Universiteit Leiden
The Netherlands

Computer Science

Multi-Objective Maintenance Optimization

Nicholas Chi-Hung Wu

Supervisor:
Dr. Furong Ye

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

15/07/2025

Abstract

This bachelor thesis investigates different multi-objective optimization methods for maintenance scheduling tasks, aiming to balance maintenance costs, frequency, and system reliability. Specifically, we formulate a multi-objective optimization model that incorporates preventative and reactive maintenance costs, failure probabilities modelled using Remaining Useful Life (RUL) prognostic, and constraints on system downtime, risk levels, and black-out periods.
We tune the evolutionary algorithms' parameters using the Sequential Model-based Algorithms Configuration (SMAC).
Performance evaluations using Hypervolume Convergence, Diversity Convergence and Inverted Generational Distance as the metrics, demonstrate that our proposed model effectively reduces maintenance costs and enhances system availability and reliability compared to traditional preventative maintenance methods.
This research demonstrates practical applications of multi-objective optimization in maintenance planning, guiding optimal operational decisions under financial and reliability constraints.

Contents

1 Introduction

Heerema Marine Contractors [Hee24], headquartered in Leiden, is a world-renowned company with operations of semi-submersible crane vessels and offshore heavy-lift projects. In operations like Heerema's, a single unplanned failure of one of the components can delay or cause significant losses in revenue or even the safety of personnel. Maintenance planning for big operations is notoriously difficult, and observing this first hand at Heerema Marine Contractors inspired this research and guided its central question:

How can we schedule maintenance efficiently while balancing cost, downtime, and risk?

Although this study uses synthetic data for testing, the methodology is deliberately generic so that it can be transferred back to the maritime or any other asset-intensive sector.
Maintenance planning is crucial in industries where the system needs to provide high efficiency. Traditional maintenance methods like reactive- and periodic maintenance often fail to achieve an optimal balance between maintenance costs, downtime, and system reliability.
Reactive maintenance can lead to unexpected failures and costly unplanned downtime, while **periodic maintenance** frequently results in unnecessary interventions and component replacements, resulting in avoidable expenses.
Whereas **Condition-Based Maintenance** (CBM), enhanced by the Remaining Useful Life (RUL) prognostics, has emerged as an more sophisticated approach for addressing these limitations. It uses real-time sensor data to determine the RUL and schedules maintenance to just before the probability of failure exceeds the threshold. This approach aims to optimize operational efficiency by striking an optimal balance between cost, risk, and downtime.
However, effectively implementing **CBM** scheduling involves complex trade-offs due to multiple conflicting objectives. These problems need a multi-objective optimization method to find the optimal trade-offs. Evolutionary Algorithms (EAs) have gained significant attention for solving these complex multi-objective optimization (MOO) problems, because they efficiently explore large and non-linear search spaces and naturally provides a set of Pareto-Optimal solutions. This maintenance scheduling problem explicitly considers practical constraints, including maintenance slot limitations, blackout periods, and maximum allowable system risk levels. To effectively solve this multi-objective maintenance scheduling problem, we compare five evolutionary algorithms: NSGA-II, NSGA-III, U-NSGA-III, SMS-EMOA, and MOEA/D. Each of these algorithms brings distinct strategies in terms of convergence speed, diversity preservation, and solution quality. To assess and compare these algorithms, we use various performance metrics including Hypervolume and Inverted Generational Distance.

This thesis therefore aims to pursue the following research goals:

1. Develop a comprehensive MOO CBM framework for maintenance scheduling, focusing on minimizing maintenance costs, maintenance frequency, and the risk of system failure.

2. Analyse and benchmark the performance of several evolutionary algorithms within that framework.

3. Automate hyperparameter tuning with the Sequential Model-based Algorithm Configuration (SMAC) tool, eliminating manual tuning and ensuring fair comparison across the different algorithms.

4. Deliver a modular codebase that can adapt to sensor data to help practitioners to develop maintenance schedules for their systems.

The remainder of this thesis is structured as follows:
Section 2 provides essential background, detailing fundamentals of maintenance scheduling, reliability modelling, multi-objective optimization concepts, evolutionary algorithms, constraint-handling techniques, and hyperparameter optimization. Section 3 presents the formal problem definition, objectives, and constraints. Section 4 describes the evolutionary algorithms used in detail. Section 4.3 elaborates on constraint-handling methodologies, particularly Deb's feasibility rule used throughout the study. Section 5 outlines the experimental setup, including performance metrics, experimental scenarios, SMAC optimization procedures, and evaluation procedures. Section 6 discusses the results and their implications. Finally, Section 7 summarizes key findings and suggests future research.

2 Background and Related Works

2.1 Maintenance Scheduling

Industrial maintenance strategies fall roughly into three broad categories: reactive maintenance, preventative maintenance, and condition based maintenance. In a **run-to-failure strategy** a component is left in service until it breaks, triggering an emergency repair. Studies such as Bianchi et al. [BPML14] show that, once the probability of failure has risen beyond a modest threshold, the expected downtime and emergency costs already exceed the cost of a scheduled intervention. At the opposite pole, **time-based preventive maintenance** replaces parts on a fixed calendar or usage interval, reducing unexpected outages but sometimes discarding components that still have remaining useful life. Sitting between these extremes, **condition-based maintenance** strategy, that continuously monitors health sensors and schedules maintenance only when the specific sensor threshold is met.

Reactive maintenance In reactive maintenance a component is allowed to operate until it fails, after which an emergency replacement is carried out. Although simple, this approach could result in unexpected downtime of a system, emergency repairs and/or additional costs.

Time-based preventive maintenance Time-based (periodic) preventive-maintenance schedules trigger work after a fixed calendar interval or usage count, for example every 5.000 flight-hours. They successfully reduce failures, but frequently replace components that still have a substantial useful life left over, resulting in inflated material cost. Their main attraction is the administrative simplicity: labour, parts and production downtime can be scheduled months in advance.
Economic studies dating back to Barlow and Proschan's age-replacement model [BH60] show that, when failure costs are moderate and reliability curves are well characterised, a carefully chosen interval can minimise the long-run cost per operating hour. In conclusion the downside of time-based maintenance is **over-maintenance**. Because the interval must protect the **average** component, slow-degrading items are often removed while still serviceable. The cross-industry evidence from the study from, Lee *et al.* [LND+06] reinforces this point. It shows that 30–50 % of calendar-driven removals reveal no defect, and argues that predictive health-management tools could cut total maintenance cost by 15–30 % while halving downtime . These inefficiencies have triggered the shift towards condition-based maintenance, where maintenance is triggered by measured degradation rather than periodically.

Condition-based Maintenance Finally with condition-based maintenance, **CBM** a component's health is monitored using sensors.
The Remaining Useful Life (RUL) prognostic is calculated, it forecasts the time-to-failure. Components are replaced as soon as their predicted failure probability exceeds an unacceptable threshold. CBM aims to intervene `just in time`, balancing the extra cost of early replacement against the risk and consequence of in-service failure. Modern high-availability industries increasingly pair condition-monitoring sensors with RUL prognostics, so that components can be replaced just before the risk of failure becomes unacceptable [LPLJ20]. This thesis therefore models only two triggers a RUL-based preventive action and a residual reactive replacement after failure, replacing the classic calendar intervals.

2.1.1 Reliability Modelling

Reliability modelling is essential for efficient maintenance scheduling, as it predicts a component's time of failure, thus help determine the optimal time for maintenance. Two of those approaches are implemented in this research to model the failure probabilities of components and are discussed in detail in Section 3.1.

Linear degradation As a baseline we, we assume the failure probability of a component rises in direct proportion to its elapsed age. This provides a straightforward approximation that facilitates maintenance scheduling. While this model works well for the task at hand, it assumes a constant degradation rate and does not capture the more complex, non-linear behaviours observed in real-world systems.

Weibull failure modelling To capture more complex characteristics of a components' failure probability we utilize the Weibull distribution, a statistical tool commonly applied in reliability engineering [SL20]. The Weibull function can capture various failure behaviours, such as components whose failure rates increase over time (indicating wear-out) or decrease (indicating early-life failures). The flexibility of the Weibull distribution makes it suitable for modelling specific behaviours of components and calculating the Remaining Useful Life (RUL) of components.

2.2 Multi-Objective Optimization

Multi-objective optimization (**MOO**) involves solving problems that require optimizing two or more conflicting objectives simultaneously. Unlike single-objective optimization, which seeks a single best solution, MOO generates a **Pareto front**, a set of optimal solutions where no objective can be improved without compromising another (discussed in further detail in Section 5.1). These solutions are called Pareto optimal, and they allow decision-makers to choose trade-offs that best align with their priorities, constraints, and/or resources. This is especially useful in complex scenarios like maintenance scheduling, where objectives such as cost, reliability, and downtime often conflict. The decision maker might prioritize cost reduction over system reliability or vice versa, depending on the operational context.
The performance of MOO algorithms is evaluated using various metrics and performance metrics. These help assess how well an algorithm approximates the true Pareto front, as well as the diversity and distribution of the solutions. According to Riquelme et al. (2015) [RVLB15], there are three key aspects that these metrics evaluate:

- Convergence: Measures how close the solutions are to the true Pareto front. The more convergent the solutions are, the better they approximate the true optimal solutions. This is typically assessed by calculating the distance between the Pareto front and the theoretical optimal front.
- Diversity (Spread): Assesses how well the solutions are distributed along the Pareto front. A good distribution of solutions ensures that the Pareto front is fully explored, and that the

decision-maker has a wide range of trade-offs to choose from. Spread measures how well the solutions cover the objective space.

- Number of Solutions: This metric refers to how many non-dominated solutions the algorithm can find. More solutions generally mean a better representation of the trade-offs between the objectives.

These metrics focus on the fundamental aspects of the solution set itself, such as its quality, diversity, and size. In addition to the basic metrics, performance metrics are used to compare different algorithms' ability to approximate the Pareto front. Performance metrics such as **Hypervolume** and **Inverted Generational Distance** (IGD) are among the most widely used [RVLB15]:

- **Hypervolume (HV)**: This performance metric measures the volume dominated by the Pareto front relative to a reference point in the objective space. The larger the hypervolume, the better the approximation of the true Pareto front. Hypervolume is considered a composite measure because it incorporates both convergence and spread, making it a powerful indicator of solution quality. The hypervolume calculation is done using the WFG algorithm introduced by While, Bradstreet, and Hingston [WBB11]. In essence, WFG orders the non-dominated points and, one by one, adds their axis-aligned hyper-rectangles while recursively subtracting any overlap, so only the new volume is counted.

- **Inverted Generational Distance (IGD)**: IGD measures the distance between the obtained solutions and the true Pareto front. It first samples the (approximated) true front into a reference set, then, for every reference point, computes the Euclidean distance to the *closest* solution in the approximation set and finally averages those distances.
  A lower IGD indicates better convergence to, and coverage of, the true Pareto front because large gaps or outliers raise the mean distance. This dual sensitivity to proximity and spread makes IGD a convenient quality indicator whenever an accurate reference front is available. The exact computation procedure is described in detail by Wang et al. [WXZ24].

By using these performance metrics, we can quantitatively compare the quality of solutions from different MOO algorithms, focusing on both convergence and diversity. These metrics are especially important when comparing Pareto fronts obtained from different algorithms, as they offer insights into how well each algorithm approximates the true front and how diverse the solutions are.

2.2.1 Evolutionary Algorithms for Multi-Objective Problems

Evolutionary Algorithms (EAs) are optimization techniques inspired by the foundation of natural evolution, using a population of candidate solutions that evolve through selection, crossover, and mutation. This population-based approach is particularly effective for addressing multi-objective optimization (MOO) problems, where multiple conflicting objectives must be optimized simultaneously.
EAs produces a set of solutions known as a Pareto Front, it provides decision-makers with multiple optimal trade-offs between objectives, enabling the selection of solutions that align with specific preferences or constraints. This characteristic to handle multiple objectives simultaneously combined with its capability to handle complex objective functions, makes EAs suitable for realistic MOO problems like maintenance scheduling.

Several established evolutionary algorithms exist for multi-objective optimization, such as NSGA-II, NSGA-III, U-NSGA-III, SMS-EMOA, and MOEA/D [ZQL+11]. These algorithms differ primarily in their selection procedures, diversity-maintenance mechanisms, convergence strategies, and overall computational efficiency. In Section 4, each algorithm is described individually, providing detailed insights into their workings and suitability for addressing our specific maintenance scheduling problem.

2.2.2 Constraint Handling

Real-world multi-objective problems always comes with specific constraints and to handle these constraints we need a method to address these violating solutions. Most evolutionary strategies fall into one of four categories, originally surveyed by Michalewicz and Schoenauer [MS96].

1. **Preserving feasibility of solutions**
   Representation, crossover, and mutation are engineered so that every offspring automatically satisfies all constraints.

2. **Penalty functions**
   Infeasible individuals are kept in the population but their fitness is downgraded by a static, dynamic, or adaptive penalty that grows with the magnitude of the violation. Penalties that are carefully selected gradually pushes the search towards feasibility.

3. **Feasible vs. infeasible ranking rules**
   Selection gives unconditional priority to feasible solutions. Infeasible candidates compete only with one another, typically via the summed violation.

4. **Hybrid methods** Combine elements of the previous three strategies, for example repairing an offspring to the nearest feasible point, applying a modest penalty to any residual violation, and still ranking feasible individuals ahead of infeasible ones.

Among the four categories, we use a *hybrid approach* that combines different constraint handling strategies depending on the algorithms' characteristics:

- For the **constraint-based algorithms** (NSGA-II, NSGA-III, U-NSGA-III, SMS-EMOA), we use the *feasible vs. infeasible ranking* approach with Deb's feasibility rule [Deb00] as provided by the `pymoo` framework [BD20].

- For **MOEA/D**, which does not natively support explicit constraint handling in its decomposition framework. To circumvent this a *hybrid method* combining penalty functions with repair operators is used. Specifically, we: (1) convert all constraints into penalty terms integrated directly into the objective functions, and (2) employ a diversity-preserving repair operator that ensures offspring feasibility while maintaining population diversity.

The details of the implementation are given in Section 4.3.

2.3 Hyperparameter Optimization with SMAC

The MOEAs in this study rely on tunable parameters, like population size, crossover- and mutation probabilities, and the simulated binary crossover- or polynomial mutation distribution indices. These hyperparameters govern the balance between exploration and exploitation, therefore determine how quickly and how well the population converges towards a diverse and dominant Pareto front. Rather than selecting the parameters by hand, the Sequential Model-based Algorithm Configuration (SMAC) method can be used to automate the algorithm tuning. SMAC is a Bayesian optimiser [LEF+22] that has become widely used in the field for automated algorithm tuning.

SMAC views the algorithm as a noisy black-box function of its hyperparameters. It repeatedly *(i)* evaluates a promising configuration, *(ii)* fits a random-forest surrogate to model performance over the untested space, and *(iii)* proposes the next configuration by maximising an objective function, such as expected improvement.

This surrogate-guided search quickly focuses on high-performing regions, while avoiding obviously poor performing parameter settings. Rook et al. [RTBG22] showed that letting SMAC pick the parameters of algorithms, such as NSGA-II yields noticeably better results than default settings on hard, multi-modal multi-objective benchmarks.

The concrete search ranges, budgets, and objectives used for tuning on this problem are described in further detail in Section 5.2.

3 Problem Modelling

In this thesis a realistic maintenance scheduling problem is addressed, involving multiple practical constraints and conflicting objectives. Organizations operating complex systems must carefully plan component maintenance to balance costs, resource utilization, and system reliability. We focus on a finite set of components, each characterized by unique properties that influence its maintenance requirements.
Over a planning horizon(PH) of discrete time points (t), we have a set of components, each with distinct installation dates, d_a^{install} and predicted Remaining Useful Life (**RUL**), $\hat{t}_a$. The RUL prognostics are randomly generated, but can be tuned to real-world characteristics of components, allowing us to estimate the probability of failure efficiently.

3.1 Component failure

Real-world components does not fail at a constant pace, factors like **wear** that accumulates with age and load, also the occasional **sudden failure** of a part with no logical reason play an important role. To study how this affects maintenance decisions we model failure probability in two ways.

1. A simple linear ageing model that serves as a baseline.
2. A failure rate based on the Weibull function.

To enhance realism, the implementation assigns each component a small, age-independent sudden-failure rate, capturing unforeseeable breakdowns that trigger immediate reactive maintenance

3.1.1 Linear Ageing Reliability Model

We adopt a linear ageing reliability model introduced by Vesely and Wolford [VW88], in which the failure probability increases proportionally with a component's elapsed age.
Specifically, the failure probability for component, a at time, t linearly increases as the component ages. The linear failure probability of a component at time t, $p_a(t)$ is calculated using:

$$p_a(t) = 1 - \frac{\max\{\hat{t}_a - (t - d_a^{\text{install}}),\, 0\}}{\hat{t}_a}, \qquad p_a(t) \in [0, 1]. \tag{1}$$

Here $\hat{t}_a - (t - d_a^{\text{install}})$ is the component's remaining useful life (RUL) at time t; dividing by the initial RUL $\hat{t}_a$ normalises the term, yielding a valid probability that rises linearly from 0 (at installation) to 1 (when the RUL clock has run out).

3.1.2 Weibull ageing

To capture wear-out that may accelerate or decelerate with age we use the two-parameter Weibull cumulative distribution, introduced in section 2.1.1.

$$p_a^{\text{wb}}(t) = 1 - \exp\Big[-\big(\tfrac{t - d_a^{\text{install}}}{\eta_a}\big)^{\beta_a}\Big], \qquad \beta_a, \eta_a > 0.$$

Every component is assigned a unique shape–scale pair, ensuring the system has a realistic variability in failure behaviour.

- **Shape** β_a (ageing mode) is sampled uniformly from 1.2 to 3.5, spanning mild ($\beta \approx 1$) through to severe ($\beta > 3$) wear-out.
- **Scale** η_a (characteristic life) is drawn uniformly from 0.6 to 1.4, in this case lifetimes about 40 % shorter up to 40 % longer than nominal.

3.1.3 Sudden failures

To emulate unforeseen breakdowns each component carries a small, age-independent *shock probability of component* a, $\rho_a \in [0.005, 0.02]$ per time step (0.5 %–2 %). With probability ρ_a the item fails instantly and immediate reactive maintenance is required. If the probability is not reached it continues to age along its original curve (linear or Weibull).

3.2 Decision Variables, Costs, and Maintenance Slots

Let A denote the set of all components, where $a \in A$. Each component a incurs a fixed preventive–maintenance cost c_a^{fix} when it is serviced on schedule, and a higher penalty c_a^{ex} if it fails unexpectedly before maintenance is performed.
Maintenance actions may only take place in predefined *maintenance slots*. Let $\mathcal{S}$ be the set of all slots; each slot is tied to a specific time-step, $d_s \in \{d_0, \ldots, d_0 + PH - 1\}$, and can handle at most κ_s components, see equation 6. For every component–slot pair we introduce the binary decision variable, $x_{a,s}$ where component a is maintained in slot s.

$$x_{a,s} = \begin{cases} 1, & \text{if component } a \text{ is maintained in slot } s, \\ 0, & \text{no maintenance.} \end{cases}$$

The collection of variables $\{x_{a,s}\}$ across $a \in \mathcal{A}$ and $s \in \mathcal{S}$ constitutes the maintenance schedule to be optimised over the planning horizon, PH.

3.3 Objectives

In a single-objective setting one could define the best schedule as the one that minimises or maximizes a single objective function. However in our maintenance-scheduling model we confront four conflicting goals. We seek the entire Pareto front of non-dominated solutions, each of which represents a different trade-off between cost, workshop usage and reliability. This Pareto set allows decision-makers to select the schedule that best matches their priorities.

- For the first objective, we aim to minimize the total expected maintenance expenditure, following the cost structure introduced in the paper of de Pater and Mitici [dPM21]. This involves accounting for both the planned maintenance costs and the potential additional costs, if components fail before it gets maintained.

$$f_1(x) = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} x_{a,s} \Big(c_a^{\text{fix}} + \big(1 - p_a(d_s)\big)\, c_a^{\text{ex}} \Big) + \sum_{a \in \mathcal{A}} \Big(1 - \sum_{s \in \mathcal{S}} x_{a,s} \Big)\, p_a(d_0 + \text{PH})\, c_a^{\text{ex}} \qquad (2)$$

In Equation (2), the expected maintenance expenditure $f_1(x)$ is expressed as the sum of two distinct contributions. The first part,

$$\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} x_{a,s}\Big(c_a^{\text{fix}} + \big(1 - p_a(d_s)\big)\, c_a^{\text{ex}}\Big),$$

accounts for all components that are deliberately scheduled for replacement or repair. Whenever $x_{a,s} = 1$, component a is maintained in slot s. In that case, we incur the fixed replacement cost c_a^{fix}, and we also pay an expected penalty $(1 - p_a(d_s))\, c_a^{\text{ex}}$.
Here, $p_a(d_s)$ is the probability that a would have already failed by time d_s, so $(1-p_a(d_s))$ is the probability that it survives until the planned maintenance date. Multiplying by c_a^{ex} converts this survival probability into the expected extra cost saved by replacing before a failure occurs.

The second part,

$$\sum_{a\in\mathcal{A}}\Big(1 - \sum_{s\in\mathcal{S}} x_{a,s}\Big)\, p_a(d_0 + \text{PH})\, c_a^{\text{ex}},$$

captures the expected penalty for any component that is never scheduled. For each such component a, the factor $\big(1-\sum_s x_{a,s}\big)$ equals one exactly when no maintenance slot is assigned. In that case, the component faces a failure probability $p_a(d_0 + \text{PH})$ over the full planning horizon, and should it indeed fail, the extra cost c_a^{ex} is incurred. Thus this term represents the expected cost of in-service failures for all unscheduled components.
By summing these two parts, $f_1(x)$ gives the total expected cost combining both the planned maintenance actions and the penalties for any components left unserviced.

- The second objective is about minimizing the frequency of performed maintenance, thereby reducing the downtime of the production. The formula is simply defined as the number of slots used for maintenance across the whole planning horizon.

$$f_2(x) = \sum_{s\in\mathcal{S}} \min\Big\{1, \sum_{a\in\mathcal{A}} x_{a,s}\Big\} \tag{3}$$

- The third objectives maximizes the system reliability by minimizing the sum of all failure probabilities of the components.

$$f_3(x) = \max_{t\in\text{PH}} \frac{1}{|\mathcal{A}|} \sum_{a\in\mathcal{A}} p_a(t) \tag{4}$$

  - **Inner sum (average risk).** At every time step, t we compute the mean failure probability of all components, $\frac{1}{|\mathcal{A}|}\sum_a p_a(t)$. The division by $|\mathcal{A}|$ normalises the metric to $[0, 1]$.
  - **Outer max (worst-case).** The maximum scans all time-steps and picks the single time-step with the highest average failure probability. By *minimising* this worst-case value, the optimisation drives every schedule toward a lower peak risk, yielding a more reliable system over the entire planning horizon.

- Finally, the fourth objective minimizes the number of reactive maintenance events throughout the planning horizon. Because such interventions disrupt production drastically, we include this objective that drives their count down. The reactive maintenance count is determined by simulating the entire maintenance schedule over the planning horizon. For each time step, the algorithm checks whether any component fails using the Weibull failure model, which includes both ageing-based failures (when RUL reaches zero) and sudden failures (stochastic events with component-specific probabilities). When a component fails, it triggers immediate reactive maintenance, and the count is incremented. This simulation-based approach captures the stochastic nature of equipment failures, making the objective more realistic than purely probabilistic calculations.
  While this objective exhibits some overlap with both the cost objective and the reliability objective, it provides a distinct perspective on operational stability. This objective stimulates schedules that maintain system reliability while avoiding unexpected breakdowns that can halt production.

$$f_4(x) = \sum_{t=0}^{\mathrm{PH}} \sum_{a \in \mathcal{A}} failed_a \tag{5}$$

where $failed_a$ indicates whether component a fails at time t during the simulation of the schedule from $t = 0$ to $t = PH$. This count is determined by running the complete maintenance schedule simulation, where component failures are governed by the Weibull failure model.

3.4 Constraints

For this problem we also implement some practical constraints regarding the scheduling of the maintenance.
The first constraint (6) enforces that no slot s is assigned more components than it can handle; here κ_s denotes the maximum number of components that workshop resources allow in slot s.

$$\sum_{a \in \mathcal{A}} x_{a,s} \;\le\; \kappa_s \quad \forall\, s \in \mathcal{S} \tag{6}$$

The time-step slot capacity constraint, equation (7), guarantees that for any discrete time t, the total number of maintenance assignments in slots happening at t does not exceed the number of slots available at that moment.
In other words, if there are $|\{s : d_s = t\}|$ slots at time t, we cannot exceed that number of simultaneous maintenance.

$$\sum_{\substack{s \in \mathcal{S} \\ d_s = t}} \sum_{a \in \mathcal{A}} x_{a,s} \;\le\; \big|\{\, s \colon d_s = t\}\big| \quad \forall\, t \in \{d_0, \ldots, d_0 + \mathrm{PH}\} \tag{7}$$

Equation (8) imposes a ceiling $R_{\max}$ on the worst–case average failure probability across all components. By bounding the maximum over t of the mean risk $\frac{1}{|\mathcal{A}|} \sum_a p_a(t)$, we ensure system reliability never falls below the predefined threshold.

$$\max_{t \in [d_0,\, d_0 + \mathrm{PH}]} \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} p_a(t) \;\leq\; R_{\max} \tag{8}$$

Finally, Equation (9) encodes blackout periods $\mathcal{B}$, during which no components may be scheduled in any slot s whose date d_s lies within these forbidden intervals. This models operational restrictions such as holidays or system-wide outages when maintenance must be suspended.

$$x_{a,s} = 0 \quad \forall\, a \in \mathcal{A},\ \forall\, s \text{ with } d_s \in \mathcal{B} \tag{9}$$

4 Algorithms

All modern MOEAs share the same high-level mechanics inherited from the genetic algorithms: they evolve a population of candidate solutions through iterative cycles of selection, variation and survival. However, they differ in how these steps are implemented and the heuristics they use to balance convergence with diversity.
We first describe the general MOEA workflow shared by all methods in figure 1, then highlight their unique differences (Table 1), followed by brief subsections discussing each algorithm individually.

4.1 Common MOEA Pipeline

Figure 1 shows the canonical generation loop shared by all multi-objective evolutionary algorithms (MOEAs) considered in this work.

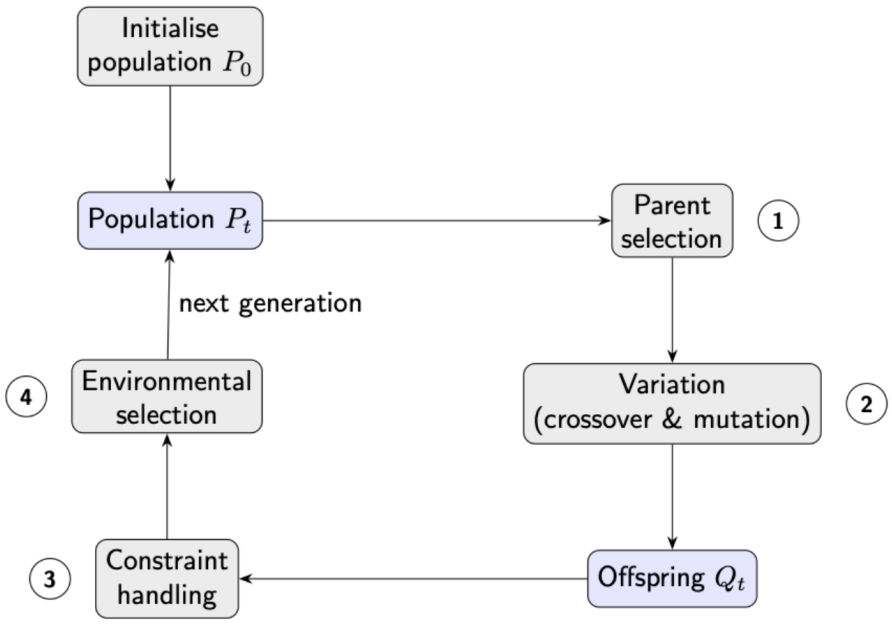


Figure 1: Generic generation loop executed by all MOEAs

At every generation `t`:

1. **Parent Selection** chooses a mating pool from the current population, P_t.
2. **Variation** applies crossover and mutation to create an offspring set, Q_t.
3. **Constraint Handling** applies Deb's feasibility rule (see Section 4.3 for details).
4. **Environmental selection** merges $P_t \cup Q_t$ and picks the next generation, P_{t+1} using each algorithms' specific survival strategy

The algorithms differ from each other in the first and fourth step. This separation lets us compare the different characteristics each modification has on the performance.

4.2 Algorithm Comparison & Algorithm Selection

In Table 1 the unique properties of the different algorithms are compared. Throughout this subsection we use N for the *population size*, K for the *number of reference directions* used by NSGA-III / U-NSGA-III,k for the *neighbourhood size* in MOEA/D and M for the number of objectives.

Algorithm	**Parent Selection**	**Survival / Diversity**	**Main hyper-parameters**	**Cost**
NSGA-II	Binary tournament on Pareto rank and crowding distance	Non-dominated sorting; crowding-distance truncation	N	$O(MN^2)$
NSGA-III	Same as NSGA-II	Reference-direction niching (fills empty niches first)	N, K	$O(MN^2)$
U-NSGA-III	Same as NSGA-II	Elitist for $M = 1$; Crowding for $M = 2$; Reference dir. for $M \geq 3$	N, K	$O(MN^2)$
SMS-EMOA	Binary tournament on hypervolume contribution	Steady-state; deletes worst ΔHV	N (implicit)	$O(MN \log N)$
MOEA/D	Tournament restricted to k neighbours	Decomposition; replacement limited to neighbourhood	N, k	$O(MNk)$

Table 1: Key characteristics of the MOEAs.

- **Selection stages.** NSGA variants rely on Pareto ranking for both parent choice and survival, differing only in how they maintain diversity (crowding vs. reference directions). SMS-EMOA skips ranking altogether and maximises the hypervolume indicator directly, while MOEA/D decomposes the problem and operates within local neighbourhoods.
- **Diversity heuristics.** Crowding distance (NSGA-II) can degrade in many-objective spaces, which motivates the reference-direction schemes of NSGA-III/U-NSGA-III. Hypervolume (SMS-EMOA) and decomposition (MOEA/D) embed diversity in their fitness definitions, avoiding an explicit niching operator.
- **Tuning effort.** All algorithms share the global parameters *population size* and the usual crossover/mutation settings. NSGA-III/U-NSGA-III additionally require the number of reference directions K, and MOEA/D needs the neighbourhood size k. SMS-EMOA has no extra knobs but becomes computationally expensive as M grows because of hypervolume calculations.

- **Complexity.** The NSGA family incurs $O(MN^2)$ cost from full non-dominated sorting; SMS-EMOA trades some of that for a $\log N$ factor thanks to its steady-state update; MOEA/D scales linearly with k, making it attractive for large populations and many objectives when a small neighbourhood suffices.

The two steps that vary between the algorithms examined in this thesis are: *parent selection* and *environmental selection*, described in the next sections.

4.2.1 NSGA-II

The Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [DPAM02] distinguishes itself by incorporating two specific features:
Fast non-dominated sorting partitions the population into Pareto fronts. The first front, F_1, contains solutions not dominated by any others, meaning that no other solution in the population is better in all objectives and strictly better in at least one. Each subsequent front $F_2, F_3, \ldots$ contains solutions dominated only by individuals from all previous fronts. This sorting mechanism efficiently organizes solutions by dominance into distinct levels.
The **crowding-distance** quantifies how isolated an individual solution is within each Pareto front. It is computed by sorting individuals within a front based on each objective. Boundary solutions are assigned infinite distances to ensure preservation, because we want to preserve a diverse front. Interior solutions are assigned a distance based on the normalized difference between their immediate neighbours in each objective. The total crowding distance is obtained by summing these contributions across all objectives, such that individuals in less crowded regions receive higher distance values and are thus more likely to be retained. NSGA-II is recommended for small– to medium-scale instances with up to three objectives, where fast convergence and a well-spread front are required at moderate computational cost.

Parent Selection NSGA-II selects parents using a binary tournament strategy. For each parent, two candidates are randomly drawn from the current population without replacement and compared first by rank, then by crowding distance.
The comparison is based first on their Pareto rank. A lower rank means the solution is less dominated and therefore preferred. If both candidates have the same rank, the one with the higher *crowding distance*, the more isolated individual in the objective space is selected. This procedure promotes both convergence and diversity, and is reused in NSGA-III and U-NSGA-III.

Environmental Selection After non-dominated sorting, the fronts are added to the next generation one by one in order of increasing Pareto rank. This process continues until adding the next front would exceed the population size limit N. If the last admissible front does not fit entirely, it is truncated by selecting the individuals with the largest crowding distances, ensuring that boundary solutions and those in less crowded regions are preserved.

4.2.2 NSGA-III

NSGA-III [DJ13] extends NSGA-II to many-objective problems ($M \geq 4$) by replacing crowding distance with a reference-direction niching scheme. It first generates an evenly spaced set of reference

directions, weight vectors with non-negative components that sum to one. Reference directions serve as fixed "anchors" for diversity. After the objectives are normalised, every solution is linked to the reference direction to which it is closest. NSGA-III should be preferred once the problem involves four or more objectives and an even coverage of the Pareto front is critical.

Parent Selection Identical to NSGA-II: binary tournaments based on Pareto rank, and crowding distance (see Section 4.2.1).

Environmental Selection After non-dominated sorting, individuals in the last admissible front are associated with their nearest reference direction (perpendicular distance). The algorithm then fills under-represented directions first, keeping at most one solution per direction. When several solutions map to the same direction, the one closest to that direction is retained. This procedure yields a uniformly distributed set of solutions across the whole Pareto front.

4.2.3 U-NSGA-III (Unified NSGA-III)

U-NSGA-III [SD15] generalizes NSGA-III so a *single* code base can solve single-, two-, and many-objective problems without switching algorithms. The key idea is to **unify** NSGA-II and NSGA-III under one framework by adapting the survival strategy to the number of objectives M:

- $M = 1$: the algorithm reduces to a simple elitist evolutionary strategy—after generating a batch of offspring, it ranks all candidates by their single objective value and keeps the best N for the next generation.
- $M = 2$: falls back to the crowding-distance truncation of NSGA-II, ensuring the classic two-objective behaviour.
- $M \geq 3$: activates the reference-direction niching of standard NSGA-III, preserving diversity in higher-dimensional fronts.

Thus, U-NSGA-III offers a versatile "one-size-fits-all" solution when the number of objectives varies between scenarios. Eliminating the need to manually swich from algorithms.

Parent Selection Identical to NSGA-II: binary tournaments based on Pareto rank, and crowding distance (see Section 4.2.1).

Environmental Selection After non-dominated sorting, U-NSGA-III chooses one of the three survival modes above according to M. For $M \geq 3$, it uses the same reference-direction filling and pruning described in Section 4.2.2; for $M = 2$ it switches to crowding-distance truncation; for $M = 1$ it keeps only the top-ranked individual(s). This adaptive mechanism lets the algorithm "unify" the behaviour of NSGA-II and NSGA-III without manual intervention.

4.2.4 SMS-EMOA

The S–Metric Selection Evolutionary Multi-objective Algorithm (SMS-EMOA [BNE07]) is an **indicator-based** method that evaluates each solution by its *hypervolume contribution* (ΔHV) is the portion of objective space that would disappear if that solution were removed. The calculation is explained in section 5.1. SMS-EMOA keeps the population size fixed at N and works in *steady-state*: each iteration produces one offspring and removes one individual, so N remains constant. SMS-EMOA is an ideal candidate when the goal is to obtain a compact but highly diverse set of trade-off solutions and when sufficient CPU resources is available for repeated hypervolume calculations.

Parent Selection Parents are chosen by binary tournaments: two candidates are sampled and the one with the larger ΔHV wins, because removing it would cause the greater loss in total hypervolume.

Environmental Selection After the offspring is added, ΔHV is recomputed for all N+1 solutions, and the individual with the *smallest* contribution is deleted. Because hypervolume rewards both convergence and spread, this single rule simultaneously drives the population toward the Pareto front and maintains diversity.

4.2.5 MOEA/D

The Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) [ZL07] tackles a multi-objective problem by decomposing it into N *scalar* sub-problems, one for each weight vector λ^i, whose non-negative components sum to one.
By default, each sub-problem is evaluated with the *weighted Tchebycheff* metric, which looks at the objective values relative to the current ideal point and takes the (largest) weighted deviation among them. Minimising this single value drives the solution toward the ideal point along the direction emphasised by λ^i. MOEA/D initialises one solution for every weight vector, so the population size equals the number of sub-problems. MOEA/D becomes advantageous for very large populations or many-objective cases where linear scalability and memory efficiency outweigh the risk of losing some global diversity.

Parent Selection For every sub-problem i, two parent solutions are sampled from the k neighbouring sub-problems whose weight vectors are closest to λ^i in Euclidean distance. This local mating concentrates search effort in adjacent regions of the objective space while still allowing information to flow through overlapping neighbourhoods.

Environmental Selection The offspring produced from these parents is evaluated under the scalarizing functions of each of the same k neighbours. If the offspring yields a lower scalar value than a neighbour's incumbent solution, it *replaces* that solution.
Because updates are strictly local, MOEA/D avoids the $O(MN^2)$ global non-dominated sorting overhead of NSGA-style algorithms and scales efficiently to many objectives and large populations. Over successive generations the population $\{x^{(1)}, \ldots, x^{(N)}\}$, evolves along its assigned weight directions Here, each $\mathbf{x}^{(i)}$ is the current decision-vector solution linked with the weight vector

λ^i Together these solutions approximate the Pareto front, while the weight-vector grid preserves diversity

4.3 Constraint Handling

All the evolutionary algorithms use **Deb's feasibility rule** [Deb00], the default constraint handler in `pymoo`. However, the MOEA/D algorithm requires a special constraint handling implementation (described in section 4.3.1), because it does not handle explicit constraints natively due to its decomposition-based approach.

Deb's feasibility rule is entirely rank-based, needs no penalty weights, and therefore works identically for every other MOEA in this study. The rule was sufficient to drive the search to fully feasible schedules and create a diverse Pareto front.
Deb's rule imposes a simple hierarchy and it proceeds in three steps:

1. **Feasibility priority.** Any feasible solution (one that satisfies all hard constraints) is always preferred over any infeasible solution, regardless of its objective values.
2. **Objective comparison within feasibility.** When comparing two feasible solutions, standard multi-objective selection operators (e.g. Pareto ranking, crowding-distance) decide which solution is better based solely on the objective vector.
3. **Violation comparison within infeasibility.** If both solutions are infeasible, the one with the smaller total constraint violation is deemed better. The total violation of a solution is computed as the sum of the magnitudes by which it exceeds each constraint bound.

Population update in our pipeline After parents and offspring have been merged, the algorithm has $2N$ candidates. These $2N$ are ordered with Deb's rule as described above, and the first N positions in that ordering become the next generation P_{t+1} (Step 3 in Fig. 1). In early generations, a few infeasible schedules may survive, but they are progressively displaced as more feasible, high-quality schedules appear.
A solution is only feasible in this problem setting when it agrees with the following constraints:

- **Slot capacity**, at most one component per maintenance slot.
- **Time-step capacity**, the number of components scheduled at any time t may not exceed the number of available slots.
- **Risk bound**, system risk must stay below the predefined threshold.
- **Black-out periods**, no maintenance inside predefined blackout windows.

4.3.1 Constraint handling MOEA/D

For MOEA/D, we implement a penalty-based constraint handling approach combined with a specialized repair operator. Instead of explicit constraints like the other algorithms, constraint violations are converted to penalty terms and added directly to the cost objective. Each constraint

violation is calculated as the degree by which a solution exceeds the constraint bound, and then multiplied by a penalty coefficient before being added to the total maintenance cost. This would mainly work as a deterrent for the algorithm to use this solution, because if we only use this strategy this would skew the balance between the objectives. Hence the penalty coefficient is chosen such that the constraint violating solutions would never be chosen. The constraint violation penalties are calculated as follows and added to the cost objective:

- **Slot capacity violations**: For each maintenance slot that contains more than one component, the penalty equals the number of excess components times the penalty coefficient.
- **Time-step capacity violations**: For each time step where more components are scheduled than available slots, the penalty equals the number of excess components times the penalty coefficient.
- **Risk bound violations**: When the maximum system risk exceeds the set threshold, the penalty equals the excess risk amount times the penalty coefficient
- **Blackout period violations**: For each component scheduled during blackout windows, a penalty is added.

All these penalties are added to the cost objective, while the other objectives (frequency, risk, and reactive count) retain their original values. This approach allows MOEA/D to see both the true objective values and the constraint violation costs, enabling informed trade-offs during the optimization process.
This penalty-based approach integrates seamlessly with MOEA/D's scalarization process while strongly discouraging infeasible solutions through the high penalty coefficient.
Additionally, MOEA/D uses a **decomposition specific repair strategy** that aligns constraint repair with the algorithm's reference directions. The repair operator analyzes each reference direction to determine which objective it emphasizes most, then applies the corresponding repair strategy:

- **Cost-based repair**: Prioritizes components with highest maintenance costs when the reference direction emphasizes cost minimization.
- **Risk-based repair**: Prioritizes components with highest failure risk when focusing on risk minimization .
- **Time-based repair**: Optimizes temporal distribution when emphasizing frequency minimization.
- **Priority-based repair**: Uses systematic component ordering when targeting reactive maintenance reduction.

This strategy ensures that constraint handling supports rather than conflicts with each sub-problem's optimization direction, improving diversity and solution quality.
This two-stage approach ensures that while the penalty mechanism provides strong selective pressure against constraint violations, the repair operator maintains solution feasibility without permanently distorting the objective balance. The high penalty value is chosen specifically to be prohibitive, preventing infeasible solutions to survive the selection, but the repair operator ensures these penalties are rarely applied to the final population.

5 Experimental Setup

All code, data-generation scripts, and experiment framework are available in the GitHub repository [Wu25] [1]. In this section, we describe how we evaluate our proposed multi-objective maintenance scheduling model and compare the performance of the evolutionary algorithms we selected.
First, we formally define Pareto optimality, clarify the performance metrics we use, explain our experimental scenarios, outline how we optimize the algorithm parameters using the Sequential Model-Based Algorithm Configuration (SMAC), and specify the software and hardware involved in the experiments.

5.1 Pareto Optimality and Performance Metrics

In multi-objective optimization, it is typically impossible to optimize all objectives simultaneously due to their conflicting nature. Therefore, the goal is to identify solutions for which no objective can be improved without sacrificing performance in at least one other. These solutions are called Pareto optimal.
Formally, let $\mathbf{f}(x) = \big(f_1(x), \ldots, f_m(x)\big)$ denote the m-objective vector associated with a feasible schedule x. A schedule $x^\star$ is Pareto optimal (as formulated in [BG15]), if there does not exist another feasible schedule x satisfying:

$$f_i(x) \leq f_i(x^\star) \;\; \forall\, i \in \{1, \ldots, m\} \quad \text{and} \quad f_j(x) < f_j(x^\star) \;\; \text{for at least one } j.$$

The set of all Pareto-optimal schedules constitutes the *Pareto front* $\mathcal{P}^\star$.
To fairly compare the performance of different multi-objective algorithms, we utilize several metrics:

Hypervolume (HV) The hypervolume metric measures the volume of the objective space dominated by a set of solutions with respect to a reference point. Initially, a fixed reference point is established. This reference point is calculated by running multiple preliminary experiments across all selected algorithms (NSGA-II, NSGA-III, U-NSGA-III, and SMS-EMOA) with mid-range hyperparameters, identifying the worst observed objective values, and extending these values by 20% to ensure all subsequent solutions clearly dominate it.
During the optimization phase, hypervolume values were computed with `pymoo`'s `HV` indicator [BD20], which internally calls the DEAP hypervolume routine [FDG+12]. The latter uses the efficient WFG algorithm proposed by Bradstreet and Hingston (2011) [WBB11] to calculate the HV. This is done with the purpose of tracking the generations on how each MOEA's approximation of the Pareto front improves over time (hypervolume convergence).
Additionally the SMS-EMOA requires the hypervolume contribution of every individual at each iteration for its survivor selection, so the indicator must be calculated during the optimization phase. After computing the raw hypervolume (the total volume dominated by the Pareto front), it is normalized by dividing by the product of the reference point's dimensions, enabling consistent comparisons across different experimental setups.

Formally hypervolume is defined as:
Let $S = \{\mathbf{f}(x) \mid x \in \mathcal{A}\} \subset \mathbb{R}^m$ be the set of non-dominated objective vectors returned by an

[1] https://github.com/NicholasWU2003/NewBachelorProject

algorithm, and let $r \in \mathbb{R}^m$ be a reference point that is dominated by every $\mathbf{f}(x) \in S$. Following [GFP21], the hypervolume indicator is

$$H(S) \;=\; \Lambda\big(\{\, q \in \mathbb{R}^m \mid \exists\, p \in S :\; p \le q \le r \,\}\big) \;=\; \Lambda\Big(\bigcup_{p \in S} [p, r]\Big), \tag{10}$$

where $\Lambda(\cdot)$ denotes the m-dimensional Lebesgue measure[Rud87] and $[p, r] = \{\, q \in \mathbb{R}^m \mid p \le q \le r\}$ is the axis-aligned box spanned by the point p and the reference point r. In the four-objective maintenance problem studied here the volume in (10) is a 4-D hyper-rectangle union. A larger $H(S)$ implies a more dominant, well-spread approximation of the true Pareto front.

Inverted Generational Distance (IGD) IGD values are computed with `pymoo`'s `InvertedGenerationalDistance` indicator [BD20], which uses a KD-tree acceleration for the nearest-neighbour queries, described in 2.2. Unlike hypervolume contribution, IGD is used *only* for post-hoc performance tracking and does not influence the evolutionary operators.

Formally, let $Z = \{z_1, \ldots, z_{|Z|}\} \subset \mathbb{R}^m$ be a non-dominated reference set approximating the true Pareto front, and $A = \{a_1, \ldots, a_{|A|}\} \subset \mathbb{R}^m$ be the solution set returned by an algorithm. Following Ishibuchi Et al. [IMN15, Eq. (13)], IGD is defined as

$$IGD(A, Z) \;=\; \frac{1}{|Z|} \sum_{j=1}^{|Z|} \min_{a_i \in A} d(a_i, z_j), \qquad d(a_i, z_j) = \|a_i - z_j\|_2. \tag{11}$$

That is, for every reference point z_j we measure the Euclidean distance to the closest solution in A and average these distances. A smaller IGD therefore indicates that A lies closer to the reference front Z.

Convergence Speed Convergence speed evaluates how rapidly an algorithm approaches both the Pareto front (in terms of solution quality) and a stable diversity level. We measure convergence speed using two complementary aspects:

- **Hypervolume Convergence:**
  - **Threshold Hypervolume ($\mathbf{HV}_{0.95}$):** 95 % of the final hyper-volume obtained at termination.
  - **Generation to $\mathbf{HV}_{0.95}$ (g_{95}):** the earliest generation at which the running hyper-volume first exceeds $\mathrm{HV}_{0.95}$ and never drops below it thereafter. Smaller g_{95} values indicate faster convergence toward the Pareto front.
- **Diversity Convergence:**
  - **Pairwise Distance Calculation:** At termination we compute the Euclidean distance between every pair of non-dominated solutions F_i and F_j in objective space $(Cost,\ Frequency,\ Risk)$:

$$d_{ij} \;=\; \sqrt{(Cost_i - Cost_j)^2 + (Freq_i - Freq_j)^2 + (Risk_i - Risk_j)^2}.$$

– **Final Diversity Metric:** The mean of all pairwise distances,

$$\text{Diversity} \;=\; \frac{1}{\binom{N}{2}} \sum_{i<j} d_{ij},$$

where N is the number of non-dominated schedules. Higher values indicate a more widely spread Pareto set.

5.2 Hyperparameter Optimization using SMAC

To ensure that evolutionary algorithms achieve optimal performance, hyperparameters such as population size, crossover probability, mutation probability, crossover distribution index (eta), and mutation distribution index require careful tuning. We utilize the Sequential Model-based Algorithm Configuration (SMAC) framework, a sophisticated approach that systematically searches the hyperparameter space using Bayesian optimization methods.
Our hyperparameter optimization (HPO) approach follows these specific steps:

1. **Configuration Space Definition**: We define the search space explicitly, with clearly bounded ranges for each hyperparameter:
   - Population size: $[20, 100]$
   - Crossover probability: $[0.6, 0.95]$
   - Mutation probability: $[0.05, 0.30]$
   - Number of generations: $[20, 50]$
   - Crossover distribution index (eta): $[5, 20]$
   - Mutation distribution index (eta): $[5, 20]$
2. **Initial Reference Point Establishment**: Before running SMAC, a fixed hypervolume reference point is established by performing five iterations for all candidate algorithms (NSGA-II, NSGA-III, U-NSGA-III, SMS-EMOA, and MOEA/D) using mid-range parameter values. The worst observed solutions from these runs define the reference point after a 24% extension, providing a consistent basis for hypervolume calculations.
3. **SMAC Optimization Process**:
   - SMAC iteratively evaluates various hyperparameter configurations on each algorithm, using the hypervolume metric (computed with the WFG [WBB11] algorithm) to measure performance.
   - Leveraging a machine learning model (random forest-based), SMAC predicts promising configurations based on previously tested setups, efficiently navigating towards optimal solutions.
   - Each algorithm is run for the selected hyperparameter configurations, and early stopping criteria are implemented based on hypervolume improvement rates, ensuring efficient utilization of computational resources.

4. **Final Selection and Verification**:

   - SMAC selects the best-performing configuration for each algorithm, recording these optimized parameters for reproducibility and further experiments.
   - The final chosen hyperparameters are verified through additional detailed evaluations, ensuring consistent and reliable performance metrics.

By explicitly following this hyperparameter optimization methodology, we ensure that algorithm comparisons are precise, fair, and accurately reflect their optimal potential.

5.3 Experimental Scenarios

To thoroughly evaluate our model, we generate synthetic test instances designed to closely emulate practical preventive maintenance scheduling scenarios. Each instance is created using our customized `data_generator.py` & `problemDef.py`, enabling controlled and systematic variations. Specifically, each scenario includes:

- **Components:** Defined by their initial Remaining Useful Life (RUL) prognostics and associated maintenance costs. Components are assumed to be already installed at the start of the experiment, and their failure behavior is modeled accordingly.
- **Maintenance Slots:** Distributed systematically over the planning horizon, incorporating configurable constraints to simulate practical limitations such as unavailable or blackout periods.
- **Failure Probabilities:** Calculated using realistic stochastic models derived from RUL prognostics, allowing accurate simulation of component degradation and failure events.

Our data generation approach supports the simulation of various component types, with parameters easily adjustable to closely match real-world components. This flexibility facilitates the evaluation of algorithm performance across diverse and realistic scenarios, ensuring robust and practical insights.

5.3.1 Problem instances Tested

The experimental design was constructed to systematically evaluate algorithm performance across multiple problem characteristics that directly impact real-world maintenance optimization scenarios:

Component Count (5–40): The range from 5 to 40 components represents the spectrum from small industrial subsystems to medium-scale manufacturing facilities. The small scale enables detailed analysis and validation of algorithm behaviour, while the large scale approaches realistic industrial complexity. This progression allows us to identify scalability limitations and computational bottlenecks that emerge as the decision space grows.

Planning Horizon (50–200): The horizon variations reflect different maintenance planning contexts: short-term operational planning (50 periods), medium-term strategic planning (100 periods), and long-term asset management (200 periods). Longer horizons increase problem complexity both computationally and structurally. Initial decisions could have a drastic effect throughout the planning horizon, so the decisions become more intricate and.

RUL characteristics To examine how lifetime uncertainty and replacement frequency affect scheduling difficulty, the experiments vary the mean (μ) and standard deviation (σ) of the components RUL distribution. A smaller mean forces more frequent interventions, while a larger σ adds more uncertainty.

Scenario	**Reliability label**	**μ, σ (periods)**	**Maintenance pressure**
Conservative (1)	High, predictable	(30, 3)	Low
Moderate (2)	Standard (baseline)	(20, 5)	Medium
Aggressive (3)	Low, erratic	(12, 8)	High

Table 2: RUL parameter settings used in the experimental study.

Each level moves the problem from a relaxed, long-horizon optimisation (Conservative) through a balanced baseline (Moderate) to a high-pressure, unpredictable environment (Aggressive).
These problem instances collectively ensure comprehensive coverage of the maintenance optimization problem space, enabling complete conclusions about algorithm performance across diverse practical scenarios.

5.4 Software and Tools

Our experiments are implemented and executed using the Python programming language [VRD09], chosen for its extensive libraries and simplicity. The primary library we use is Pymoo [BD20], an open-source optimization framework specifically designed for multi-objective optimization problems. Pymoo provides efficient implementations of algorithms like NSGA-II, NSGA-III, U-NSGA-III, SMS-EMOA, and MOEA/D, as well as essential performance metrics such as hypervolume.
Experiments are run on a `MacBook Pro (2021)`, and reproducibility is ensured by fixing random seeds. To analyze and visualize results clearly, we use standard Python visualization libraries, like Matplotlib and Seaborn, presenting trade-offs between objectives and convergence behaviour over generations.

5.5 Evaluation Procedure

To thoroughly assess algorithm performance and convergence behaviour, we follow a structured evaluation procedure detailed below:

1. **Algorithm Execution**: For each test scenario, each evolutionary algorithm (NSGA-II, NSGA-III, U-NSGA-III, SMS-EMOA, and MOEA/D) is executed with hyperparameter settings optimized through SMAC. Each algorithm runs for a predefined number of generations specified by the optimized hyperparameters.

2. **Convergence Tracking**: Throughout algorithm execution, convergence metrics are continuously recorded at each generation. Specifically, we capture:

    - **Hypervolume (HV)**: Calculated using the WFG algorithm relative to a fixed reference point, providing a robust measure of solution quality and coverage.

- **Solution Diversity**: Measured as the average pairwise Euclidean distance among solutions in the objective space (Cost, Frequency, Risk, Reactive maintenance), indicating how broadly the algorithm explores the solution space.

3. **Convergence Speed Analysis**: Convergence speed is quantified by evaluating how many generations each algorithm takes to achieve 95 % of its final hypervolume. Additional metrics such as the area under the hypervolume curve (HV AUC), improvement rates, and stagnation periods are recorded to provide comprehensive insights into the efficiency and robustness of convergence.

4. **Post-run quality comparison.** After the final generation we evaluate the approximation set with **Inverted Generational Distance (IGD)** in addition to HV. IGD is computed against a high-resolution reference front obtained by merging the non-dominated solutions of *all* runs for the same problem instance. This ensures a fair comparison for the algorithms in that specific problem instance.

5. **Multiple Runs for Robustness**: Each scenario is executed multiple times, each with a distinct random seed, ensuring statistical robustness.

6. **Visualization and Reporting**: Detailed visual summaries, including Pareto front approximations and convergence curves (hypervolume and diversity) are generated. These visualizations enable clear interpretation of algorithm behaviours and performance differences.

6 Discussion

This section interprets the experimental result in the light of the research objectives stated in the Introduction 1:

- Determine how problem size and RUL characteristics affect maintenance scheduling difficulty.
- Compare the performance of five MOEAs.
- Assess the benefits of hyperparameter with SMAC.

In this section we follow an experiment with different problem sizes each of the instances require a different reference point, which results in no cross-instance comparison between the performance indicators like HV and IGD. Therefore we normalize the HV and IGD to a [0,1] scale to preserve how far the worse performing algorithm behind the best performing one is. We also rank the algorithms to see which performs the best.

6.1 SMAC hyperparameter optimization

To test the performance difference of hyperparameter optimization using SMAC(section 5.2), we ran SMAC on one balanced problem instance: 15 components, 100 PH, and the moderate RUL scenario(5.3.1 (seed = 42). Each algorithm was given a 50 trial SMAC budget to search its optimal parameters.

Table 3 lists the parameters that achieved the highest HV after the SMAC optimization, and the initial parameters used are included for reference.

SMAC-optimised							**Standard defaults**					
Algorithm	Pop.	Gen.	P_{cross}	P_{mut}	Cx-eta	Mut-eta	Pop.	Gen.	P_{cross}	P_{mut}	Cx-eta	Mut-eta
NSGA-II	89	43	0.89	0.10	11	9	60	35	0.80	0.20	15	17
NSGA-III	95	44	0.78	0.16	15	20	60	35	0.80	0.15	12	15
U-NSGA-III	95	44	0.78	0.16	15	20	60	35	0.80	0.15	12	15
SMS-EMOA	97	44	0.68	0.26	6	16	60	35	0.80	0.15	12	15
MOEA/D	83	35	0.83	0.27	13	18	90	50	0.62	0.17	17	20

Table 3: Default versus SMAC-optimised parameter settings.

Four of the five algorithms achieved higher hypervolume after SMAC tuning, and all five obtained a lower IGD (Table 4). Thus, SMAC provides a consistent IGD improvement and, for most algorithms a HV gain. The only exception is MOEA/D: its HV decreased by 7.3 %, whereas its IGD still improved by 7.8 %. This suggests that SMAC configured MOEA/D to obtain solutions closer to the Pareto front, but at the cost of some diversity for convergence for this algorithm.
The optimized parameters intensify exploration within the sub-problems, explaining the IGD gain despite the small HV setback.
Additionally SMAC exploits problem-specific interactions: larger populations and lower mutation probabilities prevents premature convergence in the NSGA family, while higher mutation rates help SMS-EMOA. In the case of MOEA/D it improves IGD, by exploring sparsely dominated regions of the four-objective space, even if that exploration slightly lowers HV for MOEA/D.

Algorithm	$\text{HV}_{\text{default}}$	HV_{SMAC}	$\text{HV}_{\text{Improvement}}$	$\text{IGD}_{\text{default}}$	IGD_{SMAC}	$\text{IGD}_{\text{Improvement}}$
NSGA-II	0.0252	0.0263	+4.52 %	0.0027	0.0017	+37.36 %
NSGA-III	0.0205	0.0214	+4.54 %	0.0079	0.0062	+21.76 %
U-NSGA-III	0.0208	0.0233	+12.25 %	0.0068	0.0067	+0.36 %
SMS-EMOA	0.0247	0.0260	+5.08 %	0.0047	0.0035	+25.96 %
MOEA/D	0.0134	0.0124	-7.33 %	0.0234	0.0216	+7.78 %

Table 4: Performance before and after SMAC tuning.

All subsequent experiments use the SMAC-tuned settings listed above. Although these configurations are not re-optimised for every individual instance, they remain within a few percent of the optimum for NSGA-II, NSGA-III, U-NSGA-III and MOEA/D after experimenting. This keeps computational cost moderate while ensuring each algorithm operates near its optimum. SMS-EMOA is the only exception, because its selection pressure depends directly on hypervolume contribution, so its performance degrades noticeably when parameters are reused. Consequently the parameters are only re-tuned for the SMS-EMOA algorithm for each instance.

6.2 Problem size

To test the performance of the algorithms on the size of the problem we have two categories, the PH and the amount of components in the system.

6.2.1 Planning Horizon

Extending the planning horizon from 50 to 200 periods consistently reduces HV, indicating that problem difficulty rises with longer windows (Table 5). At the same time, the IGD decreases, meaning the non-dominated solution sets produced for the longer horizons lie closer to the Pareto front despite the drop in HV.
NSGA-II remains the best performer at every horizon, followed by SMS-EMOA. At 200 periods, NSGA-III edges ahead of U-NSGA-III, whereas MOEA/D stays last throughout. Overall, crowding-distance (NSGA-II) and hypervolume-selection (SMS-EMOA) handle longer horizons more efficient than the decomposition-based MOEA/D.

	Horizon 50			**Horizon 100**			**Horizon 200**		
Algorithm	**Rank**	**HV**	**IGD**	**Rank**	**HV**	**IGD**	**Rank**	**HV**	**IGD**
NSGA-II	1	0.0287	0.0020	1	0.0134	0.0056	1	0.0059	0.0010
SMS-EMOA	2	0.0246	0.0067	2	0.0116	0.0133	2	0.0053	0.0037
NSGA-III	4	0.0215	0.0142	4	0.0104	0.0103	3	0.0052	0.0048
U-NSGA-III	3	0.0242	0.0106	3	0.0109	0.0133	4	0.0049	0.0054
MOEA/D	5	0.0062	0.0480	5	0.0048	0.0521	5	0.0024	0.0157

Table 5: Algorithm performance across planning horizons (ranked by hypervolume).

6.2.2 Component count

To isolate the effect of problem size we varied the number of components, while keeping the horizon fixed at 50 periods and the RUL scenario at the moderate setting. Four levels were tested: 5, 10, 20, and 40 components. Table 6 reports the HV and IGD obtained under each size.
There we can see that as the fleet grows the problem quickly becomes harder: hyper-volume shrinks for every algorithm indicating a loss in diversity. In contrast the IGD initially improves from component count 5 to 20, but with component size 40 it jumps back up, indicating that the search for the Pareto front plateaus.

- **NSGA-II** delivers the highest HV up to the 20-component case and maintains the best IGD at every size, confirming its strong convergence pressure.
- **SMS-EMOA** overtakes NSGA-II on HV when the fleet grows to 40 components, thanks to its hypervolume–guided selection, however it still trails NSGA-II on IGD.
- **NSGA-III** and **U-NSGA-III** stay in the middle of the pack on both metrics.
- **MOEA/D** remains last throughout, underlining that the decomposition strategy scales poorly with densely constrained, large-scale schedules.

Algorithm	Small (5) HV	Small (5) IGD	Medium (10) HV	Medium (10) IGD	Large (20) HV	Large (20) IGD	Extra-L (40) HV	Extra-L (40) IGD
NSGA-II	0.070 (1)	0.004	0.042 (1)	0.002	0.021 (1)	0.003	0.007 (2)	0.012
NSGA-III	0.049 (3)	0.025	0.032 (4)	0.018	0.017 (3)	0.010	0.007 (4)	0.016
U-NSGA-III	0.052 (2)	0.021	0.033 (3)	0.017	0.017 (4)	0.011	0.007 (3)	0.013
SMS-EMOA	0.048 (4)	0.015	0.036 (2)	0.005	0.021 (2)	0.005	0.008 (1)	0.014
MOEA/D	0.024 (5)	0.043	0.012 (5)	0.043	0.007 (5)	0.047	0.004 (5)	0.016

Table 6: Hypervolume and IGD across system sizes (rank in parentheses)

6.3 RUL characteristics

To isolate the impact of component reliability we fixed the horizon at 50 periods and the fleet at 15 components, then ran the three RUL scenarios from Table 2. Table 7 reports the resulting HV and IGD scores, while figure 2 adds convergence speeds and the final diversity scores.

In the experiment we observed the impact of shortening and destabilizing the components RUL prognostic. As expected every algorithm loses hypervolume as we move from the Conservative to the Aggressive scenario. This is because shorter RULs leave less room for a good trade-off, resulting in a decrease in dominance of the hypervolume.

However for IGD the results are a bit more nuanced. Although most algorithms' HV decreases drastically, its IGD actually improves. This shows that it continues to capture solutions efficiently across the shrinking front.

In the end NSGA-II copes the best with the unstable RUL characteristics, it continues to capture the decreasing front efficiently

SMS-EMOA follows the same pattern, but couldn't perform when the RUL became aggressive.

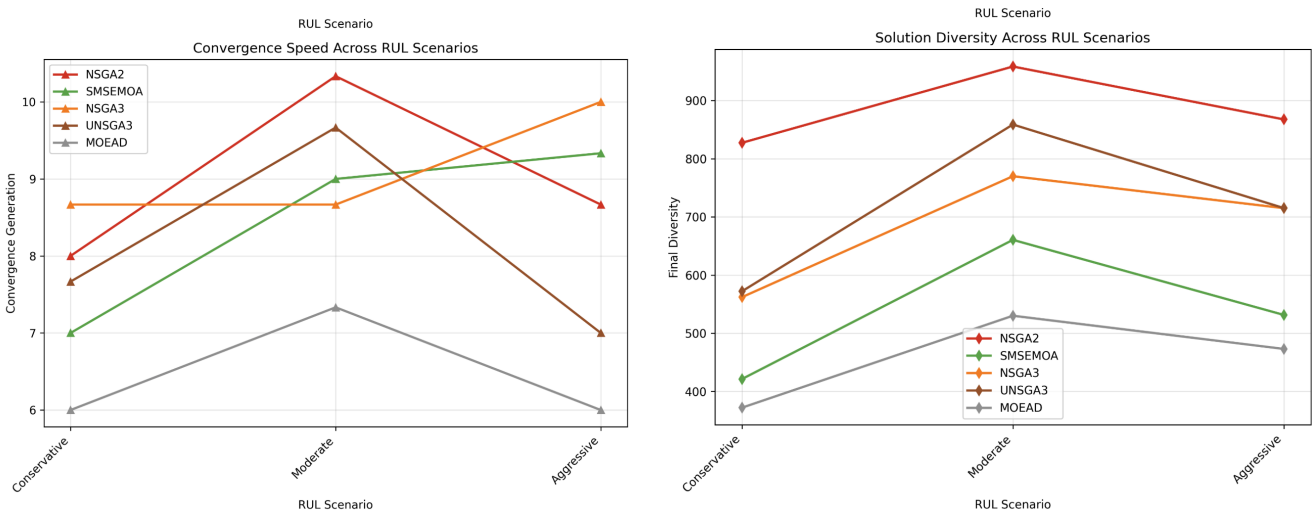
Additionally, NSGA-III and U-NSGA-III failed to capture the front in the Moderate case, causing a spike in IGD before recovering part of that loss in the most aggressive setting.

Finally MOEA/D performs the worst, its fixed weight vectors chase the wrong parts of the search space, so IGD more than doubles

To have a more in depth insight on the performance we also tracked the hypervolume convergence speed and solution diversity as described in section 5.1. The figure 2a depicts the amount of generations the algorithms needed to reach 95% of its final hypervolume. Here we can see that MOEA/D was the fastest, and NSGA-II was the slowest. Figure 2b shows the pairwise distance between the solutions. Here we can conclude that NSGA-II maintained the widest spread of solutions across the three instances. These results emphasizes that speed alone will not always result in a good outcome.

Algorithm	HV			IGD		
	Conservative	Moderate	Aggressive	Conservative	Moderate	Aggressive
NSGA-II	0.0750	0.0261 (–65 %)	0.0061 (–76 %)	0.0031	0.0023 (–26 %)	0.0005 (–78 %)
SMS-EMOA	0.0705	0.0262 (–63 %)	0.0058 (–78 %)	0.0122	0.0044 (–64 %)	0.0015 (–66 %)
NSGA-III	0.0655	0.0192 (–71 %)	0.0045 (–77 %)	0.0233	0.0135 (–42 %)	0.0054 (–60 %)
U-NSGA-III	0.0614	0.0226 (–63 %)	0.0049 (–78 %)	0.0282	0.0122 (–57 %)	0.0033 (–73 %)
MOEA/D	0.0264	0.0126 (–52 %)	0.0036 (–71 %)	0.0430	0.0475 (+10 %)	0.0104 (–78 %)

Table 7: Hypervolume (HV) and IGD across the three RUL scenarios. Percent change relative to the previous scenario is in parentheses.



(a) Convergence speed (g_{95})

(b) Solution diversity

Figure 2: Convergence speed and solution diversity across RUL scenarios.

6.4 Result Visualisation

Figure 3 illustrates four maintenance schedules generated by the framework when each specific objective is being prioritized. The red bars depict blackout periods, when no maintenance is allowed to be performed, and the blue boxes signify planned maintenance. In practice this framework can be adapted to reflect the decision makers priorities and assign each specific objective a weight to shape the maintenance schedules to their liking.

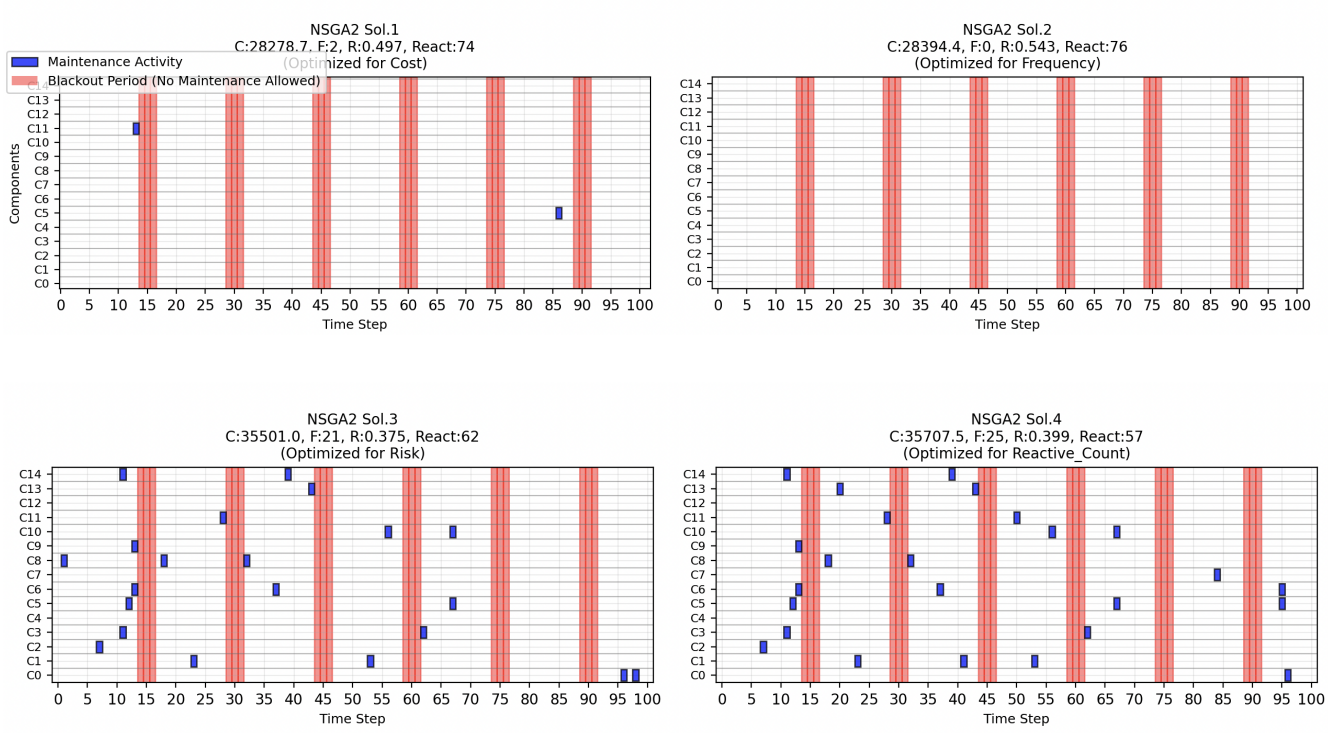


Figure 3: Maintenance schedules produced with NSGA-II.

7 Conclusion

This thesis set out to develop and evaluate a data-driven framework for a realistic maintenance scheduling problem, where the objectives guided the work:

1. Quantify how problem size and reliability uncertainty shape scheduling difficulty.
2. Benchmark a representative set of evolutionary MOEAs under those conditions.
3. Assess the practical benefit of automated hyper-parameter tuning with SMAC.

This paper provides a systematic benchmark of SMAC-tuned MOEAs on a discrete maintenance scheduling model that incorporates condition based maintenance using the RUL prognostics. It contributes a flexible framework that can be reused for varying instances, whether it would be for different components or planning horizons, as illustrated in section 6.4.

7.1 Algorithm Ranking for the studied problem

Across all scenarios NSGA-II tuned by SMAC overall delivered the highest hypervolume and the lowest IGD. Its crowding distance based niching outperformed the other algorithms.
SMS-EMOA, with its hypervolume based selection formed a consistent second place. SMS-EMOAs performance rivalled NSGA-II's on larger problem instances, but its IGD was less stable.

NSGA-III and U-NSGA-III, with its reference direction based niching provided moderate result and traded places depending on the different problem instances. This problem with four objectives didn't extract the theoretical performance it could achieve
Finally the MOEA/D came in last, it consistently underperformed. Likely because of the highly-constrained problem. Its static weight decomposition is not suited for highly constrained problems. In short, NSGA-II is the best option for the four-objective maintenance problem handled in this thesis. SMS-EMOA is a suitable alternative, when decision-makers want to maximize the Pareto-front coverage, while having the computational resources to afford the extra runtime for the hypervolume contribution calculation. (U)NSGA-III could become worthwhile when the amount of objectives increases and MOEA/D is not recommended for highly-constrained problem.

7.2 Limitation and Future works

The study relied on synthetic Weibull-based failure data. A natural next step is to feed the framework with real sensor or logbook data from for example Heerema, the maritime company mentioned in the introduction 1. Then we could observe if the observed performance across the algorithms still hold on real-world data. Furthermore scaling to a thousand-component systems is also of interest, preliminary runs suggest SMS-EMOA may overtake NSGA-II at extreme sizes.
On the modelling side, adding more objectives (e.g. repair vs. replace trade-offs) and resource constraints (limited, partially-skilled crews, imperfect repairs) would push the simulator closer to industrial reality and give algorithms such as NSGA-III a chance to leverage their many-objective strengths.
In summary, this research provides (i) a reusable benchmark framework, (ii) evidence that SMAC-tuned NSGA-II is a strong default for RUL-driven maintenance problems, and (iii) clear directions for transferring the approach to large-scale, real-world scenarios.

References

[BD20] J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020.

[BG15] S Brisset and F Gillon. Approaches for multi-objective optimization in the ecodesign of electric systems. *Eco-friendly innovation in electricity transmission and distribution networks*, pages 83–97, 2015.

[BH60] Richard Barlow and Larry Hunter. Optimum preventive maintenance policies. *Operations research*, 8(1):90–100, 1960.

[BNE07] Nicola Beume, Boris Naujoks, and Michael Emmerich. Sms-emoa: Multiobjective selection based on dominated hypervolume. *European journal of operational research*, 181(3):1653–1669, 2007.

[BPML14] Salvatore Bianchi, Roberto Paggi, Gian Luca Mariotti, and Fabio Leccese. Why and when must the preventive maintenance be performed? In *2014 IEEE Metrology for Aerospace (MetroAeroSpace)*, pages 221–226. IEEE, 2014.

[Deb00] Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2-4):311–338, 2000.

[DJ13] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4):577–601, 2013.

[DPAM02] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[dPM21] Ingeborg de Pater and Mihaela Mitici. Predictive maintenance for multi-component systems of repairables with remaining-useful-life prognostics and a limited stock of spare components. *Reliability Engineering & System Safety*, 214:107761, 2021.

[FDG+12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[GFP21] Andreia P. Guerreiro, Carlos M. Fonseca, and Luís Paquete. The hypervolume indicator: Computational problems and algorithms. *ACM Comput. Surv.*, 54(6), July 2021.

[Hee24] Heerema Marine Contractors. Making the impossible possible offshore. https://hmc.heerema.com/, 2024. Accessed 2025-07-04.

[IMN15] Hisao Ishibuchi, Hiroyuki Masuda, and Yusuke Nojima. A study on performance evaluation ability of a modified inverted generational distance indicator. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, page 695–702, New York, NY, USA, 2015. Association for Computing Machinery.

[LEF+22] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, AndrÃ© Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, RenÃ© Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[LND+06] Jay Lee, Jun Ni, Dragan Djurdjanovic, Hai Qiu, and Haitao Liao. Intelligent prognostics tools and e-maintenance. *Computers in industry*, 57(6):476–489, 2006.

[LPLJ20] Yanrong Li, Shizhe Peng, Yanting Li, and Wei Jiang. A review of condition-based maintenance: Its prognostic and operational aspects. *Frontiers of Engineering Management*, 7(3):323–334, 2020.

[MS96] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, 4(1):1–32, 1996.

[RTBG22] Jeroen Rook, Heike Trautmann, Jakob Bossek, and Christian Grimme. On the potential of automated algorithm configuration on multi-modal multi-objective optimization problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 356–359, 2022.

[Rud87] Walter Rudin. *Real and complex analysis, 3rd ed.* McGraw-Hill, Inc., USA, 1987.

[RVLB15] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance metrics in multi-objective optimization. In *2015 Latin American computing conference (CLEI)*, pages 1–11. IEEE, 2015.

[SD15] Haitham Seada and Kalyanmoy Deb. A unified evolutionary optimization procedure for single, multiple, and many objectives. *IEEE Transactions on Evolutionary Computation*, 20(3):358–369, 2015.

[SL20] Chun Su and Yang Liu. Multi-objective imperfect preventive maintenance optimisation with nsga-ii. *International Journal of Production Research*, 58(13):4033–4049, 2020.

[VRD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[VW88] William E Vesely and AJ Wolford. Risk evaluations of aging phenomena: the linear aging reliability model and its extensions. *Nuclear Engineering and Design*, 108(1-2):179–185, 1988.

[WBB11] Lyndon While, Lucas Bradstreet, and Luigi Barone. A fast way of calculating exact hypervolumes. *IEEE Transactions on Evolutionary Computation*, 16(1):86–95, 2011.

[Wu25] Nicholas Chi-Hung Wu. Multi-objective-maintenance-scheduling: Code and data. https://github.com/NicholasWU2003/NewBachelorProject, 2025.

[WXZ24] Zihan Wang, Chunyun Xiao, and Aimin Zhou. Exact calculation of inverted generational distance. *IEEE Transactions on Evolutionary Computation*, 2024.

[ZL07] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[ZQL+11] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and evolutionary computation*, 1(1):32–49, 2011.