp trading vessels measured over 1,000 voyages. This data has never been analyzed in the literature before and allows us to conduct a unique case study of the proposed methodology, which demonstrates that our optimized hull cleaning schedules can reduce **FOC** by up to 5%.

The remaining sections of this paper are organized as follows. Subsection 2.1 briefly introduces several **ML** algorithms that are later used to predict **FOC**. Afterwards, the mathematical core idea of **SHAP** values is explained in Subsection 2.2. In Section 3, the data-driven optimization algorithm is then introduced, and important theoretical properties are derived. This is followed by the description of the empirical datasets in Section 4. Subsequently, Section 5 presents the results of our numerical studies that are performed based on the previously derived theoretical setup. The impact of this work is then summarized in Section 6.

## 2. Fuel Predictions with Explainable Machine Learning

To introduce our data-driven **Cleaning Schedule Optimization (CSO)** for ships that balances environmental and economic considerations, it is necessary to construct a data-driven function  $g : \mathbb{R}^k \rightarrow \mathbb{R}$  that predicts fuel consumption for a given ship profile with  $k \in \mathbb{N}$  independent variables such as vessel speed, cargo load or trim. Accordingly, this section outlines the methodological foundation required to construct such a function and to interpret its predictions. Specifically, the first subsection briefly introduces common **ML** algorithms used in the literature to estimate  $g$ . Since most of these algorithms are black-box models, the second subsection presents an **XAI** tool that is becoming increasingly popular in the literature on **FOC** modeling. In the numerical application in Subsection 5.1, this tool is then used to verify whether the features of the ship profile exert a plausible influence on the **FOC** prediction.

### 2.1. Preliminaries on Regression Models

The estimation of any regression model requires the availability of a dataset containing  $m \in \mathbb{N}$  observations, each with  $k$  covariates and a corresponding dependent variable. In this paper, we denote this dataset by  $\mathcal{D} := \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ , where  $\mathbf{x}_t := [x_{t1}, \dots, x_{tk}] \in \mathbb{R}^{1 \times k}$  represents the ship profile at time point  $t \in \{1, \dots, m\}$ , and  $\mathbf{y} := [y_1, \dots, y_m]^\top \in \mathbb{R}_+^m$  denotes the observed fuel consumption. To refer to the collection of ship profiles for a given vessel, we use the following

notation, sometimes called a design matrix,

$$\mathbf{X} := \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1k} \\ x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mk} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_m^\top \end{bmatrix} \in \mathbb{R}^{m \times k},$$

where each row corresponds to the profile of the vessel at a given time point, and each column contains the realizations of a variable  $l$  with  $l \in \{1, \dots, k\}$ . Lastly, the collection of all predicted fuel consumptions is denoted by  $\hat{\mathbf{y}} := [\hat{y}_1, \dots, \hat{y}_m]^\top \in \mathbb{R}^m$ .

Based on the previously described model components, we can introduce a diverse set of regression estimators used in the literature to predict FOC for a given ship profile  $\mathbf{x}_t$ . In particular, we propose two simpler approaches, **Linear Regression (LinReg)** and **Least Absolute Shrinkage and Selection Operator (LASSO)**, which generate predictions by linearly combining the input variables, i.e.,  $\hat{y}_t = \beta_1 \cdot x_{t1} + \dots + \beta_m \cdot x_{tk}$ , without and with variable selection through regularization, respectively. This has the advantage that the resulting regression coefficients  $\boldsymbol{\beta} := [\beta_1, \dots, \beta_k]^\top \in \mathbb{R}^k$  can be interpreted to assess the influence of individual covariates. Besides these two linear approaches, more advanced **ML** techniques are frequently used in the literature, which apply several non-linear transformation to the input variables to obtain a prediction  $\hat{y}_t$ . Among these techniques we select three popular tree-based ensemble estimators, **RF**, **ET**, and **XGB**, whose core idea is to partition the observations into subgroups based on their covariate values. Another non-parametric **ML** approach considered in this paper is **k-Nearest Neighbors (kNN)**. It predicts  $\hat{y}_t$  for a new instance by identifying, based on a distance measure, a neighborhood of similar previously observed cases and aggregating their corresponding  $y_t$ -values. Lastly, we also evaluate two parametric non-linear **ML** models in our comparison shown in Subsection 5.1, namely the **Multilayer Perceptron (MLP)** and **Support Vector Regression (SVR)**. For the sake of brevity, we omit their detailed description here. Interested reader can find an explanation of their core ideas in Appendix A.4-A.11, which also provide additional information on the other models. Because both the analysis of biofouling-variable effects and optimization formulation are agnostic to the choice of regression technique, we refer in what follows only to the predictor  $g$ , rather than to any specific model.

## 2.2. Shapley Additive Explanations

In the previous subsection, different **ML** algorithms were briefly introduced. While the predictions of the two linear methods discussed above are easy to interpret, the remaining methods apply more complex transformations, resulting in predictions that cannot be verified by analyzing regression coefficients [Gianfagna and Di Cecco (2021, p. 33–36)]. One way of mitigating this black-box characteristics is to use modern **XAI** tools [Kamath and Liu (2021, pp. 2–4)]. These approaches aim to construct a simpler explanation model  $h : \mathbb{R}^k \rightarrow \mathbb{R}$  that can be any interpretable approximation of the original model denoted by  $g$  [Lundberg and Lee (2017, p. 2)].

Within the landscape of **XAI** tools, a distinction can be made between global and local explanation approaches [Gianfagna and Di Cecco (2021, pp. 81–82)]. Local techniques aim to explain an individual prediction  $g(\mathbf{x}_t)$ , whereas global methods seek to summarize the effects of input variables across the entire dataset [Molnar (2022, Chapter 6)]. One frequently used local model-agnostic approach are **SHapley Additive exPlanations (SHAP)**, which aim to explain the prediction for an instance  $\mathbf{x}_t$  by calculating the contribution of each variable to the model output [Molnar (2022, Chapter 9.6)]. The idea is closely related to Shapley values, which were developed in game theory to fairly distribute the total payout among players in a coalition [Rothman (2020, p. 117)]. In this game-theoretical framework, the individual contribution  $\phi_l$  of an arbitrary player  $l$  to the total payout  $f_x$  generated by a group of players  $M \in \mathbb{N}$  can be calculated as

$$\phi_l = \sum_{S \subseteq N \setminus \{l\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{l\}) - f_x(S)], \quad (1)$$

where  $N \setminus \{l\}$  are all coalitions of players that are possible without player  $l$ . Specifically, the individual contribution of player  $l$  is calculated as the difference in payouts between coalitions

with and without this player, i.e.,  $f_x(S \cup \{l\}) - f_x(S)$ . These differences are then weighted via  $\frac{|S|!(M-|S|-1)!}{M!}$ , which assigns a higher importance to cases in which player  $l$  either acts nearly alone or joins a group with many others. To connect this framework with the prediction process of a black-box model, SHAP replaces the players in a coalition with the individual variable realizations of an observation  $\mathbf{x}_t$ , and redefines the payout as the corresponding model prediction  $\hat{y}_t$  [Gianfagna and Di Cecco (2021, p. 94)]. Under additional assumptions, the resulting SHAP values are then the solution to (1) [Lundberg and Lee (2017, pp. 4–5)]. For each observation  $t$  and feature  $l$ , a corresponding  $\phi_{tl}$  value can therefore be calculated, which can subsequently be interpreted as the contribution of that feature value to the difference between the instance-specific prediction  $\hat{y}_t$  and the mean prediction [Molnar (2022, Chapter 9.6); Kamath and Liu (2021, pp. 191–192)].

### 3. Data-Driven Hull Fouling Cleaning Schedule Optimization

Section 2 presented several approaches to estimate the data-driven function  $g$ , which predicts FOC based on a given ship profile  $\mathbf{x}_t$  at time point  $t$ . Based on these insights, this section develops the Cleaning Schedule Optimization (CSO) algorithm, where  $g$  serves as a central component and whose objective is to make cleaning attractive from both an environmental and economic perspective. Towards this end, Definition 3.1 introduces a general optimization model, which is parameterized by  $n \in \mathbb{N}$  chronologically ordered voyages. The decision variables are represented by  $\mathbf{z} := [z_1, \dots, z_n]^\top \in \{0, 1\}^n$ , and defined as

$$z_j = \begin{cases} 1 & \text{we clean the ship before voyage } j \text{ for cost } c_j, \\ 0 & \text{otherwise,} \end{cases} \quad (2)$$

where  $c_j \in \mathbb{R}_+$  denotes the cost of cleaning in the port before setting out on voyage  $j \in \{1, \dots, n\}$ . The objective function of the minimization problem is the summation of two components: fuel costs and cleaning costs. The fuel costs are computed based on a function  $f : \mathbb{X} \times \mathbb{H} \rightarrow \mathbb{R}$  whose inputs are a voyage- $j$ -specific collection of ship profiles  $\mathbf{X}_j \in \mathbb{X}$  and a corresponding cleaning schedule history  $\mathbf{z}_{:j} := [z_1, \dots, z_j]^\top \in \mathbb{H}$ . The space  $\mathbb{X} := \bigsqcup_j \mathbb{R}^{m_j \times k}$  is the disjoint union over design matrices with different number of time-indexed rows  $m_j \in \mathbb{N}$  and the space  $\mathbb{H} := \bigsqcup_j \{0, 1\}^j$  is defined similarly as the union of cleaning schedule decision vectors of varying lengths  $j = 1, \dots, n$ . The function  $f$  must then map the inputs to a prediction of the total fuel costs for the voyage, which could take the following form

$$f(\mathbf{X}_j, \mathbf{z}_{:j}) := \left( \sum_{t=1}^{m_j} g(\mathbf{x}_t, \mathbf{z}_{:j}) \cdot [\text{fuel cost}] + [\text{overhead costs}] \right), \quad (3)$$

where the notation  $\mathbf{z}_{:j}$  requires that the cleaning and fuel costs of voyage  $j$  are independent of all future cleaning decisions  $z_{j+1}, \dots, z_n$ . The second component of the objective function shown in Definition 3.1 simply adds up the costs  $c_j$  of any performed cleaning  $z_j = 1$ , i.e.,  $\mathbf{c}^\top \mathbf{z} = \sum_{j=1}^n c_j \cdot z_j$ . It is noteworthy that not every port may offer a cleaning service such that in practice it might not be possible to clean the hull before certain voyages. Nevertheless, the proposed framework would still be applicable in this scenario, one just has to combine multiple voyages such that  $n$  represents the number of time points where it is possible to perform a cleaning.

#### Definition 3.1: Problem CLEANING-1

Given a dimension  $n \in \mathbb{N}$  and parameters  $(\mathbf{c}, \mathbf{X}, f) \in (\mathbb{R}^n, \mathbb{X}^n, \mathbb{X} \times \mathbb{H} \rightarrow \mathbb{R})$ , we define the following problem

$$\underset{\mathbf{z} \in \{0, 1\}^n}{\text{minimize}} \quad \underbrace{\sum_{j=1}^n f(\mathbf{X}_j, \mathbf{z}_{:j})}_{\text{Fuel Costs}} + \underbrace{\sum_{j=1}^n c_j \cdot z_j}_{\text{Cleaning Costs}}, \quad (\text{CLEANING-1})$$

where  $\mathbf{c} := [c_1, \dots, c_n]^\top \in \mathbb{R}^n$  and  $\mathbf{X} := [\mathbf{X}_1^\top, \dots, \mathbf{X}_n^\top]^\top \in \mathbb{R}^{(m_1 + \dots + m_n) \times k}$ .

A global minimum of **(CLEANING-1)** corresponds to an optimal cleaning schedule that provides the overall lowest costs and is therefore the most desirable outcome. Although the problem formulation is general and mathematically well-posed, it remains hard to solve. Theoretically, we prove in Theorem A.1 [see Appendix A] that **(CLEANING-1)** belongs to the class of NP-hard problems, which implies that no polynomial time algorithm can be expected to compute its global minimum. Furthermore, based on Theorem A.2 we show that no other algorithm can do better than the brute force approach, which exhaustively searches through all possible schedules. In particular, the combinatorial nature of  $\mathbf{z}$  means that the decision space grows exponentially with the number of voyages. Thus, finding a global minimum of **(CLEANING-1)** by a brute force approach, as is shown in Algorithm 1, requires  $n \cdot 2^n$  calls to  $f$ . To emphasize the impracticality of this approach, note that even a small instance with only 20 voyages already requires more than 1,000,000 calls to the predictive function  $f$ .

---

**Algorithm 1:** Brute Force Search

---

```

1 input  $\mathbf{X}_1, \dots, \mathbf{X}_j, \dots, \mathbf{X}_n$ ;
2  $\phi^* \leftarrow \infty$ ;
3 for  $\mathbf{z} \in \{0,1\}^n$  do
4    $\phi \leftarrow \sum_{j=1}^n f(\mathbf{X}_j, \mathbf{z}_{:j}) + \sum_{j=1}^n c_j \cdot z_j$ ;
5   if  $\phi^* > \phi$  then
6      $\phi^* \leftarrow \phi$ ;
7    $\mathbf{z}^* \leftarrow \mathbf{z}$ ;
8 return  $\mathbf{z}^*$ 

```

---

To obtain a **CSO** problem that can be solved for a realistic number of voyages  $n$ , we develop a more tractable construction from **(CLEANING-1)**. For this purpose, we introduce a new variable  $B_j \in \mathbb{B}$ , which measures the amount of hull fouling that accumulates over voyage  $j$ . This could be the count of the **DSC** but could also be generalized to a vector that contains a collection of more sophisticated measures. Moreover, we add the decision variable  $b_j \in \mathbb{R}$ , which represents the cumulative sum of hull fouling before voyage  $j$  and resets to zero if the ship is cleaned beforehand. With these two new components, we can then derive the mixed-integer problem shown in Definition 3.2. As before, the objective function sums up the fuel and cleaning costs. However, the former component is now computed based on  $\tilde{f} : \mathbb{X} \times \mathbb{B} \rightarrow \mathbb{R}$ , which no longer takes the entire cleaning history as input. Instead, it predicts the fuel costs for a given voyage  $j$  with a collection of  $m_j$  profiles  $\mathbf{X}_j$  by taking into account the accumulated hull fouling  $b_j$ . Besides this adjustment, the new problem also introduces another constraint that ensures that the cumulative hull fouling measure  $b_j$  resets to zero if the decision is made to clean, or is equivalent to the sum of  $b_{j-1}$  and  $B_{j-1}$  if the vessel continues operating without a cleaning. Under the reasonable assumption that the function  $\tilde{f}$  is monotonously increasing if the same voyage is undertaken with more hull fouling, i.e.,  $\tilde{f}(\mathbf{X}_j, 0) \leq \tilde{f}(\mathbf{X}_j, b_j) \leq \tilde{f}(\mathbf{X}_j, b'_j)$  for all  $0 \leq b_j \leq b'_j$ , it is then possible to show that a solution of **(CLEANING-2)** can be found in polynomial time with dynamic programming.

**Definition 3.2: Problem CLEANING-2**

Given a dimension  $n \in \mathbb{N}$  and parameters  $(\mathbf{c}, \mathbf{X}, \mathbf{B}, \tilde{f}) \in (\mathbb{R}^n, \mathbb{X}^n, \mathbb{B}^{n+1}, \mathbb{X} \times \mathbb{B} \rightarrow \mathbb{R})$ , we define the following problem

$$\begin{aligned}
& \underset{\mathbf{z} \in \{0,1\}^n, \mathbf{b} \in \mathbb{R}^n}{\text{minimize}} && \underbrace{\sum_{j=1}^n \tilde{f}(\mathbf{X}_j, b_j)}_{\text{Fuel Costs}} + \underbrace{\sum_{j=1}^n c_j \cdot z_j}_{\text{Cleaning Costs}} \\
& \text{subject to} && b_j = \begin{cases} b_{j-1} + B_{j-1} & \text{if } z_j = 0, \\ 0 & \text{if } z_j = 1, \end{cases}
\end{aligned} \tag{CLEANING-2}$$

where  $\mathbf{c} := [c_1, \dots, c_n]^\top$ ,  $\mathbf{B} := [B_0, B_1, \dots, B_n]^\top$ , and  $\mathbf{X} := [\mathbf{X}_1^\top, \dots, \mathbf{X}_n^\top]^\top \in \mathbb{R}^{m_1+\dots+m_n \times k}$ .

In the field of ocean engineering, dynamic programming is a powerful approach with multiple applications. For instance, the famous Dijkstra's algorithm and the A\* algorithm are both based on dynamic programming principles, and have been used to optimize weather routing and speed profiles [Fagerholt et al. (2010); Zis et al. (2020); Huotari et al. (2021)]. In general, the programming technique can be applied to all problems for which a recursive decomposition into simpler sub-problems reveals a sequence of solutions that can be re-combined to find the overall solution [Dasgupta et al. (2006, Chapter 6)]. This is known as the *optimal substructure property*; see the statement of Theorem 3.1 for a more formal construction in our context. Even though many problems, such as (**CLEANING-1**), do not satisfy this property, the situation is different for (**CLEANING-2**). Specifically, the decoupling of the dependence of  $g$  on  $\mathbf{z}$  in favor of a cumulative variable  $\mathbf{b} := [b_1, \dots, b_n]^\top$  allows dynamic programming to be applied, which is summarized by Theorem 3.1. The corresponding sub-problems can then be constructed as shown in Definition 3.3, and are denoted by  $\text{SUB-PROBLEM}[i, j]$  with  $0 \leq j < i < n$ . Intuitively, they can be interpreted as planning ahead to voyage  $i$ , and considering the optimal cleaning schedule for the future voyages  $i, \dots, n$  supposing that the last cleaning was performed before voyage  $j$ .

### Definition 3.3: SUB-PROBLEM[i, j]

Given an instance of (**CLEANING-2**) of size  $n$  and parameterized by  $(\mathbf{c}, \mathbf{X}, \mathbf{B}, \tilde{f})$ . The subproblem  $\text{SUB-PROBLEM}[i, j]$  is defined as a new instance of (**CLEANING-2**) of size  $n - i + 1$  and parameterized by  $(\mathbf{c}', \mathbf{X}', \mathbf{B}', \tilde{f})$ , where  $\mathbf{c}' = [c_i, \dots, c_n]^\top$  and  $\mathbf{X}' = [\mathbf{X}_i^\top, \dots, \mathbf{X}_n^\top]^\top$  and  $\mathbf{B}' = \left[ \sum_{k=j}^{i-1} B_k, B_i, \dots, B_n \right]^\top$ .

- Let  $\Phi[i, j] \in \mathbb{R}$  denote the optimal objective value for the  $\text{SUB-PROBLEM}[i, j]$ .
- Let  $\Psi[i, j] := [z_i^*, \dots, z_n^*]^\top \in \{0, 1\}^{(n-i+1)}$  be an optimal cleaning schedule for  $\text{SUB-PROBLEM}[i, j]$ .

From a technical perspective, each  $\text{SUB-PROBLEM}[i, j]$  aims to find the optimal subset of decision variables  $\Psi[i, j] = [z_i^*, \dots, z_n^*]^\top \in \{0, 1\}^{n-i+1}$ . Specifically, these problems assume that the vessel is cleaned before voyage  $j$  [i.e.,  $z_j = 1$  and  $b_j = 0$ ], and that no additional cleanings are performed prior to voyages  $j + 1$  through  $i - 1$  [i.e.,  $z_{j+1} = \dots = z_{i-1} = 0$ ]. Based on these assumptions, Theorem 3.1 shows that the subproblems satisfy the optimal substructure property, where the proof has been deferred to Appendix A.3. This property then enables the application of Algorithm 2 to compute the optimal overall costs  $\Phi[i, j]$  for each  $\text{SUB-PROBLEM}[i, j]$ . For more details on this technique, see Cormen et al. (2022, Section 15.3).

### Theorem 3.1

The (**CLEANING-2**) problem satisfies the optimal substructure property; that is, the objective value of  $\text{SUB-PROBLEM}[i, j]$  can be written in terms of the objective value of  $\text{SUB-PROBLEM}[i+1, i]$  and  $\text{SUB-PROBLEM}[i+1, j]$  for each pair of voyage indices  $0 \leq j < i < n$ :

$$\Phi[i, j] = \min \left\{ \tilde{f}(\mathbf{X}_i, 0) + c_i + \Phi[i+1, i], \quad \tilde{f}\left(\mathbf{X}_i, \sum_{k=j}^{i-1} B_k\right) + \Phi[i+1, j] \right\}. \quad (4)$$

Conceptually, Algorithm 2 has an outer for-loop that starts at Line 4 and an inner for-loop that begins at Line 7. The outer loop has exactly  $n$  iterations, while the inner has at most  $n$ . Since the operations inside these two loops consist only of constant-time statements and function calls to  $\tilde{f}$ , the computational complexity is characterized by  $\mathcal{O}(n^2)$  calls to  $\tilde{f}$ . In the first iteration of the outer loop, the algorithm solves the most simple sub-problems with  $i = n$ , and  $j = 0, \dots, n$ . As these are single-voyage problems, the solution is just a choice between cleaning or not cleaning. After this initial round, the algorithm then works backwards through time, solving  $\text{SUB-PROBLEM}[i, j]$  according to (5) and (6). This procedure enables the reuse of the previously computed solutions,

which reduces computation time by avoiding a full re-evaluation of the objective function. In particular, the optimal objective value for the voyage subset  $i, \dots, n$  and a cleaning before voyage  $i$  is computed in Line 5 and denoted by  $\phi^{(clean)}$ . It consists of the sum of three components: the cost of cleaning before voyage  $i$  [i.e.,  $c_i$ ], the cost of voyage  $i$  with zero hull fouling  $f(\mathbf{X}_i, 0)$ , and the optimal overall costs  $\Phi[i+1, i]$  for the voyage subset  $i+1, \dots, n$  assuming we cleaned before voyage  $i$ . Afterwards, the costs of the scenario where no cleaning is performed before voyage  $i$  is similarly computed in Line 9 and denoted by  $\phi^{(not)}$ . In Line 11 to 16, the two different outcomes are then compared and the solution for SUB-PROBLEM $[i, j]$  with lower costs is selected. Overall, this procedure provides a practical way to solve (CLEANING-2), as demonstrated in Section 5, where we apply and solve the optimization framework using real-world datasets.

---

**Algorithm 2:** Dynamic Cleaning Schedule Optimization (CSO)

---

```

1 input  $\mathbf{X}_1, \dots, \mathbf{X}_j, \dots, \mathbf{X}_n$ ;
2  $\Psi \in \{0, 1\}^{n \times n \times n}$  ;
3  $\Phi \in \mathbb{R}^{n \times n}$  ;
4 for  $i = n, \dots, 1$  do
5    $\phi^{(clean)} \leftarrow c_i + \tilde{f}(\mathbf{X}_i, 0) + \Phi[i+1, i]$  ;
6    $\mathbf{z}^{(clean)} \leftarrow (1, \Psi[i+1, i])$  ;
7   for  $j = 0, 1, \dots, i$  do
8      $b_i \leftarrow \sum_{k=j}^i B_k$ ;
9      $\phi^{(not)} \leftarrow f(\mathbf{X}_i, b_i) + \Phi[i+1, j]$  ;
10     $\mathbf{z}^{(not)} \leftarrow (0, \Psi[i+1, j])$  ;
11    if  $\phi^{(clean)} \leq \phi^{(not)}$  then
12       $\Phi[i, j] \leftarrow \phi^{(clean)}$  ;
13       $\Psi[i, j] \leftarrow \mathbf{z}^{(clean)}$  ;
14    else
15       $\Phi[i, j] \leftarrow \phi^{(not)}$  ;
16       $\Psi[i, j] \leftarrow \mathbf{z}^{(not)}$  ;
17 return  $\Psi[1, 0], \Phi[1, 0]$ 

```

---

## 4. Empirical Data Sources

After proposing a mathematical framework for optimizing the cleaning schedule of ships, this section introduces the empirical data used to evaluate its performance. Specifically, the empirical analysis relies on real-world vessel data obtained through a cooperation agreement with [Carisbrooke Shipping Ltd.](#) ↗, which is a UK-based shipping company that operates a worldwide-trading fleet of over 27 modern dry cargo and multi-purpose vessels. As part of this agreement, the industry partner granted us access to hourly voyage data for ten vessels from 2016 to 2025, which we refer to as  $V_1, \dots, V_{10}$ . While vessel  $V_8$  was excluded from the empirical evaluation due to data irregularities, each of the remaining nine ships provides approximately 80,000 observations. As these vessels do not operate based on a repetitive schedule and can be chartered by different customers, important ship characteristics like the routes, the loads and the time spent in harbor can vary greatly. This also affects the hull fouling, which develops differently over time for each vessel. Thus, the results of our models are likely less comparable to studies based on ship data with fixed schedules, e.g., ferries. In the upcoming subsections, the company data is separated into two sources and described in more detail. Afterwards, we introduce relevant weather and marine variables, which were obtained from [Open-Meteo](#) ↗. This subsection also includes non-linear variable transformations that integrate company and weather data.

### 4.1. Cleaning Reports

The first data source consists of the cleaning reports, which are generated within the company to document the cleaning procedures carried out on each vessel. Based on these reports, two types of manual cleaning events can be distinguished: [In Water Survey \(IWS\)](#) and [Dry Dock Maintenance \(DDM\)](#). As explained in Subsection 1.3, divers are deployed during an [IWS](#) to photograph and inspect the vessel, remove major biofouling, and perform necessary repairs. In contrast, [DDM](#) is

a far more involved process during which the vessel is docked for several days. In this period, the ship's entire hull is grit-blasted to abrasively remove biofouling as well as any remaining paint [Hua et al. (2018, p. 249)]. Once the hull has been cleaned, any necessary repairs are carried out and the surface is subsequently recoated [Adland et al. (2018, p. 3)]. Figure 1 shows one of the ten vessels after having been cleaned using this procedure. Compared to an IWS, a DDM is more costly and time-consuming, but it is typically also more effective in removing biofouling [Valchev et al. (2022, p. 5)]. For our empirical data, we use the cleaning records to construct two variables for each vessel: Days Since In Water Survey (DSIWS) and Days Since Dry Dock Maintenance (DSDDM). Additionally, we create the variable DSC, which combines DSIWS and DSDDM by counting the days since the most recent cleaning event.



**Figure 1.** One of the ten sister ships included in the study, photographed immediately after DDM.

#### 4.2. Ship Sensor Data

Nowadays, all ships of 5,000 gross tonnage and above must record fuel consumption data, broken down by fuel type, along with other specified information due to Regulation 22A of IMO (2016, pp. 4–5). To comply with this requirement, our industry partner has installed a Marine Fuel Efficiency Monitoring System on each vessel. This system is our second data source and enables the tracking of the FOC and several related variables, such as the Speed Through Water (STW), Speed Over the Ground (SOG), Water Under Keel (WUK) and Course Over Ground (COG), on an hourly basis using multiple onboard sensors. Table 1 provides a brief description of all relevant variables derived from this data source and specifies their respective data types. To link the ship sensor data to the variables from Subsection 4.1, we match the *Date & Time* entries with the time stamps of the cleaning records.

**Table 1.** Description of Ship Sensor Data Variables.

Variable	Type (Unit)	Description
Voyage ID	Integer	Unique identifier of the voyage per vessel.
IMO Number	Integer	A seven-digit integer unique to each ship.
Date & Time	Time Stamp	The time stamp when the entry was created.
Position	String	Geographical coordinates of the ship when the entry was created.
FOC	Float (kg/h)	Total FOC measured by flow meters on the fuel pipes.
SOG	Float (kn)	The average speed relative to the surface of the earth in knots.
STW	Float (kn)	The average speed with respect to the water in knots.
Cargo Loaded	Float (tons)	Weight of cargo loaded onto the vessel.
Draught	Float (m)	Average depth of the vessel below the waterline in meters.
Trim	Float (m)	Average difference between the aft and the forward draft in meters.
WUK	Float (m)	Average space between the ship's bottom and the seabed in meters.
COG	Float (°)	The actual direction of progress w.r.t. the earth surface in degrees.
Wind Speed	Float (m/s)	The average speed of the wind in meters per second.
Wind Direction	Float (°)	The direction of the wind in degrees.
Fuel Oil Type	String	The used fuel oil: HFO, LFO, LSHFO or MGO.
Propeller Shaft	Float (rpm)	Propeller shaft speed in rotations per minute.
Pitch Propeller	Float (%)	Propeller angle as percentage of maximum angle.

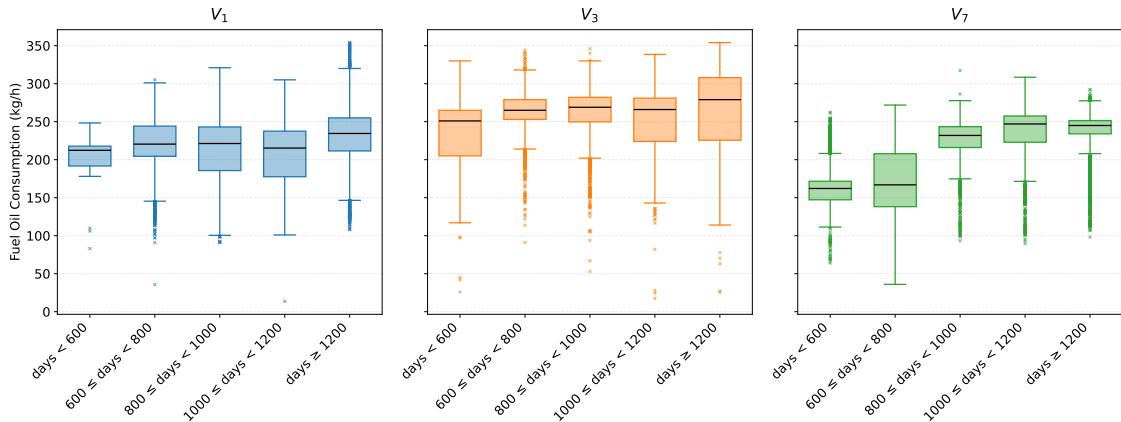
A subset of the features listed in Table 1 are categorical or represent a time stamp. Since most ML algorithms require numerical input, we transform these features using common preprocessing techniques. In particular, the *Fuel Oil Type* is encoded based on four indicator variables, Light Fuel

**Oil (LFO)**, **Low Sulfur Heavy Fuel Oil (LSHFO)** and **Marine Gasoil (MGO)**, with **Heavy Fuel Oil (HFO)** serving as the reference category. Similarly, season-specific indicator variables are generated from the *Date & Time* entries, where the winter months serve as a reference. An overview of the transformations is provided by Table 2. This summary also includes several additional variables based on **STW** that are designed to partially capture the effect of hull fouling. Specifically, these variables measure the hours spent in certain speed ranges since the most recent cleaning in order to account for self-polishing effects. The feature *HaS0* is inspired by the findings of de Haas et al. (2023), who showed that the number of anchorage days since the last vessel cleaning provides an alternative way to capture the hull fouling effect. Moreover, since we observe that the ship sensor data occasionally contains longer gaps between hourly observations due to maintenance work or other factors, we also create a feature named *HU*, which counts these unaccounted hours. Ultimately, we also add lagged versions of the **SOG** and **STW** variables as additional features to the dataset, which simply take the corresponding value from the previous time index. This allows the models to better capture an acceleration/deceleration process and may also make the dataset more robust against one-off faulty readings.

**Table 2.** Transformations of Non-Numeric Variables.

Variable	Type (Unit)	Description
<b>LFO</b>	Integer	Indicator that is “1” if <b>LFO</b> is used, otherwise “0”.
<b>LSHFO</b>	Integer	Indicator that is “1” if <b>LSHFO</b> is used, otherwise “0”.
<b>MGO</b>	Integer	Indicator that is “1” if <b>MGO</b> is used, otherwise “0”.
Spring	Integer	Indicator that is “1” if it is spring, otherwise “0”.
Summer	Integer	Indicator that is “1” if it is summer, otherwise “0”.
Autumn	Integer	Indicator that is “1” if it is autumn, otherwise “0”.
<b>HU</b>	Float (h)	The unaccounted hours since cleaning.
<b>HaS0</b>	Float (h)	The hours at speed $[0, 1]$ knots since the last cleaning.
<b>HaS6</b>	Float (h)	The hours at speed $(1, 6]$ knots since the last cleaning.
<b>HaS9</b>	Float (h)	The hours at speed $(6, 9]$ knots since the last cleaning.
<b>HaS12</b>	Float (h)	The hours at speed $(9, \infty)$ knots since the last cleaning.

Beyond the transformation of variables, we also perform common data quality and plausibility checks for the ship sensor data. This includes checking for missing or implausible values and, if necessary, performing an imputation. To limit the impact of extreme observations, we additionally apply winsorizing to selected variables. A first impression of the preprocessed data is provided by Figure 2, which illustrates the effect of the discretized **DSDDM** on the **FOC** through boxplots based on the data of vessels  $V_1$ ,  $V_3$  and  $V_7$ . Overall, the cleaned data for the selected vessels already indicates that the median **FOC** increases as more time has passed since the last cleaning. However, it is worth mentioning that such consistent plots cannot be produced for all vessels and we shall use more advanced techniques to properly model this relationship.



**Figure 2.** Three examples (left to right vessels  $V_1$ ,  $V_3$  and  $V_7$ ) where the influence of the independent variable **Days Since Dry Dock Maintenance (DSDDM)** can be observed against the **Fuel Oil Consumption (FOC)** through box and whisker plots.

### 4.3. Weather and Marine Data

After presenting the ship sensor data, this subsection introduces the final data source: weather and marine variables obtained from [Open-Meteo](#). We chose this freely available data source for our empirical analysis, as the data provider used by the industry partner did not grant permission to use their system for research purposes. Table 3 provides an overview of the obtained variables, which are linked to other data sources based on the *Date & Time* and *Location* entries. It is noteworthy that the open source data does not entirely match the time period of the ship sensor data, which is available from 2016. Specifically, the weather variables can first be extracted for the second half of 2021, while the marine features become available starting in October of the same year. More information about the data can be found in the documentation of the Historical Weather and Marine Forecast API on the [Open-Meteo](#) website.

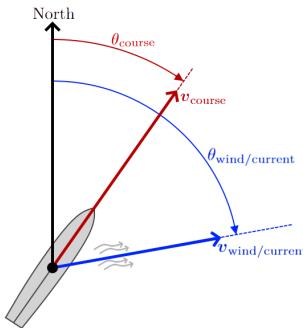
**Table 3.** Description of Weather and Marine Variables.

Variable	Type (Unit)	Description
Date & Time	Time Stamp	Time point of the data measurement.
Location	String	Geographical coordinates of measured data.
Weather code	Float	Weather condition based on the WMO code.
Temperature 2m	Float (°C)	Air temperature at 2 meters above ground.
Wind speed 10m	Float (m/s)	Wind speed at 10 meters above ground.
Wind direction 10m	Float (°)	Wind direction at 10 meters above ground.
Wave height	Float (m)	Wave height of significant mean waves.
Wave direction	Float (°)	Mean direction of mean waves.
Wave period	Float (s)	Period between mean waves.
Ocean current velocity	Float (m/s)	Velocity of ocean current.
Ocean current direction	Float (°)	Direction following the flow of the current.

Similar to Subsection 4.2, we derive additional variables based on the weather and marine features. For example, we observe that the FOC can be sensitive to high wind speeds and ocean currents, particularly in the English Channel. Depending on the situation, these forces may act in the direction of travel or against it. To include this effect, we introduce two variable transformations to our dataset, which are calculated in combination with the ship sensor data. Specifically, we compute the new features based on

$$r_{\text{wind/current}} = v_{\text{wind/current}} \cdot \cos(|\theta_{\text{course}} - \theta_{\text{wind/current}}|)$$

where  $\theta_{\text{course}}$  corresponds to the COG variable from Table 1, and  $v_{\text{wind/current}}$  and  $\theta_{\text{wind/current}}$  are the wind speed and direction (respectively ocean current velocity and direction) variables as described in Table 3.



**Figure 3.** Visualization of the relative angle between COG and wind/current direction.

## 5. Practical Application & Numerical Studies

After explaining the mathematical framework for optimizing the cleaning schedule of ships and introducing the data sources, this section presents the findings of the empirical study. In particular, the first subsection focuses on comparing the performance of the ML algorithms using the novel datasets and selects the best model to represent the data-driven function  $g$ . Based on the

chosen **ML** algorithm, the **SHAP** values of several variables are then illustrated and interpreted. Subsequently, the second subsection evaluates the results of the **CSO** to empirically verify whether the application of the proposed algorithm leads to additional cleaning events in the considered time period that are economically justified and reduce fuel consumption.

### 5.1. Selection of Fuel Prediction Function

To compare the performance of the **ML** algorithms introduced in Subsection 2.1, hyperparameter tuning is required beforehand. In contrast to model parameters, hyperparameters are not learned during model estimation but need to be specified manually prior to training [Géron (2022, pp. 35–36)]. In our study, these parameters are chosen via a random search, which samples 100 different configurations from specified distributions [see Appendix A.4–A.11] and re-estimates the model for each one. The impact of different configurations can then be compared based on the model-specific validation scores. An alternative to this procedure would be a grid search, which exhaustively explores the parameter space but becomes impractical for models with many hyperparameters, as in our case [Bergstra and Bengio (2012, p. 283)].

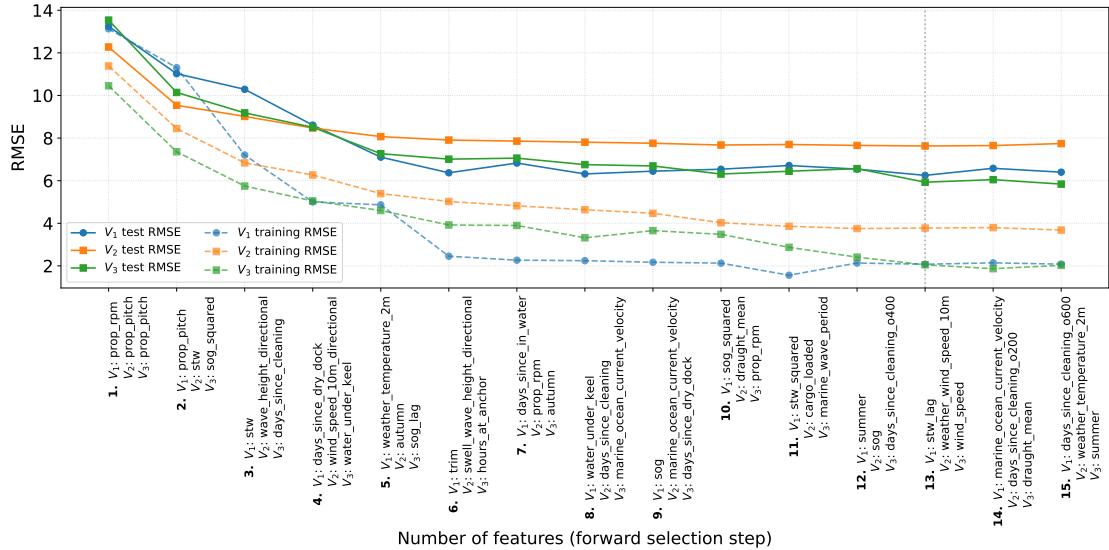
Having addressed the hyperparameter tuning, we now turn to the variable selection process in our study. Including all transformations and indicator variables presented in Section 4, the datasets contain a total of 54 variables. This high-dimensional setup can be problematic for three reasons: (i) the models may over-fit the data, (ii) training times may become long, and (iii) there may be high co-variance among variables. A common approach to mitigate such issues is **Principal Component Analysis (PCA)**. However, this method is incompatible with the type of analysis we wish to perform, as it removes the ability to study the effects of individual variables. To avoid this limitation, we apply forward stepwise feature selection [Hastie et al. (2009, Section 3.3.2)]. At each step, we trial adding one unselected variable at a time, run a new hyperparameter search, and then select the variable that results in the lowest validation score.

To perform the hyperparameter tuning and the variable selection, a validation set must be constructed. For this purpose, we first split the instances for each vessel into a training and a testing set using an approximate 85-15 ratio to retain unseen data for the final model evaluation. Specifically, we construct the subsets such that all the observations from a particular voyage [e.g., all 33 hourly instances from Portsmouth to Gijón] are assigned to either the training or the testing set, but never both, in order to avoid introducing data leakage into our comparison [Rumala (2023)]. Based on a similar procedure, we then further split the training data of each vessel into five folds to perform the feature selection and hyperparameter tuning as discussed previously. This allows us to fit the model for each configuration five times by cycling through the folds one to five as the validation set while using the remaining four folds for training [see Figure A.1]. The average validation score can then be used to assess the quality of a feature or hyperparameter choice [Bishop (2006, pp. 32–33); Hastie et al. (2009, pp. 241–242)].

Consistent with the methodology described above, Figure 4 illustrates the results of the variable selection for vessels  $V_1$ ,  $V_2$  and  $V_3$  based on the **XGB** model. In this comparison, the **Root Mean Squared Error (RMSE)** is used as performance metric and is shown on the y-axis [Chai and Draxler (2014)]. The number of selected variables is shown on the x-axis, including the name of the variable selected at each step per vessel. Overall, we find that selecting 13–15 features results in the best validation performance and that important variables like the **STW** or **Propeller Shaft** are selected first. Moreover, we observe that several variables constructed to approximate the effect of hull fouling on the **FOC** are selected across the different vessels. For instance, the selection paths of  $V_1$  and  $V_3$  show that the **DSDDM** and **DSC** are chosen as fourth and third variable in the regression model, respectively. For the hyperparameter selection, we find that the best configuration for each model varies across the nine vessels, but the optimized selection consistently offers a significant improvement over the default values.

To compare the performance of the **ML** algorithms, we select multiple performance metrics, which are commonly used in the literature [Gupta et al. (2022, p. 10); Botchkarev (2019, p. 50)]. Table 4 summarizes the model results for each vessel using the  $R^2$  score computed based on the unseen test data. In this overview, the best result for each vessel is highlighted in bold, which in-

dicates that **XGB** delivers the best performance regardless of the dataset. Similar, though slightly worse, results are provided by the **RF**, **ET**, and **SVR**. This ranking suggests the presence of non-linear relationships between some input variables and the **FOC**, as well as potential interaction effects, since the tree-based models in particular are very effective in identifying such patterns. A first noticeable performance drop is then observed for the **MLP**. In principle, this model can also be very powerful in detecting non-linear effects and interactions between variables. However, the predefined structure of the hidden layers might limit this capability for the data in our study, preventing the model from fully playing to its strengths. Another performance drop is then observed for the two linear models, **LASSO** and **LinReg**, which further promotes the previously made suppositions about the non-linear data structure and interaction effects. The overall worst  $R^2$  scores for the test data are achieved by **kNN**. For this model, we observe that fewer than five variables are selected and that the number of neighbors is very close to the upper limit, which may lead to overly generalized **FOC** predictions.



**Figure 4.** The forward feature selection process visualized through the training and test error of the **XGB** model for vessels  $V_1$ ,  $V_2$  and  $V_3$ .

Besides using the  $R^2$  score as performance metric, we also compare the model predictions based on the **RMSE** and the **Mean Absolute Error (MAE)**. The corresponding results are shown in Table A.1 and Table A.2, respectively. While the **RMSE** scores suggests an identical model ranking, the best **MAE** scores for each vessel are distributed across the tree-based models. This indicates that **XGB** generates fewer very large prediction errors, since the  $R^2$  and **RMSE** penalize such mispredictions more strongly than the **MAE**.

**Table 4.**  $R^2$  scores of the ML models on the test datasets.

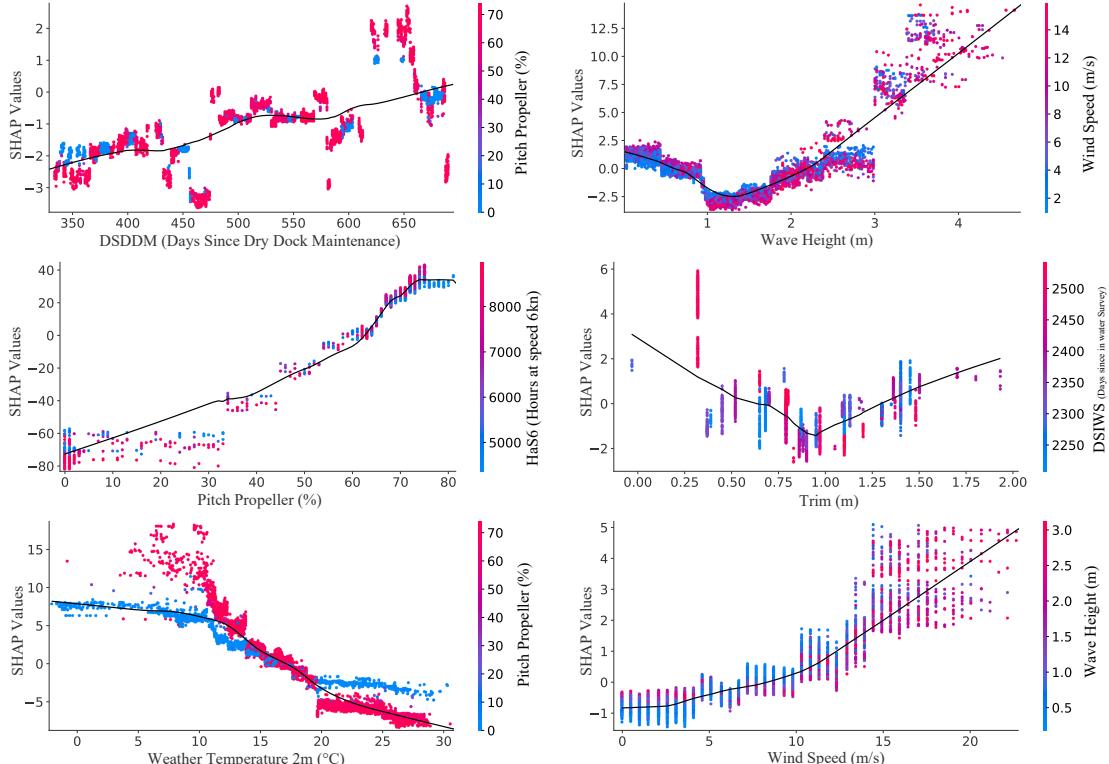
	LinReg	LASSO	kNN	MLP	SVR	ET	RF	XGB
$V_1$	0.5110	0.5148	0.3900	0.6929	0.9552	0.9671	0.9672	<b>0.9686</b>
$V_2$	0.7107	0.7085	0.3669	0.6670	0.8817	0.9049	0.9038	<b>0.9054</b>
$V_3$	0.7134	0.7139	0.3345	0.7732	0.9785	0.9854	0.9860	<b>0.9866</b>
$V_4$	0.8980	0.8960	0.6568	0.9034	0.8438	0.8961	0.9101	<b>0.9174</b>
$V_5$	0.7852	0.7842	0.3820	0.9260	0.9260	0.9774	0.9785	<b>0.9799</b>
$V_6$	0.3608	0.3638	0.3123	0.6758	0.8907	0.9104	0.9210	<b>0.9229</b>
$V_7$	0.7348	0.7330	0.5565	0.7875	0.9260	0.9303	0.9293	<b>0.9394</b>
$V_9$	0.5436	0.5447	0.3296	0.8511	0.9881	0.9915	0.9916	<b>0.9926</b>
$V_{10}$	0.7601	0.7602	0.4100	0.6744	0.9673	0.9793	0.9796	<b>0.9861</b>

Overall, the best model performance is provided by **XGB**. Thus, we select this model to represent the data-driven function  $g$  for the remainder of this paper. To obtain a complete picture of the model performance, Table 5 reports the performance scores on both the training and testing datasets for each vessel. The corresponding **Confidence Intervals (CIs)** are shown in Appendix A.14.

**Table 5.** Detailed report of the XGB performance in predicting FOC.

	Training			Test		
	RMSE (kg/h)	MAE (kg/h)	R <sup>2</sup> score	RMSE (kg/h)	MAE (kg/h)	R <sup>2</sup> score
V <sub>1</sub>	2.0070	1.2537	0.9973	6.2752	6.1702	0.9686
V <sub>2</sub>	3.8249	4.8752	0.9806	7.7433	7.5517	0.9054
V <sub>3</sub>	0.6237	0.6190	0.9996	4.0179	3.9315	0.9866
V <sub>4</sub>	2.7452	1.7175	0.9918	6.8672	5.5446	0.9174
V <sub>5</sub>	1.1335	1.1385	0.9986	6.6745	6.8213	0.9799
V <sub>6</sub>	1.7083	1.8577	0.9905	8.5962	7.8703	0.9229
V <sub>7</sub>	3.7351	2.0250	0.9836	8.2419	5.8392	0.9394
V <sub>9</sub>	2.5520	1.4508	0.9959	4.1277	2.9728	0.9926
V <sub>10</sub>	1.8322	1.5519	0.9973	5.4880	5.0052	0.9861

After evaluating the ML algorithms using performance metrics, we now examine SHAP values to further verify the effects of the input variables on the FOC. For this purpose, we apply the theoretical framework introduced in Subsection 2.2 to compute SHAP values based on the training datasets, where it becomes advantageous that XGB was selected as the underlying model. The tree-specific Python implementation enables an efficient computation of SHAP values, even for several thousand instances. These values can then be visualized through SHAP dependence plots, where the variable realizations and the SHAP values are on the horizontal and vertical axis, respectively. Figure 5 illustrates six examples of such plots using the data of vessel V<sub>7</sub>. Each plot is automatically colored based on the strongest interaction with another variable, and a local smoothing function highlights the main trends of the point cloud. Overall, the dependence plots indicate plausible relationships between the input variables and FOC. For instance, higher values of *Pitch Propeller* or *Wind Speed* are associated with higher SHAP values. This indicates that the difference between the instance-specific FOC prediction and the mean prediction increases for higher pitches and wind speeds, which is in line with the expectation. An opposite relationship is observed for *Weather Temperature*, i.e., the warmer it is, the lower the SHAP values become. This effect is further amplified when combined with higher *Pitch Propeller* values.



**Figure 5.** Six example SHAP dependence plots using the XGB model trained on the vessel V<sub>7</sub> data. On the horizontal axes are six different input variables and on the vertical axes are corresponding SHAP values, which can be interpreted as the contribution of an individual feature value to the difference between the instance-specific FOC prediction and the mean prediction. The coloring allows to check for interactions to a second input variable.

Ultimately, we also examine the dependence plot of the **DSDDM**, which is one of the selected biofouling measures. Even though its **SHAP** values are more scattered, the local smooth function still indicates the anticipated effect. We predominately observe that a higher count of the **DSDDM** coincides with a greater difference between the instance-specific **FOC** prediction and the mean prediction. At this point, it is worth noting that not all generated **SHAP** plots exhibit patterns as consistent as those shown in Figure 5. To explain this behavior, a detailed inspection of the individual split points in the regression trees would be required. However, this becomes increasingly challenging for deeper tree structures and is beyond the scope of this study.

## 5.2. Empirical Results of Cleaning Schedule Optimization

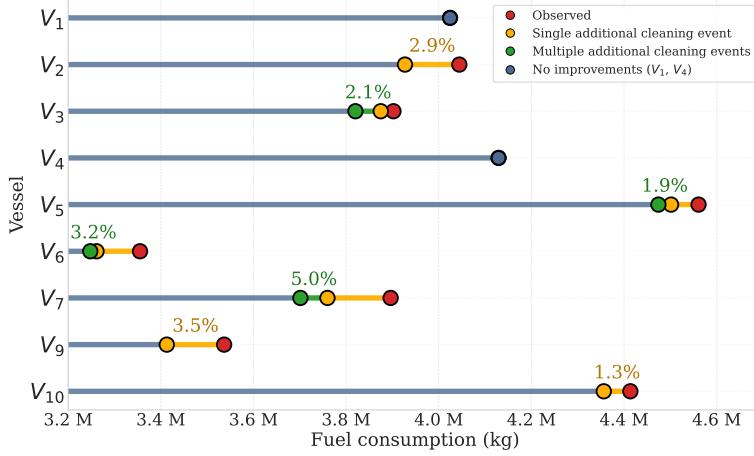
After developing a prediction model for the **FOC**, this subsection showcases how the optimization framework introduced in Section 3 can be applied to determine optimal cleaning schedules for the vessels of our industrial partner. In particular, we address the question of when a ship cleaning should be performed in order to minimize the overall operational expenditures, which consist of the cleaning and fuel costs. For this purpose, we subset the data of each ship to an approximately four-year period that begins with a dry-dock cleaning event. This has the advantage that the biofouling measures are all initiated at zero and that the [Open-Meteo](#) data is predominantly available. We then further cut the data for each vessel into time ordered distinct voyages, which are enumerated  $j = 1, \dots, n$ . The selected features relating to those voyages (e.g., *Trim*, *Cargo Load*, etc.) are stored in matrices  $\mathbf{X}_j$ , where each row represents an hourly observation within voyage  $j$ . To predict the **FOC** for a given instance, we select the **XGB** model as data-driven function  $g$ , which was discussed in more detail in Subsection 5.1.

Based on the previously described setup, it remains to determine the input parameters of the optimization problem shown in (CLEANING-2). The biofouling metric itself is denoted by  $\mathbf{b}_j$  and represents a multidimensional vector, i.e.,  $\mathbf{b}_j := [b_{j1}, b_{j2}, b_{j3}, \dots]^\top$ , in this study. In particular, the vector has seven dimensions, defined as  $\mathbf{b}_j = [\text{DSDDM}, \text{DSIWS}, \text{HU}, \text{HaS0}, \text{HaS6}, \text{HaS9}, \text{HaS12}]^\top$ , and keeps track of the ship's time spent in different operational states since the last cleaning event. These values allow the regression models to learn how fouling is built up over time, as shown in Figure 5 for the **DSDDM** variable. The increase in biofouling metric  $\mathbf{B}_j$  is then calculated according to the record of the voyage. For example, if voyage  $j$  spent ten days at anchor and five days sailing at 10 kn, the vector could contain the following values:  $\mathbf{B}_j = [15, 15, 0, 240, 0, 0, 120]^\top$ . To obtain the overall fuel costs for a voyage, the function  $f$  multiplies the **FOC** predicted by  $g$  with the fuel price as recorded on the website <https://shipandbunker.com/prices>. If a cleaning is performed before voyage  $j$ , then the cost  $c_j$  is added to the overall costs and  $\mathbf{b}_j$  is adjusted accordingly. Note that in our analysis,  $c_j$  is set to \$10,000 at all ports, which is an estimate that was derived based on values from the literature in Subsection 1.3 and can be fine-tuned in future applications. In this manner, we have defined all components for the problem of choosing the optimal cleaning schedule based on an instance  $(\mathbf{c}, \mathbf{X}, \mathbf{B}, f)$  of (CLEANING-2).

To implement the brute-force search and dynamic programming algorithm, introduced in Section 3, we utilized Python 3.12. The most up-to-date code can be accessed through the project's GitHub repository [Cleaning-Schedule-Optimization](#), but a legacy code is also provided in Appendix A.15. Note that since the vessel data contains confidential information, it is not included in this repository. In our local version, the data of our industry partner is efficiently loaded and manipulated using the **Pandas** library, and all linear algebra is computed through **NumPy**. Based on this setup, we can then run both algorithms on the data for each ship. The brute-force search, i.e., Algorithm 1, is given an hour to run but always failed to terminate in our case study. In contrast, the dynamic programming approach consistently terminated in under two minutes. Therefore, the remainder of this subsection will only focus on the results of Algorithm 2.

Figure 6 summarizes the optimization results in terms of fuel savings across the vessels of our industry partner. Overall, in seven out of nine cases, the algorithm produces a cleaning schedule that includes at least one additional cleaning event over the four-year period. In four out of nine cases, the algorithm even outputs a second cleaning event within this time period. For the vessels  $V_2, V_3, V_5, V_6, V_7, V_9, V_{10}$ , the optimized cleaning schedules then translate into a reduction of fuel consumption by 2.9%, 2.1%, 1.9%, 3.2%, 5.0%, 3.5%, 1.3%, respectively. Only for the two vessels,

$V_1$  and  $V_4$ , the algorithm could not recommend an additional cleaning as the corresponding fuel savings would have been too small to offset the cleaning costs.



**Figure 6.** The fuel savings relative to the overall fuel consumption of the nine vessels in our study.

As next step, the net savings of the company are examined based on the seven vessels, where the reduction in fuel costs outweighs the costs of cleaning. Specifically, our numerical results show that if the company had performed one extra cleaning on each of the aforementioned vessels during the four-year study period, this would have resulted in savings of approximately \$315,000. To visualize this financial impact, Table 6 provides a detailed breakdown of the algorithmic output for the vessels  $V_2$ ,  $V_3$ ,  $V_5$  and  $V_9$ . It is interesting to note that a company could reasonably brute force a local solution to the problem for exactly one cleaning event without the innovation of Algorithm 2, which is shown for  $V_3$  and  $V_5$ . However, this naive approach would not unfold the full potential to save fuel and corresponding CO<sub>2</sub> emissions. For instance, the results of  $V_3$  demonstrate that fuel consumption would then only be reduced by 0.7% instead of 2.1%.

**Table 6.** The output of our cleaning schedule optimization for four of the nine vessels. The percentages show the predicted savings in kilograms of fuel consumption (resp. costs in USD) as a percentage of the observed fuel consumption (resp. costs in USD).

	$V_2$	$V_3$	$V_5$	$V_9$
<b>Parameterisation</b>				
Number of voyages	125	103	124	71
Hours of sailing	30,458	28,062	35,301	28,019
Start date	17 Jun 2021	14 Sep 2020	24 Apr 2020	07 Sep 2021
End date	28 Dec 2024	14 Sep 2024	24 Apr 2024	28 Dec 2024
<b>No additional cleaning</b>				
Compute time (s)	1	1	1	1
Fuel (kg)	4,044,410.50	3,902,133.58	4,560,798.18	3,536,718.12
Cost (\$)	\$3,148,573.58	\$3,037,810.99	\$3,550,581.39	\$2,753,335.06
<b>One additional cleaning</b>				
Compute time (s)	70	54	65	25
Cleaning date	19 Jul 2022	05 Dec 2021	13 Jun 2022	12 Mar 2023
Fuel (kg)	3,927,073.92	3,874,779.62	4,501,393.87	3,413,095.09
Cost (\$)	\$3,067,227.05	\$3,026,515.94	\$3,514,335.12	\$2,667,094.52
$\Delta$ Fuel (kg)	(2.9%) 117,336.58	(0.7%) 27,353.96	(1.3%) 59,404.32	(3.5%) 123,623.04
$\Delta$ Cost (\$)	(2.6%) \$81,346.53	(0.4%) \$11,295.06	(1.0%) \$36,246.26	(3.1%) \$86,240.54
<b>Multiple additional cleaning</b>				
Compute time (s)	70	54	66	25
Cleaning dates (year)	-	2021, 2023	2021, 2022, 2023	-
Fuel (kg)	-	3,820,188.77	4,471,948.72	-
Cost (\$)	-	\$2,984,016.96	\$3,511,412.08	-
$\Delta$ Fuel (kg)	- (2.1%) 81,944.81	- (1.9%) 88,849.46	-	-
$\Delta$ Cost (\$)	- (1.8%) \$53,794.03	- (1.1%) 39,169.31	-	-

Altogether, the results of our study demonstrate that cleaning vessels more regularly does have a financial benefit for our industrial partner. Furthermore, performing those cleanings according to an optimal solution of (CLEANING-2) provides significant financial gains over a more naive

schedule. In the future, when the company can provide a forecast of the upcoming voyages, Algorithm 2 can then be applied to obtain optimal dates to schedule the next cleanings.

## 6. Conclusions

In this paper, we presented a novel optimization framework that determines vessel-cleaning schedules dynamically based on real-world voyage data. Compared to the fixed-time policies commonly used in industry, this framework not only quantifies the loss in fuel efficiency due to hull fouling mathematically but also adapts cleaning decisions to the vessel’s operational state. Specifically, the core components of the methodology are (i) a data-driven function that predicts vessel fuel consumption based on several input variables, including approximations of hull fouling, and (ii) an objective function that induces vessel cleaning when the associated costs can be offset by the resulting increase in fuel efficiency. Theoretically, we proved that the most general formulation of the CSO problem is NP-hard, a brute-force search guarantees finding a globally optimal solution with exactly  $n \cdot 2^n$  calls to  $f$ , and that no other deterministic algorithm can find an optimum with fewer calls. To ensure practicality of the approach for real-world deployment, we introduced reasonable structural assumptions under which the problem becomes solvable in polynomial time via dynamic programming. In particular, we demonstrated that the adjusted framework satisfies the optimal substructure property, which enables the programming approach presented in Algorithm 2 and ensures computational efficiency even for hourly data over several years.

In our numerical analysis, we evaluated the performance of the proposed optimization framework using real-world vessel data. Through a cooperation agreement with a UK-based shipping company, we obtained access to voyage records from ten tramp-trading vessels. In the academic literature, these datasets had never been used and analyzed before, and offered more vessel-specific fouling profiles than data from ships operating on fixed schedules and routes. This enabled us to contribute a novel performance comparison of several ML techniques deployed for each vessel to predict fuel consumption and evaluated as candidates for the data-driven function in our optimization framework. Across several linear and non-linear models, the tree-based approaches RF, ET and XGB achieved the best performance. In contrast, the linear models, LinReg and LASSO, generated much worse predictions, which pointed to substantial non-linear patterns and interaction effects in the underlying data. This supposition was further supported by SHAP dependence plots generated for each vessel based on the XGB model, which outperformed the other approaches in terms of RMSE, MAE and  $R^2$ . In this part of the analysis, we also examined the effects of the variables constructed to approximate hull fouling and found predominately plausible impacts on fuel consumption, providing empirical evidence for the usefulness of the variables. After selecting XGB as data-driven function for the CSO, we then also applied the dynamic programming algorithm to the data and derived vessel specific cleaning schedules. Over a time period of four years, these schedules showed that our optimization framework can reduce fuel consumption by up to 5%, even when accounting for the costs of one or two additional cleaning events.

The empirical results confirm that the proposed optimization framework provides a meaningful step towards data-driven cleaning schedules for sustainable and profitable ship operation. Despite these promising results, several directions for future research and development remain. For instance, the data-driven fuel prediction function could be combined with physical models to further improve predictive accuracy. Similarly, the approximation of hull fouling could be enhanced by incorporating and evaluating images from inexpensive underwater cameras. In terms of data, the performance of the optimization framework could also be assessed using ship data with repeating destinations, where fouling patterns are likely more predictable.

## Acknowledgments

The authors would like to thank Carisbrooke Shipping Limited<sup>1</sup>, represented by Captain Simon Merritt and Natalia Walker, for the pleasant collaboration and the provided data.

Furthermore, the authors would like to thank Selin Damla Ahipaşaoğlu for helpful remarks and discussions on the development of the dynamic programming approach.

---

<sup>1</sup>Carisbrooke Shipping Limited , 38 Medina Road, Cowes, Isle of Wight, PO31 7DA, United Kingdom.

## References

1. Adland, R., Cariou, P., Jia, H. and Wolff, F.-C. (2018). The energy efficiency effects of periodic ship hull cleaning, *Journal of Cleaner Production* **178**: 1–13. <https://doi.org/10.1016/j.jclepro.2017.12.247>.
2. Agand, P., Kennedy, A., Harris, T., Bae, C., Chen, M. and Park, E. J. (2023). Fuel consumption prediction for a passenger ferry using machine learning and in-service data: A comparative study, *Ocean Engineering* **284**: 115271. <https://doi.org/10.1016/j.oceaneng.2023.115271>.
3. AlSawaftah, N., Abuwatfa, W., Darwish, N. and Husseini, G. A. (2022). A Review on Membrane Biofouling: Prediction, Characterization, and Mitigation, *Membranes* **12**(12): 1271. <https://doi.org/10.3390/membranes12121271>.
4. Bakka, H., Rognebakke, H., Glad, I., Haff, I. H. and Vanem, E. (2022). Estimating the effect of biofouling on ship shaft power based on sensor measurements, *Ship Technology Research* **70**(3): 209–221. <https://doi.org/10.1080/09377255.2022.2159108>.
5. Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization, *Journal of Machine Learning Research* **13**(10): 281–305. <http://jmlr.org/papers/v13/bergstra12a.html>.
6. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, Information Science and Statistics, 1st edn, Springer, New York. ISBN 978-0-387-31073-2.
7. Bloomfield, N. J., Wei, S., A. Woodham, B., Wilkinson, P. and Robinson, A. P. (2021). Automating the assessment of biofouling in images using expert agreement as a gold standard, *Scientific Reports* **11**: 2739. <https://doi.org/10.1038/s41598-021-81011-2>.
8. Botchkarev, A. (2019). A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms, *Interdisciplinary Journal of Information, Knowledge, and Management* **14**: 45–79. <https://doi.org/10.28945/4184>.
9. Bressy, C., Margaillan, A., Faÿ, F., Linossier, I. and Réhel, K. (2009). 18 - tin-free self-polishing marine antifouling coatings, in C. Hellio and D. Yebra (eds), *Advances in Marine Antifouling Coatings and Technologies*, Woodhead Publishing Series in Metals and Surface Engineering, Woodhead Publishing, Cambridge, UK, pp. 445–491. <https://doi.org/10.1533/9781845696313.3.445>.
10. Chai, T. and Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature, *Geoscientific Model Development* **7**(3): 1247–1250. <https://doi.org/10.5194/gmd-7-1247-2014>.
11. Chapman, L. (2007). Transport and climate change: a review, *Journal of Transport Geography* **15**(5): 354–367. <https://doi.org/10.1016/j.jtrangeo.2006.11.008>.
12. Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, Association for Computing Machinery, New York, NY, pp. 785–794. <https://doi.org/10.1145/2939672.2939785>.
13. Coraddu, A., Lim, S., Oneto, L., Pazouki, K., Norman, R. and Murphy, A. J. (2019). A novelty detection approach to diagnosing hull and propeller fouling, *Ocean Engineering* **176**: 65–73. <https://doi.org/10.1016/j.oceaneng.2019.01.054>.
14. Coraddu, A., Oneto, L., Baldi, F. and Anguita, D. (2017). Vessels fuel consumption forecast and trim optimisation: A data analytics perspective, *Ocean Engineering* **130**: 351–370. <https://doi.org/10.1016/j.oceaneng.2016.11.058>.
15. Coraddu, A., Oneto, L., Baldi, F., Cipollini, F., Atlar, M. and Savio, S. (2019). Data-driven ship digital twin for estimating the speed loss caused by the marine fouling, *Ocean Engineering* **186**: 106063. <https://doi.org/10.1016/j.oceaneng.2019.05.045>.
16. Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2022). *Introduction to Algorithms*, 4th edn, The MIT Press, Cambridge, Massachusetts. ISBN 978-0262046305.
17. Dantzig, G. B. (1957). Discrete-Variable Extremum Problems, *Operations Research* **5**(2): 266–

288. <https://doi.org/10.1287/opre.5.2.266>.
18. Darwin, C. (1851). *A monograph on the sub-class Cirripedia, with figures of all the species*, Vol. 1 of *Ray Society Publication*, Ray Society, London, UK. <https://doi.org/10.5962/bhl.title.2104>.
19. Dasgupta, S., Papadimitriou, C. H. and Vazirani, U. V. (2006). *Algorithms*, 1st edn, McGraw-Hill Higher Education, New York, NY, USA. ISBN 978-0073523408.
20. de Haas, M., Coraddu, A., El Mouhandiz, A.-A., Dimitra Charisi, N. and Kana, A. A. (2023). Power Increase due to Marine Biofouling: a Grey-box Model Approach, *Proceedings of the 4th International Conference on Modelling and Optimisation of Ship Energy Systems*. <https://doi.org/10.59490/moses.2023.661>.
21. Demirel, Y. K., Song, S., Turan, O. and Incecik, A. (2019). Practical added resistance diagrams to predict fouling impact on ship performance, *Ocean Engineering* **186**: 106112. <https://doi.org/10.1016/j.oceaneng.2019.106112>.
22. Demirel, Y. K., Turan, O. and Incecik, A. (2017). Predicting the effect of biofouling on ship resistance using cfd, *Applied Ocean Research* **62**: 100–118. <https://doi.org/10.1016/j.apor.2016.12.003>.
23. Din, A. U., Ur Rahman, I., Vega-Muñoz, A., Elahi, E., Salazar-Sepúlveda, G., Contreras-Barraza, N. and Alhrahsheh, R. R. (2023). How sustainable transportation can utilize climate change technologies to mitigate climate change, *Sustainability* **15**(12). <https://doi.org/10.3390/su15129710>.
24. Dunn, O. J. (1961). Multiple Comparisons Among Means, *Journal of the American Statistical Association* **56**(293): 52–64. <https://doi.org/10.1080/01621459.1961.10482090>.
25. EU (2016). Commission delegated regulation (eu) 2016/2071, *Official Journal of the European Union*. [http://data.europa.eu/eli/reg\\_del/2016/2071/oj](http://data.europa.eu/eli/reg_del/2016/2071/oj).
26. Fagerholt, K., Laporte, G. and Norstad, I. (2010). Reducing fuel emissions by optimizing speed on shipping routes, *Journal of the Operational Research Society* **61**(3): 523–529. <https://doi.org/10.1057/jors.2009.77>.
27. Fahrmeir, L., Kneib, T., Lang, S. and Marx, B. D. (2021). *Regression: Models, Methods and Applications*, 2nd edn, Springer, Berlin. <https://doi.org/10.1007/978-3-662-63882-8>.
28. First, M. R., Riley, S. C., Islam, K. A., Hill, V., Li, J., Zimmerman, R. C. and Drake, L. A. (2021). Rapid quantification of biofouling with an inexpensive, underwater camera and image analysis, *Management of Biological Invasions* **12**(3): 599–617. <https://doi.org/10.3391/mbi.2021.12.3.06>.
29. Floerl, O. and Inglis, G. J. (2003). Boat harbour design can exacerbate hull fouling, *Austral Ecology* **28**(2): 116–127. <https://doi.org/10.1046/j.1442-9993.2003.01254.x>.
30. Fox, J. (2015). *Applied Regression Analysis and Generalized Linear Models*, 3rd edn, SAGE Publications, Thousand Oaks, CA.
31. Géron, A. (2022). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 3rd edn, O'Reilly Media, Sebastopol, CA.
32. Gianfagna, L. and Di Cecco, A. (2021). *Explainable AI with Python*, 1st edn, Springer, Cham. <https://doi.org/10.1007/978-3-030-68640-6>.
33. Gkerekos, C., Lazakis, I. and Theotokatos, G. (2019). Machine learning models for predicting ship main engine Fuel Oil Consumption: A comparative study, *Ocean Engineering* **188**: 106282. <https://doi.org/10.1016/j.oceaneng.2019.106282>.
34. Goren Huber, L., Kunz, S. and Dettling, M. (2017). Condition-Based Maintenance Decision Making: a Practical Approach for Marine Vessels, *30th Conference for Condition Monitoring and Diagnostic Engineering Management (COMADEM)*, Jost Institute for Tribotechnology, Lancashire, UK, pp. 377–386. <https://digitalcollection.zhaw.ch/handle/11475/8014>.
35. Granhag, L., Javadi, M. and Ytreberg, E. (2023). Best practice for cleaning of ship hulls. *Swedish Maritime and Water Authority*. <https://research.chalmers.se/publication/>

- 535739.
36. Gupta, P., Rasheed, A. and Steen, S. (2022). Ship performance monitoring using machine-learning, *Ocean Engineering* **254**: 111094. <https://doi.org/10.1016/j.oceaneng.2022.111094>.
  37. Handayani, M. P., Kim, H., Lee, S. and Lee, J. (2023). Navigating Energy Efficiency: A Multifaceted Interpretability of Fuel Oil Consumption Prediction in Cargo Container Vessel Considering the Operational and Environmental Factors, *Journal of Marine Science and Engineering* **11**(11): 2165. <https://doi.org/10.3390/jmse11112165>.
  38. Hansen, J. (2012). Game over for the climate. The New York Times. Accessed: 2025-12-29. <https://www.nytimes.com/2012/05/10/opinion/game-over-for-the-climate.html>.
  39. Hastie, T., Tibshirani, R. and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics, 2nd edn, Springer, New York, NY. <https://doi.org/10.1007/b94608>.
  40. Hua, J., Chiu, Y.-S. and Tsai, C.-Y. (2018). En-route operated hydroblasting system for counteracting biofouling on ship hull, *Ocean Engineering* **152**: 249–256. <https://doi.org/10.1016/j.oceaneng.2018.01.050>.
  41. Huang, L., Pena, B., Liu, Y. and Anderlini, E. (2022). Machine learning in sustainable ship design and operation: A review, *Ocean Engineering* **266**: 112907. <https://doi.org/10.1016/j.oceaneng.2022.112907>.
  42. Huotari, J., Manderbacka, T., Ritari, A. and Tammi, K. (2021). Convex Optimisation Model for Ship Speed Profile: Optimisation under Fixed Schedule, *Journal of Marine Science and Engineering* **9**(7): 730. <https://doi.org/10.3390/jmse9070730>.
  43. IMO (2016). Resolution MEPC.278(70) amendments to the annex of the protocol of 1997 to amend the international convention for the prevention of pollution from ships, 1973, as modified by the protocol of 1978 relating thereto, *Marine Environment Protection Committee (MEPC)*. [https://wwwcdn.imo.org/localresources/en/KnowledgeCentre/IndexofIMOResolutions/MEPCDocuments/MEPC.278\(70\).pdf](https://wwwcdn.imo.org/localresources/en/KnowledgeCentre/IndexofIMOResolutions/MEPCDocuments/MEPC.278(70).pdf).
  44. IMO (2023a). Marine Environment: Biofouling [Online]. Accessed: 2026-01-13, <https://www.imo.org/en/ourwork/environment/pages/biofouling.aspx>.
  45. IMO (2023b). Resolution MEPC.377(80) 2023 imo strategy on reduction of ghg emissions from ships, *Marine Environment Protection Committee (MEPC) Resolutions*. [https://wwwcdn.imo.org/localresources/en/KnowledgeCentre/IndexofIMOResolutions/MEPCDocuments/MEPC.377\(80\).pdf](https://wwwcdn.imo.org/localresources/en/KnowledgeCentre/IndexofIMOResolutions/MEPCDocuments/MEPC.377(80).pdf).
  46. James, G., Witten, D., Hastie, T. and Tibshirani, R. (2021). *An Introduction to Statistical Learning with Applications in R*, 2nd edn, Springer, New York, NY. <https://doi.org/10.1007/978-1-0716-1418-1>.
  47. Kamath, U. and Liu, J. (2021). *Explainable Artificial Intelligence: An Introduction to Interpretable Machine Learning*, 1st edn, Springer, Cham. <https://doi.org/10.1007/978-3-030-83356-5>.
  48. Karp, R. M. (1972). Reducibility among combinatorial problems, in R. E. Miller, J. W. Thatcher and J. D. Bohlinger (eds), *Complexity of Computer Computations*, 1st edn, The IBM Research Symposia Series, Springer, Boston, MA, pp. 85–103. [https://doi.org/10.1007/978-1-4684-2001-2\\_9](https://doi.org/10.1007/978-1-4684-2001-2_9).
  49. Kellerer, H., Pferschy, U. and Pisinger, D. (2004). *Knapsack Problems*, 1st edn, Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-540-24777-7>.
  50. Lang, X., Wu, D. and Mao, W. (2022). Comparison of supervised machine learning methods to predict ship propulsion power at sea, *Ocean Engineering* **245**: 110387. <https://doi.org/10.1016/j.oceaneng.2021.110387>.
  51. Laurie, A., Anderlini, E., Dietz, J. and Thomas, G. (2021). Machine learning for shaft power prediction and analysis of fouling related performance deterioration, *Ocean Engineering* **234**: 108886. <https://doi.org/10.1016/j.oceaneng.2021.108886>.

52. Lewis, J. A. and Coutts, A. D. M. (2009). Biofouling Invasions, in S. Dürr and J. Thomason (eds), *Biofouling*, John Wiley & Sons, Hoboken, NJ, pp. 348–365. <https://doi.org/10.1002/9781444315462.ch24>.
53. Loyola-González, O. (2019). Black-Box vs. White-Box: Understanding Their Advantages and Weaknesses From a Practical Point of View, *IEEE Access* **7**: 154096–154113. <https://doi.org/10.1109/ACCESS.2019.2949286>.
54. Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions, *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, Curran Associates Inc., Red Hook, NY, pp. 4768–4777.
55. Ma, Y., Zhao, Y., Yu, J., Zhou, J. and Kuang, H. (2023). An Interpretable Gray Box Model for Ship Fuel Consumption Prediction Based on the SHAP Framework, *Journal of Marine Science and Engineering* **11**(5): 1059. <https://doi.org/10.3390/jmse11051059>.
56. Mathews, G. B. (1896). On the Partition of Numbers, *Proceedings of the London Mathematical Society* **s1-28**(1): 486–490. <https://doi.org/10.1112/plms/s1-28.1.486>.
57. Molnar, C. (2022). *Interpretable Machine Learning*, 2nd edn, Independently published. <https://christophm.github.io/interpretable-ml-book>.
58. Molnar, J. L., Gamboa, R. L., Revenga, C. and Spalding, M. D. (2008). Assessing the global threat of invasive species to marine biodiversity, *Frontiers in Ecology and the Environment* **6**(9): 485–492. <https://doi.org/10.1890/070064>.
59. Moreira, L., Vettor, R. and Guedes Soares, C. (2021). Neural Network Approach for Predicting Ship Speed and Fuel Consumption, *Journal of Marine Science and Engineering* **9**(2): 119. <https://doi.org/10.3390/jmse9020119>.
60. Nakamura, J. and Ishida, M. (2025). Global Warming and CO<sub>2</sub> Emissions, *Blueprint for a Methanol Society: Toward Carbon-Neutrality*, 1st edn, Springer Nature, Singapore, pp. 3–26. [https://doi.org/10.1007/978-981-95-3465-4\\_1](https://doi.org/10.1007/978-981-95-3465-4_1).
61. Nikolaidis, G. and Themelis, N. (2022). Examining the performance of retrofit measures in real ship operation using data-driven models, *Ship Technology Research* **69**(3): 170–180. <https://doi.org/10.1080/09377255.2022.2109327>.
62. Olivera, D. R., Lagerström, M., Granhag, L., Werner, S., Larsson, A. I. and Ytreberg, E. (2022). A novel tool for cost and emission reduction related to ship underwater hull maintenance, *Journal of Cleaner Production* **356**: 131882. <https://doi.org/10.1016/j.jclepro.2022.131882>.
63. Papadimitriou, C. H. (1994). *Computational Complexity*, 1st edn, Addison-Wesley, Reading, Mass. ISBN 978-0201530827.
64. Parviaainen, T., Lehikoinen, A., Kuikka, S. and Haapasaari, P. (2018). How can stakeholders promote environmental and social responsibility in the shipping industry?, *WMU Journal of Maritime Affairs* **17**: 49–70. <https://doi.org/10.1007/s13437-017-0134-z>.
65. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011). Scikit-learn: Machine learning in python, *Journal of Machine Learning Research* **12**: 2825–2830. <https://doi.org/10.5555/1953048.2078195>.
66. Ravichandiran, S. (2019). *Hands-On Deep Learning Algorithms with Python*, 1st edn, Packt Publishing, Birmingham. ISBN 978-1789344158.
67. Rothman, D. (2020). *Hands-On Explainable AI (XAI) with Python*, 1st edn, Packt Publishing, Birmingham. ISBN 978-1800202764.
68. Rumala, D. (2023). How You Split Matters: Data Leakage and Subject Characteristics Studies in Longitudinal Brain MRI Analysis, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2023*, Vol. 14227 of *Lecture Notes in Computer Science*, Springer, Cham, pp. 235–245. [https://doi.org/10.1007/978-3-031-45249-9\\_23](https://doi.org/10.1007/978-3-031-45249-9_23).
69. Schultz, M. P., Bendick, J., Holm, E. and Hertel, W. (2011). Economic impact of biofouling

- on a naval surface ship, *Biofouling* **27**(1): 87–98. <https://doi.org/10.1080/08927014.2010.542809>.
70. Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression, *Statistics and Computing* **14**(3): 199–222. <https://doi.org/10.1023/B:STCO.0000035301.49549.88>.
  71. Soner, O., Akyuz, E. and Celik, M. (2019). Statistical modelling of ship operational performance monitoring problem, *Journal of Marine Science and Technology* **24**: 543–552. <https://doi.org/10.1007/s00773-018-0574-y>.
  72. Song, C. and Cui, W. (2020). Review of Underwater Ship Hull Cleaning Technologies, *Journal of Marine Science and Application* **19**: 415–429. <https://doi.org/10.1007/s11804-020-00157-z>.
  73. Srinivasan, M. and Swain, G. W. (2007). Managing the use of copper-based antifouling paints, *Environmental Management* **39**: 423–441. <https://doi.org/10.1007/s00267-005-0030-8>.
  74. Tamburri, M. N., Davidson, I. C., First, M. R., Scianni, C., Newcomer, K., Inglis, G. J., Georgiades, E. T., Barnes, J. M. and Ruiz, G. M. (2020). In-Water Cleaning and Capture to Remove Ship Biofouling: An Initial Evaluation of Efficacy and Environmental Safety, *Frontiers in Marine Science* **7**. <https://doi.org/10.3389/fmars.2020.00437>.
  75. Tibshirani, R. (1996). Regression Shrinkage and Selection Via the Lasso, *Journal of the Royal Statistical Society: Series B (Methodological)* **58**(1): 267–288. <https://doi.org/10.1111/j.2517-6161.1996.tb02080.x>.
  76. Townsin, R. L. (2003). The Ship Hull Fouling Penalty, *Biofouling* **19**(sup1): 9–15. <https://doi.org/10.1080/0892701031000088535>.
  77. Uzun, D., Demirel, Y. K., Coraddu, A. and Turan, O. (2019). Time-dependent biofouling growth model for predicting the effects of biofouling on ship resistance and powering, *Ocean Engineering* **191**: 106432. <https://doi.org/10.1016/j.oceaneng.2019.106432>.
  78. Valchev, I., Coraddu, A., Kalikatzarakis, M., Geertsma, R. and Oneto, L. (2022). Numerical methods for monitoring and evaluating the biofouling state and effects on vessels' hull and propeller performance: A review, *Ocean Engineering* **251**: 110883. <https://doi.org/10.1016/j.oceaneng.2022.110883>.
  79. Wade, C. (2020). *Hands-On Gradient Boosting with XGBoost and scikit-learn*, 1st edn, Packt Publishing, Birmingham. ISBN 978-1839218354.
  80. Wang, H., Yan, R., Wang, S. and Zhen, L. (2023). Innovative approaches to addressing the tradeoff between interpretability and accuracy in ship fuel consumption prediction, *Transportation Research Part C: Emerging Technologies* **157**: 104361. <https://doi.org/10.1016/j.trc.2023.104361>.
  81. Wichert, A. and Sa-Couto, L. (2021). *Machine Learning – A Journey to Deep Learning*, 1st edn, World Scientific, Singapore. <https://doi.org/10.1142/12201>.
  82. Wu, Y., Hua, J. and Wu, D. L. (2022). Economic analysis of ship operation using a new antifouling strategy, *Ocean Engineering* **266**: 113038. <https://doi.org/10.1016/j.oceaneng.2022.113038>.
  83. Yan, R., Wang, S. and Du, Y. (2020). Development of a two-stage ship fuel consumption prediction and reduction model for a dry bulk ship, *Transportation Research Part E: Logistics and Transportation Review* **138**: 101930. <https://doi.org/10.1016/j.tre.2020.101930>.
  84. Zhou, T., Hu, Q., Hu, Z. and Zhen, R. (2022). An adaptive hyper parameter tuning model for ship fuel consumption prediction under complex maritime environments, *Journal of Ocean Engineering and Science* **7**(3): 255–263. <https://doi.org/10.1016/j.joes.2021.08.007>.
  85. Zis, T. P. V., Psaraftis, H. N. and Ding, L. (2020). Ship weather routing: A taxonomy and survey, *Ocean Engineering* **213**: 107697. <https://doi.org/10.1016/j.oceaneng.2020.107697>.

## A. Appendix

### A.1. Theorem A.1

In this subsection of the Appendix, we prove that the most general formulation of the [CSO](#) is NP-hard. Theorem [A.1](#) summarizes this result, which implies that no polynomial time algorithm can be expected to compute the global minimum of [\(CLEANING-1\)](#).

#### Theorem A.1

The cleaning schedule decision problem “Does there exist a cleaning schedule  $\mathbf{z}$  that achieves a cost of  $C$  or less for an instance of the problem [\(CLEANING-1\)](#)?” is NP-hard.

#### Proof

To proof Theorem [A.1](#), we reduce the well-known NP-hard 0-1 knapsack decision problem [[Mathews \(1896\)](#); [Dantzig \(1957\)](#); [Kellerer et al. \(2004\)](#)] to our cleaning schedule decision problem. In the 0-1 knapsack problem of size  $n$ , we enumerate a set of items  $1, \dots, n$ . Each item  $i$  has a value  $v_i \in \mathbb{R}$  and a weight  $w_i \in \mathbb{R}$ . There is a maximum weight  $W \in \mathbb{R}$  and target value  $V \in \mathbb{R}$ . The decision variable  $u_i \in \{0, 1\}$  is 1 if item  $i$  is included in the knapsack and 0 otherwise. The bold notation  $\mathbf{w} := [w_1, \dots, w_n]^\top$ ,  $\mathbf{v} := [v_1, \dots, v_n]^\top$ , and  $\mathbf{u} := [u_1, \dots, u_n]^\top$  is used to stack those weights, values and 0-1 decisions, respectively, into vectors according to the order of the  $n$  items. The decision problem then asks, “Does there exist a selection of items  $\mathbf{u}$ , whose values total to at least  $V$  but whose weights are at most  $W$  for an instance of the problem [\(KNAPSACK\)](#)?”. The proof that this problem is NP-hard was first given by [Karp \(1972\)](#), p. 100 through a sequence of reductions.

$$\begin{aligned} & \underset{\mathbf{u}}{\text{maximize}} \quad \sum_{i=1}^n v_i \cdot u_i \\ & \text{subject to} \quad \sum_{i=1}^n w_i \cdot u_i \leq W, \\ & \quad \mathbf{u} \in \{0, 1\}^n. \end{aligned} \tag{KNAPSACK}$$

Given an instance of [\(KNAPSACK\)](#)  $(\mathbf{v}, \mathbf{w}, V, W) \in (\mathbb{R}^n, \mathbb{R}^n, \mathbb{R}, \mathbb{R})$  for  $n$  items, we transform it into an instance of the [\(CLEANING-1\)](#) decision problem  $(\mathbf{c}, \mathbf{X}, f, C) \in (\mathbb{R}^n, \mathbb{X}^n, \mathbb{X} \times \mathbb{R} \rightarrow \mathbb{R}, \mathbb{R})$  for  $n$  voyages. Notice that, since we are dealing with the decision problem, the additional parameter  $C \in \mathbb{R}$  is the target cost. This is a standard practice to convert integer optimization problems into 0-1-decision problems to prove NP-hardness [[Papadimitriou \(1994, Theorem 2\)](#)]. For the sake of this reduction, we design the ship profiles for a given voyage as a single real number ( $\mathbb{X} = \mathbb{R}$ ) rather than a data matrix so that each  $\mathbf{X}_i \in \mathbb{R}$  for  $i = 1, \dots, n$ . The transformation is then as follows:

$$C := -V, \quad \mathbf{c} := -\mathbf{v}, \quad \mathbf{X} := \mathbf{w}, \quad f(\mathbf{X}_{:i}, \mathbf{z}_{:i}) := \begin{cases} C + n \cdot \max(|\mathbf{c}|) + 1 & \text{if } \sum \mathbf{X}_i \cdot z_i > W, \\ 0 & \text{otherwise.} \end{cases}$$

Suppose there exists a selection of items  $\mathbf{u}^* \in \{0, 1\}^n$  that satisfy the 0-1 knapsack decision problem such that the weight constraint is met  $\sum w_i \cdot u_i^* \leq W$  and the target value is achieved  $\sum v_i \cdot u_i^* \geq V$ . Consider the cleaning schedule  $\mathbf{z}^* = \mathbf{u}^*$ . In the transformed instance, we have  $f(\mathbf{X}_{:i}, \mathbf{z}_{:i}^*) = 0$  for each  $i$ . Therefore, the cost of this cleaning schedule is  $\sum_{i=1}^n f(\mathbf{X}_{:i}, \mathbf{z}_{:i}^*) + \sum_{i=1}^n c_i \cdot z_i^* = 0 + \sum_{i=1}^n -v_i \cdot u_i^* \leq -V = C$  satisfying the cleaning scheduling decision problem.

Suppose instead that there does not exist a selection of items that satisfy the 0-1 knapsack decision problem. This means for all selections of items  $\mathbf{u}' \in \{0, 1\}^n$ , either the weight constraint is violated  $\sum w_i \cdot u'_i > W$  or the target value is not achieved  $\sum v_i \cdot u'_i < V$ .

Consider the corresponding cleaning schedules  $\mathbf{z}' = \mathbf{u}'$  in the transformed problem. In the first case where  $\sum w_i \cdot u'_i > W$ , the function  $f(\mathbf{X}_{:n}, \mathbf{z}'_{:n})$  evaluates to  $C + n \cdot \max(|\mathbf{c}|) + 1$  ensuring that the cost cannot be less than  $C$ . In the second case where  $\sum v_i \cdot u'_i < V$ , we find that the sum of cleaning costs is  $\sum c_i \cdot z'_i > C$ , so the overall cost cannot be less than  $C$ . Thus, the cleaning schedule problem is unsatisfied for  $\mathbf{z}' \in \{0, 1\}^n$ .

To finish the proof, we conclude that a solution to the 0-1 knapsack decision problem exists if and only if a solution to the transformed problem exists. Therefore, should there be some polynomial time algorithm to solve instances of (CLEANING-1), we could transform any instance of (KNAPSACK) of size  $n$  into an instance of (CLEANING-1) of size  $n$  and by solving it as a proxy to achieve polynomial time algorithm for 0-1 knapsack.

□

## A.2. Theorem A.2

In this subsection of the Appendix, we provide important properties for Algorithm 1. In particular, Theorem A.2 derives the number of function calls to  $f$  that are required to find the global solution of (CLEANING-1). Overall, these results further show motivate the introduction of (CLEANING-2) as a more tractable problem.

### Theorem A.2

Given an arbitrary instance  $(\mathbf{c}, \mathbf{X}, f) \in (\mathbb{R}^n, \mathbb{X}^n, \mathbb{X}^n \times \mathbb{H} \rightarrow \mathbb{R})$  of (CLEANING-1):

1. The brute force algorithm finds a global minimum with exactly  $n \cdot 2^n$  calls to  $f$ .
2. If no assumptions are made on the structure of  $f$ , then no other deterministic algorithm can guarantee to find an optimal cleaning schedule with fewer than  $n \cdot 2^n$  calls to  $f$ .

### Proof

1. On all instances, Algorithm 1 loops  $|\{0, 1\}^n| = 2^n$  times. For each loop, Line 4 evaluates the objective value, which requires  $n$  calls to  $f$ . Thus, overall, it makes exactly  $n \cdot 2^n$  calls to  $f$ .
2. Suppose, towards a contradiction, that some deterministic algorithm  $\mathcal{A}$  can find an optimal solution in fewer than  $n \cdot 2^n$  calls to  $f$ . Let  $I_1 := (\mathbf{c}, \mathbf{X}, f_{I_1})$  be an arbitrary problem instance. Let  $\mathbf{z}^*$  be the solution returned by  $\mathcal{A}$  when run on instance  $I_1$ , which achieves objective value  $\phi^*$ . Note there may be multiple optimal solutions but  $\mathcal{A}$  always returns the first one in lexicographical search order. By the pigeonhole principle there exists some voyage  $j \in \{1, \dots, n\}$  and some cleaning schedule  $\mathbf{z}' \in \{0, 1\}^n$  such that algorithm  $\mathcal{A}$  does not call  $f(\mathbf{X}_{:j}, \mathbf{z}'_{:j})$ . Consider the case where  $\mathbf{z}^* \neq \mathbf{z}'$ . Construct a new instance  $I_2 := (\mathbf{c}, \mathbf{X}, f_{I_2})$  such that  $f_{I_2}$  returns the same value as  $f_{I_1}$  on all inputs apart from  $\mathbf{X}_{:j}, \mathbf{z}'_{:j}$  where

$$f_{I_2}(\mathbf{X}_{:j}, \mathbf{z}'_{:j}) := \sum_{i \neq j} f_{I_1}(\mathbf{X}_{:i}, \mathbf{z}'_{:i}) - \sum_{i \neq j} c_i \cdot z'_i + \phi^* - 1.$$

In this new instance, the optimal objective value achieved by  $\mathbf{z}'$  is  $\phi^* - 1$ , which is less than  $\mathbf{z}^*$ . Since algorithm  $\mathcal{A}$  is deterministic and does not call  $f$  on instance  $(\mathbf{X}_{:j}, \mathbf{z}'_{:j})$ , it cannot distinguish between the two instances  $I_1$  and  $I_2$  so it must produce the same output of  $\mathbf{z}^*$  on both. But this means it fails to find the optimal schedule, for instance,  $I_2$ . Now consider the case where  $\mathbf{z}^* = \mathbf{z}'$ . We can construct a new instance  $I_3 := (\mathbf{c}, \mathbf{X}, f_{I_3})$  where  $f_{I_3} : (\mathbf{X}_{:j}, \mathbf{z}'_{:j}) \mapsto \infty$ . Using the same argument, we conclude that  $\mathcal{A}$  fails to find the optimal schedule, for instance  $I_3$ . These two cases show the contradiction.

□

### A.3. Proof of Theorem 3.1

In this part of the Appendix, we provide the proof of Theorem 3.1, which justifies the application of dynamic programming to find a solution of (CLEANING-2).

#### Proof: Theorem 3.1

*Case  $z_i = 1$ .* Assume that it is optimal to clean before voyage  $i$ , which implies that the hull fouling metric for voyage  $i$  is set to zero  $b_i = 0$ . By definition,  $[\hat{z}_{i+1}, \dots, \hat{z}_n]^\top := \Psi[i+1, i]$  is the optimal cleaning schedule for the subsequent voyages if we clean at time point  $i$ . Consider the cleaning schedule  $\mathbf{z}^{(\text{clean})} = [1, \hat{z}_{i+1}, \dots, \hat{z}_n]^\top$ , which achieves the following objective value

$$\begin{aligned}\Phi[i, j] &= \sum_{k=i}^n \tilde{f}(\mathbf{X}_k, b_k) + \sum_{k=i}^n c_k \cdot z_k \\ &= \tilde{f}(\mathbf{X}_i, b_i) + c_i + \sum_{k=i+1}^n \tilde{f}(\mathbf{X}_k, b_k) + \sum_{k=i+1}^n c_k \cdot z_k \\ &= \tilde{f}(\mathbf{X}_i, 0) + c_i + \Phi[i+1, i].\end{aligned}\tag{5}$$

*Case  $z_i = 0$ .* Assume that it is optimal to not clean before voyage  $i$ . This would imply that the hull fouling metric before voyage  $i$  takes the value  $b_i = \sum_{k=j}^{i-1} B_k$ , which is the accumulation of the fouling over the voyages  $\{j, j+1, \dots, i-1\}$ . By definition,  $[\hat{z}_{i+1}, \dots, \hat{z}_n]^\top := \Psi[i+1, j]$  is the optimal cleaning schedule for the subsequent voyages if we do not clean at time point  $i$ . Consider the cleaning schedule  $\mathbf{z}^{(\text{not})} = [0, \hat{z}_{i+1}, \dots, \hat{z}_n]^\top$ , which achieves the following objective value

$$\begin{aligned}\Phi[i, j] &= \sum_{k=i}^n \tilde{f}(\mathbf{X}_k, b_k) + \sum_{k=i}^n c_k \cdot z_k \\ &= \tilde{f}(\mathbf{X}_i, b_i) + c_i \cdot 0 + \sum_{k=i+1}^n f(\mathbf{X}_k, b_k) + \sum_{k=i+1}^n c_k \cdot z_k \\ &= \tilde{f}\left(\mathbf{X}_i, \sum_{k=j}^{i-1} B_k\right) + \Phi[i+1, j].\end{aligned}\tag{6}$$

□

### A.4. Linear Regression

A widely used statistical method for modeling the relationship between a dependent variable  $\mathbf{y}$  and a set of independent variables is the LinReg [Fahrmeir et al. (2021, p. 24)]. The main idea of the approach is to express the response variable as a non-deterministic linear function of the covariates. For an empirical dataset, this can be translates into equations of the following form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where  $\boldsymbol{\beta} \in \mathbb{R}^k$  denotes the true unknown regression coefficients, and  $\boldsymbol{\varepsilon} \in \mathbb{R}^m$  is the unknown random error term. To estimate the regression coefficients, a popular approach is to minimize the sum of squared residuals that is given as

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \sum_{t=1}^m (y_t - \mathbf{x}_t \boldsymbol{\beta})^2,$$

and is also known as Ordinary Least Squares [Fahrmeir et al. (2021, p. 26)]. The main advantages of the LinReg are its simplicity, and interpretability [Hastie et al. (2009, p. 43)].

### A.5. Least Absolute Shrinkage and Selection Operator

The **LASSO** regression introduced by Tibshirani (1996) is another common statistical method to linearly model the connection between a set of covariates and a continuous outcome. To estimate the regression coefficients, **LASSO** minimizes the penalized least squares criterion, given as

$$\underset{\beta}{\text{minimize}} \frac{1}{2} \cdot \sum_{t=1}^m (y_t - \mathbf{x}_t \beta)^2 + \lambda \cdot \sum_{l=1}^k |\beta_l|,$$

where  $\lambda \cdot \sum_{l=1}^k |\beta_l|$  with  $\lambda > 0$  represents the penalty term. This approach is similar to **LinReg** but additionally incorporates regularization to prevent overfitting and automatic feature selection [Fahrmeir et al. (2021, p. 170)]. Note that the user has to specify the parameter  $\lambda$  that decides over the strength of the penalization. In our numerical experiments, we perform a random search of  $\lambda$  values between 0.1 and 5.

### A.6. Multilayer Perceptron

A **ML** algorithm that allows for more complex relationships between input and output is the **MLP**. The main idea of this method is to use a series of non-linear transformations instead of just calculating weighted sums of the independent variables [Wichert and Sa-Couto (2021, p. 427)]. The general structure of **MLP** then can be summarized based on three components: input layer, hidden layer(s), and output layer [Ravichandiran (2019, p. 11)]. While the former component is only used to enter the variables, the hidden layers perform the non-linear transformations of the features based on activation functions [Ravichandiran (2019, pp. 12–13)]. For example, the procedure at the first hidden layer can be described as

$$h_1(\mathbf{X}\mathbf{W}_1 + \mathbf{B}_1) = \mathbf{Z}_1,$$

where  $h_1 : \mathbb{R}^{m \times k} \rightarrow \mathbb{R}^{m \times u_1}$  denotes the activation function with  $u_1$  being the number of neurons in the layer,  $\mathbf{W}_1 \in \mathbb{R}^{k \times u_1}$  represents the weight matrix, and  $\mathbf{B}_1 \in \mathbb{R}^{m \times u_1}$  is the so-called bias term. Furthermore,  $\mathbf{Z}_1 \in \mathbb{R}^{m \times u_1}$  denotes the output of the first layer which replaces  $\mathbf{X}$  as input in the second layer. The procedure then continues until the output layer is reached, which has to ensure that the outcome of the model has the correct dimensions. Due to this set-up, the **MLP** is generally more powerful than the two linear models, but it comes at the price that the user must specify several hyperparameters. Unfortunately, the optimal choices for these parameters are often unknown as they highly depend on the structure of the underlying problem. For example, there exist several options for the activation function, the number of hidden layers, and the amount of neurons per layer that significantly influence the performance of **MLP**. In practice, the hyperparameters are often selected by evaluating different combinations. In our numerical experiments, we try each of the following hyperparameter combinations:

```
"hidden_layer_sizes": [(50, 50, 50), (50, 100, 50), (25, 50, 100, 100, 50, 25)],  
"activation": ["relu", "tanh"],
```

### A.7. Support Vector Regression

Another alternative approach is **SVR** [Smola and Schölkopf (2004)], which aims to find a function that approximates the data within a specified tolerance, rather than minimizing squared residuals. Formally, **SVR** solves the following optimization problem:

$$\begin{aligned} & \underset{\beta, b, \xi, \xi^*}{\text{minimize}} \quad \frac{1}{2} \|\beta\|^2 + C \cdot \sum_{t=1}^m (\xi_t + \xi_t^*), \\ & \text{subject to} \quad y_t - (\mathbf{x}_t \beta + b) \leq \varepsilon + \xi_t, \\ & \quad (\mathbf{x}_t \beta + b) - y_t \leq \varepsilon + \xi_t^*, \\ & \quad \xi_t, \xi_t^* \geq 0, \end{aligned}$$

where  $\xi_t, \xi_t^* \in \mathbb{R}$  are slack variables allowing violations of the  $\varepsilon$ -insensitive margin, and  $C > 0$  is a regularization parameter controlling the trade-off between model complexity and tolerance to deviations. The main advantages of **SVR** comes from its robustness to outliers, and its ability to handle non-linear relationships through its dual formulation and kernel trick [Smola and Schölkopf (2004, p. 193)]. However, this approach also introduces several hyperparameters. In our numerical experiments, we use the **SVR** implementation of the Python library **scikit-learn** [Pedregosa et al. (2011)] with the following hyperparameter search space:

```
"C": (0.01, 0.1, 1, 10, 100, 1000),
"gamma": (10, 1, 0.1, 0.01, 0.001, 0.0001),
"kernel": ("rbf",)
```

### A.8. k-Nearest Neighbors

All previously explained algorithms require the estimation of parameters that constrain the model to a predefined functional form. One approach that circumvents this inflexibility is the **kNN**. It is one of the simplest non-parametric methods and essentially only requires the storage of the whole dataset [James et al. (2021, p. 105)]. The outcome of any observation  $\mathbf{x}_0$  is then predicted by a comparison to all other instances, where similarity is measured based on a distance measure. A common choice, in this regard, is the Minkowski metric (p-norm), calculated as

$$\|\mathbf{x}_0 - \mathbf{x}_t\|^p = \left( \sum_{l=1}^q |x_{0l} - x_{tl}|^p \right)^{\frac{1}{p}},$$

where  $p \in \mathbb{N}$  is the choice of norm, and  $q \in \mathbb{N}$  represents the number of features (not to be confused with  $k \in \mathbb{N}$ , which, in this case is the number of nearest neighbors). Based on such a metric a similarity ranking can then be constructed, which is used to create a neighborhood  $\mathcal{N}_k(\mathbf{x}_0)$  of the  $k$  most similar observations. Afterwards, the outcome for the instance  $\mathbf{x}_0$  is calculated as an average of the response variables from this neighborhood [Hastie et al. (2009, p. 14)]. Due to this rather simple set-up, **kNN** only requires the selection of two hyper-parameters:  $k$  and  $p$ . We tried a range values of  $k$  between 1 and 101, and  $p$  as either the manhattan or euclidean norm.

### A.9. Random Forest

The **RF** is a non-parametric approach that is based on regression trees. It is an ensemble learning technique that combines multiple **Decision Trees (DTs)** to create an overall more accurate and robust model. The main idea of a **DT** is to ask a series of yes-no questions based on the input variables that divide the observations into partitions [Bishop (2006, p. 664)]. The outcome of an instance is then predicted by the average outcome in the respective group. **RF** extends this methodology by simultaneously growing multiple de-correlated trees based on a set of bootstrap samples [Hastie et al. (2009, pp. 587–588)]. The overall prediction is then aggregated by averaging over the individual tree outputs. Another peculiarity is the introduction of more randomness by selecting random subsets of variables per yes-no-question, which leads to a greater tree diversity [Hastie et al. (2009, pp. 588–589)]. Note that due to this more sophisticated set-up, the user again has to provide several hyperparameters such as the number of de-correlated trees, the percentage of randomly selected variables that are considered per split, or the allowed depth per **DT**. In our analysis, we perform a random search of the following space:

```
"n_estimators": [200],
"max_features": np.arange(0.2, 1.05, 0.05),
"max_depth": randint(4, 30),
"min_samples_leaf": randint(5, 50),
```

### A.10. Extremely Randomized Trees

The **ET** are another tree-based method that can be seen as an extension of **RF**. It introduces even more randomness by determining the binary decision per yes-no question based on random

thresholds for each variable [Géron (2022, p. 169)]. Due to this adaption, the trees become even more diverse, and the algorithm also gets much faster than **RF**, which aims to find optimal thresholds. Nevertheless, most of the hyperparameters of **RF** remain and accordingly must be specified before the model estimation. In our numerical experiments, we search the same space of hyperparameters as given for **RF** above.

### A.11. Extreme Gradient Boosting

The last non-parametric and tree-based approach that is used in this paper is **XGB**. While **RF** and **ET** grow multiple independent **DTs**, the main idea of **XGB** is to fit a series of trees that aim to correct the predictions of their predecessors [Wade (2020, pp. 110–112)]. Formally, for an arbitrary instance  $t$  this procedure can be expressed as

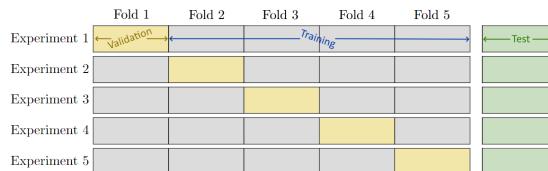
$$\hat{y}_t = \sum_{T=1}^{\bar{T}} f_T(\mathbf{x}_t) \quad \text{with } f_T \in \mathcal{F},$$

where  $\bar{T} \in \mathbb{N}$  denotes the number of additive functions, and  $\mathcal{F} = \{f_T(\mathbf{x}_t) = w_{q(\mathbf{x}_t)}\} (q : \mathbb{R}^k \rightarrow d \in \{1, \dots, D\}, w_d \in \mathbb{R})$  represents the space of regression trees with  $D \in \mathbb{N}$ . Within this space,  $q$  denotes the structure of each tree and allocates an instance to a leaf index  $d$  with a corresponding leaf score  $w_d$ . The leaf scores  $w_d$  of the  $T^{th}$  tree are then estimated based on minimizing a regularized objective function that also contains a penalty term for the complexity of the additional tree [Chen and Guestrin (2016, p. 786–787)]. Note that for this **ML** algorithm also several hyperparameters must be specified by the user [Wade (2020, ch. 6)]. For example, the number of additive trees  $\bar{T}$ , the allowed depth of an individual tree or the strength of the regularization terms must be selected. In our numerical experiments, we use the **XGB** implementation of the Python library **XGBoost** [Chen and Guestrin (2016)] with the following hyperparameter search space:

```
"colsample_bytree": np.arange(0.2, 1.05, 0.05),
"gamma": uniform(0, 2.5),
"learning_rate": uniform(0.01, 0.3),
"reg_alpha": uniform(40, 180),
"reg_lambda": uniform(0, 1),
"max_depth": randint(2, 18),
"n_estimators": randint(150, 700),
"subsample": np.arange(0.2, 1.05, 0.05)
```

### A.12. K-Fold Cross Validation

Figure A.1 illustrates the main idea of K-fold cross validation with  $K = 6$ . In this framework, one of the six folds is set aside as test dataset for the final model evaluation. The remaining folds are used for variable selection and hyperparameter tuning. In particular, the models are fitted five times for each configuration by cycling through the folds one to five as the validation set while using the remaining four folds for training. The overall validation score is then computed as the average of the individual scores and serves to assess the quality of a given feature set or hyperparameter choice [Bishop (2006, pp. 32–33); Hastie et al. (2009, pp. 241–242)].



**Figure A.1.** Visualization of K-fold cross validation with a final test set.

### A.13. Further Regression Scores

To evaluate the performance of the **ML** algorithms, we use three performance metrics:  $R^2$ , **RMSE** and **MAE**. While the results of the  $R^2$  score are directly incorporated in Subsection 5.1, the **RMSE** and **MAE** values are reported in this subsection of the Appendix for the sake of brevity. Based on the results shown in Table A.1, it can be observed that the model rankings based on the **RMSE** and the  $R^2$  score are identical. Overall, **XGB** outperforms all other **ML** models.

**Table A.1.** **RMSE** scores of the **ML** models on the test datasets.

	LinReg	LASSO	kNN	MLP	SVR	ET	RF	XGB
$V_1$	26.8676	26.7702	29.9897	17.3455	10.8206	9.4063	8.7529	<b>6.2752</b>
$V_2$	20.5689	20.6605	30.6290	18.1058	11.0696	9.0086	9.2723	<b>7.7433</b>
$V_3$	20.2513	20.2396	30.8415	12.5006	6.5161	5.8514	5.9280	<b>4.0179</b>
$V_4$	11.4218	11.5367	20.9961	11.4799	12.2190	8.5980	8.9163	<b>6.8672</b>
$V_5$	16.0106	16.0452	27.4599	15.2965	10.6673	7.8016	6.9768	<b>6.6745</b>
$V_6$	47.5881	47.4514	49.4213	27.7973	14.0532	9.7829	9.3958	<b>8.5962</b>
$V_7$	21.7915	21.8496	28.2506	14.1761	8.4934	8.6389	9.7555	<b>8.2419</b>
$V_9$	20.6359	20.8467	26.0503	13.7851	8.8230	6.2017	6.0882	<b>4.1277</b>
$V_{10}$	16.9664	16.9650	26.4931	14.5940	11.5048	5.8268	5.6535	<b>5.4880</b>

Table A.2 reports the **MAE** scores for each vessel and model. In contrast to the **RMSE** results, **XGB** does not consistently outperform the other approaches, as the other two tree-based methods provide slightly better outcomes for a few vessels. In particular, **RF** is the best model in three out of nine cases, and **ET** outperforms all other **ML** algorithms in two out of nine cases. These ranking differences can be explained by the importance of larger prediction errors. Specifically, the **RMSE** and the  $R^2$  score penalize high errors more strongly by squaring all errors, while the **MAE** only considers the absolute difference.

**Table A.2.** **MAE** scores of the **ML** models on the test datasets.

	LinReg	LASSO	kNN	MLP	SVR	ET	RF	XGB
$V_1$	20.4051	19.3193	19.8012	13.0955	8.8206	8.4063	7.7529	<b>6.1702</b>
$V_2$	9.3461	9.6350	19.4036	13.4157	7.1978	7.1606	<b>6.9578</b>	7.5517
$V_3$	9.3176	9.2890	17.7586	10.1653	6.3763	4.5847	4.4440	<b>3.9315</b>
$V_4$	7.0739	7.1368	10.5170	11.7771	5.5238	<b>4.4080</b>	4.7042	5.5446
$V_5$	8.1376	8.1762	17.3489	14.3571	7.3773	6.9749	<b>6.5662</b>	6.8213
$V_6$	26.8383	27.4823	19.0256	14.9325	12.4432	7.3608	<b>6.5948</b>	7.8703
$V_7$	11.7862	11.8897	14.6954	9.6867	8.1088	<b>5.4271</b>	6.2481	5.8392
$V_9$	11.1912	11.8410	15.5353	10.4029	5.9223	4.4587	4.4701	<b>2.9728</b>
$V_{10}$	10.7153	10.5808	18.2649	9.0718	9.1003	5.9203	5.8910	<b>5.0052</b>

### A.14. Bootstrap Confidence Intervals of Performance Metrics

For the sake of brevity, Table 5 in Subsection 5.1 only reported point estimates for the three performance metrics: **RMSE**, **MAE** and  $R^2$  score. To study the uncertainty behind these results, this section of the Appendix provides corresponding **CIs**. For this purpose, we bootstrap 100,000 independent samples with replacement based on the training and testing split of each vessel, and repeatedly score the model's performance [Fox (2015, pp. 655–656)]. In total, this leads to 36 **CIs** that are simultaneously constructed for each evaluation criterion. To ensure that the familywise error rate stays at  $\alpha = 0.05$ , the confidence level is adjusted based on the Bonferroni correction [Dunn (1961)] leading to  $\alpha = \frac{0.05}{36} \approx 0.0014$ . The low ( $\alpha$ ), medium and high ( $1 - \alpha$ ) percentiles are presented in Tables A.3, A.4 and A.5.

**Table A.3.** RMSE bootstrap CIs results for the XGB model whose training is described in Section 5.

	Training RMSE			Test RMSE		
	Low 0.14%	Med 50.00%	High 99.86%	Low 0.14%	Med 50.00%	High 99.86%
$V_1$	1.9131	2.0067	2.1675	5.8628	6.2659	6.6767
$V_2$	3.5962	3.8042	4.0527	7.0472	7.5049	7.9726
$V_3$	0.5668	0.6189	0.6792	3.5733	4.0010	4.4711
$V_4$	2.5193	2.7142	2.9633	6.4384	6.9332	7.6156
$V_5$	1.0277	1.1175	1.2120	5.9388	6.5944	7.7847
$V_6$	1.6050	1.6928	1.7873	7.7066	8.6141	9.4907
$V_7$	3.4994	3.6774	3.8838	7.1598	8.1078	9.1830
$V_9$	2.3398	2.5132	2.7144	3.7714	4.0364	4.3271
$V_{10}$	1.6891	1.7962	1.9136	5.3827	5.6037	5.8318

**Table A.4.** MAE bootstrap CIs given for the XGB model whose training is described in Section 5.

	Training MAE			Test MAE		
	Low 0.14%	Med 50.00%	High 99.86%	Low 0.14%	Med 50.00%	High 99.86%
$V_1$	1.1843	1.2530	1.3209	5.5687	6.2680	6.8778
$V_2$	4.5828	4.8522	5.0774	6.6710	7.3887	8.2529
$V_3$	0.5827	0.6131	0.6429	3.3106	3.9388	4.5517
$V_4$	1.5849	1.6863	1.7934	4.9216	5.5938	6.0942
$V_5$	1.0781	1.1233	1.1773	6.0968	6.7239	7.5140
$V_6$	1.7144	1.8600	2.0268	6.7174	7.9455	8.9072
$V_7$	1.9282	2.0127	2.1030	5.0456	5.6326	6.1831
$V_9$	1.3578	1.4203	1.4810	2.4520	2.8794	3.2993
$V_{10}$	1.4199	1.5016	1.5938	4.9048	5.2469	5.5879

**Table A.5.** R<sup>2</sup> bootstrap CIs given for the XGB model whose training is described in Section 5.

	Training R <sup>2</sup>			Test R <sup>2</sup>		
	Low 0.14%	Med 50.00%	High 99.86%	Low 0.14%	Med 50.00%	High 99.86%
$V_1$	0.9964	0.9972	0.9977	0.9617	0.9681	0.9739
$V_2$	0.9799	0.9806	0.9813	0.9053	0.9069	0.9084
$V_3$	0.9988	0.9996	0.9996	0.9827	0.9869	0.9905
$V_4$	0.9914	0.9919	0.9922	0.9139	0.9172	0.9197
$V_5$	0.9982	0.9987	0.9991	0.9721	0.9803	0.9838
$V_6$	0.9873	0.9910	0.9940	0.9026	0.9237	0.9284
$V_7$	0.9834	0.9837	0.9977	0.9355	0.9410	0.9452
$V_9$	0.9958	0.9959	0.9994	0.9920	0.9937	0.9952
$V_{10}$	0.9972	0.9974	0.9985	0.9749	0.9828	0.9896

### A.15. Python implementation

Here we provide the Python 3.12 implementation of Algorithms 1 and 2 and an example problem instance. The most up-to-date code should be accessed through the project's GitHub repository: <https://github.com/SamuelWardPhD/Cleaning-Schedule-Optimization>

```

import itertools
import numpy as np
import pandas as pd
from typing import Callable, Sequence

def example_problem_instance():
    # Number of voyages
    n = 5

    # Initial level of biofouling (100 days)
    b0 = 100

    # Cost of cleaning at each port (USD)
    c = np.array([45_000, 44_000, 39_000, 36_000, 42_000])

    # Each voyage increases biofouling level by 25 days
    B = np.array([25, 25, 25, 25, 25])

    # Example DataFrame for each of the five voyages
    X = (
        # Voyage 1
        pd.DataFrame({
            "speed_through_water_kn": [8.5, 9.2, 10.0, 11.3, 12.1, 10.8, 9.7, 11.9, 13.0, 12.4],
            "draught_plus_wave_height_m": [9.8, 10.1, 10.0, 10.4, 10.6, 10.3, 10.2, 10.7, 10.9, 10.8]
        }),
        # Voyage 2
        pd.DataFrame({
            "speed_through_water_kn": [9.0, 9.6, 10.4, 11.0, 12.0, 12.3, 11.1, 12.7, 13.2, 12.8],
            "draught_plus_wave_height_m": [9.7, 9.9, 10.0, 10.1, 10.2, 10.1, 9.8, 10.3, 10.4, 10.2]
        }),
        # Voyage 3
        pd.DataFrame({
            "speed_through_water_kn": [8.2, 8.9, 9.5, 10.1, 10.7, 10.3, 9.6, 10.8, 11.5, 11.0],
            "draught_plus_wave_height_m": [10.0, 10.3, 10.5, 10.7, 10.8, 10.6, 10.4, 10.9, 11.0, 10.8]
        }),
        # Voyage 4
        pd.DataFrame({
            "speed_through_water_kn": [9.5, 10.2, 10.8, 11.6, 12.5, 12.0, 11.3, 12.9, 13.5, 13.0],
            "draught_plus_wave_height_m": [9.6, 9.8, 10.0, 10.1, 10.2, 10.0, 9.9, 10.3, 10.4, 10.2]
        }),
        # Voyage 5
        pd.DataFrame({
            "speed_through_water_kn": [8.8, 9.4, 10.1, 10.9, 11.7, 11.2, 10.5, 11.8, 12.6, 12.1],
            "draught_plus_wave_height_m": [9.9, 10.0, 10.2, 10.3, 10.5, 10.4, 10.1, 10.6, 10.7, 10.5]
        })
    )

    # Regression function to predict voyage cost
    def f(X, b):
        fuel_cost_USD_per_kg = 0.493
        fuel_oil_consumption_kg = sum(
            X['draught_plus_wave_height_m'][i] * (X['speed_through_water_kn'][i]**2) + 36 * b
            for i in range(n)
        )
        return fuel_oil_consumption_kg * fuel_cost_USD_per_kg

    # Return a parameterization that can be passed to Algorithms 1 and 2
    return n, b0, c, X, B, f

def algorithm_1_brute_force_search(
    n: int,
    b0: float,
    c: Sequence,
    X: Sequence,
    B: Sequence,
    f: Callable,
    verbose=3,
):
    """
    :param n: Number of voyages. Should be an integer greater than one;
    :param b0: Initial value of the biofouling measure. Non-negative float;
    :param c: Numpy array where c[i] is the cost of cleaning before voyage i;
    """

```

```

:param B:           Numpy array where B[i] is the measure of biofouling that accumulates for voyage i;
:param X:           Array where X[i] is the profile of voyage i. Each element may be a DataFrame or other object;
:param f:           Function f which outputs the cost given a voyage profile;
:param verbose:    Level of output. Set to zero to suppress all print statements;
"""

# Title
if verbose:
    print("\n===== Brute Force =====")

# The objective function of the integer program
# sum_{i=1,...,n} ( f(X[i], b[i]) + c[i]*z[i] )
def objective(z):
    b = np.zeros(n, dtype=float)
    count = b0
    for i in range(n):
        if z[i] == 1:
            count = 0
        b[i] = count
        count += B[i]

    return sum((
        f(X[i], b[i]) for i in range(n)
    )) + sum((
        c[i]*z[i] for i in range(n)
    ))

# Initialise variables that will store the best solution found so far
z_best = None
obj_best = np.inf

# Iterate through every combination of decision variables
# I.e. (0, 0, ...) then (1, 0, ...) then (0, 1, ...) then (1, 1, ...) and so on
for progress, z in enumerate(itertools.product((0, 1), repeat=n)):
    obj = objective(z)

    # Maintain the best solution so far
    if obj < obj_best:
        obj_best = obj
        z_best = z

    # Update
    if verbose >= 2 and (progress+1) in (((2**n)*p)//100 for p in range(20, 101, 20)):
        print(f" {progress+1}>3} out of {2**n} combinations searched ({(progress+1)/(2**n):.1%})")

# Output results to console
if verbose:
    print(f"Optimal cleaning schedule z*: {z_best}")
    print(f"Optimal object value ($cost): {obj_best:.2f}")

return z_best, obj_best

def algorithm_2_dynamic_cleaning_schedule_optimiser(
    n: int,
    b0: float,
    c: Sequence,
    X: Sequence,
    B: Sequence,
    f: Callable,
    b0_is_B0: bool = False,
    verbose=3,
):
"""
:param n:           Number of voyages. Should be an integer greater than one;
:param b0:          Initial value of the biofouling measure. Non-negative float;
:param c:           Numpy array where c[i] is the cost of cleaning before voyage i;
:param B:           Numpy array where B[i] is the measure of biofouling that accumulates for voyage i;
:param X:           Array where X[i] is the profile of voyage i. Each element may be a DataFrame or other object;
:param f:           Function f which outputs the cost given a voyage profile;
:param b0_is_B0:    Set this to true if B[0] = b0. Otherwise, we will set B = [b0] + B;
:param verbose:    Level of output. Set to zero to suppress all print statements;
"""

# Validation
assert len(c) >= n, f"Invalid parameterization: len(c)={len(c)} should be equal to n={n}."
assert len(B) >= n, f"Invalid parameterization: len(B)={len(B)} should be equal to n={n}."
assert len(X) >= n, f"Invalid parameterization: len(X)={len(X)} should be equal to n={n}."

# Title
if verbose:
    print("\n===== Dynamic Programming =====")

# To understand the dynamic programming algorithm you must consider the following sub-problem

```

```

# CLEANING[i,j] is the sub-problem of solving for the subset of variables ( z_i, ... , z_n ) given that z_j=1
#
# Phi[i,j] is the optimal objective value for CLEANING[i,j]
# Psi[i,j] is the optimal cleaning schedule for CLEANING[i,j]
Phi = np.zeros((n, n))
Psi = [[tuple((0 for __ in range(i + 1))) for _ in range(n - i)] for i in reversed(range(n))]

# We wish B[0]=b_0 to refer to the initial biofouling and B[1] to refer to the first voyage
if not b0_is_B0:
    B = np.concatenate(([np.array([b0]), B[:-1]])

# Vector of zeros
zero = 0 if len(B.shape) == 1 else np.zeros(B.shape[1])

# Work backwards through the voyages
for i in reversed(range(n)):

    if verbose >= 2:
        print(f" --- Voyage {i + 1} ---")

    # The cost assuming we clean at voyage i is the sum of
    #   + The cost of cleaning C[i]
    #   + The fuel cost of the voyage with zero biofouling f(X[i], 0)
    #   + The optimal cost of future voyages' Phi[i + 1, i + 1]
    phi_clean = c[i] + f(X[i], zero) + (0 if i + 1 == n else Phi[i + 1, i + 1])
    z_clean = tuple((1,)) if i + 1 == n else (tuple((1,)) + Psi[i + 1][i + 1])

    # Assume that we last cleaned at before voyage
    for j in reversed(range(i + 1)):

        # The cost assuming we have not cleaned since voyage j
        #   + The fuel cost of the voyage with b biofouling f(X[i], b)
        #   + The optimal cost of future voyages' Phi[i + 1, j]
        b = np.sum(B[j:i + 1], axis=0)
        phi_fouled = f(X[i], b) + (0 if i + 1 == n else Phi[i + 1, j])
        z_fouled = tuple((0,)) if i + 1 == n else (tuple((0,)) + Psi[i + 1][j])

        if verbose >= 2:
            print("".join((
                f"j: {j}<4",
                f"b: {np.sum(B[j:i + 1]):<6}",
                f"phi_clean: {phi_clean:<10.2f}",
                f"phi_fouled: {phi_fouled:<10.2f}",
                f"B[{j}]:{i + 1}: {B[j:i + 1]}"
            )))

        # If it is optimal to clean, record so in tables Phi and Psi
        if phi_clean <= phi_fouled:
            Phi[i][j] = phi_clean
            Psi[i][j] = z_clean

        # If it is optimal to level the biofouling, record so in tables Phi and Psi
        else:
            Phi[i][j] = phi_fouled
            Psi[i][j] = z_fouled

    # Output results to console
    if verbose:
        print(" --- Results ---")
        print(f"Optimal cleaning schedule z*: {Psi[0][0]}")
        print(f"Optimal object value ($cost): {Phi[0][0]:.2f}")

return Psi[0][0], Phi[0][0]

def main():
    # Get the parameters for the example problem instance
    n, b0, c, X, B, f = example_problem_instance()

    # Solve with brute force
    algorithm_1_brute_force_search(n, b0, c, X, B, f)

    # Solve with dynamic programming
    algorithm_2_dynamic_cleaning_schedule_optimiser(n, b0, c, X, B, f)

if __name__ == '__main__':
    main()

__author__ = __maintainer__ = "Samuel Ward"
__email__ = "s.ward@soton.ac.uk"
__credits__ = ["Samuel Ward", "Marah-Lisanne Thormann"]
__version__ = "7.0.2"

```

