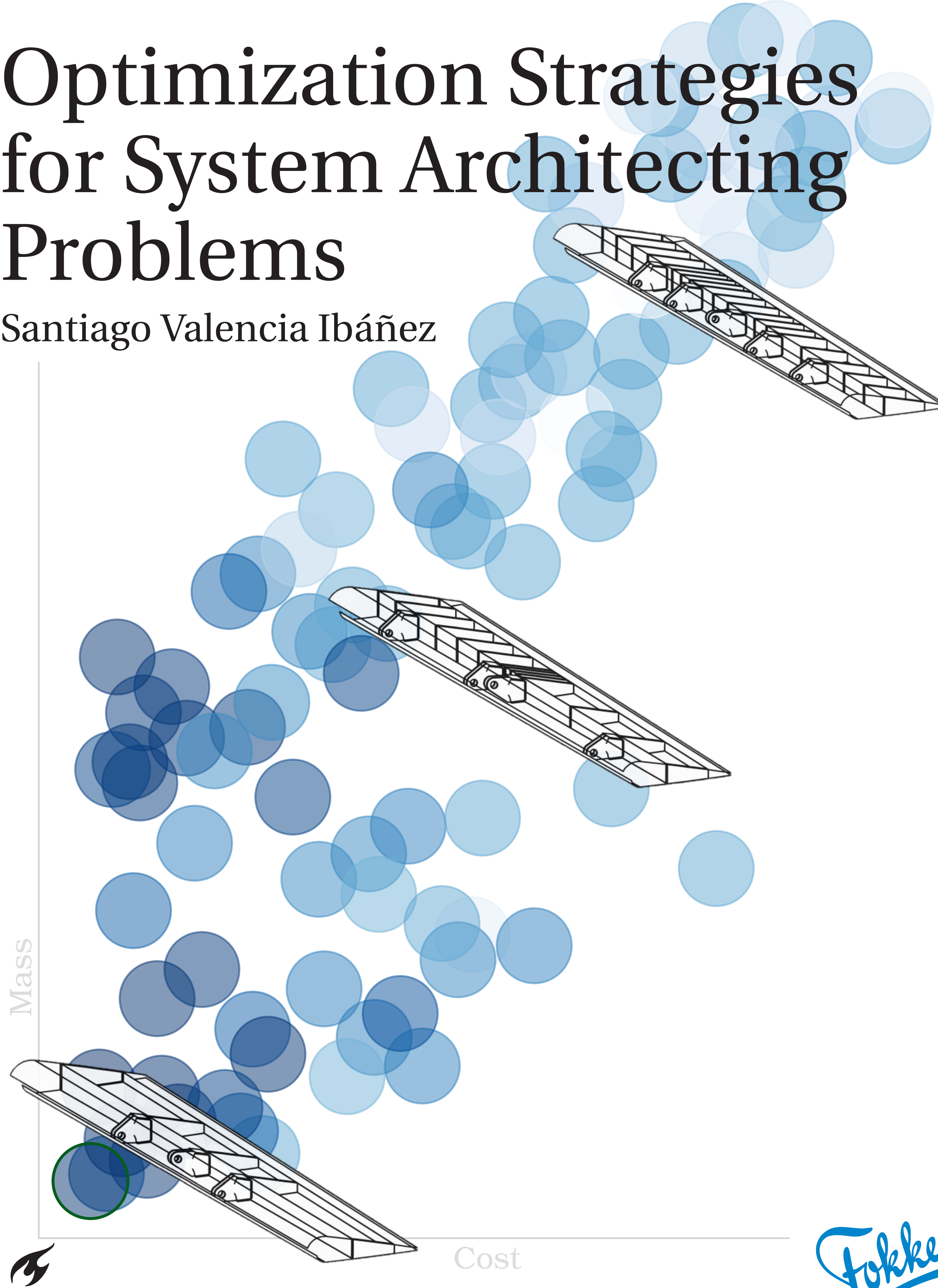
Optimization Strategies for System Architecting Problems

Santiago Valencia Ibáñez



Delft University of Technology

Optimization Strategies for System Architecting Problems

by

Santiago Valencia Ibáñez

in partial fulfillment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology,
to be defended publicly on Tuesday August 29, 2023 at 13:00.

Supervisors:	ir. A.M.R.M. Bruggeman	TU Delft
	ir. J.S. Sonneveld	TU Delft
	Dr.ir. A.H. van der Laan	GKN Fokker
Thesis committee:	Dr.ir. G. La Rocca	TU Delft, Chair & Supervisor
	Dr. F. Oliviero	TU Delft, Examiner
	Dr. M.B. Zaayer	TU Delft, Examiner

Acknowledgements

This work marks the end of my time as a master's student at TU Delft. It has been a great time for me here. I've had two years full of learning not only about aerospace engineering, but also about myself.

I was very fortunate to have many people involved and interested in my thesis work. First and foremost, I would like to thank Dr.ir. Gianfranco La Rocca for introducing me to the wonderful world of KBE, MDO, and to this project, which includes some elements of both. I am grateful for his feedback and encouragement all along this project. I would also like to extend my gratitude to Jente Sonneveld and Anne-Liza Bruggeman, my daily supervisors, for their permanent support and guidance. During this project, I worked as a graduate thesis trainee at GKN Fokker under the supervision of Dr.ir. Ton van der Laan and Dr.ir. Tobie van den Berg. I give heartfelt thanks to both for welcoming me from the first day into the Centre of Competence Design and showing me how research and industry can go hand in hand. Thanks also go to Dr. Fabrizio Oliviero and Dr. Michiel Zaayer for taking the time to be a part of my examination committee.

During this project, I had the opportunity to exchange thoughts and ideas with several people who took an interest in my work. I would like to thank Bas van Manen and Ioana Ciobotia for their feedback, questions, and ideas. I would also like to thank Jasper Bussemaker and Dr. Julien Pelamatti for making themselves available to answer my questions and discuss our work. To Sam Andersson, thank you for such interesting and fun conversations on graph reinforcement learning, even if that route did not pan out in the end for me.

I would also like to thank my friends in Delft for such a wonderful two years, especially Inge, Naomi, Tasos, Conrado, María, and Miguel. Thank you for your unwavering support and for all the good times!

I cannot end this section without addressing my family. I always felt you by my side, even while being separated by more than 9,000 kilometers. Thank you for supporting on this journey and of course, for coming to my graduation!

Finally, I want to express my deepest appreciation and admiration to my partner Diana, who has been with me in the highs and lows of this adventure. Thank you for your advice, patience, understanding, and support. And thank you for believing in me.

Thank you all! And thank you, reader, for taking an interest in this work. I am proud to present it to you.

Santiago Valencia Ibáñez
Delft, August 2023

SUMMARY

System architecting is one of the first stages of the engineering problem-solving process. Pivotal decisions regarding the system's overall configuration are taken in this phase. Consequently, decision support tools like system architecture optimization are needed to effectively assess the architectural design space. However, system architecture optimization problems include several features that make them challenging to tackle with traditional optimization techniques, including multiple objectives and mixed-discrete, hierarchical design spaces.

This work examines three algorithms capable of handling the mixed-discrete and multi-objective nature of system architecture optimization problems: genetic algorithms (such as NSGA-II), Bayesian optimization (BO), and the Deep Deterministic Policy Gradient (DDPG) reinforcement learning framework. These algorithms act as the engines that power three strategies capable of addressing hierarchical design spaces: *global exploration*, *nested optimization*, and *decision chain*. While global exploration optimizes all design variables in a single loop and uses imputation on the inactive variables, the nested optimization method divides the problem into multiple hierarchical optimization loops. In contrast, the decision chain approach reframes the problem into a sequential decision-making process within an environment where an agent's actions alter design instances.

Test problems of varying complexity are used to evaluate the suitability of these algorithms and strategies. First, an airfoil optimization problem demonstrates the basic functionality of the selected algorithms with satisfactory results comparable to those found in the literature for the same problem. Next, a mixed-discrete version of the ZDT1 multi-objective problem is evaluated for different problem sizes and proportions of continuous and discrete variables. In this family of problems, the NSGA-II algorithm emerges as the most reliable and best-performing. At the same time, DDPG struggles with many variables, and Bayesian optimization runs into computational performance issues when dealing with large discrete combinatorial spaces.

Later, the architecture optimization strategies are evaluated on two versions of the Goldstein problem, a hierarchical and multi-objective test case. In the first version, which only considers eight architectures, all three strategies show competency in solving the optimization problem, but the global exploration strategy consistently shows the best performance. The decision chain strategy is discarded from further study at this stage due to relatively low performance and implementation difficulties. The second version of the Goldstein problem can be adjusted to different degrees of hierarchy and numbers of architectures. In this case, global exploration rises again as the best-performing strategy.

Finally, a real-world aileron structural optimization study at GKN Fokker shows that the implemented global exploration and nested strategies can find better designs than a random sampling of the design space. The best design for this test problem is achieved with a two-level nested strategy, combining NSGA-II in the outer loop and Bayesian optimization in the inner loop.

In conclusion, this research proposes three adaptive strategies powered by various algorithms to solve system architecture optimization problems. The findings suggest a preference for global exploration when a complete design vector can be pre-established. Otherwise, the nested optimization strategy, which can combine different algorithms' strengths and does not need a predefined design vector, emerges as the most suitable choice for more complex scenarios.

CONTENTS

Summary ii

List of Figures v

List of Tables vii

Nomenclature viii

1 Introduction 1
- 1.1 The Structure of an Aileron . . . 1
- 1.2 Introduction to Systems. . . . 2
- 1.3 Introduction to System Architecture Optimization . . . 2
- 1.4 Challenges of System Architecture Optimization . . . 3
- 1.5 Research Objective and Question . . . 3
- 1.6 Thesis Outline . . . 4

2 Systems Engineering and System Architecture 6
- 2.1 Systems Engineering . . . 6
- 2.2 System Architecture. . . . 7
  - 2.2.1 Form and Function . . . 7
  - 2.2.2 System Architecting Frameworks . . . 7
  - 2.2.3 Obstacles of System Architecting. . . . 8
- 2.3 System Architecture Optimization . . . 8

3 System Architecture Optimization 10
- 3.1 Formulation of the System Architecture Optimization Problem. . . . 10
- 3.2 Features of System Architecture Optimization Problems . . . 11
  - 3.2.1 Mixed-Discrete Design Spaces . . . 11
  - 3.2.2 Hierarchical Design Spaces . . . 13
  - 3.2.3 Multiple Objectives . . . 14
  - 3.2.4 Hidden Constraints . . . 14
- 3.3 Algorithms for System Architecture Optimization Problems . . . 14
  - 3.3.1 Genetic Algorithms . . . 15
  - 3.3.2 Bayesian Optimization. . . . 16
  - 3.3.3 Reinforcement Learning — Deep Deterministic Policy Gradient . . . 17
- 3.4 Strategies for System Architecture Optimization . . . 18
  - 3.4.1 Global Exploration . . . 19
  - 3.4.2 Nested Optimization. . . . 19
  - 3.4.3 Decision Chain Optimization . . . 20

4 Methodology and Implementation 22
- 4.1 Algorithm Implementation . . . 22
  - 4.1.1 Problem Definition . . . 22
  - 4.1.2 Genetic Algorithm . . . 23
  - 4.1.3 Bayesian Optimization. . . . 23
  - 4.1.4 DDPG Reinforcement Learning Algorithm . . . 24
- 4.2 Strategy Implementation . . . 24
  - 4.2.1 Global Exploration . . . 24
  - 4.2.2 Nested Optimization. . . . 25
  - 4.2.3 Decision Chain Optimization . . . 26

4.3 Performance Metrics . . . 26
4.3.1 Generational Distance Plus and Inverted Generational Distance Plus . . . 26
4.3.2 Hypervolume . . . 27
4.3.3 Non-Dominated Improvement Ratio . . . 28
4.3.4 Architecture Gini Index . . . 29

5 Algorithm Test Cases 30
5.1 Airfoil Optimization Problem . . . 30
5.1.1 Problem Statement . . . 30
5.1.2 Algorithm Setup . . . 30
5.1.3 Optimization Results. . . . 31
5.2 Mixed-Discrete ZDT1 Problem . . . 33
5.2.1 Problem Definition . . . 33
5.2.2 Experimental Setup . . . 33
5.2.3 Results and Analysis . . . 34
5.2.4 Summary of Findings . . . 36

6 Goldstein Problem 37
6.1 Multi-Objective Goldstein Problem . . . 37
6.1.1 Problem Formulation . . . 37
6.1.2 Strategy Setup . . . 39
6.1.3 Optimization Results. . . . 39
6.1.4 Comparison of Results . . . 41
6.2 Tunable Goldstein Problem . . . 42
6.2.1 Experimental Setup . . . 42
6.2.2 Results and Analysis . . . 43

7 Aileron Optimization 48
7.1 Problem Formulation . . . 48
7.2 Strategy Implementation . . . 49
7.2.1 Global Exploration Strategy . . . 49
7.2.2 Nested Optimization Strategy . . . 50
7.3 Optimization Results . . . 51
7.3.1 Reference Design of Experiments: Nested Random Sampling . . . 51
7.3.2 Global Exploration . . . 53
7.3.3 Nested Optimization. . . . 54
7.4 Comparison of Results . . . 56
7.4.1 Optimization Performance. . . . 56
7.4.2 Exploration-Exploitation Trade-Off . . . 60

8 General Discussion 62
8.1 Strategy Effectiveness . . . 62
8.2 Algorithm and Strategy Implementation . . . 62
8.2.1 Algorithm Implementation . . . 63
8.2.2 Strategy Implementation. . . . 63
8.3 Potential Connections to Other Technologies . . . 65
8.4 Strategy Recommendation Flowchart . . . 65

9 Conclusions & Recommendations 67
9.1 Conslusions. . . . 67
9.2 Recommendations . . . 69

A Algorithms 71

B Example of the Nested Optimization Strategy 74

C XDSMs of the Optimization Strategies Applied to the Aileron Problem 77

References 80

List of Figures

1.1 Components of the aileron system. . . . 2

2.1 The problem-solving process in systems engineering. . . . 7

3.1 Illustration of regular simplex encoding for categorical variables. . . . 13
3.2 General flow diagram for a genetic algorithm. . . . 15
3.3 General flow diagram for Bayesian optimization. . . . 16
3.4 Flowchart for the DDPG reinforcement learning algorithm. . . . 18
3.5 Generic XDSM representation of the global exploration optimization strategy (with design vector imputation). . . . 19
3.6 Generic XDSM representation of the nested optimization strategy. . . . 20
3.7 Generic XDSM representation of the decision chain strategy. . . . 20

4.1 Illustration of the generational distance plus and inverted generational distance plus indicators. 27
4.2 Illustration of the hypervolume indicator. . . . 28
4.3 Illustration of the non-dominated improvement ratio indicator. . . . 28

5.1 Algorithm performance comparison on the maximization of an airfoil's lift-to-drag ratio. . . . 31
5.2 Comparison of the optimized airfoils. . . . 31
5.3 Pareto front of the ZDT1 problem. . . . 33
5.4 Inverted generational distance plus as a function of the percentage of continuous variables for the ZDT1 problem with 8, 32, 128, and 512 variables. . . . 35
5.5 Normalized hypervolume as a function of the percentage of continuous variables for the ZDT1 problem with 8, 32, 128, and 512 variables. . . . 35
5.6 Non-dominated improvement ratio as a function of the percentage of continuous variables for the ZDT1 problem with 8, 32, 128, and 512 variables. . . . 36

6.1 Pareto fronts of the Goldstein problem's individual architectures. . . . 38
6.2 Pareto front of the multi-objective Goldstein problem. . . . 38
6.3 Matrix representation of the Goldstein problem's design vectors to use as states in the decision chain framework. . . . 39
6.4 Objective space results for representative optimization runs with the different system architecture optimization strategies. All runs used the NSGA-II algorithm and were limited to 10,000 function evaluations. . . . 40
6.5 Non-dominated fronts for representative optimization runs with the different system architecture optimization strategies. All runs used the NSGA-II algorithm and were limited to 10,000 function evaluations. . . . 40
6.6 Multi-objective performance indicators for the proposed system architecture optimization strategies. The error bars mark a 95% confidence interval obtained from 10 optimization runs for each strategy and number of total evaluations. All runs used the NSGA-II algorithm. . . . 42
6.7 Generational distance plus as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. . . . 43
6.8 Normalized hypervolume as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. . . . 44
6.9 Non-dominated improvement ratio as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. . . . 45
6.10 Fraction of explored architectures as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. . . . 46

6.11 Fraction of explored architectures as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. 46

7.1 Objective space results for a random sampling DoE in the aileron optimization problem. 51
7.2 Random sampling DoE objective space results for the aileron problem grouped by skin, spar, and rib material library. 52
7.3 Random sampling DoE objective space results for the aileron problem grouped by total number of ribs. 53
7.4 Objective space results for a global exploration strategy run of the aileron problem using NSGA-II. 53
7.5 Objective space results for a global exploration strategy run of the aileron problem using Bayesian optimization. 54
7.6 Objective space results for a nested strategy run of the aileron problem using NSGA-II in the outer and inner level. 55
7.7 Objective space results for a nested strategy run of the aileron problem using NSGA-II in the outer level and Bayesian optimization in the inner level. 56
7.8 Non-dominated fronts for the aileron problem found by the different strategy-algorithm combinations. 58
7.9 Normalized hypervolume vs. number of function evaluations for different strategies and algorithm combinations in the aileron optimization problem. 59
7.10 Architecture Gini index vs. fraction of total architectures explored for the different strategy-algorithm combinations run on the aileron optimization problem. 61

8.1 Flowchart for the selection of system architecture optimization strategies. 66

B.1 Example of the nested optimization strategy's procedure. 76

C.1 Extended design structure matrix of the global exploration strategy implemented for the aileron optimization problem. 78
C.2 Extended design structure matrix of the nested strategy implemented for the aileron optimization problem. 79

List of Tables

5.1 Comparison of mean differences, t-statistics, and p-values for optimization algorithms vs. random sampling in the airfoil optimization problem. 31
5.2 Average execution times of the tested algorithms in the airfoil problem. 32
5.3 Validation of optimization results against published results for the same problem. 32

6.1 Variables of the Goldstein function optimization problem. 37
6.2 Architecture number definition for the Goldstein problem. 38

7.1 Variables of the aileron optimization problem. 48
7.2 Summary of optimization results for the aileron problem. 56
7.3 Non-dominated points for the aileron problem for optimization runs with different combinations of strategies and algorithms. 58

B.1 Variables of the example problem. 74

Nomenclature

Abbreviations

Abbreviation	Definition
ATC	Analytical target cascading
ASF	Achievement scalarization function
BO	Bayesian optimization
CFD	Computational fluid dynamics
CST	Class-shape transformation
DDPG	Deep deterministic policy gradient
DoDAF	Department of Defense Architecture Framework
DoE	Design of experiments
DRL-MOA	Deep reinforcement learning multi-objective algorithm
EHVI	Expected hypervolume improvement
EI	Expected improvement
FEM	Finite element method
GA	Genetic algorithm
GD	Generational distance
GD+	Generational distance plus
HV	Hypervolume
IGD+	Invertd generational distance plus
KBE	Knowledge-based engineering
MDAO	Multidisciplinary design analysis and optimization
MDM	MultiDisciplinary Modelers
MDO	Multidisciplinary design optimization
NSGA-II	Non-dominated sorting genetic algorithm II
OMG	Object Management Group
PI	Probability of improvement
PSO	Particle swarm optimization
REMBO	Random EMbedding Bayesian Optimization
RL	Reinforcement learning
SAO	System architecture optimization
SE	Systems engineering
SoS	System of Systems
THiMP	Tunable hierarchical meta problem
TOGAF	The Open Group Architecture Framework
UAF	Unified Architecture Framework
XDSM	Extended design structure matrix
ZDT1	Zitzler, Deb, and Thiele multi-objective test problem # 1

Symbols

Symbol	Definition
A	Obtained non-dominated set
C_d	Section drag coefficient
C_m	Section moment coefficient
C_l	Section lift coefficient

Symbol	Definition
$\mathscr{C}$	Continuous design space
$\mathscr{D}$	Discrete design space
d^+	Adjusted distance
$\boldsymbol{f}$	Objective function
G	Gini index
Ma	Mach number
n_{arch}	Number of architectures
n_{category}	Number of possible categories in a categorical variable
n_{crit}	Critical amplification factor
n_f	Number of objectives to evaluate
n_{hinges}	Number of hinges
$\boldsymbol{r}$	Reference point
r	Reward
Re	Reynolds number
s	State
s'	Next state
$\boldsymbol{s}$	Vector of design variables
$\boldsymbol{s}*$	Pareto optimal design variables
s_i	Design variable
$\bar{s}_i$	Upper bound of design variable
$\underline{s}_i$	Lower bound of design variable
t/c	Thickness-to-chord ratio
$\bar{y}$	Hinge spanwise position
Z	Reference non-dominated set
α	Angle of attack
η_0	Non-dominated improvement ratio

1

Introduction

The early phases of the design process of large-scale engineering systems include several decisions that determine the system's high-level configuration. These decisions include technology selections, material choices, and operational mode settings. The number of potential architectures that can arise at these early stages is immense [1]; the possible decisions compound each other to form a multiplicative surge of options [2]. As a result, system architecting typically restricts this large tradespace before any detailed analysis takes place. However, because most of a system's cost and performance are determined at these early design stages [3, 4], more effective ways of exploring and evaluating this vast decision space are crucial. This chapter introduces the work carried out in this research to identify, formulate, and implement optimization strategies useful for system architecting problems.

1.1. The Structure of an Aileron

An aircraft's aileron is a hinged control surface typically located on the trailing edge of the aircraft's wing, near its outboard edge. Ailerons control the aircraft's roll motion around its longitudinal axis, allowing it to bank during flight. When the pilot moves the control yoke to one side, the aileron on that end moves upward while the corresponding aileron on the opposite wing moves downward [5]. This differential movement generates a lift imbalance and causes the aircraft to roll in the desired direction. Adjusting the ailerons' positions allows the pilot to maneuver the aircraft laterally and maintain stability during various flight conditions.

The main components of an aileron, as shown in Figure 1.1, are the following:

- **Hinges:** Hinges are mechanical devices that attach the aileron to the main wing and allow it to pivot about a fixed point, facilitating its up-and-down movement to control the aircraft's roll motion.

- **Actuators:** Actuators provide the necessary mechanical force to move the control surfaces and control the aircraft's roll motion. They are responsible for converting input signals from the pilot or automated flight control systems into the physical movement of the aileron surfaces.

- **Skins:** Skins make up the outer surface of the aileron. They give the aileron its aerodynamic shape.

- **Ribs:** Ribs are vertical structures positioned chordwise and form the basic skeleton of the aileron, maintaining its basic shape. They also support the skins, assisting in the distribution of aerodynamic and control forces. In the aileron, ribs can be classified as *main ribs* or *secondary ribs*. The main ribs are located at the aileron's root and tip and behind all hinge and actuator brackets. Meanwhile, the secondary ribs are positioned in the spaces formed between the main ribs, also known as *bays*.

- **Spars:** Spars are the primary structural elements of the aileron, usually extending in the spanwise direction. They provide rigidity, strength, and form to the aileron. Spars also support the attached skins and ribs and carry bending and torsional loads.

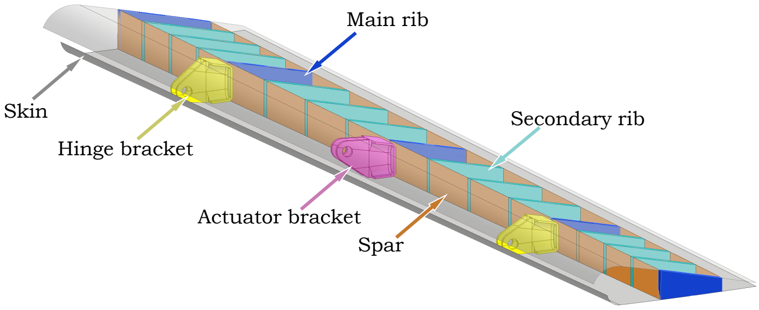


Figure 1.1: Components of the aileron system.

1.2. Introduction to Systems

An aileron's structure can be seen as a *system*, or a collection of connected elements and their relationships working together to achieve a common goal [6]. In general, the functionality or capability of a system is greater than that of the sum of its individual components [3, 7]. The aileron illustrates this concept since each of its components plays a specific role and interacts with the others to provide the aircraft with a means of lateral control.

Advances in engineering and technology have made it possible to design and build increasingly intricate systems. As in the aileron's case, these systems often include components related to multiple disciplines and require careful coordination to function effectively. Consequently, a methodical approach is required to formulate and design complex systems. This need motivates the branch of engineering known as *systems engineering*, a transdisciplinary field that guides the development of complex systems [6, 8]. Within systems engineering, the early stages of design responsible for decisions regarding the overall configuration are known as the *system architecting* phases [9]. System architecting can be framed as a decision-making process that identifies, analyzes, and selects a system's potential high-level configurations. From this perspective, a system's first few architectural decisions determine the majority of its cost and performance [3].

However, there is little knowledge of the system's behavior at these early design stages. Consequently, the architecting problem is subject to expert bias when choosing from a vast array of design options that generate a sizeable combinatorial design space [10]. As the design of a system increases its detail, this knowledge grows, but the ability to make impactful decisions decreases [11]. As a result, decision support tools for effective system architecting are needed.

1.3. Introduction to System Architecture Optimization

Among these decision support tools, *system architecture optimization* (SAO) stands out due to its ability to systematically search the design space and deal with conflicting objectives [12]. Nevertheless, significant challenges still have to be addressed in this area. For instance, systems architecting activities are usually detached from the concept development stage of the design process. Well-established tools such as multidisciplinary design analysis and optimization (MDAO) are typically implemented for cases where the system architecture has already been defined [13]. Connecting the systems architecting phases and the concept development stages of the systems design process is an ongoing field of investigation.

SAO aims to bring together the system architecting, concept development, and sizing stages of the systems design process [10]. Therefore, the high-level architectural and lower-level sizing variables are all considered within the same optimization routine. With system architecture optimization, the design process becomes more front-loaded. The goal of front-loaded design is to identify and understand system requirements, pain points, and goals as early as possible in the design process so that designers can create solutions that are more likely to meet those needs and succeed in the problems they seek to solve [14].

SAO makes the design process more front-loaded by bridging activities typically carried out one after the other. Traditionally, a system architecture is defined first, then its components are sized and optimized. However, this approach can lead to significant delays and bottlenecks because the assumptions made during the architecting phase may be revealed to be false or incorrect by the time concept development is carried out. Another scenario in which this segregated approach could become disadvantageous is when a particular architecture is selected and later developed without exploring other solutions in the tradespace that may have been better suited to fulfill the system's requirements and objectives. This case may happen due to a general lack of system knowledge or expert bias at the early design stages. Expert bias occurs when a system's design is influenced by its predecessors without considering alternatives that deviate from what is already known [12].

1.4. Challenges of System Architecture Optimization

When systems architecting and concept development are folded into the same process using SAO, the architectural design space is more thoroughly and rigorously searched, leading to lower uncertainty in later design activities and a better understanding of the system as a whole. However, some difficulties arise when system architecture optimization is used. For instance, system architecture optimization problems include both continuous and discrete variables, which complicates the use of common gradient-based optimizers. Furthermore, there may be a hierarchy in the problem's design space, such that some variables may influence whether other variables are present, or *active*, in the problem. For example, in the two-hinge aileron configuration depicted in Figure 1.1, the variable corresponding to a third hinge's position would have no meaning and thus be inactive. This potential hierarchy may also affect the problem's constraints or the bounds of some of its variables.

Additionally, because the architecting process involves recognizing and tracking system requirements and operating conditions, SAO problems tend to have multiple, potentially conflicting, design objectives. For example, in an aircraft's design, minimizing its cost and nitrogen oxide emissions might be desired. These goals are not aligned with each other, so instead of producing a single design, the optimization needs to produce a set of designs that reflects the trade-off between these objectives.

Finally, engineering optimization frequently uses black-box analysis tools to determine the values of objectives, constraints, and other quantities of interest. The black-box nature of these tools means there is no closed-form expression of the objectives and constraints available, and they may be expensive to evaluate. Furthermore, these tools may fail unexpectedly, for example, due to failing to achieve convergence. This failure introduces hidden constraints into the problem, meaning that there are zones in the design space that are not feasible even though no constraints explicitly show this. Even though this characteristic is not unique to system architecture optimization problems because it is common to many other engineering optimization cases, it is important to address it within the framework of SAO.

These four features of system architecture optimization problems call for specialized solution methods that can handle a mix of continuous and discrete variables, navigate hierarchical design spaces, accommodate multiple objectives, and adequately interface with black-box analysis tools that may have failure regions.

The non-hierarchical aspects of SAO problems can be effectively addressed with a variety of optimization algorithms including population-based approaches like genetic algorithms (GAs) [10, 15–17], surrogate-based techniques like Bayesian optimization (BO) [18–21], and, more recently, reinforcement learning (RL) procedures [22–24].

At the same time, the hierarchical design spaces of system architecture optimization problems require additional considerations regarding how to perform an effective search in these intricate spaces. Thus, a distinction is made between *algorithms* and *strategies* for system architecture optimization.

In this work, the following convention is used:

- An **optimization algorithm** is a method used to refine solutions based on specific criteria or heuristics iteratively.
- An **optimization strategy** is a higher-level approach or framework that employs one or more optimization algorithms to address specific problem structures or characteristics, especially related to the problem's hierarchical design space.

1.5. Research Objective and Question

Although previous work has identified and implemented families of strategies that address these hierarchies [1, 25–27], there is little consensus on the most suitable method to navigate them, especially in the multi-objective case. Therefore, this thesis focuses on formulating and implementing optimization strategies to address the idiosyncrasies of system architecture optimization problems. As a result, this research aims to propose, identify, and verify, through implementation, suitable strategies to address system architecture optimization problems. More specifically, this work aims to answer the following research question:

> **What are the most suitable optimization strategies for addressing the mixed-discrete, multi-objective, failure region, and hierarchical aspects of system architecture optimization problems?**

To better guide the search for these strategies, the research question is divided into the following sub-questions:

- *What are the main features of each strategy, and how can these be used to classify them?*
- *How can the different strategies be implemented using available software tools?*
- *How do the different strategies scale with the composition of the design space?*
- *What are the advantages and disadvantages of these strategies?*

Knowledge of how these strategies compare to each other will allow for an informed selection of the most convenient strategy for a given system architecture optimization problem, thus aiding the objective of front-loading the engineering problem-solving process and achieving a more advanced exploration in the early design phases.

1.6. THESIS OUTLINE

With the research objective and question in mind, this thesis is divided into the following chapters:

- **Systems Engineering and Systems Architecture:** This chapter provides an overview of the main concepts and features of systems engineering and its relationship to system architecting and the overall engineering problem-solving process.
- **System Architecture Optimization:** This chapter introduces the terminology, notation, and concepts related to system architecture optimization problems. It treats their mixed-discrete, hierarchical, multi-objective, and hidden constraints aspects in detail and presents a unified formalism that describes these problems based on their variables' types and roles. Furthermore, it identifies three algorithms suited to handle SAO problems and presents three types of strategies that deal with their hierarchical spaces in different ways.
- **Methodology and Implementation:** Here, the practical implementation aspects of both the optimization algorithms and the system architecture optimization strategies are discussed. Additionally, the metrics used to assess the performance of the optimization runs are introduced.
- **Algorithm Test Cases:** This chapter focuses on two test cases of non-hierarchical problems designed to evaluate different aspects of the selected optimization algorithms. First, a single-objective airfoil optimization problem is studied. This problem's results are compared to results from the literature, and some of the main characteristics of the different optimization algorithms are identified. The second test case in this chapter tackles a mixed-discrete version of the multi-objective ZDT1 benchmark problem [28] to determine how the different algorithms behave as a function of the number and type of variables in the problem.
- **Goldstein Problem:** The Goldstein problem proposed by Pelamatti [25] for system architecture problems is expanded with an additional objective, and the different optimization strategies are compared using the NSGA-II algorithm as their base. Next, the performance of each strategy as a function of the problem's degree of hierarchy and number of available architectures is studied with another modified version of the Goldstein problem that makes it possible to change these characteristics while maintaining others relatively constant.
- **Aileron Optimization:** This chapter delves into a real-world use case of system architecture optimization: the structural design of an aileron like the one shown in Figure 1.1. Different optimization strategies, using various algorithms to power them, are tested and compared to each other and validated against a reference design of experiments (DoE) that samples randomly from the design space.

- **General Discussion:** Given the results presented in the earlier chapters, this chapter carries out a higher-level discussion on the proposed optimization strategies' effectiveness, how the optimization algorithms and strategies were implemented, and potential connections to other technologies. The chapter closes with a decision-making flowchart that explains how to select a system architecture optimization strategy based on the insights drawn from this work.

- **Conclusions & Recommendations:** By summarizing the key findings and insights of the research, this chapter emphasizes the main contributions of this work to the field of system architecture optimization. The practical implications of these findings are outlined. Recommendations are made based on the comparative performance of the different algorithms and strategies tested throughout the study. Critical reflections on the research process and its limitations provide insights into areas that might benefit from further investigation.

2 SYSTEMS ENGINEERING AND SYSTEM ARCHITECTURE

Before exploring system architecture optimization, this chapter defines the broader subject of systems engineering (SE) that encapsulates system architecting activities and establishes the place of system architecture within the engineering problem-solving process.

2.1. SYSTEMS ENGINEERING

As mentioned in Chapter 1, a system represents interconnected elements and their associations, collectively striving to reach a shared purpose [6]. The field of *systems engineering* focuses on the design, creation, and management of complex systems. It can be defined as a meta-discipline that integrates all disciplines and covers the entire development process of a system, from concept to disposal, with a holistic approach that considers both technical and economic aspects [29].

The discipline of systems engineering has three key characteristics that differentiate it from other engineering disciplines [6]:

1. **SE takes a perspective on the system as a whole.** The most crucial aspect to be addressed by a systems engineer is the total operation of the system [30]. Systems engineering views the system from the outside and inside. Looking from the outside, SE analyzes the system's interactions with other systems and the environment. Meanwhile, looking from the inside, SE focuses on how the system's components interact to produce the desired result. However, systems engineering is concerned with more than just the engineering design of the system. Instead, it takes a more comprehensive approach to include external factors that affect this design, such as customer needs or requirements, regulations, operational capabilities, interfacing systems, and the operational setting, among others [6, 30].

2. **SE leads a system's development using a problem-solving process** [9]. An analysis and design procedure and an integration and verification method are carried out within this process. Ultimately, the problem-solving process results in a functional design that satisfies the user's needs and complies with all requirements. [31]. However, this process is not entirely quantitative because many design decisions at this stage involve domain knowledge, disciplinary experience, and quantities that are difficult to relate to each other [6].

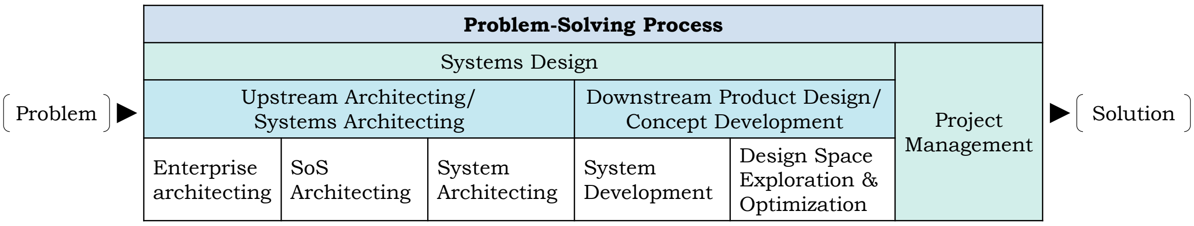
3. **SE's methods, techniques, and tools are highly interdisciplinary** [9]. Systems engineering integrates and links conventional engineering disciplines. Complex systems typically incorporate elements that draw from different fields, so it is up to the systems engineer to reconcile these disciplines and coordinate the different elements or subsystems such that they mutually support each other and the overall design is consistent [6].

Systems engineering is based on *systems thinking*, which treats problems explicitly as systems and emphasizes the importance of the whole and the significance of the relationships between said whole's constituents [3, 9, 32]. In a system like the aircraft aileron described in Chapter 1, systems thinking ensures that each component is designed and analyzed not in isolation but relating it to how it interacts with the other components and their function within the structure. For example, spars must be designed to withstand the bending and twisting forces during flight. However, the design process for spars cannot occur independently; it also needs to consider the loads transferred by the skins and their connection with the ribs.

2.2. System Architecture

As anticipated, systems engineering carries out a problem-solving process supported by systems thinking. This problem-solving process can be divided into two stages: *systems design* and *project management* [9]. The systems design phase poses fundamental questions about the problem and its possible solutions. On the other hand, the project management phase deals with bringing the designed system to reality. As a result, the project management step can be interpreted as the total administrative and organizational steps taken to plan, direct, supervise, and steer a project concerning scope, duration, and cost [9, 32].

This work mainly concerns the systems design phase of the problem-solving process. As its name indicates, a system design is created in this phase, considering possible options, strategies, and trade-offs. Haberfellner et al. divide this phase into two stages: *systems architecting* and *concept development* [9]. Analogously, Ciampa et al. call these two stages *upstream architecting* and *downstream product design* [33]. The upstream architecting stage is further divided into *enterprise architecting*, *system-of-systems (SoS) architecting*, and *system architecting*. In contrast, the downstream product design stage is partitioned into *system development* and *design space exploration and optimization* [33]. Figure 2.1 graphically summarizes the problem-solving process stages and substages.



Problem ▶

<table>
<tr><th colspan="6">Problem-Solving Process</th></tr>
<tr><td colspan="5">Systems Design</td><td rowspan="3">Project Management</td></tr>
<tr><td colspan="3">Upstream Architecting/
Systems Architecting</td><td colspan="2">Downstream Product Design/
Concept Development</td></tr>
<tr><td>Enterprise architecting</td><td>SoS Architecting</td><td>System Architecting</td><td>System Development</td><td>Design Space Exploration & Optimization</td></tr>
</table>

▶ Solution

Figure 2.1: The problem-solving process in systems engineering. Based on Figure 3.1 of [9] and Figure 5 of [33].

2.2.1. Form and Function

As seen in Figure 2.1, the first stage in the systems engineering problem-solving process's systems design step is *system architecting*. In this stage, the system's architecture is established according to given policy and needs at the enterprise level, goals and capabilities at the system-of-systems level, and scenarios and requirements at the system level [33]. An architecture is understood as the specification in which a system is organized. It includes its main parts, their connections, how they cooperate to fulfill requirements, and the driving principles behind their design and evolution [6, 34]. More succinctly, as Crawley et al. put it, a system architecture is "a mapping between form and function [3]."

In this context, *function* is the action, procedure, operation, or transformation a system carries out to produce or contribute to performance [3, 9]. Conversely, *form* is defined as a system's physical or informational manifestation that exists or has the potential to exist and is crucial to accomplishing a function. Before a function is executed, a form has to exist [3]. On the other hand, a form devoid of function has no value [9].

The main objective of systems architecting is to introduce an architecture that connects the objectives, needs, and requirements derived from a problem to a potentially feasible solution [3]. However, the product of a system architecting process is not a final system design that can be built with no further steps. Instead, the systems architect delivers a set of *abstracted* systems designs that are refined during the concept development (or downstream product design) step of the systems design stage in the problem-solving process [35]. The downstream product design refinement may eventually reveal that the selected architecture was unfeasible.

2.2.2. System Architecting Frameworks

Because architecting takes place at the beginning of the problem-solving process, the system architecture team has extensive freedom to build and integrate products. This freedom creates a challenge regarding the standardization of systems architecting development. Several architecture frameworks have been established to address this challenge. These aim to homogenize the architecture design procedure and the end products associated with architectures [6]. However, this does not mean frameworks produce “common templates” for system architecture design. Instead, they are sets of conventions or standards that dictate principles, practices, conventions, and approaches for developing and describing a system architecture [8]. Architecture frameworks typically offer a methodology, different viewpoints (conventions for data definition and exchange), a meta-model or ontology (a common reference data model for describing the system architecture), and a glossary (a collection of standard terms and their definitions) [36]. Examples of commonly used system architecture frameworks include DoDAF (Department of Defense Architecture Optimization Framework), developed by the United States Department of Defense; TOGAF (The Open Group Architecture Framework), developed by The Open Group; and the OMG Unified Architecture Framework (UAF), developed by the Object Management Group [6, 29].

Regardless of the framework chosen to design a system architecture, the end goal of the architecting process remains the same: to create a solution principle for the problem at hand [9]. The system architecture design process can be interpreted as a set of decisions the architect takes based on models, experience, intuition, stakeholder requirements, regulations, budget, and many other factors. Consequently, the architect must be adequately equipped with a toolkit that supports the decision process to allow a sensible exploration of the architecture *tradespace*, or set of available architectures [3, 37].

2.2.3. Obstacles of System Architecting

Architects face three significant obstacles that can severely restrict a thorough (and productive) search of the tradespace:

Combinatorial Explosion of Alternatives The architecture tradespace usually includes several variables of a discrete nature. As a result, the number of architectures to explore can be subject to a “combinatorial explosion of alternatives”, in which the number of candidate architectures makes it intractable to realistically conduct an exhaustive search for the best one according to a given metric [2, 21].

Knowledge Paradox Although decisions taken during the architecting phase significantly impact the design and performance of the final system, there is a large degree of uncertainty regarding the suitability of these decisions concerning the design objectives. This uncertainty emerges because, during early design stages, the models and available knowledge of the system’s behavior are less detailed and precise than they are further along the design process. Additionally, introducing architectural changes late in the design process can be extremely difficult or costly [12, 38], resulting in a “knowledge paradox”: as design knowledge increases when a system’s design becomes more detailed, the ability to act upon that knowledge decreases [11].

Multiple Design Objectives Selecting a system architecture results from evaluating several, often conflicting or interdependent, criteria [39]. This multitude of criteria implies that performing an adequate tradespace investigation and analysis is a complex process that requires advanced tools to perform design space explorations and multi-objective optimization.

2.3. System Architecture Optimization

As previously discussed, systems architecting involves selecting a system configuration to solve a given problem. Because many architectures may provide satisfactory solutions to the problem at hand, the question of which one is the *best* arises naturally. In this context, *system architecture optimization* (SAO) emerges as an attempt to solve this dilemma. SAO employs methods from numerical optimization to search the tradespace for the architectures that best fulfill one or more design objectives determined by the requirements and expectations of various stakeholders [3].

For instance, Villeneuve and Mavris show a methodology to quickly evaluate and select system architectures for launch vehicles using Monte Carlo sampling called RASAC [38]. Similarly, McManus et al. develop a procedure to rapidly identify the best architectures in a tradespace given a set of requirements and objectives called MATE-CON [37]. However, these approaches only work well for a relatively small number of architectures because the first step of the process is to enumerate all possible architectures. These sampling methods become intractable when the architecting problem involves thousands or more possible architectures.

Furthermore, unlike traditional optimization problems in which the design space is fixed, system architecture tradespaces can make architecture optimization subject to varying design spaces depending on the kind of architecture selected. Abdelkhalik gives an overview of system architecture optimization and introduces the notion of variable design spaces, in which the number of design variables is itself a design variable [40]. A more general perspective of this feature of system architecture optimization problems is that of *hierarchical* design spaces, where one variable can dictate whether other variables are *active* in the design [41, 42].

As mentioned earlier, SAO problems include certain features that set them apart from basic optimization problems. Bussemaker et al. identify the following characteristics for system architecture optimization problems [41]:

- **Black-box evaluations:** The objective and constraint functions are non-linear and challenging to represent in a closed form, meaning that obtaining or calculating derivative information may be computationally expensive or complicated. This difficulty arises because several black-box analysis tools may contribute to the evaluation functions.
- **Mixed-discrete design spaces:** Some of the variables in the design space may be integer-valued or categorical, which complicates the use of gradient-based optimizers.
- **Multiple objectives:** Requirements from different stakeholders may translate to different competing objectives.
- **Hierarchical design spaces:** Some design variables may depend on others. Therefore, in some architecting problems, the size and composition of the design space may change depending on the value of certain design variables.
- **Hidden constraints:** There may be *hidden constraints* present in the design space. Constraints not explicitly given to the optimizer, such as a requirement of convergence in an analysis tool, may affect the optimization's performance.

As a result, specialized optimization strategies incorporating the above features are required to carry out system architecture optimization effectively. With system architecting now well understood in the context of the engineering problem-solving process and a notion of the general challenges faced by system architecture, Chapter 3 will introduce the formalism used throughout this work to tackle system architecture optimization problems.

3

System Architecture Optimization

Chapter 2 already anticipated that system architecture optimization is a decision support tool that can address the inherent challenges of system architecting by systematically searching the design space. This chapter gives a formulation of the system architecture optimization problem and discusses its features in depth. It also introduces both algorithms and strategies that can handle SAO problems.

3.1. Formulation of the System Architecture Optimization Problem

Optimization is the selection of the most suitable solution out of a set of candidate solutions, the *design space* (which is delimited by given *bounds*), according to a given criterion or set of criteria encoded by an *objective function* and satisfying certain *constraints* [43]. From a design standpoint, optimization is an automated method of determining the best design to address a specific issue [44]. Before establishing the general problem statement for system architecture optimization, the fundamental concepts of *design space*, *bounds*, *objective function*, and *constraints* mentioned previously are defined as [45]:

- **Design Space:** The *design space* $\mathscr{S}$ is a set whose elements are vectors $\boldsymbol{s}$ of *design variables* that numerically encode the characteristics of a particular solution.

- **Bounds:** Each component s_i of $\boldsymbol{s}$ may be delimited by upper and lower *bounds* $\bar{s}_i$ and $\underline{s}_i$, respectively, such that $\underline{s}_i \leq s_i \leq \bar{s}_i$.

- **Objective Function:** The function that maps design vectors to a quantitative metric that determines the goodness or fitness of a given design is called the *objective function* and is denoted by $\boldsymbol{f}$. The objective function typically maps elements of the design space $\mathscr{S}$ to the real set. However, the objective function can include more than one quantity of interest, in which case it is denoted as a vector function $\boldsymbol{f} : \mathscr{S} \rightarrow \mathbb{R}^{n_f}$, where n_f is the number of objectives to be evaluated.

- **Constraints:** Many optimization problems are subject to *constraints*, or functions of the design variables that have to be restricted in some way to ensure the feasibility of the design. Consequently, a *feasible region* can be identified within the design space. Design vectors within this region satisfy all constraints. There are two types of constraints: equality and inequality. *Equality constraints* restrict a function to a fixed value and can be formulated as $h(\boldsymbol{s}) = 0$. *Inequality constraints* require a function to be less than or equal to a given value and can be specified as $g(\boldsymbol{s}) \leq 0$.

With the definitions given above, the SAO problem can be formally stated as follows:

$$\begin{array}{lll}
\text{Minimize} & \boldsymbol{f}(\boldsymbol{s}) & \boldsymbol{f} : \mathscr{S} \rightarrow \mathbb{R}^{n_f} \\[2ex]
\text{With respect to} & \boldsymbol{s} \in \mathscr{S} & \\[2ex]
\text{Subject to} & \underline{s}_i \leq s_i \leq \bar{s}_i & i = 1, \ldots, n_s \\
 & g_j(\boldsymbol{s}) \leq 0 & j = 1, \ldots, n_g \\
 & h_l(\boldsymbol{s}) = 0 & l = 1, \ldots, n_h.
\end{array} \tag{3.1}$$

The nature of the design space $\mathscr{S}$ and the multi-objective function $\boldsymbol{f}$ will be explored in the following section.

3.2. Features of System Architecture Optimization Problems

As mentioned in Section 2.3, SAO problems include certain features that set them apart from basic optimization problems [41, 46]. For instance, the design spaces of these problems tend to be *mixed-discrete* and *hierarchical*, which complicates the use of gradient-based optimizers. Additionally, because requirements from different stakeholders must be considered during the architecting phase of the systems design process, different and potentially competing objectives are included in the optimization. Finally, engineering simulation and analysis tools are typically used in the constraint and objective evaluations of system architecture optimization problems. These black-box functions often make the problems challenging to represent in a closed form. They may introduce *hidden constraints* related to the failure and success of the analysis tools.

This section discusses these four features of SAO problems. Section 3.2.1 examines these problems' mixed-discrete design spaces. Then, Section 3.2.2 explains their hierarchical nature. Afterward, Section 3.2.3 introduces the notion of multi-objective optimization, and Section 3.2.4 goes over the concept of hidden constraints that may arise in these problems.

3.2.1. Mixed-Discrete Design Spaces

The early stages of design for complex engineering systems involve making many decisions that will ultimately determine the system's final architecture. These decisions can include discrete design variables such as material selection and continuous variables like structural sizing parameters. As a result, the design problem can be framed as an optimization problem with objective and constraint functions that depend on continuous and discrete variables, such as integer values or categorical variables. Therefore, a systematic methodology is needed to explore the large design space and support quantitative trade-off analyses to make informed decisions [46].

When the design space $\mathscr{S}$ of Equation 3.1 is mixed-discrete, $\mathscr{S}$ is represented as the Cartesian product of a continuous design space $\mathscr{C}$ and a discrete design space $\mathscr{D}$, such that $\mathscr{S} = \mathscr{C} \times \mathscr{D}$. In this case, $\mathscr{C}$ is a subset of $\mathbb{R}^{n_c}$, with $\mathscr{C} \subseteq \mathbb{R}^{n_c}$, where n_c denotes the number of continuous variables. Conversely, $\mathscr{D}$ is defined as a subset of the Cartesian product of integer sets, or $\mathscr{D} \subseteq \mathbb{Z}^{n_d}$, with n_d being the number of discrete variables [46].

Types of Discrete Variables

Three types of discrete variables can compose the discrete search space $\mathscr{D}$, namely *binary*, *integer*, and *categorical* [45]:

- **Binary Variables:** Binary (or Boolean) variables can only take on the values of 0 or 1. They can represent either true or false choices, such as whether to include a particular component in a system architecture.
- **Integer Variables:** Integer variables can only take on integer values and represent quantities associated with whole numbers, such as the number of secondary ribs in an aileron's structure.
- **Categorical Variables:** Categorical variables are limited to a distinct set of non-numerical values. For instance, categorical variables can model dates, names, or choices only from a given set of options.

Encoding Techniques for Categorical Variables

Categorical variables have to be encoded to be effectively treated in optimization problems. There are many ways to encode categorical variables in numerical representations [47], but three common encoding techniques are further discussed: *integer encoding*, *binary-constraint encoding*, and *regular simplex encoding*.

Integer Encoding In *integer encoding*, a unique integer represents each possible value of the categorical variable [21, 48]. For instance, using integer encoding, the three values of a material selection discrete variable (`thermoplastic`, `thermoset`, and `thermoset with hoyecomb`) would be represented as follows:

- `thermoplastic`: 0
- `thermoset`: 1

- `thermoset with honeycomb`: 2

One possible disadvantage of integer encoding is that it introduces a notion of order to otherwise unordered variables, which the optimization algorithm can erroneously exploit. Consequently, optimization algorithms or approximation methods that rely on calculating distances between design points in the design space can produce drastically different results depending on the arbitrary integer values assigned to each categorical variable [48].

Binary-Constrained (One-Hot) Encoding Another common way to encode discrete variables for machine learning and optimization problems is to use *binary-constrained* or *one-hot* encoding. In this approach, each possible value of the discrete variable is represented as a unique binary vector, with a "1" corresponding to that value and a "0" in all other positions. For example, the material selection variable mentioned earlier can be one-hot encoded as follows:

- `thermoplastic`: $[1,0,0]$
- `thermoset`: $[0,1,0]$
- `thermoset with honeycomb`: $[0,0,1]$

An additional equality constraint for each encoded variable appears when this encoding is used for optimization problems. These constraints ensure that only one of the possible "options" for each of the discrete variables is used [45]:

$$\sum_{i=1}^{n} s_i = 1, \tag{3.2}$$

where i denotes the indices of one binary-constrained variable in the design vector, and n represents the number of discrete choices available for the variable. In the example above, the constraint described in Equation 3.2 allows feasible designs such as `thermoplastic`: $[0,1,0]$ but rejects nonsensical points like $[1,1,0]$.

An advantage of binary-constrained encoding is that it does not introduce a cardinality to categorical variables. However, one-hot encoding significantly increases the number of variables in the design space. Nevertheless, despite the increase in the problem's size, the binary problem can become simpler to solve than its integer-encoded counterpart when using optimization techniques such as branch-and-bound [45]. Additionally, specialized design space partitioning methods have been developed to deal with one-hot encoded variables efficiently [49].

Regular Simplex Encoding An approach that slightly reduces the number of parameters required to model categorical variables with respect to binary encoding while addressing the arbitrary ordering issue of integer encoding is *regular simplex encoding*. In this method, the n_{category} alternatives of each categorical variable are mapped as the vertices of a *regular simplex*[1] in a $(n_{\text{category}} - 1)$ space, such that all potential values are equidistant [47, 48].

For the material selection categorical variable described above, the three alternatives `thermoplastic`, `thermoset`, and `thermoset with honeycomb` can be encoded as the vertices of an equilateral triangle as shown in Figure 3.1 with the following coordinates:

- `thermoplastic`: $(0.71, 0.96)$
- `thermoset`: $(0.8, 0.2)$
- `thermoset with honeycomb`: $(0.1, 0.5)$

This encoding transforms the categorical variables into two continuous variables. When the optimizer proposes a combination of these two continuous variables, the corresponding category is decoded as the vertex of the simplex closest (by Euclidean distance) to the given combination. In Figure 3.1, the shaded areas represent the coordinates that would be decoded as the corresponding category shown in the same color.

The selection of vertices for each category is arbitrary as long as the resulting simplex is regular [47]. However, care must be taken to set the bounds of the new continuous variables such that the area corresponding

[1] The generalization of a regular triangle in higher dimensions.

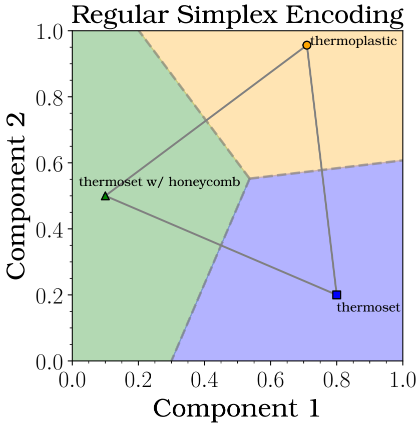


Figure 3.1: Illustration of regular simplex encoding for categorical variables.

to each category in the parameterized space is equal. This way, bias toward any of the available options is avoided. Additionally, it is worth noting that categorical variables with several possible options may not be suited for regular simplex encoding, as the notion of Euclidean distance can break down due to the "curse of dimensionality": in high dimensions, the distances between points start converging, meaning that the concept of nearness or farness loses its meaning [50].

3.2.2. Hierarchical Design Spaces

System architecting is a decision-making process [3]. In the context of SAO, this means that the design space is not only mixed-discrete as discussed in Section 3.2.1, but it also contains variables whose values affect other variable's presence, bounds, or constraints. This variability in the design space's size or composition introduces a notion of *hierarchy*, in which some of the problem's variables directly affect others [1, 10, 21, 51].

The structural optimization of an aircraft's aileron exemplifies this hierarchy in the design space. In this problem, the designer could consider configurations characterized by the number of hinges n_{hinges} and their respective spanwise positions $\{\bar{y}_1, \ldots, \bar{y}_{n_{\text{hinges}}}\}$. If the possible numbers of hinges are $n_{\text{hinges}} = 2$ and $n_{\text{hinges}} = 3$, the *active* design variables in each instance would be different:

- In case $n_{\text{hinges}} = 2$, the vector of active variables would be $[n_{\text{hinges}}, \bar{y}_1, \bar{y}_2]$.
- In case $n_{\text{hinges}} = 3$, the vector of active variables would be $[n_{\text{hinges}}, \bar{y}_1, \bar{y}_2, \bar{y}_3]$.

This example shows how, in system architecture problems, the choice of one variable can affect the size and composition of the design space. With the goal of handling these hierarchical design spaces, Audet et al. [52] present a framework that identifies the variables of a general system architecture optimization problem by their type and role. A variable's *type* indicates the kinds of values it can take on and whether there is a notion of ordinality among the variable's possible values. Variable types include *real*, *integer*, and *categorical*, as discussed in Section 3.2.1. The *role* of a variable, on the other hand, refers to how it affects or is affected by the values of other variables in the problem. A variable can have one of three different roles [52]:

- **Neutral**: The variable is always active in the problem and does not affect other variables.
- **Meta**: The value of the variable *decrees* whether one or more variables in the problem are active.
- **Decreed**: The variable is conditionally active depending on the values of one or more meta variables.

Even though Audet et al.'s formalism establishes that variables can only have one role, it is possible to find hierarchical spaces where a decreed variable is also a meta variable. For instance, in an aileron design problem, the variable that determines the number of secondary ribs in a given bay is both decreed and meta. It is decreed by the choice of number of hinges because the existence of the bay where the ribs will be placed may be subject to how many hinges are present in the aileron. At the same time, the number of ribs in a given bay affects the number and layout of the skin and spar material zones, so this variable is meta in the sense that it affects which material zone sizing variables are active.

3.2.3. Multiple Objectives

As shown in Equation 3.1, system architecture optimization problems may include multiple, potentially conflicting, objectives. Multiple objectives arise in these problems because, in the architecting phase of systems design, multiple stakeholder expectations and requirements must be considered.

For example, in the design of an aileron, an engineer might consider its structural weight and the cost it would take to produce. Additionally, one design might be required to fulfill more than one function, like a flaperon, an aircraft moveable that combines an aileron's control role with a flap's high-lift capabilities. This motivates the need for a multi-objective optimization framework where multiple objectives can be evaluated simultaneously, and optimal designs concerning these objectives can be identified [46].

Pareto Optimality

In a multi-objective optimization problem, the function of interest is a vector $\boldsymbol{f}$ with n_f components, each representing an individual objective to minimize. Consequently, there is a need to measure the optimality of a vector-valued objective function so that all its (possibly conflicting) components are considered. The concept of Pareto optimality addresses this issue. A design point $\boldsymbol{s}^*$ is *Pareto optimal* (or *non-dominated*) if and only if there does not exist any other point $\boldsymbol{s}$ in the design space such that $f_i(\boldsymbol{s}) \leq f_i(\boldsymbol{s}^*)$ for all f_i in $\boldsymbol{f} = \left[f_1, \ldots, f_{n_f}\right]^T$. In other words, for a point to be Pareto optimal, there must be no other point in the design space that improves at least one component of $\boldsymbol{f}$ while leaving the others unchanged [53]. The set of Pareto optimal points is called the *Pareto set*, and the corresponding vector objective function values are called the *Pareto front* [54].

3.2.4. Hidden Constraints

The analysis tools in many engineering optimization problems are simulation-based: the constraints or objectives are evaluated with specialized computational tools. For instance, an aerodynamic shape optimization might rely on computational fluid dynamics (CFD) to calculate drag, and a structural optimization might use finite element method (FEM) tools to calculate stresses and strains within the structure.

These tools enable a detailed understanding of the problem domain, but they also introduce a unique class of constraints, often termed as *hidden* or *implicit* constraints [55]. They are not explicitly defined in the problem formulation, but they inherently exist due to the nature of the computational simulation tools or real-world considerations.

A hidden constraint is considered violated when a simulation does not converge or fails to produce a meaningful result, resulting in unevaluated objectives or constraints [41]. These hidden constraints, while not defined directly, significantly impact the feasible design space and the performance of the optimization algorithm. As such, addressing them often requires incorporating robustness strategies within the optimization process. This might involve adaptive sampling techniques, surrogate modeling, or advanced numerical methods capable of handling nonevaluable points.

Some optimization algorithms can handle hidden constraints by using extreme barrier approaches [41, 56], while others can incorporate surrogate modeling techniques that predict which design points are nonevaluable [55, 57].

3.3. Algorithms for System Architecture Optimization Problems

Optimization algorithms can be classified as *gradient-based* and *gradient-free*. Gradient-based algorithms make use of derivative information in order to perform an efficient local search. These methods use the gradient of the objective function, a vector of partial derivatives, to iteratively improve the solution [40]. If the objective function is defined and differentiable, it decreases fastest along the negative direction of its gradient. Gradient-based algorithms leverage this behavior to take small steps along the design space in a direction guided by the objective function's gradient. These methods have widespread use in many different applications [58].

However, these algorithms may get trapped in local optima when the objective function is non-convex. Furthermore, these methods may not be adequate when the function is not continuous because derivatives become undefined or hard to compute. Consequently, gradient-based algorithms are not directly suited to solve system architecture optimization problems because they are ineffective in dealing with their mixed-discrete nature.

On the other hand, gradient-free optimization algorithms are a class of optimization methods that do not calculate gradients as a part of their iterative process. These algorithms are typically used when the function to be optimized is complex and does not have a smooth gradient or when the gradient is difficult or expensive to calculate [44]. Instead of using gradients, these algorithms typically use a combination of local search and heuristics to explore the design space and find reasonable solutions [59].

With this in mind, three different algorithms were chosen to study their potential to solve mixed-discrete, multi-objective problems with hidden constraints. The chosen methods are genetic algorithms (GAs), Bayesian optimization (BO), and the deep deterministic policy gradient (DDPG) deep reinforcement learning algorithm. The rationale for selecting these algorithms stems from their distinct approaches to navigating the design space. While GAs are population-based, Bayesian optimization uses a surrogate model of the objective functions and constraints to guide the search. DDPG leverages deep learning to approximate an optimal policy that produces the desired designs. Despite their radically different approaches, they are all suitable for addressing the complexities of mixed-discrete, multi-objective optimization problems with hidden constraints. Furthermore, although they cannot directly address the hierarchical design spaces of system architecture optimization problems, they can all be used to power strategies specialized in dealing with this type of design space.

3.3.1. Genetic Algorithms

Genetic algorithms are optimization techniques in which designs are represented as individuals in a population. Throughout several generations, these individuals are subjected to *mating* operations that produce new individuals for the following generations [60]. The three basic mating operations are *selection*, *crossover*, and *mutation* [61, 62]:

- **Selection:** The members of the population that will reproduce, or *parents*, are chosen according to their fitness.
- **Crossover:** Offspring designs are created using the combined characteristics of two or more parents.
- **Mutation:** Small random changes are applied to the offspring to maintain genetic diversity and explore new areas of the design space.

After the mating process, a survival operator determines which individuals will proceed to the next generation to repeat the process. Figure 3.2 shows a general flow diagram for a genetic algorithm. A more detailed explanation of the genetic algorithm's procedure can be found in Algorithm A.1. The random initialization of the first population and the mating operations in genetic algorithms make them inherently stochastic.

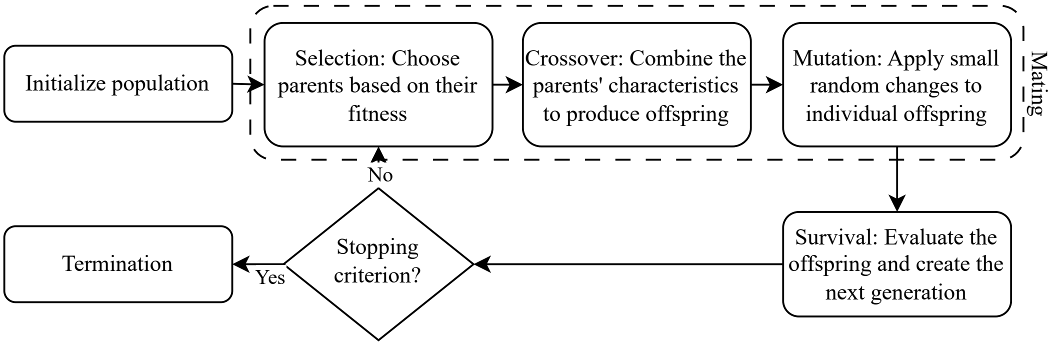

Initialize population
Selection: Choose parents based on their fitness
Crossover: Combine the parents' characteristics to produce offspring
Mutation: Apply small random changes to individual offspring
Mating
No
Stopping criterion?
Yes
Termination
Survival: Evaluate the offspring and create the next generation

Figure 3.2: General flow diagram for a genetic algorithm.

The genetic algorithm is well-equipped for several aspects of system architecture optimization problems. For instance, it can handle the mixed-discrete nature of these problems by using custom crossover and mutation operators [63–65]. Additionally, adaptations of the genetic algorithm known as multi-objective genetic algorithms, such as the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [66] used later in this work, use non-dominated sorting to include multiple objectives in the optimization process[2]. Finally, the possibility of having failed constraint or objective evaluations is handled well by genetic algorithms. These failed solutions can be set up to be rejected automatically in the survival operator with an extreme barrier approach that assigns $+\infty$ or `NaN` to the objectives or constraints of failed evaluations [41]. As a result, the genetic algorithm and its variations are suitable candidates for powering strategies for system architecture optimization.

[2] See Algorithm A.2.

Although genetic algorithms are simple to implement, they can be data-intensive, possibly even more so than gradient-based methods with finite differences, since they may require large populations to sample the design space effectively. They may need several generations to converge toward an optimum.

3.3.2. Bayesian Optimization

Bayesian optimization is a general framework that employs Bayesian inference and probabilistic modeling to efficiently solve optimization problems with costly objective functions [67]. In this framework, the objectives and constraints are approximated with surrogate regression models that estimate their values and give a measure of the associated uncertainty [68]. A general flow diagram depicting the Bayesian optimization algorithm is illustrated in Figure 3.3. Additionally, Algorithm A.3 gives a more detailed overview of this procedure.

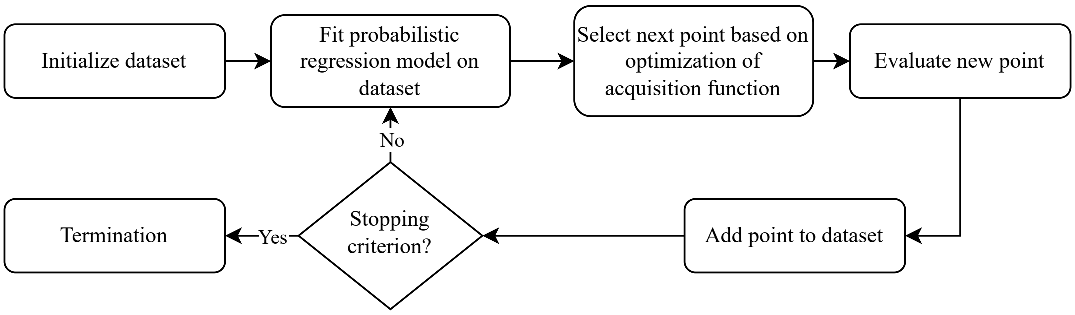

Initialize dataset
Fit probabilistic regression model on dataset
Select next point based on optimization of acquisition function
Evaluate new point
No
Termination
Yes
Stopping criterion?
Add point to dataset

Figure 3.3: General flow diagram for Bayesian optimization.

As shown in Figure 3.3, the main idea behind Bayesian optimization is to use a probabilistic surrogate model of the objectives and constraints to guide the selection of the next point to evaluate. This probabilistic model gives an estimate of the objective and constraints' expected values as well as their uncertainty. Bayesian optimization algorithms leverage the uncertainty quantified by the probabilistic model to balance exploration (searching regions of the parameter space where the function value is highly uncertain) and exploitation (focusing on regions where the objective function values are expected to be low).

However, it is necessary to produce an initial dataset before using such a probabilistic model in the Bayesian optimization procedure. Such a dataset can be obtained by carrying out a design of experiments (DoE), such as random sampling, factorial designs, or Latin hypercube samplings. After the DoE has been executed, the optimizer can use the dataset to fit its probabilistic model and subsequently carry out the optimization.

A standard regression model employed in Bayesian optimization is Gaussian process regression, also known as Kriging [69]. Gaussian process regression can both interpolate and extrapolate on an existing dataset. It uses a *kernel function* to represent the assumed spatial relationship between different data points and models the (assumed to be unknown) objective function and constraints [52]. In Bayesian optimization, the estimates and uncertainties the surrogate model provides are combined into an acquisition function optimized in every iteration to select the next point to evaluate [68]. Standard acquisition functions include the *probability of improvement* (PI) and *expected improvement* (EI). While optimizing the PI aims to maximize the probability that the new design point will improve on the current optimum, optimizing the EI aims to maximize the magnitude of the improvement from the current optimum to the objective function value provided by the new design point [67, 68]. Bayesian optimization is a flexible framework that can handle many aspects of system architecture optimization problems. For example, Bayesian optimization can manage mixed-discrete problems using specialized mixed-discrete kernels or relaxing the discrete variables into continuous latent variables [21, 70–73].

Furthermore, acquisition functions such as expected hypervolume improvement (EHVI) have been developed to deal with multi-objective problems [74, 75]. Finally, if it is likely that some design evaluations will fail, Bayesian optimization can include in its acquisition function a Gaussian classification model that determines whether a potential design point evaluation is likely to fail [56, 57]. Bayesian optimization algorithms are designed to efficiently search the design space with a small number of evaluations. This efficient search makes them potent options for problems where there might be a time or computational resource budget on the number of evaluations available for a given problem. Nonetheless, their performance relies on various hyperparameters, such as the choice of acquisition and kernel functions [68].

Bayesian optimization runs are non-deterministic because the dataset initialization is normally done randomly or stochastically. The acquisition function optimization process may use stochastic algorithms that employ random search directions or random initializations.

3.3.3. Reinforcement Learning — Deep Deterministic Policy Gradient

The final optimization algorithm studied in this work is the Deep Deterministic Policy Gradient (DDPG) reinforcement learning algorithm. A short overview of reinforcement learning is given before introducing DDPG to clarify some important concepts and provide definitions relevant to this field.

Overview of Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning that focuses on obtaining a *policy* that maps *states* to *actions* in order to maximize some *reward function* [76]. In this context, an artificial intelligence system, or *agent*, learns within an interactive *environment*. These terms are defined as follows [76, 77]:

- **Policy:** A policy is a plan that defines how the agent should act in a given state. It is a function that maps the state space to the action space.
- **State:** States represent the various conditions or situations that an agent can encounter within the environment. The set of all possible states is called the *state space*.
- **Action:** Actions represent an agent's possible moves or decisions in response to a given state. The set of all possible actions is called the *action space*.
- **Reward:** The reward quantifies the value or quality of taking a particular action in a given state. It provides feedback to the agent, guiding it to take actions favorable to the problem it aims to solve.
- **Agent:** The agent is the decision maker that interacts with the environment, making observations of its states and taking actions to learn a policy that maximizes the expected accumulated reward.
- **Environment:** The environment represents the external system with which the agent interacts. It reacts to the agent's actions by shifting to new states and providing feedback encoded in the reward function.

Typically, reinforcement learning environments are set up to satisfy the *Markov property*, meaning that the future state depends solely on the current state and action and not on the sequence of states that preceded it [76]. In other words, the system's dynamics must be memoryless, so all relevant information for its temporal evolution is encapsulated in the current state. Therefore, if a design optimization problem can be framed as a sequence of decisions that satisfy the Markov property, it could be solved with a reinforcement learning algorithm [22].

Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a reinforcement learning algorithm introduced by Lillicrap et al. for continuous control [78]. DDPG is designed for continuous action spaces: the actions an agent can make as part of the design process can be represented as vectors in a real coordinate space [78]. Additionally, the DDPG algorithm uses an actor-critic architecture, where the agent consists of an *actor model* and a *critic model*. The actor model determines the optimal policy by mapping states to actions, while the critic model establishes the value function of state-action pairs [79]. The learning process in DDPG happens through experience replay: as the agent interacts with its environment, state-action-reward-next state (s, a, r, s') tuples are stored in a buffer and later sampled randomly to update the actor and critic models. Another critical feature of DDPG is its use of target models. These models are copies of the actor and critic models with weights that are slowly updated during training to help stabilize the process [78]. DDPG also encourages off-policy action exploration [78]; the actions proposed by the actor are slightly altered with correlated noise sampled from an Ornstein-Uhlenbeck process [80]. Figure 3.4 presents a schematic representation outlining the general flow of the DDPG algorithm. A more comprehensive overview of this algorithm is given in Algorithm A.4.

DDPG and other continuous action space algorithms can deal with mixed-discrete problems by using continuous action spaces that result in mixed-discrete actions or by adapting the model to produce discrete outputs based on, for example, a softmax activation layer [81]. For the multi-objective aspect of system architecture optimization problems, DPPG and similar reinforcement learning algorithms can use a reward function that encodes the various objectives into a single scalar value or decompose the multi-objective problem

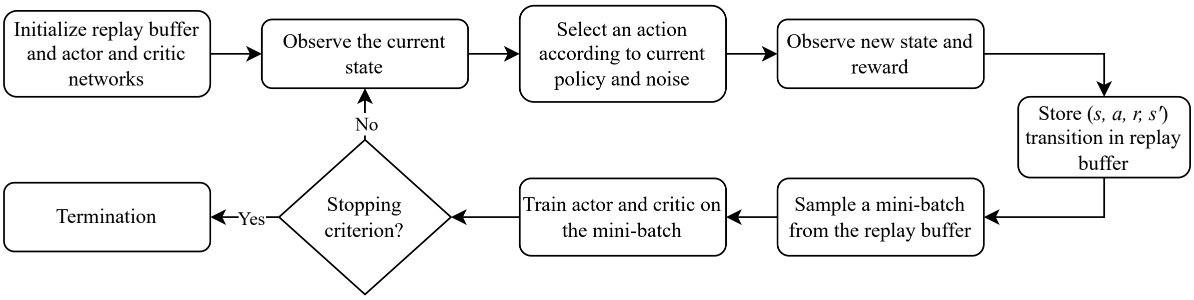

Initialize replay buffer and actor and critic networks
Observe the current state
Select an action according to current policy and noise
Observe new state and reward
Store (s, a, r, s′) transition in replay buffer
Sample a mini-batch from the replay buffer
Train actor and critic on the mini-batch
Stopping criterion?
No
Yes
Termination

Figure 3.4: Flowchart for the DDPG reinforcement learning algorithm.

into several scalar problems. The trained actor and critic models from one scalar problem can be a starting point for a similar problem, thus avoiding the need to re-learn policies from scratch [24, 82].

Regarding the possibility of failure regions in the problem's constraints or objectives, a large penalization can be incorporated into the reward function to discourage the agent from exploring designs near points whose evaluations have failed. The novel component that DDPG and other reinforcement learning methods bring to the solution of system architecture optimization problems is that, besides providing optimal solutions, they provide *policies* to produce new designs. If the environments in which the reinforcement learning agents act are designed carefully, these optimal policies can generalize to scenarios not encountered during training and still provide feasible design points under varying conditions [24]. However, it is worth noting that the stability of reinforcement learning training processes can be susceptible to several hyperparameters, including model architecture (types, numbers, orders, and activation functions of the neural networks' layers), learning rates, reward function structure, and state and action space definitions, among others [83]. Consequently, creating a suitable reinforcement learning environment and training procedure can become an architecture optimization problem of its own.

DDPG (and other reinforcement learning algorithms) are non-deterministic for various reasons:

- **Initialization of neural networks:** DDPG uses neural networks with weights that are typically initialized at random. Different initialization can lead to different training trajectories and final policies.
- **Exploration noise:** As discussed above, DDPG uses an inherently stochastic exploration noise to explore the action space more effectively.
- **Mini-batch sampling:** State-action-reward-new state tuples are randomly sampled from the replay buffer. Different samples can affect the gradient estimates and, consequently, the weight updates.
- **Stochastic optimization of the neural network parameters:** Commonly used optimizers to set the neural networks' parameters, such as Adam [84], introduce randomness in how they update the parameters. Adam, for instance, keeps running estimates of the first and second moments of the gradient and uses these to adapt the learning rates for each parameter. The initialization and update of these moments are sensitive to the order and exact values of the gradients, leading to variations in the training process.

3.4. Strategies for System Architecture Optimization

As discussed in Section 3.3, the genetic, Bayesian, and DDPG optimization algorithms are well-suited to handle the mixed-discrete, multi-objective, and hidden constraints aspects of system architecture optimization problems. However, these algorithms on their own cannot effectively address the hierarchical design spaces and the changes in active design variables, bounds, and constraints that come with them. As a result, *strategies* are needed to define how the hierarchical space will be searched. From the literature, three main families of strategies are identified as suitable frameworks with which to deal with the hierarchical spaces of system architecture optimization problems [1, 21, 22, 26]. They are:

- **Global Exploration:** All design variables are treated within a single optimization loop.
- **Nested Optimization:** The design variables are split into levels according to variable type or role. One optimizer controls the variables at each level.

- **Decision Chain Optimization:** The architecting problem is formulated as a sequence of decisions that gradually change the system's design.

3.4.1. Global Exploration

The *global exploration* strategy uses a single optimization loop to optimize all the design variables simultaneously [25]. This concurrence means, however, that a large portion of the variables observed by an optimizer may be inactive at any given iteration, making it difficult for the algorithm to select the next points to evaluate effectively [1]. For instance, a surrogate-based optimization method like Bayesian optimization may create an incorrect approximation of the design space based on erroneous attributions of changes in the objectives or constraints to variables that are not active [41]. Likewise, genetic algorithms employed in a global exploration strategy may carry out crossover and mutation operations on inactive variables that lead to evaluations of the same design, thus wasting computational resources [1, 25].

Adding design vector *imputation* to this naive approach in the global exploration strategy leads to enhanced performance. The idea behind imputation is to set the inactive variables to a constant value before performing function evaluations and returning the design vector to the optimizer. This way, no resources are wasted on evaluating duplicate design vectors [41]. When using surrogate-based techniques like Bayesian optimization, imputation of the design vectors also helps the regression model build a more accurate representation of the design space because fixing the inactive variables avoids attributing changes in the objectives and constraints to changes in these idle variables. This strategy has been successfully used to solve a multi-objective aircraft engine architecting problem [10, 85].

One disadvantage of the global exploration strategy is that it needs an explicit definition of the hierarchical space before starting the optimization run. This definition is necessary because the global design vector's size and component variables must be established before starting the optimization. Figure 3.5 shows a generic extended design structure matrix (XDSM) for the global exploration strategy with imputation.

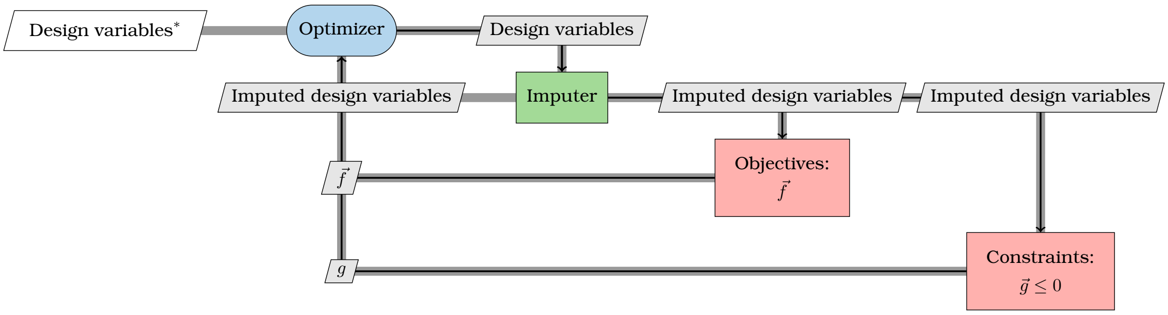


Figure 3.5: Generic XDSM representation of the global exploration optimization strategy (with design vector imputation).

3.4.2. Nested Optimization

In the *nested optimization* strategy, the variables of the design space are split into two or more levels according to their hierarchy or type [25]. Notable examples of this strategy include Frank et al.'s nested architecting procedure [26], Michalek and Papalambros' combination of an outer-loop branch and bound algorithm with an analytical target cascading (ATC) MDO architecture in the inner loop for optimizing hierarchical systems [86], and Pelamatti's budget allocation strategy for Bayesian optimization of hierarchical problems [25].

The previously mentioned implementations of the nested strategy explicitly enumerate the entire space of architectures [25], prune the tradespace based on domain knowledge [26], or execute a complete optimization at the subproblem level for every top-level iteration [86]. These characteristics are not desirable when dealing with general system architecture optimization problems because the entire architectural space might be too large to contemplate, there might not be enough domain knowledge available to streamline the architectural design space, and carrying out full optimizations at the subproblem level for every architecture might result in wasting computational resources and time.

The nested strategy implemented in this work addresses the limitations mentioned above. Enumerating all architectures or carrying out design space pruning is avoided by using a genetic algorithm at the top-level loop in the strategy, which allows the exploration of different architectures via random mutations. This

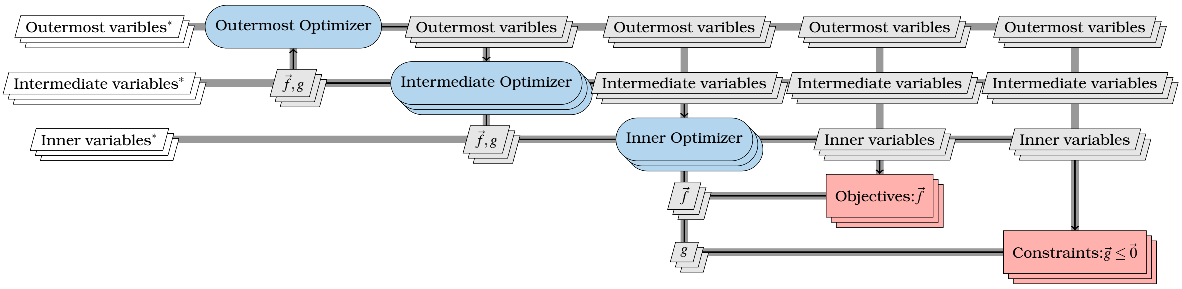


Figure 3.6: Generic XDSM representation of the nested optimization strategy. Three optimization levels are illustrated here, but the strategy works with any number of levels starting at two.

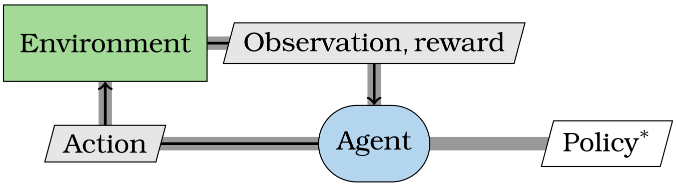


Figure 3.7: Generic XDSM representation of the decision chain strategy. The problem's objectives and constraints are encoded in the reward signal.

approach is similar to the one proposed by Frank et al. in [26]. However, this work's implementation uses NSGA-II at the top level to directly determine which architectures will be visited in the downstream optimization loops. In this work's approach, populations are generated at the innermost optimization loop and are passed to the outer loops. Then, when the populations are passed for the outer loops, their inner variables are replaced by the outer-loop variables that generate them.

Figure 3.6 shows an XDSM representation of the nested optimization strategy. Although three levels are illustrated in this case, the nested strategy can handle an arbitrary number of levels starting at two. Appendix B shows an example of how the nested strategy carries out the optimization process.

3.4.3. Decision Chain Optimization

The *decision chain* optimization strategy frames the architecting problem as a sequence of decisions. This strategy formulates the problem as a Markov process, and the optimizer does not directly control the design vector. Instead, the current design point is part of an *environment* with which an *agent* can interact via *actions*. These actions change the environment, and the agent perceives the result as a *reward* signal and an *observation* indicative of the environment's *state*. The process is repeated throughout several *episodes*, which are finalized after a set number of steps or when the environment reaches a state that forces termination. After a successful run, the agent will have obtained a *policy* that maximizes its expected reward. Figure 3.7 shows an XDSM representation of the decision chain strategy.

The terminology in the decision chain strategy is very similar to the one introduced in Section 3.3.3. This is no coincidence, as the decision chain strategy can be seen as the framework within which system architecture optimization problems are transformed into reinforcement learning problems. For instance, this technique has been applied to optimize complex systems such as chemical processes [27, 81], microfluidic devices [87], microchips [23], metasurface holograms [88], and neural network architectures [89, 90].

However, if the environment is entirely deterministic (i.e., the starting point is always the same, and state-action pairs always lead to the same new states), any optimization algorithm can be used to obtain a policy consisting of a fixed set of actions that lead to the highest episodic reward.

Nonetheless, generalization capabilities are also dropped when reinforcement learning is dropped from the decision chain strategy. This loss happens because, without reinforcement learning, the policy is simply a list of actions the agent takes without considering the environment itself. With reinforcement learning, meanwhile, the policy is a mapping from states to actions that can be robust to noise, changes in initial conditions, and even changes in the environment's definition. For example, Li et al. train a reinforcement learning agent on a 40-city version of the multi-objective traveling salesperson problem that goes on to outperform an NSGA-II optimization in a 200-city version of the problem that the agent never saw during training [24].

4

Methodology and Implementation

The algorithms and strategies for system architecture optimization discussed in Chapter 3 require a rigorous methodology for their practical implementation and evaluation. This chapter explains the procedures and steps taken to implement and evaluate these algorithms and strategies within the domain of system architecture optimization.

4.1. Algorithm Implementation

The algorithms described in Section 3.3 were implemented in Python. The implementation of the genetic algorithms was based on Pymoo's[1] interface [62]. Meanwhile, the implementation of the Bayesian optimization algorithm was based on Trieste's[2] implementation [91], but was modified to work under Pymoo's interface. The DDPG algorithm was developed using TensorFlow[3] [92] to create and train the agents, and its implementation was based on Singh[4] [93]. The Stable Baselines 3[5] version of DDPG was also used for the multi-objective problems.

4.1.1. Problem Definition

The optimization problems were formulated in two different ways, as instances of a custom class inheriting from Pymoo's `ElementWiseProblem` class and as environments based on Gymnasium's[6] `Environment` class.

Pymoo-Based Problem Definition

The basic class used to represent optimization problems inherits from Pymoo's `ElementWiseProblem` class. It adds a `might_fail` attribute to identify problems that may include hidden constraints. The problem's variables are defined as a Python dictionary. This dictionary's keys are the variables' names, and its corresponding values are the variables themselves. The variables are defined as instances of Pymoo's `Variable` class, more specifically `Real` for continuous variables and `Integer` for integer or categorical variables. This means that the categorical variables for all problems treated in this work were represented using the integer encoding method explained in Section 3.2.1.

Gymnasium-Based Environment Definition

As discussed in Section 3.3.3 and Section 3.4.3, reinforcement learning and decision chain frameworks require an *environment*. In Python, the Gymnasium library is a common way to set up environments for RL. The environments used in this work inherit from Gymnasium's `Env` class, which defines an environment's action and state spaces and its reward function.

[1] https://pymoo.org/. Accessed 2023-07-31.

[2] https://secondmind-labs.github.io/trieste/1.1.2/index.html. Accessed 2023-07-31.

[3] https://www.tensorflow.org/. Accessed 2023-07-31.

[4] https://keras.io/examples/rl/ddpg_pendulum/. Accessed 2023-07-31.

[5] https://stable-baselines3.readthedocs.io/en/master/. Accessed 2023-08-02.

[6] https://gymnasium.farama.org/. Accessed 2023-07-31.

4.1.2. Genetic Algorithm

Genetic algorithms are widely used for many optimization tasks where gradients are not readily available [94]. For example, the Python library multi-objective optimization library Pymoo has various genetic algorithms, starting from a simple single-objective version, mixed-discrete GAs, and the well-known and robust NSGA-II algorithm [62]. As a result, the implementation of genetic algorithms for this work was straightforward: it was only necessary to inherit from Pymoo's `MixedVariableGA` class for the single-objective case and from its `NSGA2` class for the multi-objective case. The Pymoo-based implementation of the genetic algorithms used in this work made them instantly compatible with Pymoo's problem definition format and its `minimize` function. Thanks to using Pymoo's classes and programming interface, this work's implementation of genetic algorithms was simple.

4.1.3. Bayesian Optimization

As mentioned earlier, this work used the Trieste Python library as its starting point for implementing the Bayesian optimization algorithms [91]. The Trieste library is known for its efficient Bayesian optimization algorithms, designed to handle complex optimization problems.

A custom interface was programmed to make Trieste's optimizer work smoothly with Pymoo's `Algorithm` class and `minimize` function. This way, the integration between the two libraries was seamless. The interface bridged the differences in data structures and function calls, ensuring that optimization tasks could be performed with the same workflow style regardless of the choice of genetic or Bayesian optimization algorithms. Mapping the functionalities of Trieste to the requirements of Pymoo's `Algorithm` class and `minimize` function made it possible to leverage the strengths of both libraries in a unified framework.

Trieste includes different acquisition functions that allow Bayesian optimization algorithms under different numbers of objectives, constraints, and possible hidden constraints. The acquisition functions implemented for this work were the following, depending on the problem's characteristics:

- **Single objective, no constraints:** Expected Improvement. This acquisition function measures the anticipated decrease in the objective value over the current best-known solution. It is commonly used in single-objective optimization, guiding the search toward regions with higher potential for improvement [68, 95].

- **Single objective with constraints:** Expected Improvement with Probability of Feasibility. This function combines the benefits of Expected Improvement while considering the likelihood that a solution meets the given constraints, ensuring that feasible solutions are prioritized [96].

- **Multiple objectives without constraints:** Expected Hypervolume Improvement. This function targets areas of the objective space that can advance the Pareto front and focuses on achieving optimal tradeoffs between the objectives [20].

- **Multiple objectives with constraints:** Expected Constrained Hypervolume Improvement. This acquisition function operates similarly to Expected Hypervolume Improvement but introduces an additional consideration for constraints, ensuring that the multi-objective solutions proposed also adhere to the given constraints [75].

- **Hidden constraints:** Probability of Failure. If a problem potentially includes a failure region, this acquisition function can be used with the other functions described above. It estimates the likelihood of a design yielding invalid evaluations in either the objectives or the constraints, helping the optimization algorithm to cautiously navigate areas of the design space that may pose unforeseen challenges [57]. Alternatively, hidden constraints can be handled through an extreme barrier approach, in which failed evaluations are assigned a worst-case scenario value [56].

One important limitation in this work's Trieste-based implementation of Bayesian optimization is the efficient handling of large combinatorial spaces. For mixed-discrete problems, Trieste requires an explicit enumeration of all the members of the discrete design space $\mathscr{D}$. Therefore, when a problem has several discrete variables, each with many possible alternatives, the Trieste-based implementation struggles with scalability. The need for explicit enumeration can lead to a combinatorial explosion, where the number of possible combinations grows exponentially with the increase in discrete variables and their potential values. As a result, optimization tasks that involve large combinatorial spaces can become computationally prohibitive for this setup.

4.1.4. Deep Deterministic Policy Gradient Reinforcement Learning Algorithm

As mentioned earlier, the environments required for reinforcement learning optimization approaches and the decision chain strategy were programmed using the Gymnasium library. A DDPG actor-critic neural network architecture was developed in the Python machine learning library TensorFlow for the single-objective problems. The multi-objective problems were addressed using Stable Baselines 3's implementation of DDPG. Stable Baselines 3 is a set of open-source implementations of well-known reinforcement learning algorithms maintained by the German Aerospace Center's Institute of Robotics and Mechatronics [97].

For single-objective problems, the DDPG algorithm implementation requires only an environment and a problem-specific reward function that encapsulates the problem's objectives and constraints in a single scalar value. The multi-objective case is more involved since it is necessary to account for the multiple objective functions while retaining the scalar reward structure. As introduced in Section 3.3.3, reinforcement learning methods can tackle multi-objective problems by decomposing the problem into several scalar subproblems [24, 82].

The multi-objective implementation developed for this work is based on the approach proposed by Li et al. [24], in which the authors train RL agents to solve scalarized problems that are close to each other sequentially. The policy learned for one scalarized problem can be used as a starting point for solving the next scalarized problem, thereby leveraging the knowledge gained from previous iterations and expediting the overall optimization process.

In this method, a Pareto front is not directly approximated. Instead, multiple scalarized objectives are defined, and the RL agents focus on optimizing each scalarized problem one at a time. As the agent moves from one scalarized problem to the next, the proximity of these problems allows it to exploit knowledge from prior experiences, effectively transferring their learning [24]. This approach can be likened to curriculum learning, where simpler tasks guide the agents' learning progression towards more complex ones [98].

The scalarization function used in this work is not the weighted sum employed by Li et al. Instead, the achievement scalarization function (ASF) is used. This function has the advantage that it always returns Pareto optimal solutions, given that the reference point provided is feasible [99]. Unlike the weighted sum, which can sometimes lead to non-Pareto optimal solutions, the ASF guarantees that the minimum of the scalarized problem lies on the Pareto front.

It is imperative to recognize that DDPG and other reinforcement learning algorithms are challenging to implement for several reasons. These include:

- **Hyperparameter tuning:** An implementation of a given reinforcement learning algorithm depends on several hyperparameters, ranging from the algorithm's settings such as learning rate, discount factor, and target update rate, to the architecture of their neural networks, such as number of layers, number of units per layer, and type of layers. In fact, the hyperparameter tuning problem of reinforcement learning (and other machine learning applications) can be framed as an architecture optimization problem [52].

- **Reward shaping:** Choosing a reward function that properly reflects all the problem's objectives and constraints in a single scalar is a crucial yet complicated activity in the implementation of reinforcement learning algorithms. An incorrect reward function may lead the agent to learn suboptimal policies even if the RL algorithm itself is sound.

- **Training instability:** Many reinforcement learning algorithms contain several sources of instability, making it challenging to implement them in a way that they produce reliable results after multiple training runs.

4.2. Strategy Implementation

With the three algorithms to be studied implemented, it is now possible to incorporate the strategies that will address the hierarchical design spaces of system architecture optimization problems.

4.2.1. Global Exploration

In Section 3.4.1, it was shown that the global exploration strategy uses a single optimizer to iterate through all design variables. Two approaches can be taken with the global exploration strategy: a naive approach, where the optimizer is allowed to operate on inactive variables, and an imputation approach, where the design vectors proposed by the optimizer are corrected to reflect that changes to inactive variables are ineffectual.

The imputation approach was chosen because it leads to fewer unnecessary function evaluations and a more accurate modeling of the design space [41].

The global exploration strategy uses only one optimizer to find points in the design space. As a result, most optimization algorithms will work "out-of-the-box" with this strategy. However, to carry out the imputation approach within this strategy, an *imputer* is required. The imputer is responsible for identifying inactive variables and assigning them a constant value.

Within the Python multi-objective optimization package Pymoo [62], the imputer's operation can be implemented within a `Repair` object. Pymoo's `Repair` is built to act after an algorithm proposes new designs and before the function evaluations are carried out. The `Repair` object defines a function to determine how the proposed design or designs will be altered. This function is entirely problem-dependent, as different architecture optimization problems have different sets of meta, decreed, and neutral variables.

Another important feature of the global exploration strategy is that before running the optimization, all possible design variables must be known and enumerated [40]. This enumeration is necessary because the global exploration strategy operates on constant-sized design vectors, so the entirety of the design space must be included in these vectors. Depending on the problem, the enumeration of all possible variables may be achieved by inspection or by using a custom function. In some cases, however, this process might be exceedingly difficult or even impossible due to large amounts of branching and hierarchies in the architecting problem.

Therefore, the procedure to implement the global exploration strategy in a given system architecture optimization problem was the following:

1. Get an enumeration of all possible design variables.
2. Select an optimization algorithm and implement it using Pymoo's interface.
3. Create a `Repair` object that captures the relationship between the problem's meta and decreed variables.

The main implementation challenge of the global exploration strategy is its first step: getting an enumeration of all possible design variables. This step is straightforward for problems with few architectures: an exhaustive enumeration of all the architectures and an inspection of their corresponding variables is sufficient. However, when the number of possible architectures grows, and the amount of hierarchy in the problem increases, the task of finding all possible variables can quickly become unmanageable.

4.2.2. NESTED OPTIMIZATION

As discussed in Section 3.4.2, the nested optimization strategy splits a problem's variables into different levels according to their role or type. Consequently, an n-level nested optimization strategy uses n different optimizers. These optimizers can, in principle, be any suitable optimization algorithm. However, for this work's particular implementation of the nested optimization strategy, a genetic algorithm is always used in the outermost loop to control which architectures are evaluated. This decision was taken because the genetic algorithm's random initialization and evolutionary operators allow it to explore the entire architectural space without resorting to pruning or exhaustive enumeration.

The procedure to implement the nested exploration strategy in a given system architecture optimization strategy was the following:

1. Split the design space into two or more levels following the problem's hierarchy or variable types.
2. Set up a genetic algorithm for the outermost level and any suitable optimization algorithm in the other levels.
3. Define the proportion of evaluations in the outer level to those in the inner level.

One advantage of this work's implementation of the nested strategy is that it does not require all the architectures to be enumerated explicitly or to prune the tradespace beforehand. The outermost genetic algorithm ensures that promising architectures will be found and poorly performing ones will be ignored.

A considerable modification of the genetic algorithm that runs the strategy's outer loop is that it allows duplicate designs. This allows the outer optimizer to act only on information directly available to it without requiring knowledge of the inner-level design variables. Appendix B shows an example of how this process is carried out.

The main implementation challenge for the nested strategy is determining a suitable ratio of inner and outer level evaluations. Too many outer level evaluations with too few inner level evaluations can lead to a superficial exploration of the design space, with many architectures being selected but not sufficiently explored. On the other hand, allocating too many evaluations to the inner level and limiting the outer level evaluations can result in over-optimizing a narrow set of architectures without sufficiently exploring other potentially promising designs. This difficulty highlights that the nested strategy inherently favors exploitation in the exploration-exploitation trade-off. However, there is some control over the strategy's degree of exploitation by adequately setting up the proportion of outer to inner level evaluations.

4.2.3. Decision Chain Optimization

Similar to the implementation of the DDPG reinforcement learning algorithm discussed in Section 4.1.4, the decision chain strategy transforms the optimization problem into a series of steps taken within an environment. As mentioned in Section 4.1.1, the environment definition for the decision chain strategy uses Gymnasium's `Env` as a base class. Within this `Env` class, the action and state spaces are defined such that:

1. The action space defines actions in a constant-sized vector that translates to modifications of the hierarchical design instances.
2. The state space flexibly represents design instances that may have different numbers of design variables.

The architectural problem's action space was crafted based on Stops et al.'s methodology [27]. In this framework, a two-component action is employed: the initial component indicates the design variable targeted for modification, while the subsequent component specifies the value to assign to that variable.

Two distinct formulations were explored regarding the state space, albeit with moderate success. Initially, the approach proposed by Andersson and Whyte [81] was adopted, where designs are represented as graphs that capture the hierarchical interconnections among various variables. Alternatively, the designs were characterized using variable-sized matrices. Each column retains details relevant to the active design variables in this schema.

The hierarchy in system architecture optimization problems makes it challenging to find adequate definitions of the state and action spaces that can be used in a reinforcement learning context. The inherent layered structure often leads to a vast and intricate state space that can be complex to represent accurately.

4.3. Performance Metrics

By definition, multi-objective problems simultaneously deal with two or more objectives. Consequently, optimized solutions to these problems are not generally a single point but sets of two or more Pareto-optimal points. As a result, metrics to assess the quality of these sets are needed. This section introduces three performance metrics used in the test cases discussed later in this work to compare the optimization results of different strategies and algorithms qualitatively. Additionally, the architecture Gini index is introduced as a way to gauge the level of exploration and exploitation that a given optimization exhibits during its operation. Pymoo's [62] documentation page on performance indicators[7] inspired this section's structure.

4.3.1. Generational Distance Plus and Inverted Generational Distance Plus

If a reference non-dominated set $Z = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{|Z|}\}$ for the optimization problem is known, the quality of an obtained non-dominated set $A = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{|A|}\}$ can be assessed with a measure of the average distance in the objective space between the points in A and their closest corresponding points in Z, or *generational distance* (GD) [42]. In general, lower values of the GD indicator would signify higher-quality non-dominated sets. However, it has been shown that the generational distance is not a Pareto-compliant indicator [100], meaning that, in some cases, a solution set entirely dominated by another may have a lower-valued GD.

To remedy this, the generational distance plus (GD+) indicator proposed by Ishibuchi et al. is given by [101]:

$$\text{GD+}(A) = \frac{1}{|A|}\left(\sum_{i=1}^{|A|} {d_i^+}^2\right)^{1/2}, \tag{4.1}$$

[7] `https://pymoo.org/misc/indicators.html`. Accessed 2023-07-28.

where d_i^+ is the adjusted distance between a particular point, $\mathbf{a}_i$, and the closest reference point to it within Z, which holds the corresponding value $\mathbf{z}_i$. The distance d_i^+ is adjusted to include dominance relationships between the points in A and the points in Z. When a reference point dominates a solution, the unmodified Euclidean distance is used. Conversely, in cases where the solution is non-dominated with respect to the reference set, the shortest distance from the reference point to the region dominated by the solution is determined. This distance may be interpreted as the degree to which the solution falls short (i.e., how its objective values lack when compared to the reference point). Only those objective values of the solution that are inferior to the reference point contribute to the calculation of this distance [101].

Conversely, the inverted generational distance plus (IGD+) metric reverses the GD+, assessing the distance from any point within the reference Pareto front to its nearest point in the obtained non-dominated front [101]. As an illustration, Figure 4.1 shows the non-dominated front obtained after a fictitious optimization run on the ZDT1 problem [28] and the reference Pareto front for the same problem. The red lines connect the points from the optimization non-dominated front to their corresponding closest points on the reference front. The blue lines do the opposite; they connect the points from the reference Pareto front to their closest point on the optimization non-dominated front. On average, the adjusted distance between the points in the optimization non-dominated front and their closest corresponding points in the reference Pareto front is 0.168, which is the value of the GD+ indicator for this case. Similarly, the adjusted distance between the points on the reference Pareto front and the points on the optimization's non-dominated front gives a value of 0.159 for the IGD+ indicator.

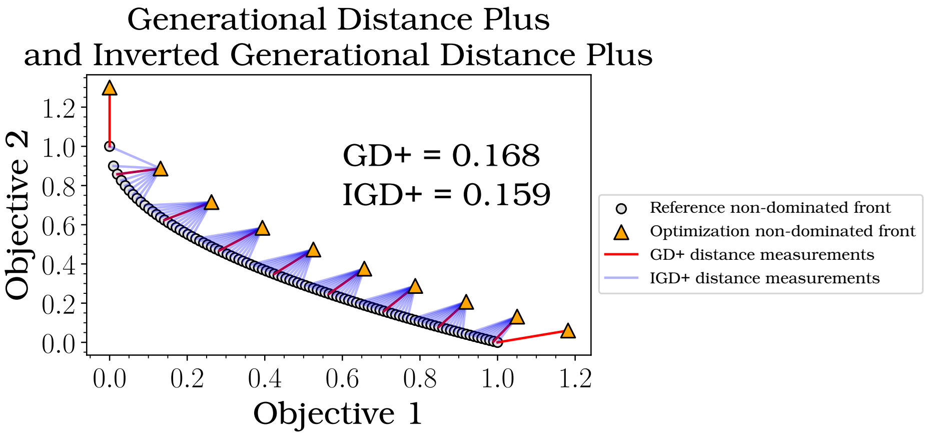


Figure 4.1: Illustration of the generational distance plus and inverted generational distance plus indicators.

It is important to remember that the GD+ and IGD+ indicators depend on a reference non-dominated front. If no reference front is available, this metric cannot be calculated. Furthermore, if the only available reference front is not a good approximation of the problem's actual Pareto front, the indicator's value could be misleading. Additionally, in problems with different objectives having different units or orders of magnitude, it is worthwhile to consider a possible scaling or normalization of the objectives to avoid biasing the indicator toward the objectives with the largest magnitudes.

4.3.2. Hypervolume

The hypervolume (HV) indicator provides the volume of the section of the objective space weakly dominated by a particular approximation of the Pareto set [102]. It measures the quality of a set $A = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{|A|}\}$ of $|A|$ non-dominated objective vectors produced in a run of a multi-objective optimizer. In a minimization problem with d objectives, this indicator consists of the measure of the region simultaneously dominated by A and bounded above by a reference point $\mathbf{r} \in \mathbb{R}^d$ [42, 103]. The hypervolume indicator can be normalized with respect to the hypervolume of a reference non-dominated front.

Figure 4.2 shows the weakly dominated area of the objective space for reference point $[1.2, 1.2]$ for the same optimization non-dominated front shown in Figure 4.1. In this case, the hypervolume of the obtained solution is 0.96.

The main advantage of the hypervolume indicator is that it does not need any reference sets, so it can be used even if the problem's solution (or a reasonable approximation) is unavailable. Only a reference point is

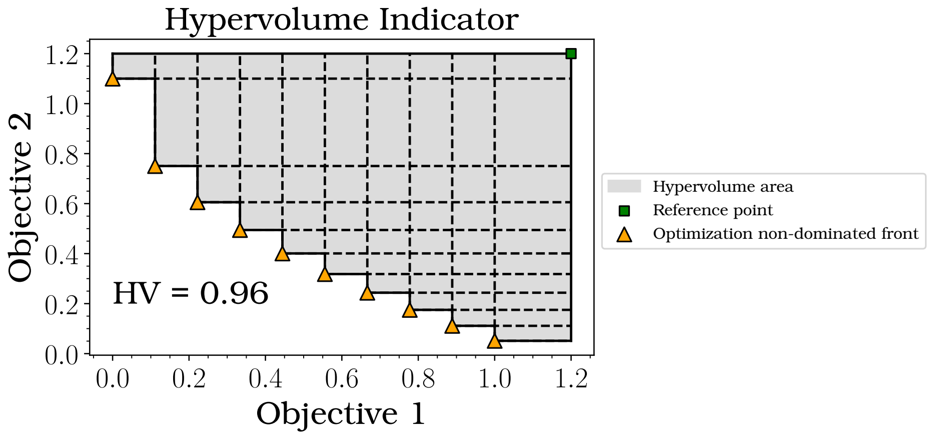


Figure 4.2: Illustration of the hypervolume (HV) indicator.

required, and it can be placed on the objective space after a non-dominated front has been found. Nonetheless, the HV metric suffers from a high computational cost. In two- or three-objective cases, the hypervolume is relatively straightforward to compute, but as the number of objectives increases, calculating the hypervolume becomes computationally intensive [103]. Additionally, the scaling of objectives can significantly influence the hypervolume indicator. If the objectives are not correctly scaled, the hypervolume indicator might favor one objective over another, leading to bias in evaluating the solutions.

4.3.3. NON-DOMINATED IMPROVEMENT RATIO (η_0)

The non-dominated improvement ratio η_0 uses the objective space results of a reference design of experiments (DoE) to assess the quality of a non-dominated set $A = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{|A|}\}$ obtained from a multi-objective optimization run. A combined non-dominated set is formed with the points in A and the points from the reference DoE. The indicator η_0 is the ratio of the points in this combined non-dominated set that come from A over the total number of points in the combined non-dominated set [26, 104].

Figure 4.3 shows how the η_0 indicator is calculated for the same optimization non-dominated front shown in Figures 4.2 and 4.3. In this case, the 100-point DoE yielded a point that dominates six of the ten points in the optimization's non-dominated front. The combined non-dominated front includes this point from the DoE and four points from the optimization results. As a result, the η_0 metric for this case has a value of $(4)/(4+1) = 0.8$.

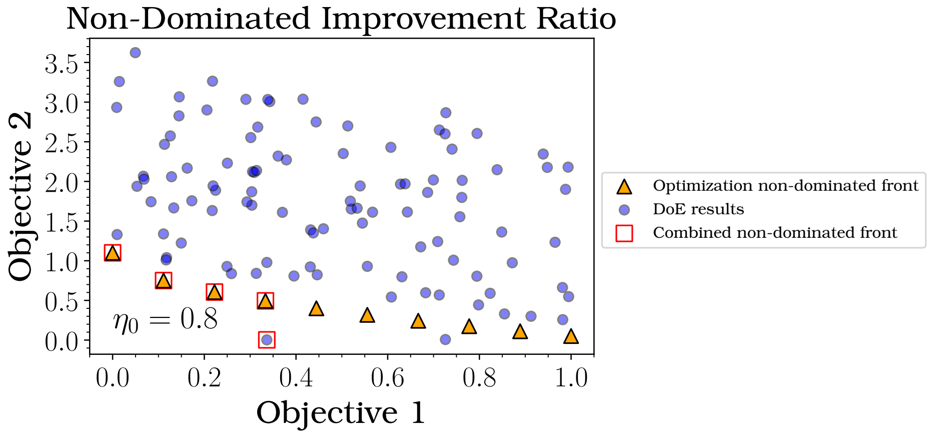


Figure 4.3: Illustration of the non-dominated improvement ratio (η_0) indicator.

The non-dominated improvement ratio is highly dependent on the reference DoE. For example, removing the non-dominated DoE point from Figure 4.3 would increase the obtained front's η_0 to 0.875. However, if the DoE's results include many more points than the obtained non-dominated front and they are used across multiple optimization runs, the η_0 indicator can provide a measure of the quality of the obtained solutions without recurring to (possibly unknown) reference Pareto sets or potentially costly hypervolume calculations.

4.3.4. ARCHITECTURE GINI INDEX

The Gini index, originally developed as a measure of income inequality [105], can also be adapted to quantify the inequality in the distribution of evaluations across different architectures in system architecture optimization problems. It provides a metric to understand how evenly or unevenly evaluations are spread across candidate architectures.

The Gini index G is calculated based on the Lorenz curve, which is derived from a cumulative distribution of ordered values. For system architecture optimization, architectures are first ordered based on the number of times they have been evaluated. Let n_{arch} be the number of architectures and n_i be the number of evaluations for the i^{th} architecture, sorted in non-decreasing order. Then, the Gini index is given by [106]:

$$G = \frac{\sum_{i=1}^{n_{\text{arch}}} \sum_{j=1}^{n_{\text{arch}}} \left| n_i - n_j \right|}{2 n_{\text{arch}}^2 \bar{n}}, \tag{4.2}$$

where $\bar{n}$ is the mean number of evaluations across all architectures.

A Gini index of 0 represents perfect equality (all architectures are evaluated the same number of times). In contrast, an index of 1 means maximal inequality (all evaluations are concentrated in a single architecture). Consequently, the Gini index applied to system architecture optimization problems can help quantify the degree of exploration or exploitation produced by a given optimization routine. While random sampling of the design space is expected to yield low Gini indices, indicating thorough exploration, effective optimizations will have higher Gini indices, signifying some degree of exploitation of promising areas of the design space.

5

Algorithm Test Cases

This chapter studies two test cases that verify and validate the performance and robustness of the optimization algorithms introduced in Sections 3.3 and 4.1. These test cases provide a rigorous examination of the algorithmic properties under varied conditions and help identify their strengths and potential limitations. Moreover, they ensure that the algorithms are theoretically promising and practically applicable. To begin, Section 5.1 studies a simple airfoil optimization problem to verify whether the algorithms implemented work under standard conditions and to validate them against published results. Next, Section 5.2 examines a mixed-discrete version of the ZDT1 benchmark problem to investigate how the algorithms respond to changes in the problem's variable composition and size.

5.1. Airfoil Optimization Problem

As a first approximation to optimization with the genetic, Bayesian, and DDPG algorithms, a relatively simple problem was chosen. This single-objective and continuous problem aims to optimize an airfoil's lift-to-drag ratio in incompressible flow. This test case can be used to verify that the implementations of the algorithms work correctly and to validate the optimization's results against published results of the same problem [107, 108].

5.1.1. Problem Statement

This problem's objective is to maximize an airfoil's lift-to-drag ratio using a NACA 2412 airfoil as a reference point. The problem is subject to two constraints: a maximum thickness-to-chord ratio t/c of 0.12 and a minimum moment coefficient C_m of -0.13. The airfoil's aerodynamic properties are calculated using XFOIL[1] with Reynolds number $\mathrm{Re} = 5.5 \times 10^5$, Mach number $\mathrm{Ma} = 0$, critical amplification factor $n_{\mathrm{crit}} = 9$, and angle of attack $\alpha = 2°$, following the simulation configuration details from Castellanos [107] and Traisak et al. [108]. The parameterization for the airfoil designs was carried out with eight coefficients following Kulfan's Class-Shape Transformation (CST) method [109]. Equation 5.1 gives a formal statement of the optimization problem.

$$\begin{aligned} &\text{Minimize} && -C_l/C_d(\boldsymbol{x}) \\ &\text{With respect to} && \boldsymbol{x} = 8 \text{ CST coefficients} \\ &\text{Subject to} && 0.13 - C_m(\boldsymbol{x}) \leq 0 \\ &&& t/c(\boldsymbol{x}) - 0.12 \leq 0. \end{aligned} \tag{5.1}$$

5.1.2. Algorithm Setup

As discussed in Section 3.3, the three tested algorithms have stochastic behavior. As a result, each one was executed ten times in the airfoil optimization problem. The stopping criterion used in all cases was a maximum evaluation number of 500.

The genetic, Bayesian, and DDPG algorithms were configured as follows for this problem:

[1] https://web.mit.edu/drela/Public/web/xfoil/. Accessed 2023-07-31.

Genetic Algorithm A population size of 10 was used for 50 generations to complete the evaluation budget.

Bayesian Optimization The dataset for the algorithm was initialized with ten points chosen randomly from the design space. Each of the ten runs started with a different set of randomly selected points.

DDPG The action space was defined as a vector of size eight, with each component corresponding to one of the airfoil's CST coefficients. The state space was defined as the set of possible airfoil designs given by their CST parameterization. Each training run consisted of 500 episodes, with one function evaluation per episode. The first 50 episodes of each run were conducted by taking random actions to enhance the experience diversity later in the training.

Additionally, ten separate random samplings of the design space were conducted to establish a baseline for assessing the algorithms' performance. Each sampling drew 500 points from the design space from a uniform random distribution. If an algorithm can find a lower objective value than the random sampling process for the same number of evaluations, then its performance is considered adequate.

5.1.3. Optimization Results

Figure 5.1 shows the aggregated behaviors for the optimization progress of the different algorithms. The solid lines represent the interquartile mean of each algorithm's progress, while the shaded areas represent the interquartile ranges of the optimization runs. The optimized airfoils obtained with each algorithm are plotted and compared against the reference design of the NACA 2412 in Figure 5.2.

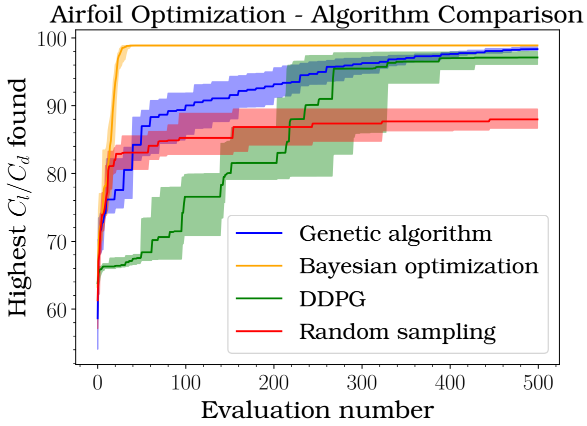


Figure 5.1: Algorithm performance comparison on the maximization of an airfoil's C_l/C_d. Solid lines correspond to the interquartile mean across ten optimization runs; shaded areas represent the interquartile ranges of the runs.

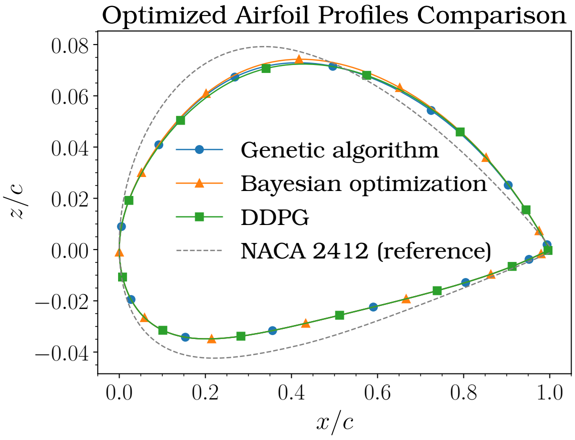


Figure 5.2: Comparison of the optimized airfoils.

Statistical analysis was conducted using two-sample t-tests with a 95% confidence interval, considering the alternative hypothesis that the optimization algorithms' means are higher than the means of random sampling. As shown in Table 5.1, the obtained p-values for all comparisons were consistently small, rejecting the null hypothesis at a 95% confidence level. These findings verify that the optimization algorithms significantly outperformed random sampling in solving the benchmark problem. As a result, all three algorithms effectively fulfill their intended purpose on a basic black-box, non-architectural problem.

Table 5.1: Comparison of mean differences, t-statistics, and p-values for optimization algorithms vs. random sampling in the airfoil optimization problem. $\Delta C_l/C_d$ is the difference between the mean highest C_l/C_d found by a given algorithm and that found by random sampling.

Algorithm	$\Delta C_l/C_d$	t-statistic	p-value
Genetic algorithm	10.156	12.052	1.4×10^{-8}
Bayesian optimization	10.897	13.977	5.31×10^{-8}
DDPG	7.798	5.014	9.01×10^{-5}

As seen in Figure 5.1, the convergence behavior of all three algorithms varies considerably. The Bayesian optimization procedure is the fastest-converging one, requiring, on average, 21 function evaluations to find a point that evaluates within 5% of the maximum C_l/C_d value found across all runs. Meanwhile, the genetic and DDPG algorithms take 230 and 268 function evaluations, respectively, to reach this point. Another interesting feature of Figure 5.1 is the observed variance across runs. The Bayesian optimization routine produces consistent results, as shown by its small interquartile range for all evaluations. The genetic algorithm has somewhat more variance, likely due to the sources of randomness intrinsic to the algorithm in its selection, crossover, and mutation operations. Finally, the DDPG runs were the least consistent, exhibiting, in some cases, a greater variance than that of the random sampling. This behavior is expected since, like many reinforcement learning algorithms, DDPG is highly stochastic. Sources of randomness in the DDPG runs include its neural network weight initializations, the stochastic gradient descent optimizers used during model training, and the noise added to the actor's deterministic policy.

Besides the difference in the convergence behaviors of the three algorithms studied, it is essential to note that each has advantages and disadvantages regarding ease of implementation and runtime. For instance, although Bayesian optimization achieves the fastest and most consistent convergence (measured by number of evaluations), its time per iteration slows considerably after around 100 evaluations because it needs to refit its Gaussian process model on all data points in every iteration. The DDPG algorithm shows the promise of reinforcement learning for design optimization problems but brings considerable difficulties concerning its implementation. For a successful DDPG run, it is necessary to first define an environment with its corresponding action and state spaces and a reward function. Furthermore, the user is responsible for defining the neural network architectures and hyperparameters that determine the agent's training process. The genetic algorithm is the most straightforward and consistent to implement among the three alternatives studied. All the user needs to specify is a population size and a termination criterion.

Table 5.2 shows the average execution times of each algorithm for the 500 function evaluation budget established for this problem. As mentioned earlier, the Bayesian optimization adds considerable computational overhead to the optimization routine, while DDPG adds a moderate time to the routine, and the genetic algorithm completes the optimization in the fastest time. However, these results are highly hardware- and implementation-dependent. For example, it is expected that leveraging a graphics processing unit (GPU) for the DDPG and Bayesian optimization runs would cut down on the computational overhead of these methods. Depending on the problem, the higher computational overhead of Bayesian optimization may be offset by the smaller number of evaluations it requires to converge towards an optimal solution.

Table 5.2: Average execution times of the tested algorithms in the airfoil problem. All runs were carried out in a 2.80 GHz Intel i7 CPU.

Algorithm	Average execution time (s)
Genetic algorithm	61.3
Bayesian optimization	1985.14
DDPG	141.6

The results found in this section can be validated against those reported by Castellanos [107] and Traisak et al. [108] for the same problem. Castellanos uses a genetic algorithm for the optimization, while Traisak et al. employ a particle swarm optimization (PSO) procedure. As seen in Table 5.3, the maxima obtained with the algorithms implemented in this work exceed the highest C_l/C_d of 82.22 reported by Traisak et al. as well as Castellanos' 95.30. The high difference between Traisak et al.'s results and the rest of the lift-to-drag ratios can be explained by the small number of generations used by the authors; the algorithm was stopped before reaching convergence. The difference between Castellanos' work and this work is smaller and indicates an agreement in the values of C_l/C_d that can be obtained in this optimization problem.

Table 5.3: Validation of optimization results against published results for the same problem.

Algorithm	Highest C_l/C_d found
Genetic algorithm - Castellanos [107]	95.30
PSO - Traisak et al. [108]	82.22
Genetic algorithm - this work	98.84
Bayesian optimization - this work	99.05
DDPG - this work	98.50

5.2. Mixed-Discrete ZDT1 Problem

Before delving into system architecture optimization problems, the algorithms used in Section 5.1 to solve the single-objective, continuous airfoil problem were tested in a more challenging problem that incorporates multi-objective and mixed-discrete elements. The problem in question is the ZDT1 benchmark [28], a multiple-variable problem with a convex Pareto front.

5.2.1. Problem Definition

The ZDT1 problem's formal definition is [28]:

$$\begin{array}{lll} \text{Minimize} & \boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x})]^T & \boldsymbol{f}: \mathbb{R}^n \rightarrow \mathbb{R}^2 \\ & \text{Where} & f_1(\boldsymbol{x}) = x_1 \\ & & f_2(\boldsymbol{x}) = \left(1 + \frac{9}{n-1} \sum_{i=2}^{n} x_i\right)\left(1 - \sqrt{\frac{x_1}{1+\frac{9}{n-1}\sum_{i=2}^{n} x_i}}\right) \\ \text{With respect to} & \boldsymbol{x} \in \mathbb{R}^2 & \\ \text{Subject to} & 0 \leq x_i \leq 1 & i = 1, \ldots, n, \end{array} \tag{5.2}$$

where n is the problem's number of variables. The Pareto set for this problem is defined as:

$$0 \leq x_1^* \leq 1 \text{ and } x_i^* = 0 \text{ for } i = 2, \ldots, n. \tag{5.3}$$

The corresponding Pareto front is given by:

$$\begin{cases} f_1 \in [0,1] \\ f_2 = 1 - \sqrt{f_1} \end{cases} . \tag{5.4}$$

Figure 5.3 shows the Pareto front of the ZDT1 problem.

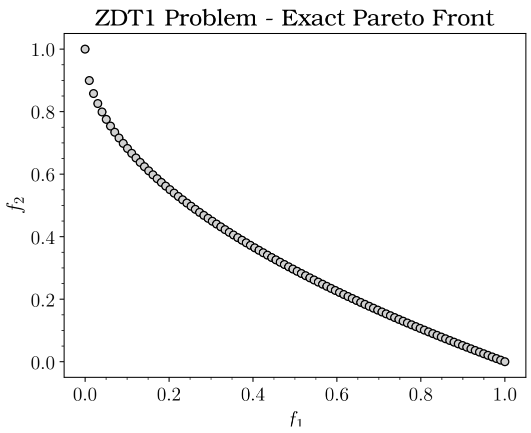


Figure 5.3: Pareto front of the ZDT1 problem.

5.2.2. Experimental Setup

For the purposes of the study, versions of the ZDT1 problem with 8, 32, 128, and 512 variables were studied. Additionally, the problem was adapted so that a portion of its variables would be binary-discrete instead of continuous. For each of the studied numbers of variables, versions of the problem with varying proportions of continuous-to-discrete variables were optimized.

Due to the stochastic nature of the studied optimization algorithms, each version of the problem was optimized eight times. Initially, all algorithms were assigned a budget of 5000 total evaluations per run, but the Bayesian algorithm's budget was cut to 500 evaluations due to its poor time performance on large datasets.

The performance of each algorithm was measured using the (normalized) hypervolume, generational distance plus, and non-dominated improvement ratio indicators discussed in Section 4.3.

Multi-Objective and Mixed-Discrete Adaptations of the Base Algorithms

The algorithms explored in Section 5.1 must be adapted to effectively deal with multi-objective and mixed-discrete problems. For this reason, the following modifications were applied to each algorithm:

Genetic Algorithm The Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [66] was employed as the multi-objective genetic algorithm for this study. Some of this method's key features are using domination during the selection process, preserving solution diversity in its populations, and using elitism [45, 110].

Bayesian Optimization The Expected Hypervolume Improvement (EHVI) acquisition function mentioned in Sections 3.3.2 and 4.1.3 was used instead of the single-objective Expected Improvement function. The objectives were still modeled as Gaussian processes. A significant limitation was found in Trieste's implementation of Bayesian optimization for discrete spaces: all possible combinations of the discrete variables have to be enumerated explicitly. This enumeration limits the direct applicability of Trieste's version of Bayesian optimization to relatively small discrete spaces. As a result, for this problem, the discrete variables were encoded with continuous relaxation for the Bayesian optimization algorithm.

DDPG The DDPG reinforcement learning algorithm was combined with Li et al.'s deep reinforcement learning multi-objective algorithm (DRL-MOA) [24] to handle the problem's multiple objectives. However, as mentioned in Section 4.1.4, the selected scalarization function was the achievement scalarization function (ASF) instead of the weighted sum approach used in the original implementation of DRL-MOA. The environment's action space was modeled as a continuous space, with each action corresponding to a decision variable in the optimization problem. The states were simply defined as the problem's design vectors, and the rewards corresponded to the negative scalarized objectives.

5.2.3. Results and Analysis

The inverted generational distance plus, hypervolume, and η_0 performance indicators were used to assess the performance of the three studied optimization algorithms in the mixed-discrete ZDT1 problem across different numbers of variables and compositions of the design space. Unfortunately, problems were encountered when using Bayesian optimization to solve the 128- and 512-variable versions of the problem. Iterations would either be too long (more than 10 minutes on a 4.10 GHz CPU to complete an iteration that proposed only one new point to evaluate) or cause memory problems that would interrupt the run (the program would consume all 32 GB of available RAM). Consequently, the results for the Bayesian optimization method are only reported for the 8- and 32-variable versions of the problem.

Inverted Generational Distance Plus

Figure 5.4 shows the inverted generational distance plus obtained by the NSGA-II, Bayesian optimization, and DDPG optimization algorithms as a function of the ZDT1 problem's number of variables and its distribution of continuous and discrete variables. In the 8-variable case, all algorithms achieve an IGD+ close to 0 for all distributions of continuous and discrete variables. However, the NSGA-II algorithm is significantly more consistent in its results than the Bayesian optimization or the DDPG algorithms. Additionally, a subtle upward trend can be observed for the Bayesian optimization and DDPG algorithms as the proportion of continuous variables in the problem increases. This indicates that, for the eight-variable case, the DDPG and Bayesian optimization algorithms perform better when there are more discrete variables.

In the 32-variable version of the problem, the DDPG algorithm's performance decreases considerably for all proportions of continuous-to-discrete variables. The trend observed in the 8-variable case for this algorithm is no longer present when there are 32 variables; the indicator's value is approximately unchanged when the percentage of continuous variables changes. Even though the Bayesian optimization's performance also decreases for the 32-variable case according to the IGD+ indicator, the drop is not as sharp as it is for the DDPG algorithm. The Bayesian algorithm's trend of worse performance when more continuous variables are present in the problem is also visible for the 32-variable version of the problem. Meanwhile, the NSGA-II algorithm continues to exhibit IGD+ values close to 0 for the 32-variable problem, showing the strongest performance of the three algorithms. In the 128- and 512-variable cases, the DDPG algorithm's performance, as measured by the IGD+ indicator, continues to decrease but remains stable across different proportions of continuous-to-discrete variables, much like what was observed in the 32-variable scenario. Similarly, NSGA-II keeps its good performance in the larger problems.

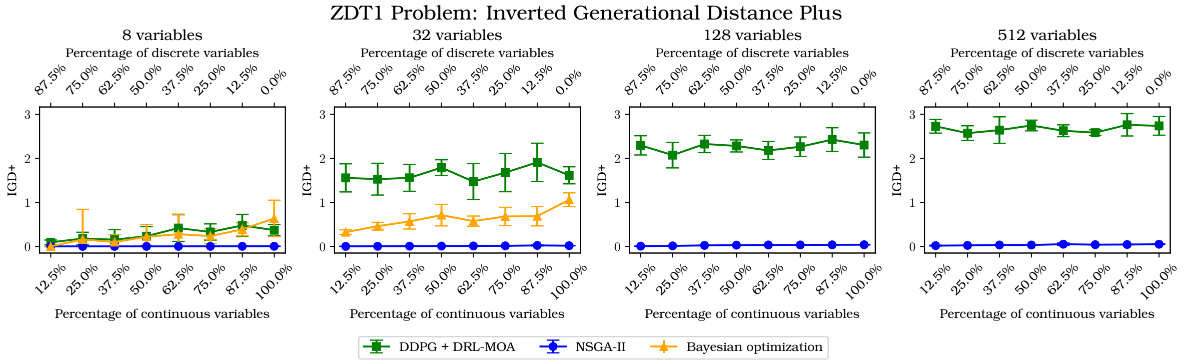


Figure 5.4: Inverted generational distance plus as a function of the percentage of continuous variables for the ZDT1 problem with 8, 32, 128, and 512 variables. The error bars indicate a 95% confidence interval around the mean result of eight optimization runs. The NSGA-II and DDPG runs used 5000 evaluations, while the Bayesian optimization runs used 500.

Hypervolume

The results for the hypervolume indicator of the multi-objective algorithms studied in this section are plotted in Figure 5.5. The hypervolume values were normalized with respect to the exact Pareto front's hypervolume. This indicator's results tell a similar story to what was found with the IGD+ metric. The NSGA-II algorithm consistently finds normalized hypervolumes close to 1, indicating non-dominated fronts very close to the problem's exact Pareto front. Meanwhile, both the Bayesian optimization algorithm and DDPG show a decrease in performance with an increase in the number of variables. Additionally, the Bayesian optimization algorithm performs worse for high proportions of continuous variables in both the 8-variable and 32-variable cases.

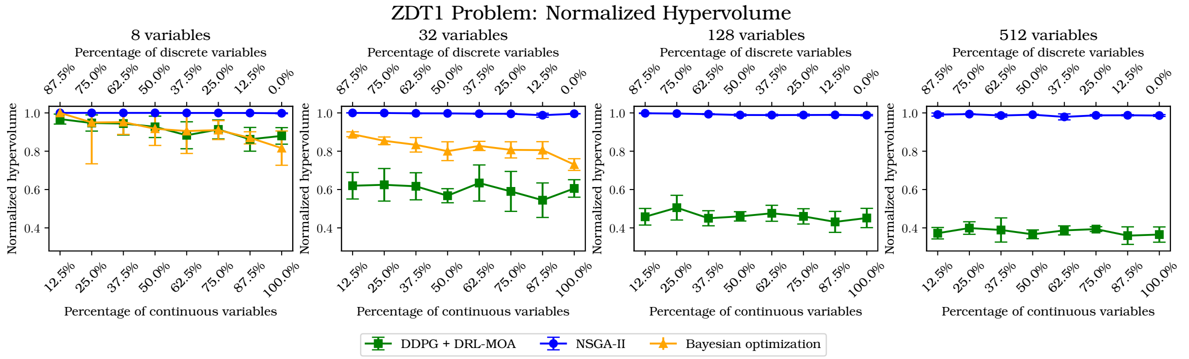


Figure 5.5: Normalized hypervolume as a function of the percentage of continuous variables for the ZDT1 problem with 8, 32, 128, and 512 variables. The error bars indicate a 95% confidence interval around the mean result of eight optimization runs. The NSGA-II and DDPG runs used 5000 evaluations, while the Bayesian optimization runs used 500.

Non-Dominated Improvement Ratio (η_0)

A 5000-evaluation random sampling DoE was conducted for every variation of the ZDT1 problem to calculate the η_0 indicator of the studied optimization algorithms. As discussed earlier, the η_0 indicator combines a reference DoE's non-dominated front with the one from the assessed optimization algorithm. The value of η_0 for the algorithm is the percentage of points from the combined non-dominated front found by the optimization. As a result, values of η_0 greater than 50% indicate acceptable performance of the optimization algorithm, as they suggest that the algorithm was able to find more points on the combined non-dominated front than a random search.

Figure Figure 5.6 shows the results of the η_0 indicator across all tested scenarios. For the 8-variable version of the ZDT1 problem, all three algorithms consistently produce η_0 values considerably above 50%. This suggests that all three algorithms perform adequately in relatively low-dimensional problems, being more efficient than random exploration. However, as the problem dimensionality increases to 32 variables, the DDPG algorithm begins to lag. Its η_0 values drop close to the 50% mark, especially in cases with a high proportion of discrete variables. For the 128-variable and 512-variable cases, the DDPG's η_0 values fall well below 50%, indicating that a random search would find a better non-dominated set than this algorithm.

This decline reiterates DDPG's challenge in handling larger action spaces and further reinforces its observed performance degradation in higher-dimensional problems. Surprisingly, and contrary to the pattern observed in the other two indicators, the Bayesian optimization algorithm improves its η_0 between the 8-variable and 32-variable cases. However, unlike IGD+ and hypervolume, which take into account the quality and spread of the solutions found, the η_0 primarily focuses on the dominance of the algorithm's solutions over those from a reference DoE. This means that an η_0 of 1 can be achieved by finding a single point that dominates all those coming from the DoE, which does not necessarily reflect a good-quality non-dominated front. On the other hand, NSGA-II, mirroring its consistent performance in the other indicators, retains η_0 values near 1 even in the 128- and 512-variable cases. This indicates its robustness and scalability across various problem sizes and compositions.

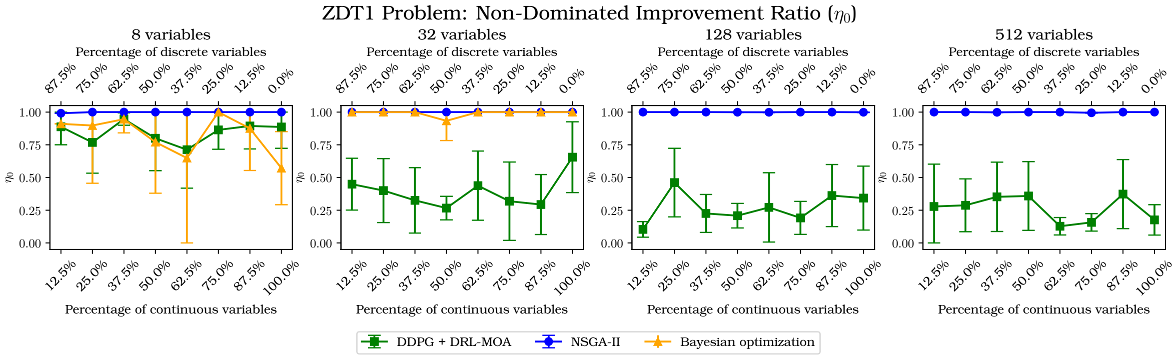


Figure 5.6: Non-dominated improvement ratio (η_0) as a function of the percentage of continuous variables for the ZDT1 problem with 8, 32, 128, and 512 variables. The error bars indicate a 95% confidence interval around the mean result of eight optimization runs. The NSGA-II and DDPG runs used 5000 evaluations, while the Bayesian optimization runs used 500.

5.2.4. Summary of Findings

The comprehensive analysis of the three optimization algorithms—NSGA-II, Bayesian optimization, and DDPG—revealed notable trends and insights across the different performance indicators. All three algorithms showcased good performance for the 8-variable version of the ZDT1 problem across all performance metrics. However, as the problem's dimensionality increased, NSGA-II consistently emerged as the most robust algorithm, maintaining superior performance even in higher-dimensional scenarios. DDPG and Bayesian optimization, on the other hand, struggled to maintain their performance with increasing dimensionality, revealing challenges associated with scalability.

For the inverted generational distance plus metric, NSGA-II was consistently strong in performance across all tested scenarios. Meanwhile, DDPG faced a decline in performance as the problem's number of variables increased. Its results were less affected by the distribution of continuous-to-discrete variables in higher-dimensional scenarios. Bayesian optimization displayed a drop in performance when there was an increase in continuous variables, indicating potential difficulties in scenarios with a higher proportion of continuous variables. Consistent with the IGD+ results, NSGA-II displayed strong results with normalized hypervolumes close to 1. Both DDPG and Bayesian optimization demonstrated decreased performance with increased problem dimensionality. Additionally, Bayesian optimization faced challenges in scenarios with a higher proportion of continuous variables.

As measured by the η_0 indicator, all algorithms performed well in the 8-variable scenario. DDPG's η_0 declined in the 32-variable scenario, especially in configurations with a higher proportion of discrete variables. NSGA-II again displayed its strength, maintaining high η_0 values even in higher-dimensional problems.

In summary, while all algorithms performed well in low-dimensional problems, NSGA-II proved to be the most versatile and robust optimization algorithm for the ZDT1 problem. Both Bayesian optimization and DDPG demonstrated their potential but faced significant challenges, particularly as problem dimensionality increased. The results for Bayesian optimization in this study contrast with those found for the simpler airfoil problem of Section 5.1, showing how changes in problem characteristics can lead to considerable changes in the relative performance of different optimization algorithms.

6

Goldstein Problem

In Chapter 5, it was shown that this work's implementations of the genetic algorithm, Bayesian optimization, and DDPG can address the multi-objective, mixed-discrete aspects of system architecture optimization problems. This chapter expands the investigation into hierarchical problems by studying adaptations of the variable-size design space Goldstein function introduced by Pelamatti [111]. The first adaptation, discussed in Section 6.1, adds a second objective to the original problem. The second adaptation, investigated in Section 6.2, creates a meta-problem based on the Goldstein problem to identify how changing specific characteristics of a hierarchical problem affects the performance of different system architecture strategies.

6.1. Multi-Objective Goldstein Problem

The multi-objective Goldstein problem presented here extends the Goldstein test problem introduced by Pelamatti [21].

6.1.1. Problem Formulation

The original problem aims to minimize a function $f(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w})$, with $\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5]$, $\boldsymbol{z} = [z_1, z_2, z_3, z_4]$, and $\boldsymbol{w} = [w_1, w_2]$. The variables' types, roles, and domains are shown in Table 6.1. As illustrated by Table 6.1, the vector $\boldsymbol{w}$ contains the problem's meta variables, while $\boldsymbol{x}$ and $\boldsymbol{z}$ contain its decreed and neutral variables. While $\boldsymbol{x}$ is composed of continuous variables, $\boldsymbol{z}$ is composed of integer variables.

Table 6.1: Variables of the Goldstein function optimization problem. Taken from Pelamatti et al.'s problem definition [21].

Variable	Domain	Type	Role
w_1	{0, 1, 2, 3}	Integer	Meta
w_2	{0, 1}	Integer	Meta
z_1	{0, 1, 2}	Integer	Decreed
z_2	{0, 1, 2}	Integer	Decreed
z_3	{0, 1, 2}	Integer	Neutral
z_4	{0, 1, 2}	Integer	Neutral
x_1	[0, 100]	Continuous	Neutral
x_2	[0, 100]	Continuous	Neutral
x_3	[0, 100]	Continuous	Decreed
x_4	[0, 100]	Continuous	Decreed
x_5	[0, 100]	Continuous	Decreed

The Goldstein problem is stated as follows [21]:

$$\begin{aligned} &\text{Minimize} && f(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w}) \\ &\text{With respect to} && \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w} \\ &\text{Subject to} && g(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w}) \leq 0, \end{aligned} \tag{6.1}$$

with

$$f(\boldsymbol{x},\boldsymbol{z},\boldsymbol{w}) = \begin{cases} f_1(x_1,x_2,z_1,z_2,z_3,z_4) & \text{if } w_1 = 0 \text{ and } w_2 = 0 \\ f_2(x_1,x_2,x_3,z_2,z_3,z_4) & \text{if } w_1 = 1 \text{ and } w_2 = 0 \\ f_3(x_1,x_2,x_4,z_1,z_3,z_4) & \text{if } w_1 = 2 \text{ and } w_2 = 0 \\ f_4(x_1,x_2,x_3,x_4,z_3,z_4) & \text{if } w_1 = 3 \text{ and } w_2 = 0 \\ f_5(x_1,x_2,x_5,z_1,z_2,z_3,z_4) & \text{if } w_1 = 0 \text{ and } w_2 = 1 \\ f_6(x_1,x_2,x_3,x_5,z_2,z_3,z_4) & \text{if } w_1 = 1 \text{ and } w_2 = 1 \\ f_7(x_1,x_2,x_4,x_5,z_1,z_3,z_4) & \text{if } w_1 = 2 \text{ and } w_2 = 1 \\ f_8(x_1,x_2,x_3,x_4,x_5,z_3,z_4) & \text{if } w_1 = 3 \text{ and } w_2 = 1 \end{cases} \tag{6.2}$$

and

$$g(\boldsymbol{x},\boldsymbol{z},\boldsymbol{w}) = \begin{cases} g_1(x_1,x_2,z_1,z_2) & \text{if } w_1 = 0 \\ g_2(x_1,x_2,z_1,z_2) & \text{if } w_1 = 1 \\ g_3(x_1,x_2,z_1,z_2) & \text{if } w_1 = 2 \\ g_4(x_1,x_2,z_1,z_2) & \text{if } w_1 = 3 \end{cases}. \tag{6.3}$$

$f_1,\ldots,f_8$ and $g_1,\ldots,g_4$ are given in [21]'s appendix. For this problem, it is noteworthy that none of the architectures defined by the combinations of w_1 and w_2 use all the variables in $\boldsymbol{x}$ or $\boldsymbol{z}$. As a result, the problem's full design vector always includes inactive variables.

The multi-objective adaptation of the Goldstein problem was carried out by adding a second objective that can be expressed as:

$$f_2 = f(100-\boldsymbol{x},\boldsymbol{z},\boldsymbol{w}) + \frac{5}{7} \times \text{architecture number}, \tag{6.4}$$

where f is the original objective function shown in Equation 6.2, and the architecture number is defined from the w_1, w_2 pairs as shown in Table 6.2. The same numbering is used in this section's figures. The second objective function in Equation 6.4 maps the continuous variables $\boldsymbol{x}$ to the other end of their bounds, thus ensuring that the two objectives will compete with each other.

Table 6.2: Architecture number definition for the Goldstein problem.

w_1	0	1	2	3	0	1	2	3
w_2	0	0	0	0	1	1	1	1
Architecture number	1	2	3	4	5	6	7	8

As seen in Table 6.2, the multi-objective Goldstein problem has 8 different architectures. Figure 6.1 shows the Pareto fronts of each of the problem's architectures. These Pareto fronts are used as references against which the different optimization strategies can be compared. Figure 6.2 shows the combined Pareto front of the multi-objective Goldstein problem. The figure shows that the Pareto front includes designs from three different architectures.

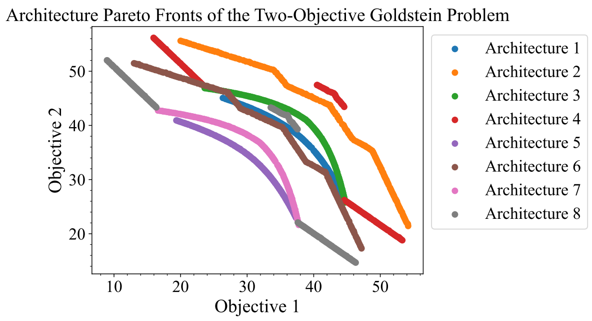


Figure 6.1: Pareto fronts of the Goldstein problem's individual architectures.

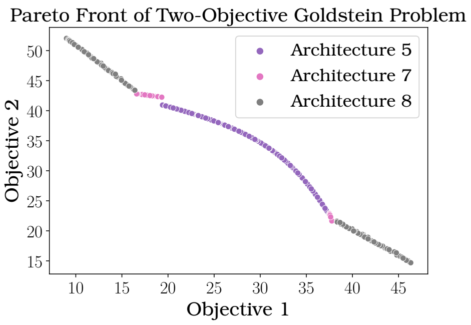


Figure 6.2: Pareto front of the multi-objective Goldstein problem.

6.1.2. Strategy Setup

The Pareto front from Figure 6.2 was used as a reference to assess the performance of the three strategies introduced in Section 3.4. For this assessment, the NSGA-II algorithm was used in all strategies due to its ease of implementation and acceptable convergence behavior. The strategies were run with a total function call budget of 500, 1,000, 5,000, and 10,000 evaluations. Because the NSGA-II algorithm is intrinsically stochastic, ten optimization runs were carried out for each strategy and evaluation budget combination.

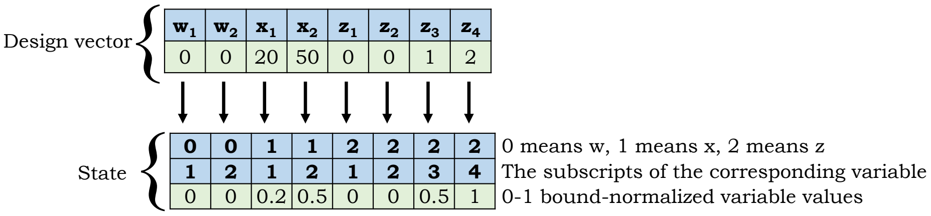
In this problem, the setup for each strategy was as follows:

- **Global Exploration:** The global exploration strategy used a population of 100 for its optimizer. An imputer addressed the decreed variables. Before function evaluations, the imputer set the continuous inactive variables to 50 (the midpoint between their upper and lower bounds) and the discrete inactive variables to 0.

- **Nested Optimization:** The nested optimization strategy used two loops. An outer loop with population size 10 controlled the w_1 and w_2 meta variables, and an inner one with population size 10 controlled the neutral variables and the decreed variables stemming from the upstream selection of w_1 and w_2. The inner optimizer could only advance one generation per outer loop evaluation.

- **Decision Chain:** The decision chain strategy defined the environment as a space of $n \times 3$ matrices that encode the possible design vectors, where n is the number of variables present in a particular design instance of the Goldstein problem. The first two rows of these matrices encode the variable name, and the last row gives a bound-normalized value of the variable's value. An example of this representation is depicted in Figure 6.3.

  The actions the agent can take in this environment are represented as vectors in $\mathbb{R}^2$ bounded between 0 and 1. The first component of the action vector selects a variable to change, and the second component assigns a value to said variable. If an action changes a meta variable that causes variables not previously present in the design instance to come up, these new variables take on a default value set at the middle of their bounds. In each episode, the agent has 10 steps to change the design, and function evaluations are only carried out in the final step. All episodes start from the same reference design. The NSGA-II optimization algorithm used in the decision chain strategy also used a population size of 100.



w_1	w_2	x_1	x_2	z_1	z_2	z_3	z_4
0	0	20	50	0	0	1	2

0	0	1	1	2	2	2	2	0 means w, 1 means x, 2 means z
1	2	1	2	1	2	3	4	The subscripts of the corresponding variable
0	0	0.2	0.5	0	0	0.5	1	0-1 bound-normalized variable values

Figure 6.3: Matrix representation of the Goldstein problem's design vectors to use as states in the decision chain framework.

6.1.3. Optimization Results

As mentioned in Section 6.1.2, 10 runs were carried out for each optimization strategy. Figure 6.4 shows the objective space results for representative runs of each of these strategies. Qualitatively, the global strategy appears to converge more quickly to the reference Pareto front and to efficiently explore a broader region of the design space than the other two strategies. This is evident in Section 6.1.3, where more evaluations are shown over a broader area of the objective space, and there are no signs of concentrations of evaluations around the Pareto fronts of the dominated architectures. On the other hand, the nested and decision chain strategies show distinctive patterns that indicate several evaluations at or near the Pareto fronts corresponding to architectures later dominated by others. This evidences that the nested and decision chain strategies rely more heavily on exploiting known information than exploring unknown design space areas.

Figure 6.5 shows the non-dominated fronts obtained by the same runs from Figure 6.4. In the runs shown, the global exploration strategy achieves the closest approximation to the reference Pareto front, followed by the nested strategy, and with the decision chain approach getting the non-dominated front farthest away from the reference Pareto front.

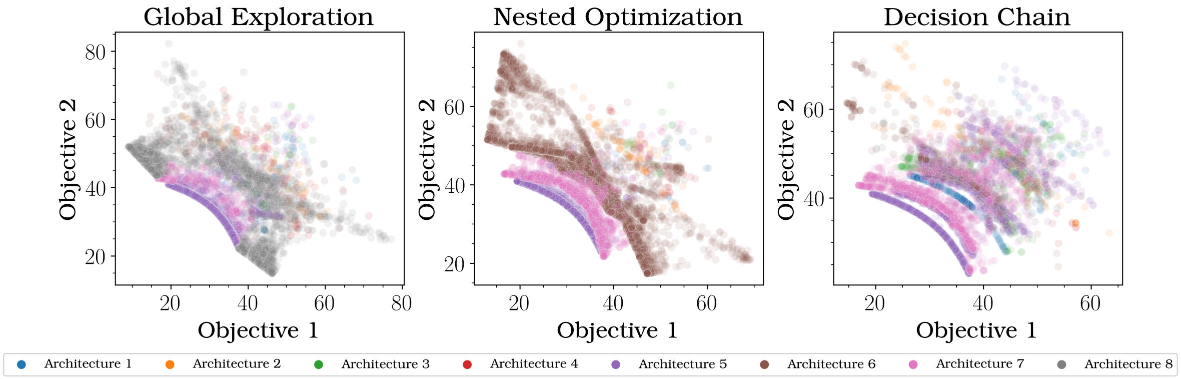


Figure 6.4: Objective space results for representative optimization runs with the different system architecture optimization strategies. All runs used the NSGA-II algorithm and were limited to 10,000 function evaluations.

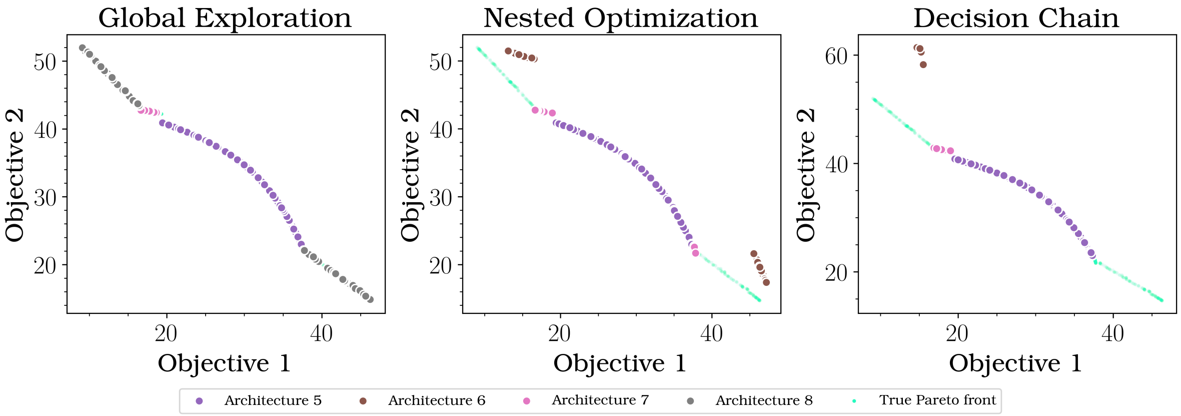


Figure 6.5: Non-dominated fronts for representative optimization runs with the different system architecture optimization strategies. All runs used the NSGA-II algorithm and were limited to 10,000 function evaluations.

As illustrated in Figure 6.5, the decision chain and nested optimization failed to include architecture 8 in their non-dominated solutions for these particular runs. This difficulty was a recurring theme in all runs, and even the global exploration strategy struggled to find the non-dominated points for architecture 8. A likely cause might be the fact that architecture 6 dominates all other architectures except for architecture 8 in a large, central region of the objective space, as seen in Figure 6.1, where the separate Pareto fronts of each of the problem's architectures are shown.

6.1.4. Comparison of Results

The performance of the different strategies was compared using the metrics described in Section 4.3: generational distance plus, (normalized) hypervolume, and η_0.

To recapitulate:

- Generational distance plus (GD+) measures the average distance from any point in a non-dominated set to its closest point in the reference Pareto front.
- The hypervolume indicator measures the hypervolume (area in a two-objective case and volume in a three-objective case) of the region of the objective space dominated by the examined non-dominated front and bounded above by a given reference point.
- The η_0 indicator measures the proportion of the points on the non-dominated front coming from the combination of a predefined design of experiments (DoE) and the non-dominated front under study; thus, η_0 can vary between 0 and 1, where 1 means that all the points in the combined front come from the tested algorithm.

Figure 6.6 shows the multi-objective performance indicators described above for each of the proposed system architecture optimization strategies with error bars indicating a 95% confidence interval around the mean metrics obtained after 10 runs for each strategy for 100, 500, 5,000, and 10,000 function evaluations. The hypervolume metric was normalized with respect to the reference Pareto front's hypervolume, and the reference point was set at $[60,60]$ in the objective space. The trends in Figure 6.6 verify the expected behavior of all strategies: as the number of function calls increases, the generational distance approaches 0, and the normalized hypervolume and η_0 approach 1. As anticipated qualitatively in Figures 6.4 and 6.5, the global exploration strategy tends to perform better than the other two strategies, and the decision chain approach tends to be the worst-performing one.

Nevertheless, as shown in Figure 6.6, each strategy's performance usually falls within the error bars of the others, indicating little statistical difference in the performance indicators for all strategies. A notable exception to this pattern can be identified in the behavior of the decision chain strategy at 500 evaluations. In this case, this strategy significantly underperforms in all indicators with respect to the other two strategies. This difference in performance might be explained by the fact that the decision chain's design vector (the series of actions that define the policy to be followed) is longer than the design vectors of the other two strategies. Additionally, the decision chain strategy's design vector is order-dependent: the meaning of the design variables along the design vector changes depending on the values of variables found earlier along it. These factors may potentially complicate the search process and decrease convergence speed.

With this analysis of the Goldstein problem, it can be established that the implemented system architecture optimization strategies effectively handled the hierarchical, mixed-discrete, and multi-objective nature of system architecture optimization problems. According to the three evaluated performance indicators, the global exploration strategy tends to perform best, while the decision chain framework shows the lowest level of performance. Given the high implementation difficulty of setting up and implementing the decision chain strategy, together with its inferior results compared to the other two strategies, it was decided that more complex problems like the tunable Goldstein problem in Section 6.2 and the aileron structural optimization in Chapter 7 would not be treated with the decision chain approach.

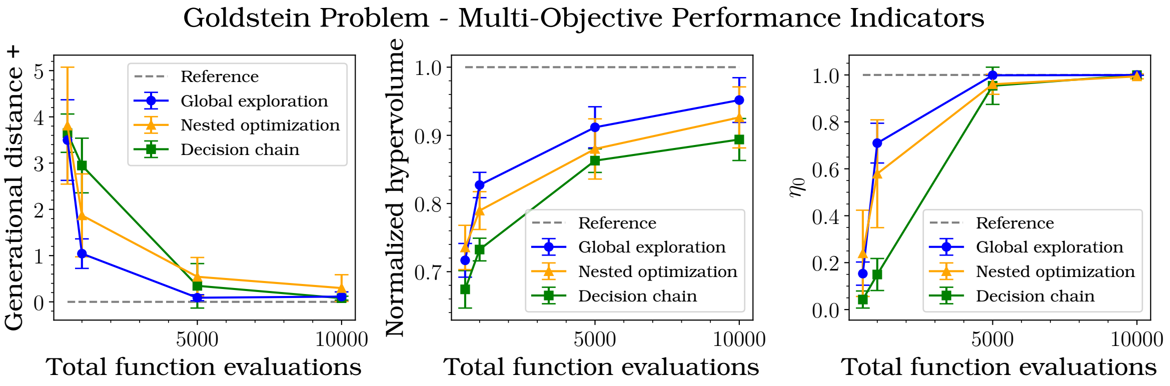


Figure 6.6: Multi-objective performance indicators for the proposed system architecture optimization strategies. The error bars mark a 95% confidence interval obtained from 10 optimization runs for each strategy and number of total evaluations. All runs used the NSGA-II algorithm.

6.2. TUNABLE GOLDSTEIN PROBLEM

The tunable Goldstein problem adapts the multi-objective problem discussed in Section 6.1. The tunable version of the problem can be formulated with desired characteristics, such as its *imputation ratio* and its ratio of continuous to discrete variables. An architecture optimization problem's imputation ratio is the proportion of all the apparent discrete designs available to the actual number of valid designs. It can be interpreted as the probability that choosing the design's discrete variables at random will lead to imputation. For instance, an imputation ratio of 100 means that out of every 100 discrete variable combinations, only one is valid, while the others include inactive variables [112].

A problem's imputation ratio is thus a measure of its degree of hierarchy. The lowest possible value for the imputation ratio is 1, meaning that all discrete variable combinations are valid, or, in other words, that there are no meta variables in the problem. By adjusting a problem's imputation ratio while maintaining its other attributes relatively stable, it is possible to gain insights into how different system architecture optimization strategies perform based on the problem's level of hierarchy.

The tunable version of the Goldstein problem is based on Bussemaker's tunable hierarchical meta problem (THiMP) [112]. It fixes the original problem's variables at those present in architecture 7 (as defined in Table 6.2). Then, it allows changing the number of continuous and discrete variables in this base problem. The new continuous variables are obtained by splitting the original problem's variables into equal intervals within their bounds. Meanwhile, the new discrete variables are obtained by copying the original problem's variables as many times as needed. The problem's function evaluations are carried out by adding the continuous variables and averaging the discrete variables.

6.2.1. EXPERIMENTAL SETUP

The performance of the global exploration and nested optimization strategies was assessed as a function of the imputation ratio and the number of architectures present in the tunable Goldstein problem. Unfortunately, this work's implementation of Bayesian optimization, which relies on Trieste [91], requires an explicit enumeration of the entire combinatorial search space. As the number of discrete variables grows, this explicit enumeration becomes more memory-intensive, which prohibits the exploration of problems with a larger number of architectures using the framework developed in this work. The computational constraints thus introduced are a significant limitation for Trieste's handling of discrete variables when scaling up the problem's complexity. As a result, the tunable version of the Goldstein was only studied using the NSGA-II algorithm in both the nested and global exploration strategies.

Instances of the tunable problem were created with varying numbers of architectures and imputation ratios. The number of options for each architecture variable was kept constant at two choices per variable. For a fixed number of architectures, the imputation ratio was varied between 2 and 2^{12}. Problems with 64, 512, and 4096 possible architectures were studied.

For each imputation ratio and number of architecture combinations, eight optimization runs were carried out using the global exploration and nested strategies. The global exploration strategy used a population size of 100, while the nested strategy used a population size of 10 for its inner and outer loop. The nested strategy's inner loop was allowed to advance for one generation per outer loop iteration. This setup maintained the number of function evaluations per generation equal among both strategies. The total number of evaluations was fixed at 10,000 for both strategies.

After the optimization runs were done, the hypervolume, GD+, and η_0 metrics for each strategy were calculated and averaged, aggregating by imputation ratio and number of architectures. The results of this study are presented in Section 6.2.2.

6.2.2. Results and Analysis

As mentioned in Section 6.2.1, eight optimization runs were carried out for versions of the tunable Goldstein problem with changing numbers of total architectures and imputation ratios. The performance of the strategies was measured using the generational distance plus, (normalized) hypervolume, and η_0 introduced earlier. Additionally, the exploration-exploitation behavior of the strategies was quantified by measuring the fraction of the total architectures evaluated by the optimization strategies as well as the Gini index of the distribution of evaluations across architectures.

Multi-Objective Performance Indicators

Three multi-objective performance indicators were examined to provide a comprehensive understanding of the performance of the global exploration and nested strategies. These indicators offer insights into how each strategy copes with the challenges of varying architectures and imputation ratios in the tunable Goldstein problem.

Generational Distance Plus Figure 6.7 shows how the global exploration and nested strategies, both using the NSGA-II algorithm, behave as a function of the tunable problem's imputation ratio and total number of architectures. In general, across architectures and imputation ratios, the global strategy significantly outperforms the nested one. While the global strategy consistently achieves near-zero generational distances, the nested optimization strategy's mean generational distances vary from 0.3 to 1.4, with considerable variation across runs.

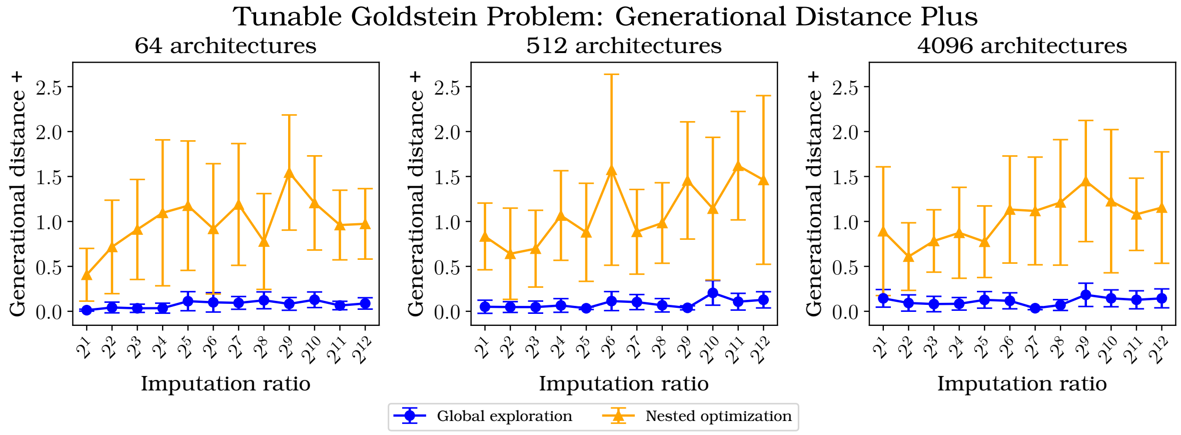


Figure 6.7: Generational distance plus as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. The error bars indicate a 95% confidence interval around the mean. Both the global exploration and the nested optimization strategies used the NSGA-II algorithm and carried out 10,000 total function evaluations.

The global exploration strategy's generational distance plus remains relatively constant as a function of the imputation ratio, while it increases subtly as the problem's number of architectures increases. Meanwhile, the nested strategy shows a slight upward trend in its generational distance plus an increasing imputation ratio. This trend suggests that as the hierarchy of the problem intensifies, the nested strategy faces more challenges in converging to the Pareto front, leading to a larger GD+. However, there is no significant trend in the nested strategy's generational distance plus as the number of architectures increases, possibly indicating that the architectural combinatorial complexity is less of a determinant factor in its performance than the hierarchy introduced by the imputation ratio.

Hypervolume Figure 4.2 illustrates the performance of both the global exploration and nested optimization strategies, employing the NSGA-II algorithm, in response to variations in the tunable problem's imputation ratio and overall number of architectures. As for the GD+ indicator, the global exploration strategy consistently outperforms the nested optimization strategy across all the imputation ratios and numbers of architectures studied. Furthermore, the uncertainty intervals for the global exploration strategy are markedly smaller than those of the nested one, suggesting that this method performs more reliably.

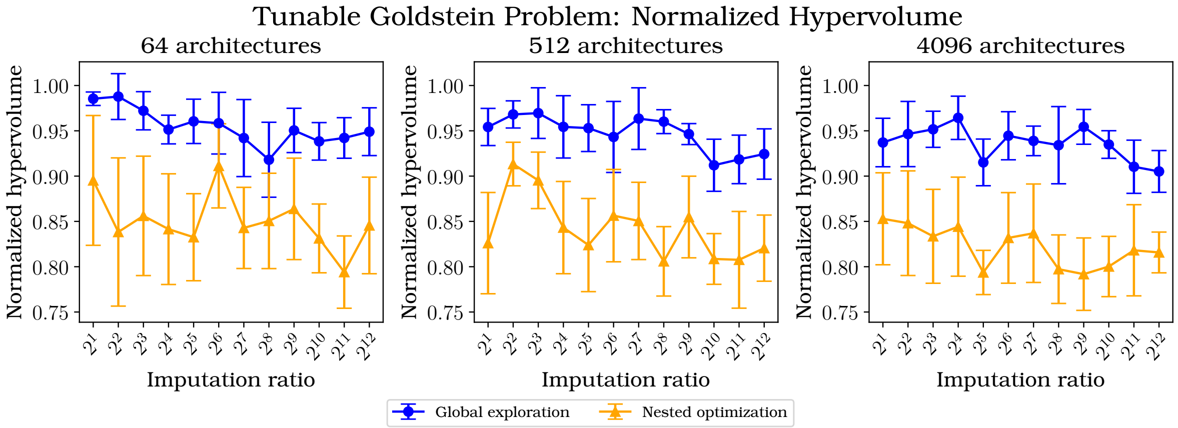


Figure 6.8: Normalized hypervolume as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. The error bars indicate a 95% confidence interval around the mean. Both the global exploration and the nested optimization strategies used the NSGA-II algorithm and carried out 10,000 total function evaluations.

With a rise in the number of architectures, there is a corresponding decline in the hypervolume of the global exploration strategy. For instance, at an imputation ratio of 2, the mean normalized hypervolume for the global exploration strategy stands at 0.98 with 64 architectures, drops to 0.96 for 512 architectures, and further reduces to 0.94 when there are 4096 architectures. The nested strategy's performance as a function of the number of architectures is more erratic: there is no consistent trend with increasing architectures. Meanwhile, for all the numbers of architectures studied, both strategies show a slight decrease in their hypervolume with increasing imputation ratio. This finding shows that a problem's imputation ratio plays a considerable role in influencing the performance of both strategies. As the imputation ratio increases, indicating a heightened hierarchical structure in the problem, both the global exploration and the nested strategies find it more challenging to find the Pareto front.

Non-Dominated Improvement (η_0) As discussed in Section 4.3.3, the η_0 indicator measures the quality of an optimization run's non-dominated front against that of a reference design of experiments. For each instance of the tunable Goldstein problem, characterized by its number of architectures and imputation ratio, reference DoEs were carried out using a nested random sampling of the hierarchical design space as proposed by Sonneveld et al. [113]. The η_0 metric as a function of the tunable Goldstein problem's number of architectures and imputation ratio is shown in Figure 6.9. As with the GD+ and hypervolume indicators, the global exploration strategy consistently outperforms that nested optimization strategy across all configurations. While the global exploration method reliably achieves the highest possible η_0 of 100%, the nested strategy's mean η_0 fluctuates between 91% and 98% without a discernible pattern either as a function of the number of architectures or with respect to the imputation ratio.

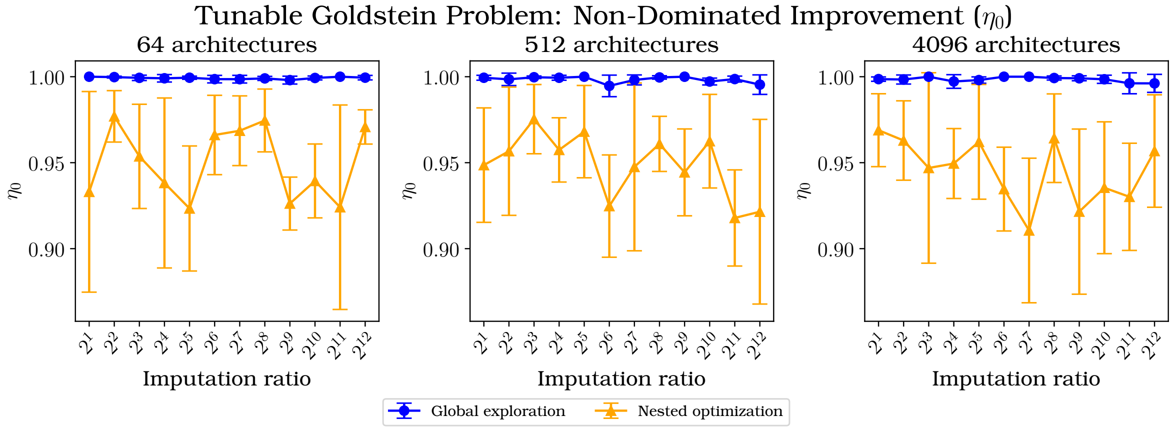


Figure 6.9: Non-dominated improvement ratio (η_0) as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. The error bars indicate a 95% confidence interval around the mean. Both the global exploration and the nested optimization strategies used the NSGA-II algorithm and carried out 10,000 total function evaluations.

Summary of Findings Throughout the different evaluations, it is evident that the global exploration strategy exhibits superior performance over the nested strategy across various imputation ratios and numbers of architectures. This superior performance is evident in metrics like the generational distance plus, hypervolume, and the η_0 indicator. However, it is crucial to acknowledge the nested strategy's efficacy. Despite its underperformance compared to global exploration, the nested strategy demonstrates its ability to locate points in the objective space proximate to the Pareto front. This capability underscores its effectiveness, particularly when compared to conducting a random sampling DoE for the same number of evaluations. In other words, while the global strategy stands out in direct comparison, the nested strategy remains viable. This insight is significant, as not all architecting problems may accommodate the global exploration strategy easily because this strategy requires identifying the comprehensive design vector that includes every architecture's variables. This task can often be challenging due to high hierarchies or intricately connected meta variables. In such scenarios, the nested strategy may be the only viable option, and this study has shown that, while its performance is not as good as that of the global exploration strategy, it is still a valuable method for navigating the design space.

EXPLORATION-EXPLOITATION TRADE-OFF

In any optimization procedure, there is a trade-off between exploration and exploitation. Exploration focuses on investigating new areas of the design space, seeking diverse solutions, and broadening the search horizon. In contrast, exploitation aims to refine known good solutions, concentrating the search around promising areas. Balancing these two aspects is crucial for ensuring both the diversity and quality of solutions. The fraction of the total architectures explored and the architecture Gini index, as defined in Section 4.3.4, can be used to gauge the degree of exploration and exploitation in an optimization procedure.

Fraction of Total Architectures Explored The fraction of the total architectures explored directly indicates the breadth of the search. A higher fraction indicates that the optimization strategy is leaning more towards exploration, covering a broader range of potential solutions. Conversely, a lower fraction might suggest that the strategy primarily focuses on refining a specific subset of architectures, leaning more toward exploitation.

Figure 6.10 shows the fraction of architectures explored by the global exploration and nested strategies for versions of the tunable Goldstein problem with 64, 512, and 4096 architectures and imputation ratios ranging from 2 to 2^{12}. For the same number of architectures, the fraction of architectures explored remains nearly constant for the nested and global exploration strategies. However, the global strategy always evaluates more architectures than the nested method. This indicates that the global exploration strategy is more exploratory than the nested strategy. One possible explanation for this notable difference is that the nested strategy partitions the evaluation budget into two levels, meaning that once it selects an architecture in its outer level, it has to perform a given number of evaluations of that architecture in its inner level. Meanwhile, because the global strategy effectively "flattens" the problem's hierarchy, it can produce populations with diverse architectures in every generation.

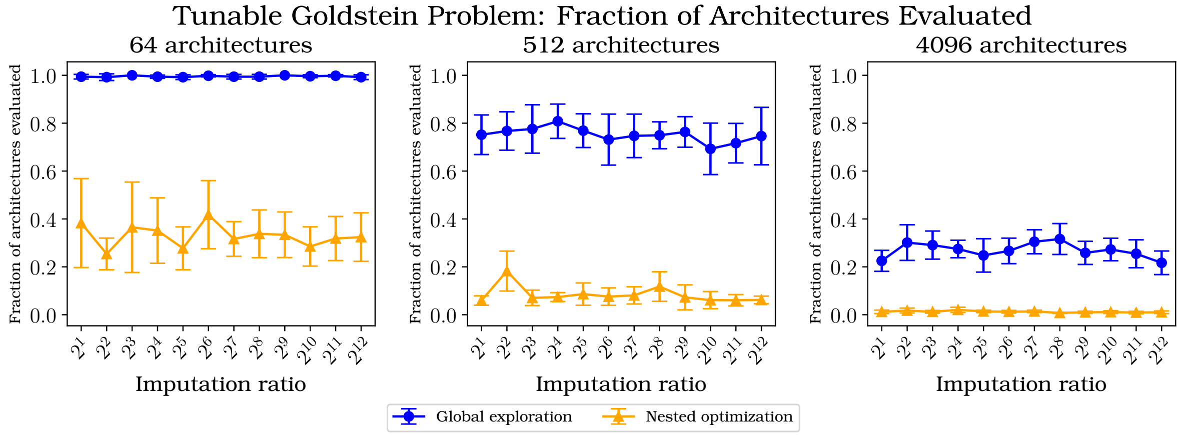


Figure 6.10: Fraction of explored architectures as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. The error bars indicate a 95% confidence interval around the mean. Both the global exploration and the nested optimization strategies used the NSGA-II algorithm and carried out 10,000 total function evaluations.

Interestingly, Figure 6.10 shows that the fraction of explored architectures decreases for both tested strategies as the tunable problem's number of architectures increases. This is to be expected, as a higher number of architectures for the same evaluation budget means that each architecture receives, on average, fewer evaluations. The optimization algorithms must spread their fixed budget of evaluations over a larger number of potential solutions, inevitably resulting in a reduced exploration rate.

Architecture Gini Index The architecture Gini index captures the unevenness or inequality in the distribution of evaluations across architectures. A Gini index closer to 0 indicates a more uniform distribution, suggesting that the strategy distributes its evaluations equally across architectures and thus emphasizes exploration. In contrast, a Gini index closer to 1 signifies a more skewed distribution. This skew indicates that the optimization strategy concentrates its evaluations on a specific subset of architectures, implying a higher degree of exploitation.

Figure 6.11 shows that the Gini index for both strategies rises as the number of architectures increases. This trend suggests a greater concentration of evaluations on specific architectures or regions of the design space. It can be inferred that the strategies shift towards a more exploitative behavior with the growth in architectural possibilities. This concentration might be due to the strategies identifying particular architectures that show promise early in the optimization process and then dedicating a disproportionate number of evaluations to refining these promising solutions.

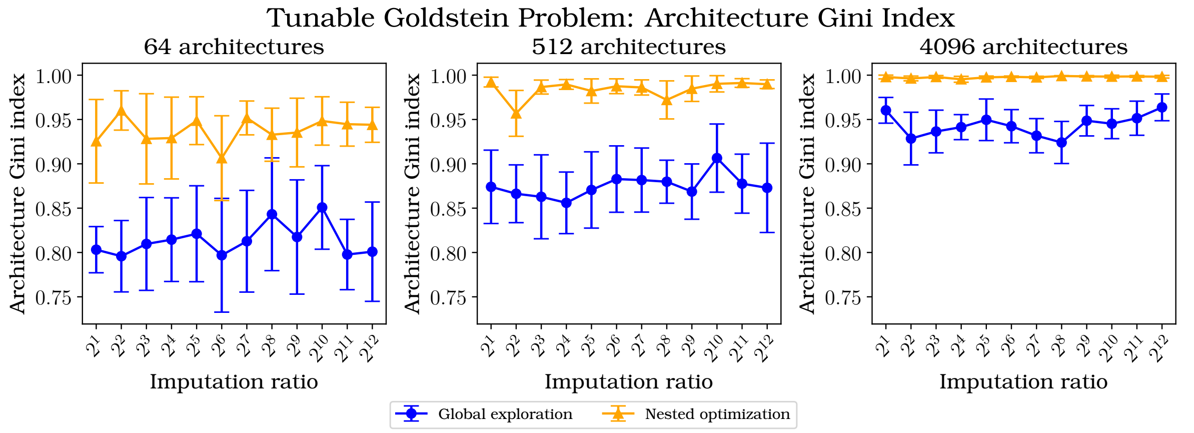


Figure 6.11: Fraction of explored architectures as a function of imputation ratio for the tunable Goldstein problem with 64, 512, and 4096 architectures. The error bars indicate a 95% confidence interval around the mean. Both the global exploration and the nested optimization strategies used the NSGA-II algorithm and carried out 10,000 total function evaluations.

Consistent with what was observed in Figure 6.10, the global exploration's Gini index remains lower than that of the nested optimization strategy, further evidencing that the nested optimization strategy relies more heavily on exploitation than the global approach. Like the fraction of architectures explored, the Gini index remains relatively constant as the tunable problem's imputation ratio increases.

A significant observation from the results is the effectiveness of both strategies in identifying designs near the Pareto front, even when faced with problems with many architectures and the associated increase in exploitative behavior. Specifically, as the number of architectures in the problem grows, the Gini index edges closer to 1, and the fraction of explored architectures nears 0. Nevertheless, the GD+, hypervolume, and η_0 metrics consistently indicate robust performance for both strategies. This highlights their ability to locate high-quality solutions even when most of the architectural space remains unexplored.

Summary of Findings In optimization, balancing exploration and exploitation is a central challenge. The measurements from both the fraction of architectures evaluated and the Gini index reveal two trends. First, as the tunable problem's architectural count increases, global exploration and nested strategies tend to explore fewer architectures. Second, there is a noticeable rise in the imbalance of how evaluations are distributed among these architectures. Despite this increase in exploitation, the global strategy consistently surpasses the nested approach in its breadth of exploration. Nonetheless, both strategies exhibit a resilient capability to identify solutions proximate to the Pareto front—even in the face of increasing architectural possibilities and a heightened exploitative focus. This underscores the efficacy of these strategies in diverse optimization landscapes.

7

Aileron Optimization

This chapter discusses the application of the global exploration and nested optimization strategies in a system architecture optimization problem under study at GKN Fokker and Delft University of Technology within the DEFAINE[1] project that deals with the structural design of an (uncrewed) aircraft's aileron.

7.1. Problem Formulation

As discussed in Chapter 1, the structural design of an aileron can be seen as a system architecture problem due to the hierarchy between several of the variables in the problem. For instance, if the number of hinges connecting the aileron to the main wing is considered a design variable, the number and location of the *material zones* on the aileron's spars and skins will be dependent on the value assigned to the number of hinges. Furthermore, the spaces between the primary ribs (one at each spanwise end of the aileron and one behind each of its hinges and actuators), or *bays*, will change according to the selected number of hinges, allowing for more or fewer secondary ribs to be placed in these areas. Additionally, an engineer working on this problem may choose the material type for all three of the aileron's main components: spars, skins, and ribs. Every choice of material type will lead to a different set of material zone thicknesses. Moreover, the hinge and spar positions are also subject to change within a continuous range. Finally, this design problem aims to minimize the aileron's mass and its manufacturing cost simultaneously.

Table 7.1 shows the variables of the aileron problem, as well as their domains, types, and roles. The roles in Table 7.1 follow the convention discussed in Section 3.2.2, where the meta role means a variable's value affects the presence or bounds of other variables, and the decreed property means a variable's presence or value in the problem is affected by another variable. It is worth noting that the number of ribs per bay in the aileron optimization problem depends on the choice of the number of hinges, and it affects the layout and number of skin and spar material zones. As a result, this variable has been marked as meta **and** decreed.

Table 7.1: Variables of the aileron optimization problem.

Variable	Domain	Type	Role
Skin material library	{Thermoplastics, Thermosets, Thermoset core}	Categorical	Meta
Rib material library	{Thermoplastics, Thermosets, Thermoset core}	Categorical	Meta
Spar material library	{Thermoplastics, Thermosets, Thermoset core}	Categorical	Meta
Number of hinges	{2, 3, 4}	Integer	Meta
Normalized spanwise position of hinge i	[0, 1]	Continuous	Decreed
Number of ribs in bay i	{0, 1, 2, 3}	Integer	Meta **and** decreed
Skin material zone thicknesses	Depends on skin material library	Categorical	Decreed
Rib material zone thicknesses	Depends on rib material library	Categorical	Decreed
Spar material zone thicknesses	Depends on spar material library	Categorical	Decreed

Equation 7.1 formally defines the aileron optimization problem. In addition to the design variables and objectives already discussed, this problem has one constraint: the structure's minimum reserve factor (RF) must be greater than 1. This constraint has two load cases: maximum upwards dynamic pressure and maximum downwards dynamic pressure.

[1] https://www.defaine.eu/. Accessed 2023-07-30.

$$\begin{array}{lll}
\text{Minimize} & \boldsymbol{f} = [\text{mass}, \text{cost}] & \\[1ex]
\text{With respect to} & \text{Number of hinges} & \text{(meta, integer)} \\
 & \text{Skin material library} & \text{(meta, categorical)} \\
 & \text{Spar material library} & \text{(meta, categorical)} \\
 & \text{Rib material library} & \text{(meta, categorical)} \\
 & \text{Hinge positions} & \text{(decreed, continuous)} \\
 & \text{Number of ribs per bay} & \text{(meta } \textbf{and} \text{ decreed, integer)} \\
 & \text{Material zone thicknesses} & \text{(decreed, categorical)} \\[1ex]
\text{Subject to} & g = \text{Minimum reserve factor} - 1 \leq 0 &
\end{array} \tag{7.1}$$

In this aileron optimization problem, the top-level meta variables are the number of hinges and the material library selection for each component. As a result, the possible architectures for the aileron are defined by the combinations of these meta variables. Consequently, given that each of these variables has three discrete options, the total number of architectures in this problem is $3^4 = 81$.

For this problem, the aileron designs and their related disciplinary tools were accessed via GKN Fokker's in-house knowledge-based engineering application, MultiDisciplinary Modelers (MDM) [114]. More information on this application, particularly the moveable generator from which the aileron was instantiated, is available in [115]. The cost calculations were carried out with the the open-source tool CATMAC, which employs data available in the public domain to provide estimates of the manufacturing costs of aircraft components [116].

7.2. Strategy Implementation

It was decided that only the global exploration and nested strategies would be implemented. The decision chain strategy was discarded due to the amount of setup and testing that must be done before carrying out the actual optimization. The decision chain approach requires the creation of an environment, a problem-specific formulation of the state and action spaces, and an appropriate reward function. These must be refined iteratively to ensure the agent will interact with the environment as expected. Because function evaluations in this problem take minutes, this iterative process would have been prohibitively resource- and time-intensive.

Figures C.1 and C.2 show the XDSM representations of the global exploration and nested optimization strategies implemented to solve the aileron optimization problem. Both XDSMs show that a heuristic search solver component handles the material zone thicknesses. The procedure followed by the solver is further explained in Algorithm A.5. This solver was incorporated for two reasons: first, MDM includes a basic sizing tool that directly searches for material zone thicknesses that satisfy the minimum reserve factor constraint using only one FEM run per configuration. This tool was leveraged to increase the feasible points analyzed within the optimization runs. Second, although an approach similar to the basic sizing tool might have been incorporated into the optimization routine, it would have required considerable storage space (on the order of hundreds of gigabytes). This storage requirement arises because the basic sizing tool needs the disk files of a particular FEM run to obtain the load paths it will use for its calculations. This means that for every configuration seen by the optimizer, the particular FEM run would have to be saved in the disk indefinitely in case that configuration is revisited.

7.2.1. Global Exploration Strategy

An extended design structure matrix (XDSM) representation of the global optimization strategy implemented for the aileron optimization problem is shown in Figure C.1 (Appendix C, page 78). The diagram illustrates that the global strategy places all design variables (except material zone thicknesses) in a single design loop.

Imputer

An imputer assigns fixed values to inactive variables according to the problem's hierarchy. In this case, two groups of variables may be inactive for a given design:

- **Hinge span fractions:** The active hinge span fraction variables depend on the selected number of hinges n_{hinges}. For instance, if the chosen number of hinges is 2, the hinge span fractions for the non-existent hinges 3 and 4 will be imputed.

- **Number of ribs per bay:** The number of ribs per bay variables also depend on the selected number of hinges. In a given design, there are $n_{\text{hinges}} + 2$ bays. Consequently, when $n_{\text{hinges}} = 2$ or $n_{\text{hinges}} = 3$, there will be inactive variables corresponding to the number of ribs per bay of unavailable bays.

The imputer determines which variables are inactive by comparing the longest possible design vector against the variables available in the current design of the aileron, represented by a moveable instance in MDM. For this particular problem, the longest possible design vector can be obtained without hurdles: a moveable can be initialized with the maximum number of hinges and maximum number of ribs in each of the resulting bays. Then, the calls to the attributes of the aileron's design instance in MDM provide the necessary list of the longest possible design vector for this problem.

Even though determining the longest possible design vector was straightforward in this case, there may be problems where achieving this is not as direct. In more complex design scenarios, factors such as more intricate interdependencies between components might complicate the determination of the longest possible design vector. For such cases, relying solely on a simple initialization based on maximum values might not suffice. A more detailed analysis, possibly employing optimization algorithms or heuristic approaches, may be needed to identify the potential design space accurately.

EVALUATION BUDGET

For this problem, two runs of the global exploration strategy were carried out. In the first one, the optimizer in Figure C.1 was a mixed-discrete NSGA-II algorithm. In the second run, the optimizer was a multi-objective Bayesian optimization algorithm guided by the expected constrained hypervolume improvement acquisition function.

Preliminary objective and constraint evaluations established that evaluating a single design point would take around three minutes. For the NSGA-II algorithm, a population size of 210 and 13 generations was thus chosen to run the problem in approximately a working week. This setup gives 2730 total evaluations, the total evaluation budget for the Bayesian optimization algorithm. It is worth noting that Bayesian optimization typically requires fewer design points than genetic algorithms, but the same number of total function calls was allocated to both algorithms to compare their behavior and performance directly.

7.2.2. NESTED OPTIMIZATION STRATEGY

Figure C.2 (Appendix C, page 79) shows the XDSM representation of the nested strategy applied to the aileron optimization problem. For the aileron problem, this strategy partitions the design space according to its hierarchy: the problem's meta variables are placed in the outer loop, while its decreed variables are put in the inner loop.

OPTIMIZATION LEVELS

As in the implementation of the global exploration strategy for this problem, the material zone thickness variables were handled by a heuristic search algorithm. In this case, the outer-level variables are the meta variables identified in Table 7.1: the number of hinges and the material library selection for each of the aileron's components. Meanwhile, the inner optimization loop handles the hinge spanwise positions and the number of ribs per bay.

EVALUATION BUDGET

Two runs of the nested optimization strategy were carried out for this problem. In both runs, the outermost variables were controlled using the NSGA-II algorithm. However, they differed in their inner loop approaches: one run employed the NSGA-II algorithm, whereas the other utilized Bayesian optimization with *expected constrained hypervolume improvement* as its acquisition function.

To make the nested optimization generational behavior comparable to that of the global exploration strategy, a budget of 210 function evaluations per outer loop iteration was set. In the two-level nested strategy, these 210 evaluations must be split between both levels: the outer-loop population size multiplied by the inner-loop evaluation budget should equal 210. As a result, the outer loop population was set at 35, while the inner loop evaluation budget per iteration was set at 6. As in the global exploration strategy, the number of outer-loop generations was set at 13 to allow the problem to run for approximately one working week.

7.3. Optimization Results

This section presents the results of the two optimization strategies implemented for the aileron optimization problem. The optimization results are compared to each other and to a random sampling design of experiments.

7.3.1. Reference Design of Experiments: Nested Random Sampling

Because the solution to the aileron optimization problem is unknown, a reference DoE was carried out to set a baseline for the performance of the different optimization strategies. Similar to the methodology followed in [113, 114], a nested approach was conducted to conduct the DoE. The levels in this nested approach are the same ones as those described in Section 7.2.2. The heuristic search solver for the material zone thickness variables discussed earlier in this chapter was also incorporated into the DoE. In each nested level, a uniform random sampling was conducted to select the points to be evaluated. The evaluation budget for the DoE was the same as the one set up for the optimization runs; a total of 2730 points were evaluated.

The nested random sampling DoE can be interpreted as the worst-case scenario for the optimization problem, as it does not use prior knowledge, heuristics, or learning mechanism to guide the search. Instead, it relies purely on random chance to find high-performance designs. As a result, this reference DoE ensures that any improvement observed in the optimization runs is not merely due to chance.

This nested random sampling DoE acts as a control group in an experimental setup. It makes it possible to quantify the effectiveness and efficiency of the implemented optimization strategies by offering a direct comparison against a purely random search.

General Results of the Random Sampling DoE

Figure 7.1 shows the objective space results of the random sampling DoE. 78.5% of the DoE's evaluations were feasible. 1.47% of the evaluations were unfeasible, meaning that the heuristic search did not succeed in finding material thicknesses that would yield a minimum reserve factor greater than 1. The remaining 20.03% of the DoE's evaluations failed. Upon closer analysis, the reason for the failures was identified as a problem within the FEM workflow. For some configurations, the meshing process did not produce an adequate mesh quality, leading to further errors in the process.

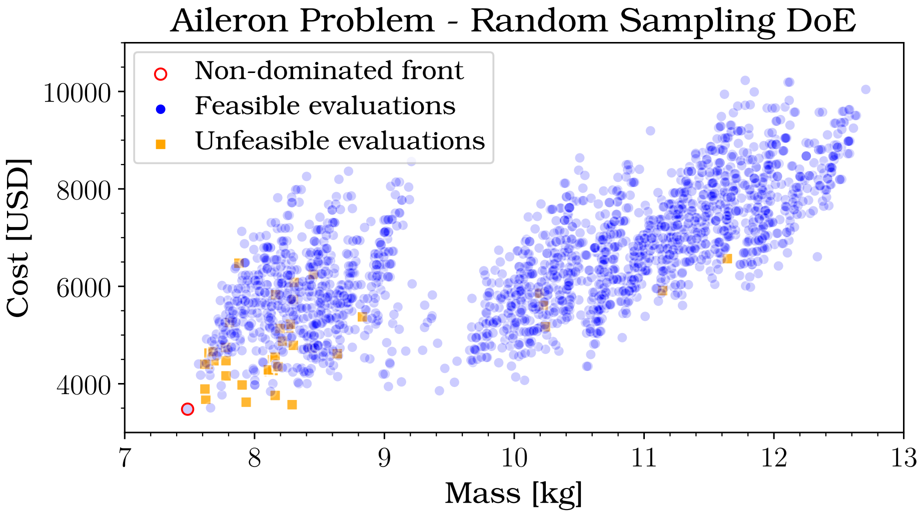


Figure 7.1: Objective space results for a random sampling DoE in the aileron optimization problem.

The objective space results shown in Figure 7.1 demonstrate a relationship between the aileron's mass and cost. Qualitatively, the plot shows that as the aileron's mass increases, its cost also tends to increase. The Pearson correlation coefficient between the mass and cost of the feasible evaluation was calculated to assess this relationship. This coefficient measures the linear relationship between two variables and ranges between -1 to 1. Values close to 1 indicate strong positive linear correlations, while values close to -1 indicate strong negative linear correlations. Values around 0 indicate no linear relationship between the variables. For the feasible results of the random sampling DoE, the Pearson correlation coefficient was 0.68. This value suggests a moderate to strong linear correlation between the objectives, which may result in a single optimum point that satisfies both minimum mass and minimum cost. This interpretation is further supported when considering that CATMAC, the open-source tool used to estimate the aileron's cost, uses the parts' mass as an important feature within its calculations [116].

DoE Results Grouped by Design Variables

The objective space results of the random sampling DoE can be analyzed as a function of the design variables that make up the aileron optimization problem. This analysis can predict the general characteristics of optimal designs, even before any optimization procedure. Understanding how various design variables influence the problem's objectives can lead to insights into the likely regions where optimal solutions might be found.

DoE Results by Material Library An interesting relationship between the material libraries of the aileron's different components is visible from the random sampling DoE, as shown by Figure 7.2. The figure shows that skin and spar material library selections are crucial factors in determining the aileron's mass. The plot colored by skin material library shows that three distinct groups appear in the objective space according to the library choice: designs with thermoplastic composite skins are the lightest, followed by those with thermoset composite skins. All the heaviest designs feature thermosets with honeycomb skins. These large groups are further divided by the choice of spar material library; once again, the lightest designs have thermoset composite spars. Finally, the rib material library does not produce clearly discernible patterns in the objective space, suggesting that its influence on the aileron's mass and cost is relatively minor compared to the skin and spar materials.

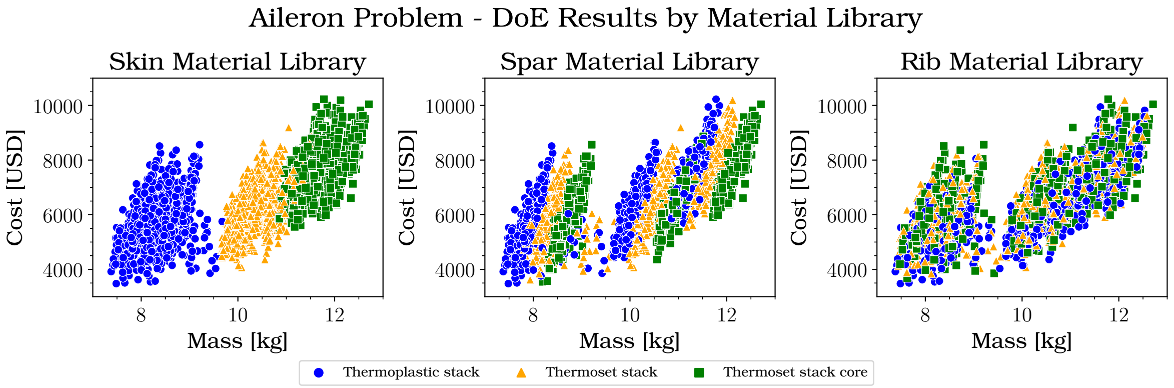


Figure 7.2: Random sampling DoE objective space results for the aileron problem grouped by skin, spar, and rib material library.

It is worth noting that the choice of thermoplastic composite materials for both the skin and spar results in the most lightweight designs. This outcome likely arises from the inherent properties of thermoplastics, which generally offer a favorable strength-to-weight ratio when compared to other materials. On the other hand, including honeycomb structures in thermoset skins increases the weight, potentially due to the added mass from the honeycomb core, while not providing a commensurate increase in structural performance for this specific application.

DoE Results by Total Number of Ribs The aileron's total number of ribs is the sum of its main ribs (located at its spanwise end and behind each hinge or actuator bracket) and its secondary ribs (located at the bays between the main ribs). Figure 7.3 shows how the number of ribs affects the design objectives of mass and cost. The cost vs. mass plot shows that the number of ribs does not significantly affect the aileron's mass; there is a more or less even distribution of numbers of ribs across the range of masses found in the DoE. In contrast, the aileron's cost shows a noticeable dependency on the total number of ribs, as shown by the cost vs. total number of ribs plot. This scatter plot also illustrates how part of the spread in cost as a function of ribs can be attributed to the skin material library selection.

From these results, it can be expected that the optimizers will attempt to produce feasible designs with as few ribs as possible to reduce the cost objective.

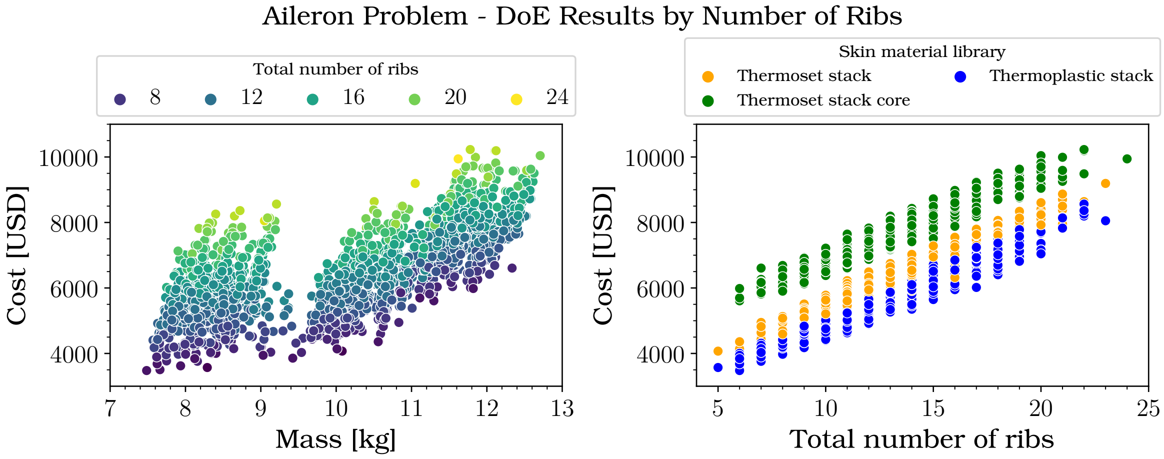


Figure 7.3: Random sampling DoE objective space results for the aileron problem grouped by total number of ribs.

7.3.2. GLOBAL EXPLORATION

As discussed earlier, two optimization runs were carried out using the global exploration strategy: one with NSGA-II as an algorithm and the other employing Bayesian optimization instead. In this strategy, all the design variables are considered simultaneously in a single optimization loop, and any inactive variables are handled through imputation to avoid superfluous evaluations, wasteful operations, and inaccurate modeling of the design space.

GLOBAL EXPLORATION WITH NSGA-II

Figure 7.5 shows the objective space results for all the function evaluations carried out with the global exploration strategy using the NSGA-II algorithm for the aileron optimization problem discussed in this chapter. 89.9% of the evaluations were feasible, 3.4% were unfeasible, and the remaining 6.7% failed to evaluate the constraint function.

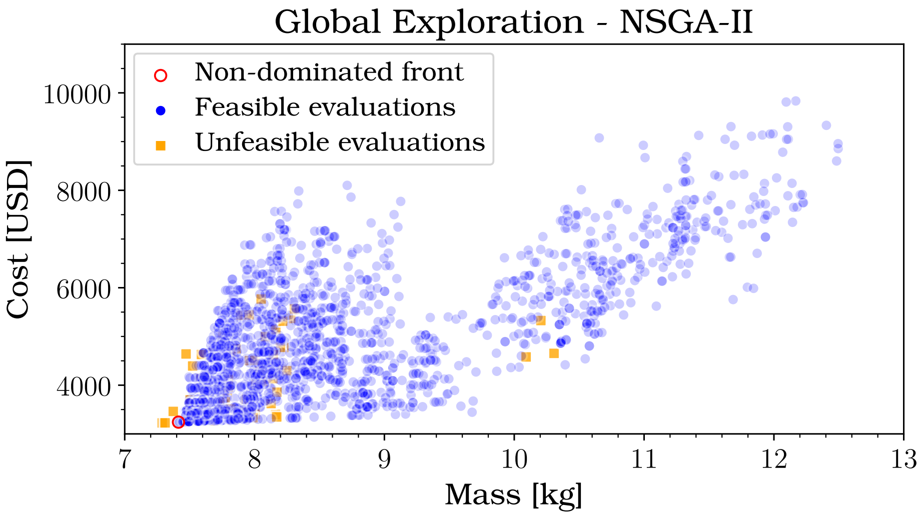


Figure 7.4: Objective space results for a global exploration strategy run of the aileron problem using NSGA-II.

The global exploration strategy powered by the NSGA-II algorithm produces markedly different objective space results than the random sampling DoE. The optimization ignores points with high mass and cost in favor of points with low mass and cost. There is a high density of evaluations of designs with a mass below 8 kilograms and cost below 4,000 USD. Additionally, it appears that the optimization found a “floor” to the cost as a function of the mass, as evidenced by an emerging straight line of feasible evaluations at the bottom of the plot.

In this case, the global exploration strategy with NSGA-II produced a non-dominated front with a single point. This point dominates the non-dominated point found by the random sampling, and it is accompanied by 26 other designs that dominate the best point found by the random sampling DoE. Even though this run produced several good design points, the global exploration strategy still maintained a level of exploration that allowed it to investigate relatively unknown areas of the design space. Consequently, it can be said that the global exploration strategy with NSGA-II favors exploration in the exploration-exploitation trade-off.

The results of this run show that the global exploration strategy paired with the NSGA-II algorithm is an effective way to deal with system architecture optimization problems like this chapter's aileron optimization.

Global Exploration with Bayesian Optimization

Figure 7.5 shows the objective space results of an optimization run with the global exploration strategy using Bayesian optimization as its algorithm. In this case, the Bayesian optimization routine seems to have arrived faster than its NSGA-II counterpart at the portion of the design space that produces the lowest masses and costs. In this case, there is even less spread in the objective space, and it only takes a few evaluations for the Bayesian optimization algorithm to move on from areas that are not promising. However, in this run, the Bayesian optimization algorithm produces many more unfeasible or failed points than the NSGA-II run and the random sampling DoE. For this run, 38.86% of the points were feasible, 29.96% were unfeasible, and 31.17% were failed.

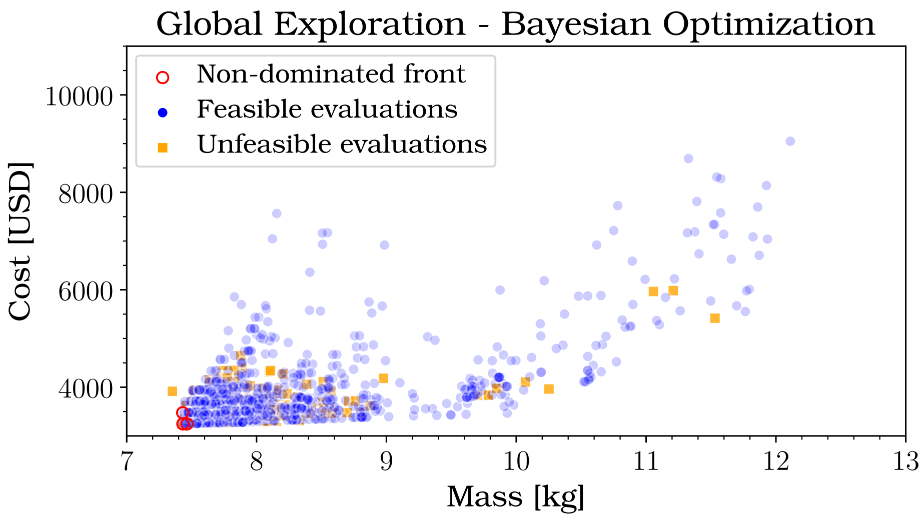


Figure 7.5: Objective space results for a global exploration strategy run of the aileron problem using Bayesian optimization.

This behavior might be due to the nature of Bayesian optimization, which builds a probabilistic model of the objective function and uses it to predict the most promising regions of the design space. While this approach is highly efficient in identifying optimal or near-optimal solutions, it can sometimes lead the search into regions where constraints are violated or the system exhibits behaviors that are challenging for the underlying evaluation tools. Another possibility is that the Bayesian algorithm has encountered a “wall” that delimits the feasible region of the objective space. Most of its attempts to improve over its current non-dominated set lead to unfeasible or failed evaluations. The rapid concentration of evaluations in the most promising area of the objective space showcases the efficiency of Bayesian optimization and suggests that the global exploration strategy with this algorithm favors more exploitation over exploration.

In this run, the global exploration strategy with Bayesian optimization yielded a non-dominated front with three points, all dominating the non-dominated point from the reference DoE. Additionally, 29 of the feasible points found by this run dominate the DoE's non-dominated point, indicating a significant improvement in the search for optimal designs over the naive random sampling approach.

7.3.3. Nested Optimization

The nested optimization strategy removes the need to handle inactive variables because it partitions the design space such that the meta variables dictate the creation of subproblems with only the relevant decreed variables present. This approach also opens an opportunity to mix and match different types of algorithms along its levels, as each level can be tailored to the specific characteristics and challenges of its associated subproblem. By decomposing the design space in this manner, the strengths of various optimization algorithms can be leveraged, potentially accelerating convergence and improving solution quality.

For instance, in the higher levels where the design space may be more abstract or categorical, genetic algorithms might be more appropriate due to their exploratory nature. In contrast, gradient-based methods or Bayesian optimization might be more effective in the lower levels, where the design variables might be continuous and the design space smoother.

As discussed in Section 7.2.2, two runs were carried out for the aileron problem using the nested optimization framework with two levels. The first run used NSGA-II for both levels, while the second used NSGA-II in its outer level and Bayesian optimization in its inner level.

Nested Optimization with NSGA-II in Both Levels

Figure 7.6 shows the objective space results for a two-level nested optimization of the aileron problem using NSGA-II in both of its levels. The objective space in this run is much more sparsely distributed than in the random sampling DoE and the global NSGA-II run. As in the run with the global exploration strategy with Bayesian optimization, the nested NSGA-II concentrates most of its evaluations in the corner of the objective space where both mass and cost are low. The nested optimization run achieved a very high percentage of feasible evaluations, with 95.9% of its designs being feasible. Meanwhile, 1.0% of the evaluations were unfeasible, and 3.2% failed. This indicates that the nested structure, combined with the NSGA-II approach at both levels, effectively navigates the design space while largely adhering to the problem's constraints.

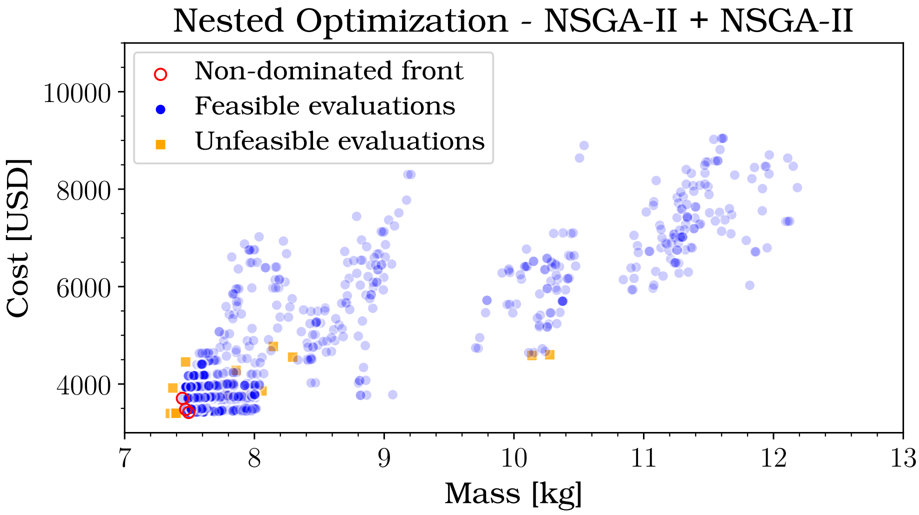


Figure 7.6: Objective space results for a nested strategy run of the aileron problem using NSGA-II in the outer and inner level.

The distribution of points suggests that the nested NSGA-II approach rapidly identifies and focuses on the promising regions of the design space, resulting in fewer evaluations in the less desirable areas. This concentration of evaluations in the low-mass and low-cost corner of the objective space is indicative of the algorithm's capability to find optimal or near-optimal solutions efficiently. As a result, the nested strategy with NSGA-II in both levels favors exploitation over exploration in its search of the design space.

This nested optimization run did not produce any feasible design points that dominate the reference DoE's non-dominated front. However, all three points in this run's non-dominated front do improve upon the DoE's best point in at least one of the objectives.

Nested Optimization with NSGA-II in the Outer Level and Bayesian Optimization in the Inner Level

Figure 7.7 shows the results of an optimization run of the aileron problem using the nested strategy with NSGA-II at the outer level and Bayesian optimization at the inner level. The distribution of points in the objective space is similar to that of the other run with the nested strategy (Figure 7.6), highlighting the influence of the top-level architectural variables on the problem's objectives. 61.47% of the evaluated designs were feasible, 27.11% were unfeasible, and the remaining 11.43% were failed.

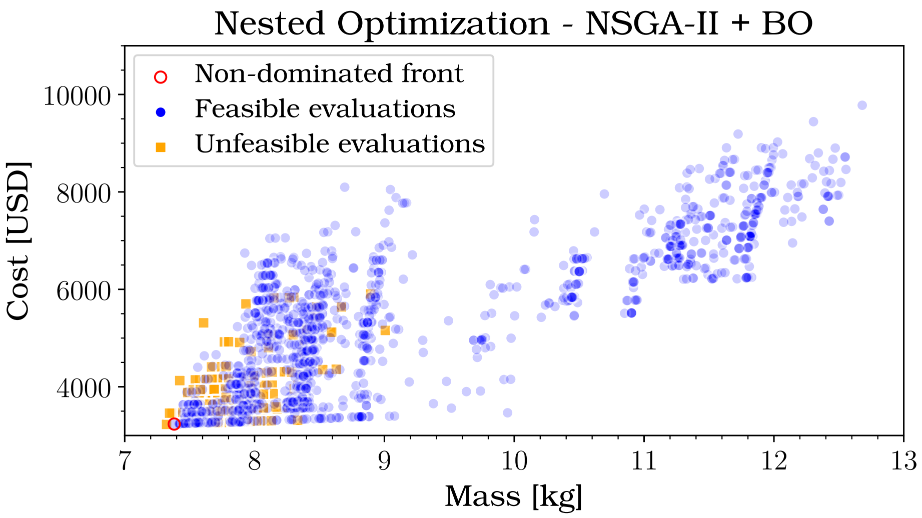


Figure 7.7: Objective space results for a nested strategy run of the aileron problem using NSGA-II in the outer level and Bayesian optimization in the inner level.

One interesting difference between this run's results and the two-level NSGA-II run is the concentration of points around a suboptimal area in the objective space. Figure 7.7 shows that several designs led to masses greater than 8 kg and costs of over 5,000 USD, well above the best points found by the other strategies, which have masses of around 7.5 kg and costs of around 3,500 USD. This behavior highlights the risks of employing strategies and algorithms that heavily favor exploitation over exploration: by becoming too myopic in their search, they can get trapped in local optima and miss out on globally better regions of the design space.

Nonetheless, this nested optimization run found a non-dominated front with one point that dominates the best point found by the reference DoE. In total, this run of the nested optimization strategy found 81 points that dominate the DoE's non-dominated front.

7.4. Comparison of Results

The results of the runs presented in Section 7.3 are compared in this section. The hypervolume (scaled and normalized with respect to the reference DoE) and η_0 indicators are used to assess the performance of the different strategy and algorithm combinations. Because no solution is known for this problem, the generational distance plus metric is not available in this test case.

7.4.1. Optimization Performance

Table 7.2 summarizes the optimization results for each strategy and algorithm combination tested in the aileron optimization problem.

Table 7.2: Summary of optimization results for the aileron problem.

Strategy	**Random Sampling DoE**	**Global Exploration**		**Nested Optimization**	
Algorithm(s)	**None**	**NSGA-II**	**Bayesian Optimization**	**NSGA-II + NSGA-II**	**NSGA-II + Bayesian Optimization**
Percentage of feasible points	78.50%	89.89%	38.86%	95.90%	61.47%
Percentage of unfeasible points	1.47%	3.41%	29.96%	0.95%	27.11%
Percentage of failed points	20.03%	6.70%	31.17%	3.15%	11.43%
Non-dominated points	1	1	4	3	1
Points dominating the reference DoE	-	27	29	0	81
η_0	-	100%	100%	75%	100%
Normalized hypervolume	1.000	1.045	1.041	1.036	1.052

Various insights can be derived from the table in terms of the distribution of feasible, unfeasible, and failed evaluations; in the characteristics of the non-dominated sets and points found in each run; and in terms of performance indicators.

Distribution of Feasible, Unfeasible, and Failed Evaluations

Comparing the proportions of feasible, unfeasible, and failed evaluations carried out by each of the strategy-algorithm combinations studied can lead to a better understanding of how each approach navigates the design space and handles the problem's constraints and failure regions.

Percentage of Feasible Points Table 7.2 shows that the nested optimization strategy using NSGA-II in both of its levels yields the highest percentage of feasible designs, at 95.90%. This high percentage suggests that this strategy-algorithm combination effectively finds valid design points. Additionally, this result may indicate a high degree of exploitation of areas of the design space that are known to produce feasible evaluations. The lowest percentage of feasible points comes from the global exploration strategy with Bayesian optimization, at 38.86%. In this case, the surrogate model behind the optimization algorithm might have struggled with complexities in the minimum reserve factor constraint, especially considering that some evaluations can fail when evaluating the constraint. This observation is supported by the relatively low percentage of feasible points (61.47%) found by the hybrid NSGA-II + Bayesian Optimization setup for the nested strategy.

Percentage of Unfeasible Points According to Table 7.2, the nested strategy with NSGA-II in both of its levels produced the lowest percentage of unfeasible points, at 0.95%. Meanwhile, both of the runs that used Bayesian optimization yielded high percentages of unfeasible evaluations, at 29.96% for the global strategy and 27.11% for the nested strategy. This result can indicate difficulties in modeling the constraint as a Gaussian process, especially considering that the constraint calculation process is the primary source of possible design evaluation failures.

Percentage of Failed Points The DoE has a failure rate of 20.03%, giving an estimate of the proportion of the points in the design space that are subject to failed evaluations. In contrast, the nested optimization strategy with NSGA-II in both levels has a lower failure rate of 3.15%. Similarly, the global exploration strategy with NSGA-II has a 6.17% failure rate. This indicates that the NSGA-II algorithm filters the failed points effectively within its evolutionary scheme. Meanwhile, the strategy-algorithm combinations that used Bayesian optimization have higher proportions of failed evaluations: the nested strategy with Bayesian optimization in its inner level has a failure rate of 11.43%, and the global exploration strategy with Bayesian optimization has the highest evaluation failure rate, with a value of 31.17%. These differences in the percentage of failed points, driven mainly by the choice of algorithm, highlight how a problem's hidden constraints might make it more difficult for a Bayesian optimization algorithm to succeed, even though techniques are available to handle failure regions.

Characteristics of the Non-Dominated Sets and Fronts

As shown in Table 7.2, the number of non-dominated points found by every optimization run (and the reference DoE) was considerably low for a multi-objective problem: three of the approaches found only one non-dominated point, and the largest non-dominated sets, obtained by the global exploration strategy with Bayesian optimization, and the nested strategy with NSGA-II, contain only three members. The small size of the non-dominated sets may be explained by the fact that, as discussed in Figure 7.1, there is a moderate to strong correlation between the aileron's mass and cost, which may make the problem only apparently multi-objective: it is likely that a single point simultaneously minimizes both of this problem's objectives.

The non-dominated fronts found by each optimization run are shown in Table 7.3 and Figure 7.8. Out of all the points in these non-dominated fronts, there is one with a lower mass and cost than all the others. The non-dominated point found by the nested approach with NSGA-II + Bayesian optimization dominates all the other Pareto points found by the other strategies, making it the most optimal solution discovered during the entire study.

Across all runs, most non-dominated points come from the architecture defined by the thermoplastic stack material library in the skins, spars, and ribs, and by two hinges. In fact, only one of the non-dominated points found by the different strategies belongs to an architecture not defined by these parameters. Furthermore, all optimization runs obtained feasible designs with no secondary ribs. This result is consistent with what was predicted from the results of the reference DoE in Section 7.3.1, where it was shown that selecting thermoplastics favors the aileron's mass, and selecting few ribs favors its cost.

Additionally, the second hinge's spanwise position is largely consistent throughout the different non-dominated points found. This can indicate the structural importance of the hinge's placement and how the

Table 7.3: Non-dominated points for the aileron problem for optimization runs with different combinations of strategies and algorithms. TP stands for thermoplastic, TS stands for thermoset, and TS Stack Core represents thermoset with honeycomb. The best point overall is highlighted.

Strategy	Algorithm	Number of hinges	Skin material library	Spar material library	Rib material library	Hinge 1 span fraction	Hinge 2 span fraction	Number of secondary ribs	Mass [kg]	Cost [USD]	Min. RF
Random Sampling DoE	None	2	Thermoplastic stack	Thermoplastic stack	Thermoplastic stack	0.29	0.63	1	7.48	3479	1.08
Global Exploration	NSGA-II	2	Thermoplastic stack	Thermoplastic stack	Thermoplastic stack	0.30	0.64	0	7.41	3242	1.01
	Bayesian optimization	2	Thermoplastic stack	Thermoplastic stack	Thermoplastic stack	0.17	0.60	1	7.43	3476	1.03
	Bayesian optimization	2	Thermoplastic stack	Thermoplastic stack	Thermoplastic stack	0.28	0.65	0	7.46	3246	1.02
	Bayesian optimization	2	Thermoplastic stack	Thermoplastic stack	Thermoplastic stack	0.29	0.63	0	7.43	3247	1.08
Nested Optimization	NSGA-II + Bayesian optimization	2	Thermoplastic stack	Thermoplastic stack	Thermoplastic stack	0.31	0.66	0	7.38	3235	1.01
	NSGA-II + NSGA-II	2	Thermoplastic stack	Thermoplastic stack	Thermoset stack core	0.20	0.60	0	7.49	3417	1.00
	NSGA-II + NSGA-II	2	Thermoplastic stack	Thermoplastic stack	Thermoplastic stack	0.18	0.59	1	7.47	3479	1.00
	NSGA-II + NSGA-II	2	Thermoplastic stack	Thermoplastic stack	Thermoplastic stack	0.18	0.59	2	7.45	3706	1.00

optimizer seeks to fully utilize the load-bearing capabilities of the main ribs instead of including secondary ribs.

Figure 7.8 shows how the non-dominated fronts found by each strategy compare to each other. As mentioned earlier, the best solution found by the global strategy using the NSGA-II algorithm dominates all other solutions. However, the figure shows that all solutions are considerably close to each other, especially in the mass dimension. Only 100 grams separate the lightest and heaviest of these designs. Meanwhile, the difference in cost across designs is more significant, with 464 USD separating the least and most expensive points.

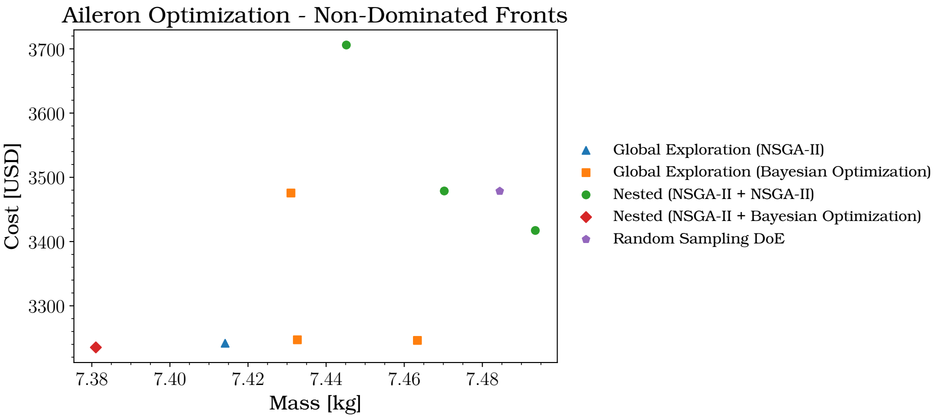


Figure 7.8: Non-dominated fronts for the aileron problem found by the different strategy-algorithm combinations.

In general, the characteristics of the non-dominated points obtained by the different strategies are similar to each other. This similarity suggests that all the strategy and algorithm combinations tested in this problem were able to find a portion of the architectural design space, predominated by one of 81 possible architectures, that produces designs with low mass and cost.

Performance Indicators

As already discussed, the generational distance plus metric cannot be used for this problem because its actual Pareto front is unknown. However, the η_0 and hypervolume metrics can be used to assess the performance of the different strategies and algorithms chosen for this problem.

Non-Dominated Improvement Ratio (η_0) As outlined in Section 4.3, the η_0 indicator quantifies the fraction of points from a given optimization run that remain non-dominated when combined with a reference DoE. In this case, the random sampling DoE discussed in Section 7.3.1 was used as the reference. Because there are so few points in the non-dominated sets, the interpretation of the η_0 indicator is limited in this case. However, it is worth noting that every optimization run except for the nested strategy with NSGA-II in both levels achieved an η_0 of 100%, meaning that all non-dominated points found were better than the non-dominated point from the DoE. Even though the nested NSGA-II did not achieve an η_0 of 100%, all its three non-dominated points were still part of the combined non-dominated front, leading to an η_0 of 75%. These results indicate that all optimization strategies were effective in finding promising points in the design space relative to the naive random sampling DoE, which validates the implementation and application of these optimization strategies for the aileron problem. While the random sampling DoE provided a preliminary understanding of the design space, the optimization strategies explored it more effectively. This improvement is evident from the η_0 values obtained. Despite the limited interpretability due to the small number of points in the non-dominated sets, these high percentages demonstrate the capability of the optimization runs to locate good design alternatives in hierarchical, mixed-discrete, multi-objective problems subject to hidden constraints, as in the aileron test case.

Hypervolume The hypervolume measures the area of the objective space dominated by a given set of points. As a result, it was necessary to normalize the cost and mass objectives to have similar orders of magnitude (and no units) to get an accurate sense of this metric. The normalization was carried out with respect to the mean mass and cost found by the DoE. Then, the DoE's hypervolume was used as a reference to scale the hypervolumes obtained by the optimization runs. As seen in Table 7.2, all optimization runs produced hypervolumes higher than that of the random sampling DoE. However, the relative improvements were small, ranging from 3.6% for the nested strategy with NSGA-II in both levels to 4.5% for the global exploration strategy with NSGA-II.

A more telling way to examine the hypervolume obtained by the different strategies is to explore how it changes as a function of the number of evaluations made. Figure 7.9 plots the hypervolume's evolution as each optimization run advances. This figure reveals that, although all optimization runs achieved a hypervolume higher than that of the random sampling DoE within the 2730-evaluation budget, some strategy and algorithm combinations managed to surpass the reference hypervolume faster than others.

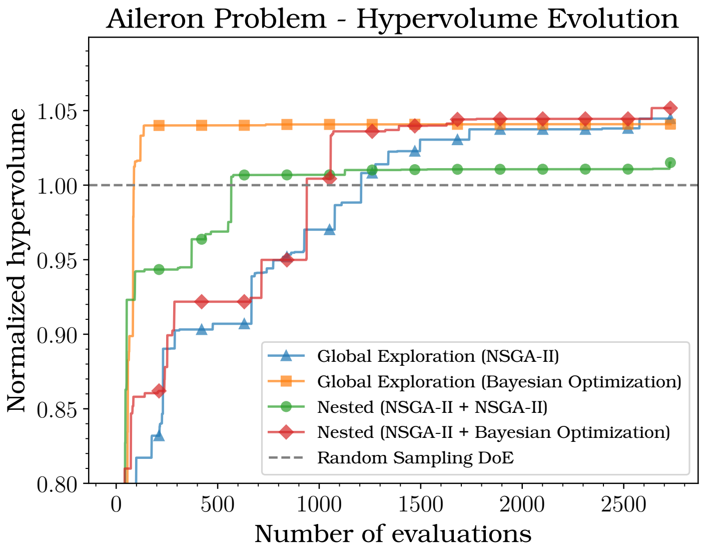


Figure 7.9: Normalized hypervolume vs. number of function evaluations for different strategies and algorithm combinations in the aileron optimization problem. The markers represent multiples of 210 evaluations, the size of one generation in the runs that use evolutionary algorithms.

The approach that used the lowest number of evaluations to surpass the DoE's hypervolume was the global exploration strategy with Bayesian optimization, which crossed this threshold after only 87 design point evaluations. Meanwhile, the global exploration strategy with NSGA-II spent the most evaluations before surpassing the DoE's hypervolume, requiring 1260 evaluations to cross the same threshold. The nested run with NSGA-II in both levels surpassed the DoE's hypervolume in 566 evaluations, while the other nested run (using NSGA-II and Bayesian optimization) surpassed it in 938 evaluations.

The significant difference in the hypervolume evolution of the global strategy runs is reminiscent of what was observed in Figure 5.1 for the airfoil optimization problem, where the single-objective Bayesian algorithm got close to the optimum design point in considerably fewer evaluations than the genetic algorithm.

One plausible reason for this trend could be the information-driven nature of Bayesian optimization. While genetic algorithms, like NSGA-II, rely on generating and evolving populations of solutions using genetic operators, Bayesian optimization is more data-focused. It creates a probabilistic regression model of the objective function based on prior evaluations and leverages this model to predict regions in the design space that are most likely to yield improvements. This model-driven approach allows Bayesian optimization to effectively combine both exploration (investigating regions with high uncertainty) and exploitation (focusing on regions known to perform well) in its search strategy.

7.4.2. Exploration-Exploitation Trade-Off

Throughout this chapter, the different optimization strategy and algorithm combinations have been qualitatively analyzed in terms of the exploration and exploitation trade-off present in all optimization and search procedures. While exploration focuses on investigating new, unknown areas of the design space, exploitation concentrates on refining the solutions in already-known promising regions.

Two measures of exploration and exploitation are used next to obtain a quantitative understanding of where each strategy stands in this trade-off:

- **Architecture Gini index:** The Gini index, commonly used in economics to gauge income inequality, measures how a given distribution deviates from a perfectly equal distribution. A Gini index of 0 represents perfect equality, while an index of 1 indicates absolute inequality [117]. As discussed in Section 4.3.4, the Gini index can be applied in architecture optimization to measure how evenly evaluations are distributed across the different possible architectures. For instance, in a scenario where each possible architecture has been evaluated an equal number of times, the Gini index would approach 0, indicating a balanced exploration of all architectures. On the other hand, if all evaluations are concentrated on just a few architectures while others remain largely unexplored, the Gini index would approach 1, signaling a disproportionate focus on specific parts of the design space and thus an emphasis on exploitation.

- **Fraction of architectures evaluated:** The fraction of architectures evaluated gives the proportion of possible architectures with at least one evaluation during the optimization or DoE run. This measure also varies between 0 and 1, where 0 indicates no architectures were evaluated, and 1 indicates that all architectures were visited. As a result, optimization methods with a high degree of exploration tend to have a fraction closer to 1, showcasing their thorough coverage of the design space. In contrast, methods more focused on exploitation might present a lower fraction, implying a concentrated effort on refining specific architectural configurations without extensively looking beyond the areas of the design space that are already known to the optimizer.

Optimization methods with high exploitation will have a high Gini index and a low fraction of evaluated architectures. In contrast, methods with high exploration will have a low Gini index and a high fraction of evaluated architectures. Following Chepko's [1] work, these two measures are plotted against each other in Figure 7.10 for the optimization strategy and algorithm combinations employed in the aileron architecting problem.

The scatter plot in Figure 7.10 shows how each combination stands in the exploration-exploitation trade-off.

Random Sampling DoE In Figure 7.10, the DoE presents the highest level of exploration and the lowest level of exploitation: it evaluated all possible architectures and maintained a relatively even distribution of the evaluation budget across architectures, as evidenced by its low Gini index of 0.34. This behavior is desirable in a design of experiments, where the aim is to gain knowledge of the design space as a whole.

Global Exploration (NSGA-II) Like the random sampling DoE, the global exploration strategy with NSGA-II evaluated all possible architectures. However, this method's moderate Gini index of 0.58 suggests that it balances exploration and exploitation, leaning slightly towards exploitation. Although it evaluated all architectures, the evaluation distribution is unequal across them.

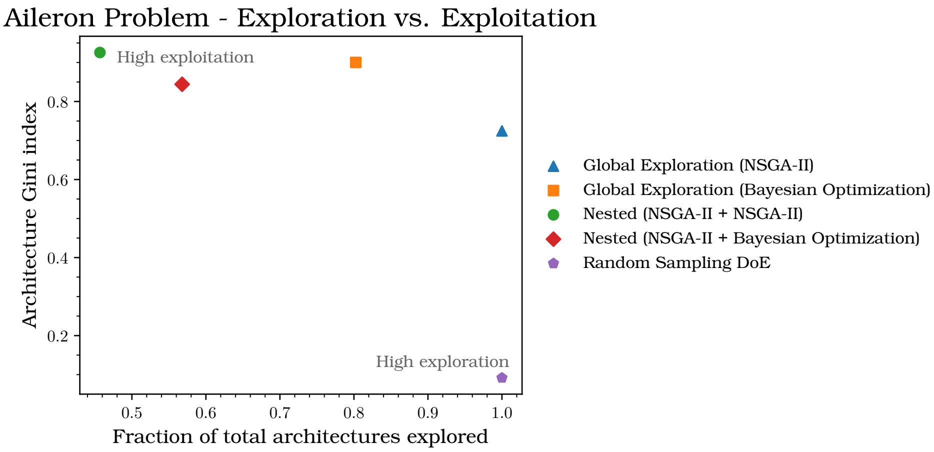


Figure 7.10: Architecture Gini index vs. fraction of total architectures explored for the different strategy-algorithm combinations run on the aileron optimization problem.

Global Exploration (Bayesian Optimization) The global exploration strategy with a Bayesian optimization algorithm evaluated 80% of all possible architectures. Despite this degree of exploration, this method's high Gini coefficient of 0.80 suggests that while many architectures were evaluated, the number of evaluations was skewed towards specific architectures, indicating a strong element of exploitation.

Nested Optimization (NSGA-II + NSGA-II) The two-level NSGA-II nested strategy is the most exploitative method in the aileron problem. Its low fraction of evaluated architectures (46%) indicates that the method was highly selective. Its high Gini index of 0.80 indicates that most evaluations were concentrated on a small subset of architectures.

Nested Optimization (NSGA-II + Bayesian Optimization) The nested strategy with NSGA-II in the outer level and Bayesian optimization in the inner level also evaluated a relatively low number of the possible architectures with a fraction of 52%, suggesting a degree of exploitation. However, the Gini index of this method was the lowest among the optimization runs, and at a value of 0.569 indicates some exploration. As a result, this method's architecture fraction and Gini index suggest that it provides balanced exploration and exploitation.

8

General Discussion

After the test case results showcased in Chapters 5, 6 and 7, this chapter discusses the general trends found across the different optimization runs carried out for each problem. The results are analyzed as a whole in terms of strategy effectiveness, strategy and algorithm implementation, advantages and disadvantages of the different strategies, and opportunities for collaborative design. Finally, a decision-making procedure is proposed to help guide the selection of an optimization strategy for a given architecting problem.

8.1. Strategy Effectiveness

Chapter 6 showed that the three studied architecture optimization strategies can find non-dominated sets effectively, with better performance than a random sampling of the design space. For the multi-objective Goldstein problem in Section 6.1, all three strategies achieved performance indicator values that evidenced this effectiveness. However, the decision chain and nested strategies tended to be more exploitative than the global exploration strategy, at the cost of, in some cases, failing to find designs corresponding to one of the Pareto-optimal architectures.

When the Goldstein problem was scaled exponentially in Section 6.2, the nested and global exploration strategies continued to show good performance, even when dealing with versions of the problem with thousands of possible architectures and considerably high imputation ratios. The performance indicators for these two strategies suggest that, when a problem scales, the relative performance of the nested and global exploration strategies remains similar. The lower uncertainty in the results for the global exploration strategy evidences that its emphasis on exploration over exploitation makes it a more reliable procedure, capable of consistently achieving high performance metrics.

For the aileron problem studied in Chapter 7, the nested and global strategies were, once again, more effective at finding good solutions than a reference design of experiments. This problem showed that, even when the objectives are aligned, the strategies can find designs close to the (single-point) Pareto set. However, it remains to be seen if a potential objective reduction procedure could benefit the optimization process in cases like this one. It would be valuable to study the application of objective reduction frameworks like the one proposed by Brockoff and Zitzler [118] in such contexts.

Chapter 7 also showed that the choice of optimization algorithm within one of the strategies can significantly affect a given strategy's behavior. For example, the nested strategy run with a Bayesian optimizer in its inner loop converged towards the best region of the design space much faster than the same strategy with NSGA-II in its inner loop. Nevertheless, the general characteristics of each strategy are still maintained regardless of the algorithm powering them. For instance, as shown in Figure 7.10, the nested strategy runs are considerably more exploitative than their global exploration counterparts.

8.2. Algorithm and Strategy Implementation

The different test problems studied in this work helped identify which algorithms and strategies were more straightforward to implement than others.

8.2.1. Algorithm Implementation

The airfoil optimization problem from Section 5.1 showed that while the Bayesian optimization and genetic algorithm can use the same problem definition and minimization procedure, the DDPG reinforcement learning approach requires the construction of a problem-specific definition of an environment, together with its own state, action, and reward definitions, as discussed in Section 4.1.4. While the problem definition used by the genetic and Bayesian algorithms is always the same (state the problem's variables and how to evaluate its objectives and constraints), the reinforcement learning environment definition is an open question. There are several ways to formulate the environment to get the agent to train in the desired way, and it is not immediately obvious what the most effective one is. Therefore, the problem definition for a reinforcement learning algorithm is considerably more intricate than that of a more traditional optimization routine.

Meanwhile, as seen in Section 5.2, the implementation of Bayesian optimization used for this work suffers from poor scalability regarding the problem's number of variables, especially if there is a large number of discrete parameters. This issue arises because Trieste's definition of the discrete subspace requires a complete enumeration of all possible discrete combinations. For instance, Section 5.2's 512-variable problem where 87.5% of the variables are discrete would require an enumeration of the 2^{488} (10^{146}) possible discrete design alternatives. For reference, the length of the largest possible Python list[1] is around 9×10^{18}. Even though one could get around this issue by carrying out a continuous relaxation of the discrete variables [41] or by using modifications of the basic Bayesian optimization algorithm tailored to high-dimensional problems, such as Random EMbedding Bayesian Optimization (REMBO) [119], it is still complicated to implement Bayesian optimization in problems with large design vectors.

At the same time, the genetic algorithms used for this work remain robust to implement and execute for differently sized design spaces and continuous-discrete compositions. The vast differences in ease and feasibility of implementation can be used together with the performance trends observed to help guide the selection of an algorithm for a given problem. For instance, Bayesian optimization has an edge over genetic algorithms when the problem's functions are expensive to evaluate and there are relatively few design variables. Meanwhile, genetic algorithms like NSGA-II excel when the design space is more extensive, and the objectives and constraint evaluations are not computationally costly.

8.2.2. Strategy Implementation

Because each of the studied system architecture optimization strategies handles the hierarchical design space differently, their implementations contrast with each other.

Global Exploration

In system architecture optimization problems, an explicit definition of the design space is sometimes feasible. In such cases, all variables, including the inactive ones, can be incorporated into a single design vector. Under these conditions, the global exploration method is the most straightforward strategy to implement. Besides the base optimization algorithm, the global strategy only requires the definition of an imputer capable of discerning which variables are active and which are not in a given design instance.

The imputer needs a way to identify the problem's meta and decreed variables to accomplish this task. This identification can be achieved by using the already available design space definition or leveraging technologies such as knowledge-based engineering (KBE). If a product's design space is already represented within a KBE application, as in the aileron case from Chapter 7, KBE's distinctive rules-based approach to modeling [120] provides a simple way to determine which variables are active and which are not. Furthermore, if the full design space is not known in advance, a KBE application may help perform a quick exploration to assess all the variables required for the global strategy's design vector.

Nested Optimization

The implementation of the nested optimization strategy is more flexible than that of the global strategy. As shown in Chapter 7, this strategy can be leveraged to "mix-and-match" different algorithms and use their strengths synergistically. For instance, while a particular algorithm might excel at exploring the top layers of the hierarchy due to its global search capabilities, another algorithm might be better suited to delve into the finer details of a nested subspace because of its local search mechanisms. Even though this work did not consider gradient-based algorithms, the nested strategy makes it possible to separate the problem's continuous variables into a single level such that gradient-based optimization can be applied effectively.

[1] This number can be checked by importing the `sys` module and printing `sys.maxsize`.

One implementation challenge of the nested optimization strategy is that the best way to partition the design space into levels may not be obvious. Even though the meta variables would logically be in outer levels and the corresponding decreed variables would be in the inner levels, the place to put the neutral variables is less evident. For example, in the Goldstein problem from Section 6.1, the nested strategy placed the meta variables w_1 and w_2 in the outer loop and the rest of the variables in the inner loop. However, another valid approach would have been placing the discrete neutral variables z_3 and z_4 in the outer loop. Further research is thus required on how to systematically and optimally partition the design space for the nested optimization strategy, especially when dealing with neutral variables.

Another aspect that must be considered when designing a nested optimization strategy for a given problem is the definition of the number of evaluations assigned to each level. An appropriate allocation of evaluations per level is crucial because the distribution of evaluations directly influences the depth and breadth of exploration in each of the hierarchy's levels.

Allocating too many evaluations to the outer loop, for instance, may result in an exhaustive search of meta variables, but it could leave insufficient resources for a comprehensive exploration of the inner layers, where the finer details of the solution reside. On the other hand, if most evaluations are assigned to the inner loops, the algorithm might overly refine a limited set of solutions without capturing the global characteristics and potential of the design space.

Decision Chain

As established previously, the decision chain strategy is the most complex to implement because it requires an environment with action, state, and reward definitions that may not translate directly from the original problem's definition. Reframing the design problem as a Markov decision process requires a deep understanding of both the original design problem and the decision-making dynamics.

As explained in Section 3.4.3, in the decision chain strategy, the state represents the current design decisions and their implications. Each action corresponds to a possible design decision, and the transition between states reflects the result of taking a particular action in the current state. The reward, on the other hand, quantifies the benefit (or penalty) of transitioning from one state to another due to a particular action.

The implementation complexity of the decision chain strategy arises in four critical areas:

- **State space representation:** An appropriate definition of states in system architecture optimization problems is complex due to their hierarchical nature. This may mean that states have different sizes or compositions, which limits the applicability of regular vector representations of states. Instead, as discussed in Section 4.2.3, states can be represented as graphs or matrices of varying dimension. However, this more elaborate representation introduces its own complications, including the need for specialized embedding techniques like graph or convolutional neural networks in case reinforcement learning is used under this strategy [27, 81].

- **Action definition:** Establishing a way to represent design decisions as actions in the decision chain approach is an open problem, and there are many possible ways to attempt to formulate the action space. Nevertheless, predicting if a particular action space definition will be successful before executing the optimization run is difficult.

- **Reward formulation:** In the decision chain strategy, rewards must be crafted to capture the agent's desired behavior. If traditional reinforcement learning methods are used with the decision chain approach, this means that all of a problem's objectives and constraints must be summarized in a single scalar. Once again, finding a reward structure that works well for a particular problem is more art than science and requires extensive experimentation.

- **Transition dynamics:** The relationships between design variables, especially in complex systems, can lead to non-linear and complicated transition dynamics. Predicting the outcome of a decision, especially when considering interactions between variables, adds another layer of complexity.

Even though the decision chain approach can be used with any optimization algorithm in deterministic problems, like its implementation with NSGA-II in Section 6.1, this strategy does not present significant performance gains over the much simpler to implement global exploration and decision chain strategies. As a result, it can be concluded that the decision chain approach should only be used when reinforcement learning techniques are going to be applied.

8.3. Potential Connections to Other Technologies

The three studied system architecture optimization strategies have the potential for numerous integration possibilities that can further enhance the systems design process. These include knowledge-based engineering (KBE) applications, collaborative web environments, and multidisciplinary design optimization (MDO) packages.

KBE Applications As discussed earlier, KBE methodologies introduce a structured way of translating design expertise and logic into computational tools. This structure aligns well with the rigorous formulation needed in optimization strategies and can be leveraged in all three strategies to simplify the optimization process. In the global exploration strategy, for example, a KBE application can help the imputer determine which variables are active in a given design instance. Similarly, the KBE application can act as the nucleus that originates the optimization subproblems in the nested strategy based on the variables that arise in the design instance after the outer-loop architectural variables have been set. Finally, the decision chain approach can benefit from KBE tools by using the KBE model as the source of truth that dictates the transition dynamics from one state to the next, ensuring that each decision aligns with best practices and adheres to specific design constraints.

Web-Based Collaborative Environments The use of web-based platforms, such as KE-chain[2], brings forth the advantage of collaborative design. Collaborative design is especially advantageous when using the nested strategy because different organizations or stakeholders can take ownership of separate optimization loops within the hierarchical design space. By distributing these tasks, the nested strategy can fully exploit the advantages of a collaborative design approach. Stakeholders can iterate on their specific domains, and the results can be communicated seamlessly through the platform. This decentralization of the optimization process also ensures that domain experts are directly involved, resulting in better-informed and more efficient design decisions.

MDO Packages When an architecting problem involves interdependent analysis tools, it becomes necessary to coordinate their execution to ensure that the design evaluations are consistent and meaningful. Multidisciplinary Design Optimization (MDO) packages, like KADMOS [121], offer solutions to manage and integrate these tools efficiently. For example, a generic framework that supports the formulation of the inner-loop optimization problems in the nested strategy has been developed using KADMOS [113]. For more complex problems, MDO packages like KADMOS can help ensure that the system architecting process remains rigorous, coordinated, and efficient, especially when addressing multiple interconnected disciplines.

8.4. Strategy Recommendation Flowchart

This work's results show that, in general, the global exploration strategy outperforms both the nested optimization and the decision chain strategies. Meanwhile, the decision chain approach is the most difficult and worst-performing of the three strategies. For this reason, even though it can be implemented with any optimization algorithm, its relatively low performance and high implementation difficulty make it the appropriate choice only when a reinforcement learning algorithm will be used to solve the optimization problem. To recapitulate, reinforcement learning can only be applied within the decision chain strategy, so if the study uses this paradigm, the decision chain is the only possible choice.

On the other hand, if reinforcement learning is not used, global exploration and nested optimization strategies are available. Based on this work's results, the global exploration strategy performs better according to most metrics. It is more reliable than the nested optimization strategy at finding near-optimal solution sets across different problems and degrees of hierarchy in the design space. Therefore, the recommendation derived from this point is to use the global exploration strategy whenever possible. This strategy can only be employed when a full design vector that captures all possible design variables is known before starting the optimization. As a result, this possibility can be used as the criterion that determines whether the global exploration or nested optimization strategies should be used: when no explicit definition of the entire design space is available, the nested strategy should be used.

After selecting a strategy, choosing an adequate algorithm or algorithms lies ahead. If the decision chain strategy is chosen, a myriad of reinforcement learning algorithms is available, each with its advantages and

[2] `https://pykechain.readthedocs.io/en/main/`. Accessed 2023-08-10.

drawbacks and its highly problem- and hyperparameter-dependent performance. Similarly, if the global exploration or nested optimization strategies are selected, there is a wide range of possible optimization algorithms. As shown in this work's test problems, there is no "one size fits all" algorithm that always works best. For instance, even though the Bayesian optimization algorithm showed a strong performance in the airfoil optimization problem from Section 5.1, it greatly struggled with the mixed-discrete, multi-objective ZDT1 problem discussed in Section 5.2. Therefore, some degree of algorithm exploration is required once an architecture optimization strategy has been selected. Figure 8.1 summarizes the recommended decision-making process for selecting a system architecture optimization strategy based on this work's findings.

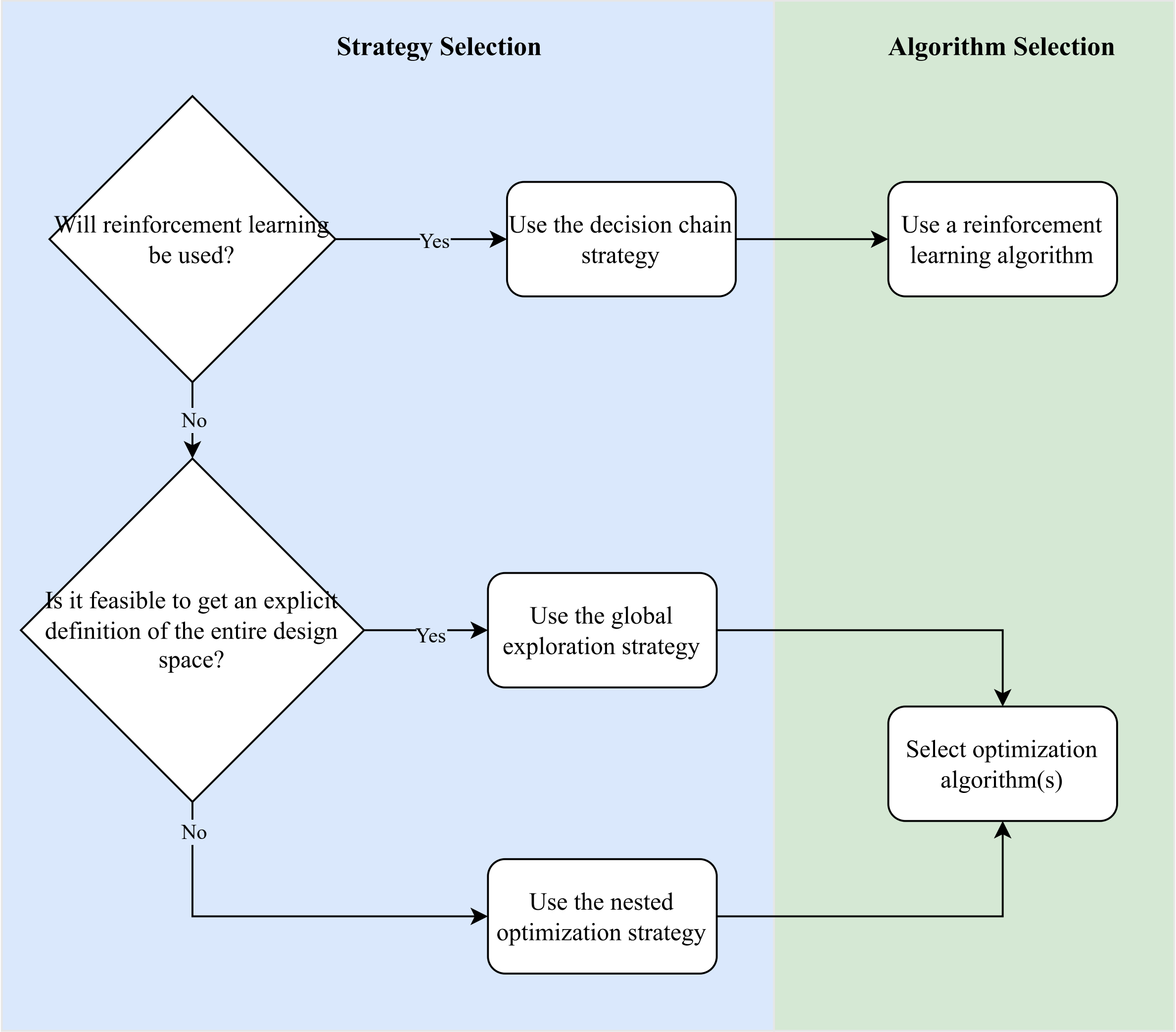

Strategy Selection
Algorithm Selection
Will reinforcement learning be used?
Yes
Use the decision chain strategy
Use a reinforcement learning algorithm
No
Is it feasible to get an explicit definition of the entire design space?
Yes
Use the global exploration strategy
Select optimization algorithm(s)
No
Use the nested optimization strategy

Figure 8.1: Flowchart for the selection of system architecture optimization strategies.

9

Conclusions & Recommendations

This chapter concludes this thesis' work on finding suitable optimization strategies to address system architecture optimization problems. First, the research questions introduced in Chapter 1 are answered, and then recommendations are given regarding possible directions for future research.

9.1. Conslusions

Given the high impact of the decisions taken during the system architecting phase of the systems design process, tools that can ensure these decisions are optimal and informed are essential. As discussed throughout this thesis, system architecture optimization strategies provide a structured framework to explore, evaluate, and select architectural configurations out of large mixed-discrete and hierarchical design spaces. Three main strategies—global exploration, nested optimization, and decision chain—were proposed to handle the hierarchical design spaces of system architecture optimization problems in conjunction with three possible families of optimization algorithms.

The questions posed at the beginning of this work can now be answered as conclusions to the research presented:

- *What are the main features of each strategy, and how can these be used to classify them?*
  While managing multiple objectives and mixed-discrete variables is left to the optimization, the optimization strategies are responsible for handling the hierarchical design spaces. Thus, the main features of each of the optimization strategies studied involve how they handle the hierarchical design space:

  - The **global exploration** strategy puts all design variables in a single design vector handled by a single optimization loop. The inactive or irrelevant variables for a selected architecture are set to a constant value by an *imputer*.
  - The **nested optimization** strategy partitions the hierarchical space into different levels, each managed by a different optimizer. The design's information is passed along the levels, allowing for fine-tuning and specialized optimization at each stage.
  - The **decision chain** strategy reframes the architecting problem as a sequence of decision-making steps. These decisions are represented as *actions* that alter *states* and produce *rewards* inside a computational *environment* and thus make the architecting compatible with reinforcement learning techniques.

- *How can the different strategies be implemented using available software tools?*

  Three main software packages allowed for the implementation of these strategies: Pymoo, Trieste, and Gymnasium:

  - The global exploration and nested optimization strategies can be implemented using well-known optimization packages such as **Pymoo**. While global exploration requires the definition of an imputer, nested optimization needs a way to allow information to travel between the different optimization levels. Pymoo provides the tools to build such mechanisms effectively. Pymoo also includes implementations of several multi-objective algorithms, such as NSGA-II, which can be used as the engine behind the optimization strategies.

- **Trieste** was successfully used to implement mixed-discrete, multi-objective Bayesian optimization algorithms that could be used with the different strategies. However, certain limitations of Trieste's implementation were identified. First, the discrete search spaces in Trieste must be completely enumerated, which restricts its use to relatively small combinatorial spaces. Second, the computational overhead of Bayesian optimization is considerable, especially after hundreds of evaluations. This overhead stems from the iterative model updates and the need to solve acquisition function optimization problems, which can take considerable time. For larger-scale problems or when many evaluations are needed, using Trieste or other Bayesian optimization frameworks may become computationally impractical.
- **Gymnasium** was employed to formulate the environments needed for the decision chain strategy. Its application programming interface provided the tools to define these environments succinctly, their state and action spaces, and their rewards.

- *How do the different strategies scale with the composition of the design space?*

The tunable version of the Goldstein problem was used to assess how the global exploration and nested optimization strategies scale with the problem's number of architectures and its imputation ratio, which compares the size of the apparent discrete search space and the size of the actual combinatorial search space after accounting for inactive variables. It was found that the performance of these strategies, as measured by three different indicators, only slightly decreases with an increasing imputation ratio but noticeably degrades with an increased number of architectures. It was found that, for this tunable problem, the global exploration strategy consistently outperforms the nested approach across all tested imputation ratios and numbers of architectures. Additionally, for both strategies, an increasing number of architectures leads to an increased reliance on exploitation over exploration as measured by the architecture Gini index and the proportion of architectures evaluated.

- *What are the advantages and disadvantages of these strategies?*

Thanks to the radically different approaches that each strategy takes to handle the hierarchical search spaces of architecting problems, each of the three strategies studied has its distinct advantages and disadvantages:

- The **global exploration** strategy was found to perform robustly under different problem conditions and to be straightforward to implement. However, its main disadvantage is that its use is limited to problems where the full design vector can be formulated before the optimization begins. Therefore, problems with unknown numbers of variables due to complex architectural spaces cannot be solved with this strategy.
- The **nested optimization** strategy is highly adaptable and excels at handling problems with intricate hierarchical structures, even if the optimization starts without knowledge of all the variables that might appear in the problem. It is also advantageous because it allows different optimizers to be used for different levels such that their respective benefits can be leveraged simultaneously. However, the nested approach makes optimization runs greedier, focusing more on exploitation than on exploration. While this can help the optimization process find the best solutions quickly, it can also lead to several evaluations in suboptimal areas of the design space. This behavior causes the nested strategy's performance to be more irregular than that of its global exploration counterpart.
- The **decision chain** strategy offers a unique perspective, transforming the design problem into a sequence of decisions. This sequential decision-making process can be useful for problems where the order of decisions matters or when thinking in terms of actions and consequences is beneficial. Furthermore, this strategy is the only one that allows the use of reinforcement learning to solve architecting problems. As established previously, reinforcement learning can be helpful in system architecting problems because, after the training process, it provides a decision-making model that may generalize to similar, previously unseen problems. Nevertheless, the strategy's

primary disadvantage lies in the complexity of its setup. It requires the definition of states, actions, and rewards, which might not be intuitive for traditional design problems and may require an extensive and iterative process of experimentation to yield good results.

The answers to these questions lead to an answer to this work's main research question:

> **What are the most suitable optimization strategies for addressing the mixed-discrete, multi-objective, failure region, and hierarchical aspects of system architecture optimization problems?**

Even though the global exploration, nested optimization, and decision chain strategies are all suitable for addressing the various intricacies of system architecture optimization problems, each one is most suitable for a particular set of conditions. As anticipated in the previous chapter:

- The **global exploration** strategy generally has the best performance and the most straightforward implementation, so it should be the default option to tackle architecting problems.

- The **nested optimization** strategy allows for more flexibility than the global exploration strategy at the expense of a more uncertain and slightly worse performance. However, in contrast to the global exploration strategy, it can operate when a complete design space specification is unavailable. Thus, the nested strategy is the best option when such a specification is unavailable or difficult to obtain.

- According to this work's results, the **decision chain** strategy does not provide performance or ease of implementation benefits over the other two strategies. However, it is the only strategy that opens the possibility of using reinforcement learning in system architecture optimization problems. Therefore, this strategy should only be used when the architecting problem is going to be addressed with reinforcement learning.

9.2. RECOMMENDATIONS

This work presented and implemented three possible optimization strategies to address system architecture problems. These strategies have the potential to help bridge the systems architecting and concept development phases of the systems design process so that more informed decisions can be taken early on in the course of product development. Nonetheless, there are still several areas that require further exploration and refinement:

- *Study of significantly larger problems* System architecting problems can include thousands of interconnected variables. As system architecture problems grow in complexity, so does the challenge of managing such complexity. Future research should focus on developing algorithms and techniques that can handle larger design spaces without significantly increasing computational time. Furthermore, truly multidisciplinary problems should be studied within the framework laid out by the different strategies showcased in this work.

- *Inclusion of more advanced optimization algorithms* As discussed earlier, genetic algorithms and basic Bayesian optimization have certain limitations. For example, Bayesian optimization scales poorly with an increasing number of variables, and genetic algorithms require a large number of evaluations to function properly. For this reason, other, more advanced optimization algorithms should be studied as alternative engines for the strategies proposed in this work. The previously mentioned Random EMbedding Bayesian Optimization (REMBO) is particularly interesting, designed to extend Bayesian optimization's functionality to high-dimensional problems and other surrogate-based techniques.

- *Deep dive into deep reinforcement learning*

  Although this work showed the promise of reinforcement learning for optimization in the airfoil and mixed-discrete ZDT1 test cases, there is still much to explore, especially in representing design spaces and design instances as graphs and using such graphs to drive the training process. One significant advantage of using a reinforcement learning framework is the possibility of transfer learning, where a pre-trained model on one problem can be fine-tuned to solve a related but different problem. This could dramatically reduce training times and data requirements for new architecting scenarios if they share some common features with previously encountered problems.

- *Integration with other tools*

  The system architecture optimization strategies presented in this work have the potential to be integrated with other tools, such as knowledge-based engineering applications, MDAO workflow formulation tools, and collaborative design frameworks. A study on how these strategies can be integrated with such tools can lead to further front-loading of the systems design process.

- *Analysis on changing analysis tools*

  This work assumed that the analysis tools needed to evaluate every architecture were the same. However, in more complex scenarios, the disciplines and tools are subject to change depending on the choice of architecture. As a result, more work is needed to understand how adding that degree of freedom affects the applicability of the proposed strategies.

- *Adoption of adaptive or hybrid strategies*

  The strategies presented in this work were tested statically, i.e., they did not change during the optimization run. However, it may be possible to use adaptive or hybrid schemes that change a strategy's characteristics as the optimization advances. For example, a global exploration strategy could start using an approximation of the full design vector. When it finds a previously unseen variable, it could retroactively add it to the already evaluated points as an imputed quantity. Additionally, the strong exploration capabilities of the global approach could be combined with the exploitative nature of the nested strategy to form a hybrid strategy.

This thesis delved into three distinct strategies capable of addressing system architecture optimization problems, thus making a step towards bridging the upstream architecting and downstream concept development stages of the design of complex systems. By carefully studying each strategy, insights into its strengths, limitations, and potential applications have been identified. Doing so has opened paths for informed decision-making during the earliest stages of system design. These strategies, enriched by various computational tools working together, hold promise for a novel design approach where intuition is augmented by powerful optimization techniques.

A

ALGORITHMS

This appendix includes pseudocode descriptions of some of the algorithms described throughout this work. Algorithm A.1 shows a genetic algorithm, Algorithm A.2 shows the NSGA-II algorithm, Algorithm A.3 depicts the steps taken by a Bayesian optimization routine, and Algorithm A.4 details the training procedure for the DDPG reinforcement learning algorithm. Algorithm A.5 explains the steps followed to obtain the material thicknesses in the aileron optimization problem.

Algorithm A.1 Genetic algorithm. Adapted from [45, 122].

```
Require: Maximum number of iterations $k_{\max}$, population size $N_p$
  Initialize population $P = \{\boldsymbol{s}^{(0)}, \ldots, \boldsymbol{s}^{(N_p)}\}$
  for generation $= 0, \ldots, k_{\max}$ do
      Calculate $\boldsymbol{f}(\boldsymbol{s}^{(i)}) \forall \boldsymbol{s}^{(i)} \in P$
      Choose $N_p/2$ parent pairs from $P$ for crossover                ▷ Selection
      Generate a new population $P_{\text{new}}$ of $N_p$ children       ▷ Crossover
      Randomly apply mutations to the chromosomes of $P_{\text{new}}$    ▷ Mutation
      $P \leftarrow P_{\text{new}}$
  end for
```

Algorithm A.2 NSGA-II algorithm. Adapted from [66].

```
Require: Maximum number of generations $k_{\max}$, population size $N_p$
  Initialize population $P = \{\boldsymbol{s}^{(0)}, \ldots, \boldsymbol{s}^{(N_p)}\}$
  Evaluate fitness of individuals in $P$
  Rank individuals based on non-domination in $P$
  for generation $= 1, \ldots, k_{\max}$ do
      Select parents using binary tournament selection based on rank and crowding distance
      Apply crossover on selected parents to create offspring
      Apply mutation on offspring
      Create a combined population $R$ of parents and offspring
      Rank individuals in $R$ based on non-domination
      P $\leftarrow$ top $N_p$ individuals from $R$ based on rank and crowding distance
  end for
```

Algorithm A.3 Bayesian optimization algorithm. Adapted from [68].

```
Require: Maximum number of iterations $k_{\max}$
  Initialize dataset $\mathscr{D}$
  for iteration $k = 0, \dots, k_{\max}$ do
      Find $\boldsymbol{s}_t$ by optimizing the acquisition function over the Gaussian process:
          $$\boldsymbol{s}_t \leftarrow \operatorname{argmax}_{\boldsymbol{s}} u(\boldsymbol{s}|\mathscr{D}_{0:k-1})$$
      Augment the dataset: $\mathscr{D}_{0:t} \leftarrow \{\mathscr{D}_{0:t-1}, (\boldsymbol{s}_t, \boldsymbol{f}(\boldsymbol{s}_t))\}$
      Update the Gaussian process model
  end for
```

Algorithm A.4 DDPG algorithm (reproduced from [78]).

```
  Randomly initialize critic network $Q(\mathbf{s}, \mathbf{a}|\theta^Q)$ and actor $\mu(\mathbf{s}|\theta^\mu)$
  Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
  Initialize replay buffer $\mathscr{R}$
  for episode = 1, $M$ do
      Initialize a random process $\mathscr{N}$ for action exploration
      Receive initial observation state $s_1$
      for $t = 1, T$ do
          Select action $a_t = \mu(s_t|\theta^\mu) + \mathscr{N}_t$ according to the current policy and exploration noise
          Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
          Store transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ in $\mathscr{R}$
          Sample a random minibatch of $N$ transitions $(\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}_{i+1})$ from $\mathscr{R}$
          Set $y_i = r_i + \gamma Q'\left(\mathbf{s}_{i+1}, \mu'\left(\mathbf{s}_{i+1}|\theta^{\mu'}\right)|\theta^{Q'}\right)$
          Update critic by minimizing the loss: $L = \frac{1}{N}\sum_{i=0}^{N}\left(y_i - Q\left(\mathbf{s}_i, \mathbf{a}_i|\theta^Q\right)\right)^2$
          Update the actor policy using the sampled policy gradient:
              $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i=0}^{N} \nabla_a Q(\boldsymbol{s}, \boldsymbol{a}|\theta^Q)|_{\boldsymbol{s}=\boldsymbol{s}_i, \boldsymbol{a}=\mu(\boldsymbol{s}_i)} \nabla_{\theta^\mu}\mu(\boldsymbol{s}|\theta^\mu)|_{\boldsymbol{s}=\boldsymbol{s}_i}$$
          Update the target networks:
              $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
              $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
      end for
  end for
```

Algorithm A.5 Heuristic search algorithm for minimum reserve factor constraint satisfaction.

```
Require: Maximum number of iterations $k_{\max}$
  $k \leftarrow 0$
  Calculate the structure's load paths
  Initialize all material thicknesses at the thinnest available option
  Calculate the reserve factor of each material zone
  $\text{RF}_{\min} \leftarrow$ minimum reserve factor found
  while $k < k_{\max}$ or $\text{RF}_{\min} < 1$ do
      for each material zone do

          if the material zone's minimum reserve factor $< 1$ then
              Increase the material zone's thickness to the next thickest value
          end if
      end for
      Calculate the reserve factor of each material zone
      $\text{RF}_{\min} \leftarrow$ minimum reserve factor found
      $k \leftarrow k + 1$
  end while
```

B

Example of the Nested Optimization Strategy

To better illustrate this work's implementation of the nested strategy, a simple example is presented. The optimization problem for this example has the variables listed in Table B.1 and can be formally stated as:

$$\begin{aligned} &\text{Minimize} && f(a,b,c,d,e) \\ &\text{With respect to} && a,b,c,d,e. \end{aligned} \tag{B.1}$$

Table B.1: Variables of the example problem.

Variable	Domain	Type	Role
a	0, 1	Integer	Meta (always active)
b	0, 1	Integer	Meta (always active)
c	[0, 1]	Continuous	Neutral (always active)
d	[0, 1]	Continuous	Decreed (active when $a = 0$)
e	[0, 1]	Continuous	Decreed (active when $b = 1$)

For simplicity, the example problem is taken as single-objective, but the same logic applies for multi-objective problems.

In this example, the nested strategy is set up in two levels such that the problem's meta variables, a and b, are assigned to the outer loop, while the other variables are handled in the inner loop. To keep the problem small, a population size of 3 is used for the outer loop, and a population size of 2 is used for the inner loop. The schematic step-by step process followed by the nested is illustrated in Figure B.1. The steps are as follows:

1. **Initial outer-loop population:** First, the outer-loop population is initialized. In this example, the initial outer population is $\{[a=0,b=0],[a=0,b=1],[a=1,b=1]\}$.

2. **Inner loop optimizations advance:** Next, for every member of the outer population, a subproblem is formulated and an inner-level optimization advances according to the chosen inner optimizer parameters. The inner subproblems only use the variables that are active given their corresponding outer variables, so, for example, the $[a=0,b=0]$ inner-loop optimization uses variables c and d while the $[a=0,b=1]$ uses c, d, and e.

3. **Ranking and survival:** Once all the inner-level optimizations have been advanced, a ranking and survival process is carried out. Here, all the obtained designs are considered and compared to each other. The fittest members of this combined population advance to the next generation, while the others are discarded.

4. **New outer-loop population:** A new outer-level population emerges out of the ranking and survival process. It is important to note that this work's nested strategy allows for duplicate outer-level design vectors. For instance, in this example, the resulting outer-loop population is $\{[a = 0, b = 1], [a = 0, b = 0], [a = 0, b = 1]\}$. In practice, this means that more of the evaluation budget per generation will be assigned to the repeated architectures.

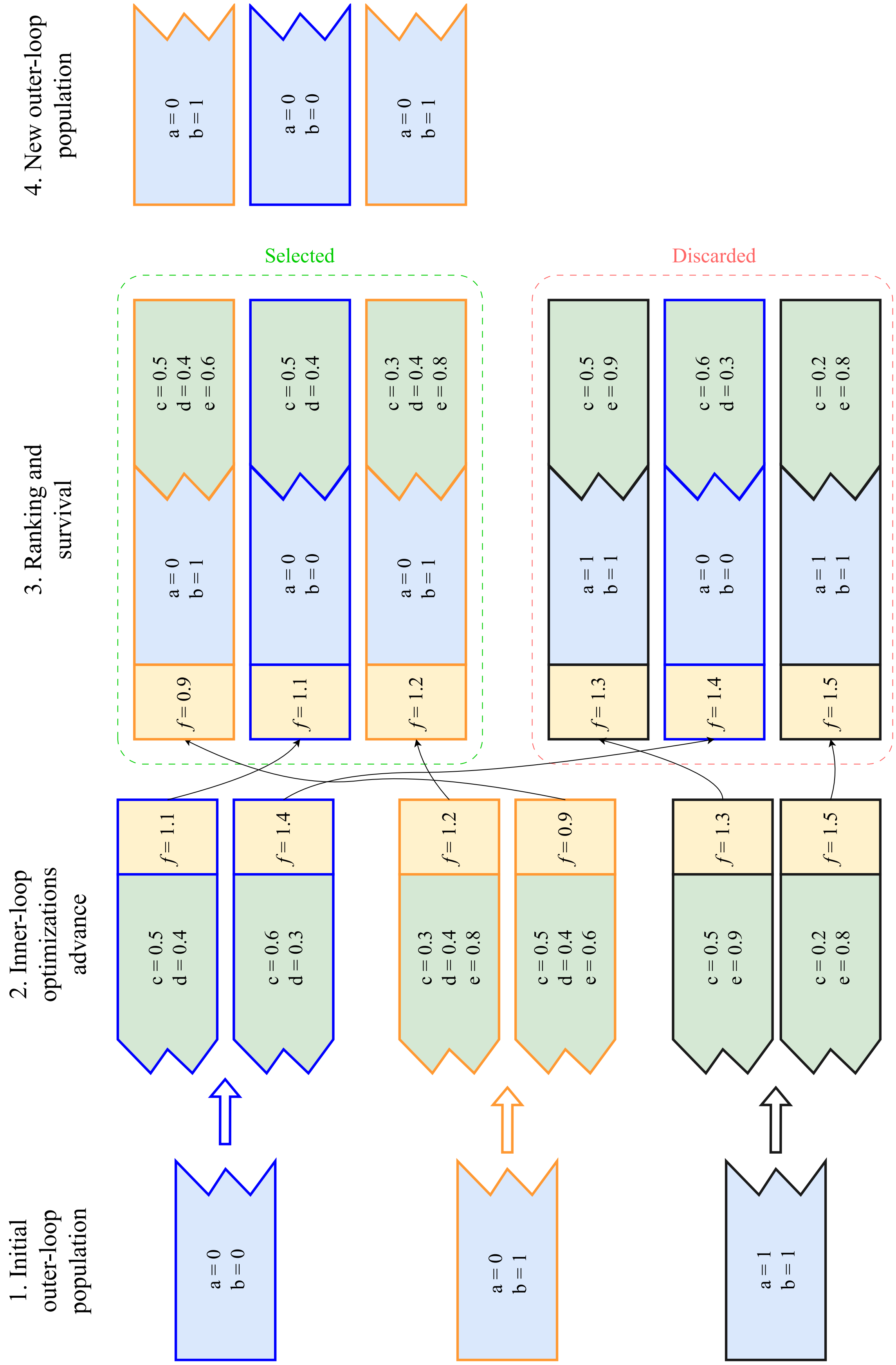


Figure B.1: Example of the nested optimization strategy's procedure.

C

XDSMs of the Optimization Strategies Applied to the Aileron Problem

On page 78, Figure C.1 shows the XDSM for the implementation of the global exploration strategy for the aileron problem. Page 79 contains Figure C.2, which illustrates the XDSM for the implementation of the nested optimization strategy for the aileron problem.

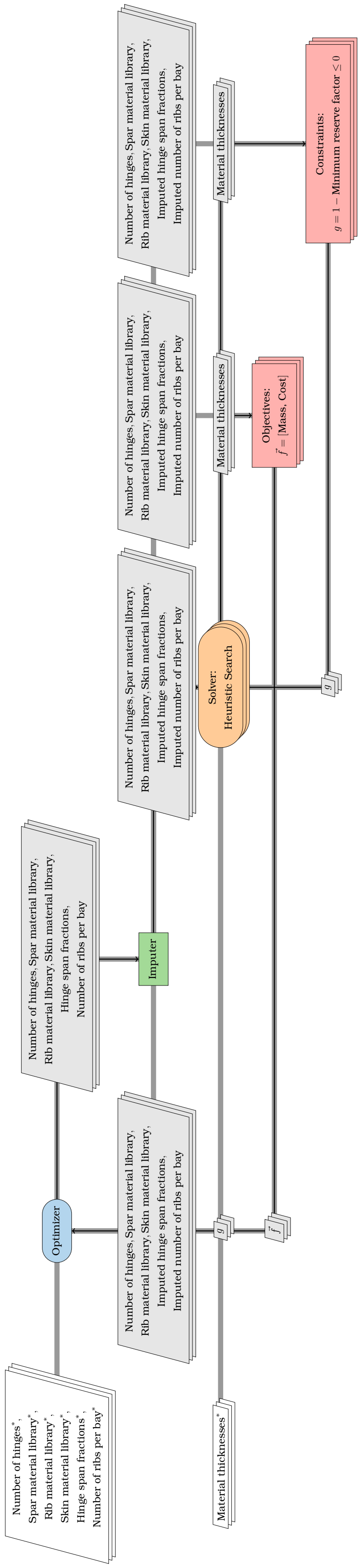

Number of hinges*,
Spar material library*,
Rib material library*,
Skin material library*,
Hinge span fractions*,
Number of ribs per bay*
Material thicknesses*
Optimizer
Number of hinges, Spar material library,
Rib material library, Skin material library,
Hinge span fractions,
Number of ribs per bay
Number of hinges, Spar material library,
Rib material library, Skin material library,
Imputed hinge span fractions,
Imputed number of ribs per bay
g
$\vec{f}$
Imputer
Solver:
Heuristic Search
Material thicknesses
Objectives:
$\vec{f} = [\text{Mass, Cost}]$
Constraints:
$g = 1 -$ Minimum reserve factor ≤ 0

Figure C.1: Extended design structure matrix of the global exploration strategy implemented for the aileron optimization problem.

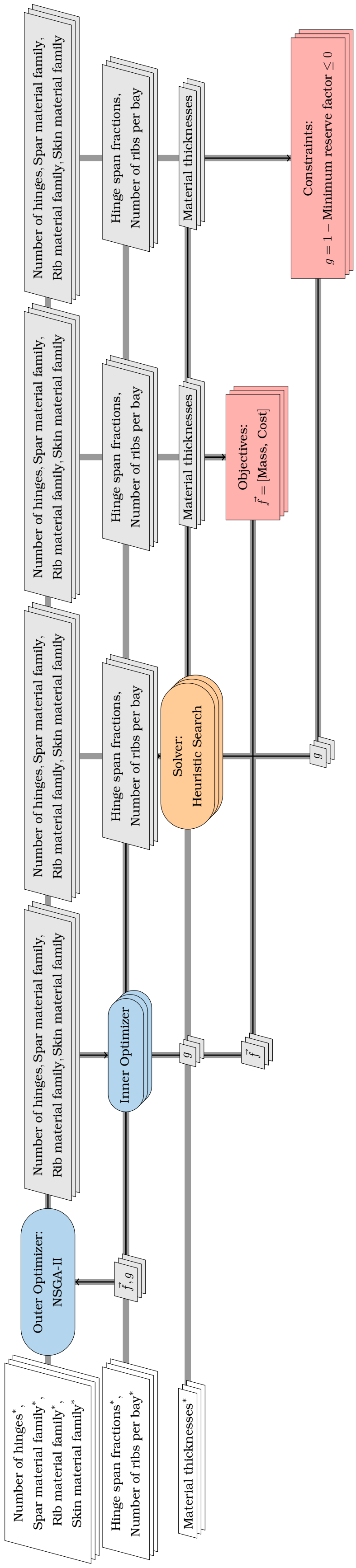

Number of hinges*, Spar material family*, Rib material family*, Skin material family*
Hinge span fractions*, Number of ribs per bay*
Material thicknesses*
Outer Optimizer: NSGA-II
$\vec{f}, g$
Number of hinges, Spar material family, Rib material family, Skin material family
Inner Optimizer
g
$\vec{f}$
Number of hinges, Spar material family, Rib material family, Skin material family
Hinge span fractions, Number of ribs per bay
Solver: Heuristic Search
g
Number of hinges, Spar material family, Rib material family, Skin material family
Hinge span fractions, Number of ribs per bay
Material thicknesses
Objectives: $\vec{f} = [\text{Mass, Cost}]$
Number of hinges, Spar material family, Rib material family, Skin material family
Hinge span fractions, Number of ribs per bay
Material thicknesses
Constraints: $g = 1 - \text{Minimum reserve factor} \leq 0$

Figure C.2: Extended design structure matrix of the nested strategy implemented for the aileron optimization problem.

References

[1] Chepko, A., *Technology Selection and Architecture Optimization of In-Situ Resource Utilization Systems*, Master's thesis, Massachusetts Institute of Technology, 2009.

[2] Iacobucci, J. V., *Rapid Architecture Alternative Modeling (RAAM): A Framework for Capability-based Analysis of System of Systems Architectures*, Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, 2012.

[3] Crawley, E., Cameron, B., and Selva, D., *System Architecture, Global Edition*, Pearson Education, London, England, Dec. 2015.

[4] Nadir, W. D., *Multidisciplinary structural design and optimization for performance, cost, and flexibility*, Master's thesis, Massachusetts Institute of Technology, 2005.

[5] Cook, M., *Flight Dynamics Principles*, Butterworth-Heinemann, Waltham, MA, USA, 3rd ed., 2013. doi:10.1016/c2010-0-65889-5.

[6] Kossiakoff, A., Seymour, S. J., Flanigan, D. A., and Biemer, S. M., *Systems Engineering: Principles and Practice*, John Wiley & Sons, Hoboken, NJ, 2020.

[7] Borky, J. M. and Bradley, T. H., *Effective Model-Based Systems Engineering*, Springer International Publishing, Cham, Switzerland, 2019. doi:10.1007/978-3-319-95669-5.

[8] Weilkiens, T., Lamm, J., Roth, S., and Walker, M., *Model-based System Architecture*, Wiley Series in Systems Engineering and Management, John Wiley & Sons, Hoboken, NJ, 2nd ed., April 2022.

[9] Haberfellner, R., de Weck, O., Fricke, E., and Vössner, S., *Systems Engineering*, Springer International Publishing, 2019. doi:10.1007/978-3-030-13431-0.

[10] Bussemaker, J. H., de Smedt, T., La Rocca, G., Ciampa, P. D., and Nagel, B., "System Architecture Optimization: An Open Source Multidisciplinary Aircraft Jet Engine Architecting Problem," *AIAA AVIATION 2021 FORUM*, American Institute of Aeronautics and Astronautics, July 2021, pp. 1–20. doi:10.2514/6.2021-3078.

[11] La Rocca, G., *Knowledge Based Engineering Techniques to Support Aircraft Design and Optimization*, Ph.D. thesis, Delft University of Technology, Delft, the Netherlands, 2011.

[12] Bussemaker, J. H. and Ciampa, P. D., "MBSE in Architecture Design Space Exploration," *Handbook of Model-Based Systems Engineering*, Springer International Publishing, 2022, pp. 1–41. doi:10.1007/978-3-030-27486-3_36-1.

[13] Chaudemar, J.-C. and de Saqui-Sannes, P., "MBSE and MDAO for Early Validation of Design Decisions: a Bibliography Survey," *2021 IEEE International Systems Conference (SysCon)*, IEEE, April 2021, pp. 1–8. doi:10.1109/syscon48628.2021.9447140.

[14] Thomke, S. and Fujimoto, T., "The Effect of "Front-Loading" Problem-Solving on Product Development Performance," *Journal of Product Innovation Management*, Vol. 17, No. 2, March 2000, pp. 128–142. doi:10.1111/1540-5885.1720128.

[15] Brownlee, A. E. and Wright, J. A., "Constrained, mixed-integer and multi-objective optimisation of building designs by NSGA-II with fitness approximation," *Applied Soft Computing*, Vol. 33, Aug. 2015, pp. 114–126. doi:10.1016/j.asoc.2015.04.010.

[16] Gentile, L., Greco, C., Minisci, E., Bartz-Beielstein, T., and Vasile, M., "Structured-chromosome GA optimisation for satellite tracking," *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ACM, July 2019, pp. 1955–1963. doi:10.1145/3319619.3326841.

[17] Ellithy, A., Abdelkhalik, O., and Englander, J., "Multi-Objective Hidden Genes Genetic Algorithm for Multigravity-Assist Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 45, No. 7, July 2022, pp. 1269–1285. doi:10.2514/1.g006415.

[18] Jones, D. R., Schonlau, M., and Welch, W. J., "Efficient Global Optimization of Expensive Black-Box Functions," *Journal of Global Optimization*, Vol. 13, No. 4, 1998, pp. 455–492. doi:10.1023/a:1008306431147.

[19] Saves, P., Bartoli, N., Diouane, Y., Lefebvre, T., Morlier, J., David, C., Nguyen Van, E., and Defoort, S., "Bayesian optimization for mixed variables using an adaptive dimension reduction process: applications to aircraft design," *AIAA SCITECH 2022 Forum*, American Institute of Aeronautics and Astronautics, Jan. 2022, pp. 1–22. doi:10.2514/6.2022-0082.

[20] Yang, K., Emmerich, M., Deutz, A., and Bäck, T., "Multi-Objective Bayesian Global Optimization using expected hypervolume improvement gradient," *Swarm and Evolutionary Computation*, Vol. 44, Feb. 2019, pp. 945–956. doi:10.1016/j.swevo.2018.10.007.

[21] Pelamatti, J., Brevault, L., Balesdent, M., Talbi, E.-G., and Guerin, Y., "Bayesian optimization of variable-size design space problems," *Optimization and Engineering*, Vol. 22, No. 1, July 2020, pp. 387–447. doi:10.1007/s11081-020-09520-z.

[22] Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E., "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, Vol. 134, Oct. 2021, pp. 105400. doi:10.1016/j.cor.2021.105400.

[23] Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Le, Q. V., Laudon, J., Ho, R., Carpenter, R., and Dean, J., "A graph placement methodology for fast chip design," *Nature*, Vol. 594, No. 7862, June 2021, pp. 207–212. doi:10.1038/s41586-021-03544-w.

[24] Li, K., Zhang, T., and Wang, R., "Deep Reinforcement Learning for Multiobjective Optimization," *IEEE Transactions on Cybernetics*, Vol. 51, No. 6, June 2021, pp. 3103–3114. doi:10.1109/tcyb.2020.2977661.

[25] Pelamatti, J., *Mixed-variable Bayesian optimization - Application to aerospace system design*, Ph.D. thesis, Université de Lille, 2020.

[26] Frank, C., Marlier, R., Pinon-Fischer, O. J., and Mavris, D. N., "An Evolutionary Multi-Architecture Multi-Objective Optimization Algorithm for Design Space Exploration," *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, American Institute of Aeronautics and Astronautics, Jan. 2016, pp. 1–19. doi:10.2514/6.2016-0414.

[27] Stops, L., Leenhouts, R., Gao, Q., and Schweidtmann, A. M., "Flowsheet synthesis through hierarchical reinforcement learning and graph neural networks," 2022. doi:10.48550/arxiv.2207.12051.

[28] Zitzler, E., Deb, K., and Thiele, L., "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evolutionary Computation*, Vol. 8, No. 2, June 2000, pp. 173–195. doi:10.1162/106365600568202.

[29] Weilkiens, T., *Systems Engineering with SysML/UML*, The MK/OMG Press, Morgan Kaufmann, Oxford, England, 2 2008.

[30] Ryschkewitsch, M., Schaible, D., and Larson, W., "The Art and Science of Systems Engineering," *Systems Research Forum*, Vol. 03, No. 02, Dec. 2009, pp. 81–100. doi:10.1142/s1793966609000080.

[31] Forsberg, K. and Mooz, H., "The Relationship of Systems Engineering to the Project Cycle," *Engineering Management Journal*, Vol. 4, No. 3, Sept. 1992, pp. 36–43. doi:10.1080/10429247.1992.11414684.

[32] Walden, D., Roedler, G., Forsberg, K., Hamelin, R., and Shortell, T., editors, *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and activities*, John Wiley & Sons, San Diego, CA, 4th ed., 2015.

[33] Ciampa, P. D., La Rocca, G., and Nagel, B., "A MBSE Approach to MDAO Systems for the Development of Complex Products," *AIAA Aviation 2020 Forum*, American Institute of Aeronautics and Astronautics, June 2020, pp. 1–31. doi:10.2514/6.2020-3150.

[34] Firesmith, D., "The Method Framework for Engineering System Architectures," Software Engineering Institute, Carnegie Mellon University (Keynote Slides), 3 2009.

[35] Maier, M. W. and Rechtin, E., *The Art of Systems Architecting*, Systems Engineering, CRC Press, Boca Raton, FL, 3rd ed., Jan. 2009.

[36] Jankovic, M. and Hein, A. M., "Architecting Engineering Systems: Designing Critical Interfaces," *Handbook of Engineering Systems Design*, Springer International Publishing, 2022, pp. 381–405. doi:10.1007/978-3-030-81159-4_14.

[37] McManus, H. L., Hastings, D. E., and Warmkessel, J. M., "New Methods for Rapid Architecture Selection and Conceptual Design," *Journal of Spacecraft and Rockets*, Vol. 41, No. 1, Jan. 2004, pp. 10–19. doi:10.2514/1.9203.

[38] Villeneuve, F. and Mavris, D., "A New Method of Architecture Selection for Launch Vehicles," *AIAA/-CIRA 13th International Space Planes and Hypersonics Systems and Technologies Conference*, American Institute of Aeronautics and Astronautics, May 2005, pp. 1–17. doi:10.2514/6.2005-3361.

[39] Moullec, M.-L., Jankovic, M., and Eckert, C. M., "The impact of criteria in system architecture selection: observation from industrial experiment," *20th International Conference on Engineering Design (ICED 2015)*, Milan, Italy, July 2016, pp. 1–12.

[40] Abdelkhalik, O., *Algorithms for variable-size optimization*, CRC Press, Boca Raton, FL, April 2021.

[41] Bussemaker, J. H., Bartoli, N., Lefebvre, T., Ciampa, P. D., and Nagel, B., "Effectiveness of Surrogate-Based Optimization Algorithms for System Architecture Optimization," *AIAA AVIATION 2021 FORUM*, American Institute of Aeronautics and Astronautics, July 2021, pp. 1–29. doi:10.2514/6.2021-3095.

[42] Audet, C., Bigeon, J., Cartier, D., Le Digabel, S., and Salomon, L., "Performance indicators in multi-objective optimization," *European Journal of Operational Research*, Vol. 292, No. 2, 2021, pp. 397–422. doi:https://doi.org/10.1016/j.ejor.2020.11.016.

[43] El-Halwagi, M. M., "Overview of optimization," *Process Integration*, edited by M. M. El-Halwagi, Vol. 7 of *Process Systems Engineering*, Academic Press, 2006, pp. 285–314. doi:https://doi.org/10.1016/S1874-5970(06)80012-3.

[44] Hendrix, E. M. T. and Toth, B., *Introduction to nonlinear and global optimization*, Springer Optimization and Its Applications, Springer, New York, NY, 1st ed., April 2010.

[45] Martins, J. R. R. A. and Ning, A., *Engineering Design Optimization*, Cambridge University Press, 1 2022.

[46] Brevault, L., Pelamatti, J., Hebbal, A., Balesdent, M., Talbi, E.-G., and Melab, N., "MDO Related Issues: Multi-Objective and Mixed Continuous/Discrete Optimization," *Springer Optimization and Its Applications*, Springer International Publishing, 2020, pp. 321–358. doi:10.1007/978-3-030-39126-3_9.

[47] McCane, B. and Albert, M., "Distance functions for categorical and mixed variables," *Pattern Recognition Letters*, Vol. 29, No. 7, May 2008, pp. 986–993. doi:10.1016/j.patrec.2008.01.021.

[48] Filomeno Coelho, R., "Metamodels for mixed variables based on moving least squares," *Optimization and Engineering*, Vol. 15, No. 2, June 2013, pp. 311–329. doi:10.1007/s11081-013-9216-8.

[49] Okada, S., Ohzeki, M., and Taguchi, S., "Efficient partition of integer optimization problems with one-hot encoding," *Scientific Reports*, Vol. 9, No. 1, Sept. 2019. doi:10.1038/s41598-019-49539-6.

[50] Aggarwal, C. C., Hinneburg, A., and Keim, D. A., "On the Surprising Behavior of Distance Metrics in High Dimensional Space," *Database Theory — ICDT 2001*, Springer Berlin Heidelberg, 2001, pp. 420–434. doi:10.1007/3-540-44503-x_27.

[51] Abdelkhalik, O., "Hidden Genes Genetic Optimization for Variable-Size Design Space Problems," *Journal of Optimization Theory and Applications*, Vol. 156, No. 2, Aug. 2012, pp. 450–468. doi:10.1007/s10957-012-0122-6.

[52] Audet, C., Hallé-Hannan, E., and Digabel, S. L., "A General Mathematical Framework for Constrained Mixed-variable Blackbox Optimization Problems with Meta and Categorical Variables," *Operations Research Forum*, Vol. 4, No. 1, Feb. 2023. doi:10.1007/s43069-022-00180-6.

[53] Arora, J. S., "Multi-objective Optimum Design Concepts and Methods," *Introduction to Optimum Design*, Elsevier, 2017, pp. 771–794. doi:10.1016/b978-0-12-800806-5.00018-4.

[54] Arias-Montano, A., Coello, C. A. C., and Mezura-Montes, E., "Multiobjective Evolutionary Algorithms in Aeronautical and Aerospace Engineering," *IEEE Transactions on Evolutionary Computation*, Vol. 16, No. 5, Oct. 2012, pp. 662–694. doi:10.1109/tevc.2011.2169968.

[55] Müller, J. and Day, M., "Surrogate Optimization of Computationally Expensive Black-Box Problems with Hidden Constraints," *INFORMS Journal on Computing*, Vol. 31, No. 4, Oct. 2019, pp. 689–702. doi:10.1287/ijoc.2018.0864.

[56] Wakabayashi, Y. K., Otsuka, T., Krockenberger, Y., Sawada, H., Taniyasu, Y., and Yamamoto, H., "Bayesian optimization with experimental failure for high-throughput materials growth," *npj Computational Materials*, Vol. 8, No. 1, Aug. 2022. doi:10.1038/s41524-022-00859-8.

[57] Chakrabarty, A., Bortoff, S. A., and Laughman, C. R., "Simulation Failure Robust Bayesian Optimization for Estimating Black-Box Model Parameters," *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2021, pp. 1533–1538. doi:10.1109/SMC52423.2021.9658893.

[58] Yang, X.-S., "Optimization Algorithms," *Computational Optimization, Methods and Algorithms*, Springer Berlin Heidelberg, 2011, pp. 13–31. doi:10.1007/978-3-642-20859-1_2.

[59] Kramer, O., Ciaurri, D. E., and Koziel, S., "Derivative-Free Optimization," *Computational Optimization, Methods and Algorithms*, Springer Berlin Heidelberg, 2011, pp. 61–83. doi:10.1007/978-3-642-20859-1_4.

[60] Rios, L. M. and Sahinidis, N. V., "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, Vol. 56, No. 3, July 2012, pp. 1247–1293. doi:10.1007/s10898-012-9951-y.

[61] Simon, D., *Evolutionary Optimization Algorithms*, Wiley-Blackwell, Hoboken, NJ, 4 2013.

[62] Blank, J. and Deb, K., "Pymoo: Multi-Objective Optimization in Python," *IEEE Access*, Vol. 8, 2020, pp. 89497–89509. doi:10.1109/access.2020.2990567.

[63] Storn, R. and Price, K., "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization*, Vol. 11, No. 4, 1997, pp. 341–359. doi:10.1023/a:1008202821328.

[64] Ebrahimi, A., Tamnanloo, J., Mousavi, S. H., Miandoab, E. S., Hosseini, E., Ghasemi, H., and Mozaffari, S., "Discrete-Continuous Genetic Algorithm for Designing a Mixed Refrigerant Cryogenic Process," *Industrial & Engineering Chemistry Research*, Vol. 60, No. 20, May 2021, pp. 7700–7713. doi:10.1021/acs.iecr.1c01191.

[65] Deb, K. and Agrawal, R. B., "Simulated Binary Crossover for Continuous Search Space," *Complex Systems*, Vol. 9, No. 2, 1995, pp. 115–148.

[66] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T., "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, April 2002, pp. 182–197. doi:10.1109/4235.996017.

[67] Garnett, R., *Bayesian Optimization*, Cambridge University Press, 2022, in preparation.

[68] Brochu, E., Cora, V. M., and de Freitas, N., "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," 2010. doi:10.48550/arxiv.1012.2599.

[69] Kleijnen, J. P., "Kriging metamodeling in simulation: A review," *European Journal of Operational Research*, Vol. 192, No. 3, Feb. 2009, pp. 707–716. doi:10.1016/j.ejor.2007.10.013.

[70] Cuesta Ramirez, J., Le Riche, R., Roustant, O., Perrin, G., Durantin, C., and Glière, A., "A comparison of mixed-variables Bayesian optimization approaches," *Advanced Modeling and Simulation in Engineering Sciences*, Vol. 9, No. 1, 6 2022, pp. 1–29. doi:10.1186/s40323-022-00218-8.

[71] Deshwal, A., Belakaria, S., and Doppa, J. R., "Bayesian Optimization over Hybrid Spaces," *Proceedings of the 38th International Conference on Machine Learning*, edited by M. Meila and T. Zhang, Vol. 139 of *Proceedings of Machine Learning Research*, PMLR, 18–24 Jul 2021, pp. 2632–2643.

[72] Zaefferer, M. and Horn, D., "A First Analysis of Kernels for Kriging-Based Optimization in Hierarchical Search Spaces," *Parallel Problem Solving from Nature – PPSN XV*, Springer International Publishing, 2018, pp. 399–410. doi:10.1007/978-3-319-99259-4_32.

[73] Saves, P., Diouane, Y., Bartoli, N., Lefebvre, T., and Morlier, J., "A mixed-categorical correlation kernel for Gaussian process," *Neurocomputing*, Vol. 550, 2023, pp. 126472. doi:https://doi.org/10.1016/j.neucom.2023.126472.

[74] Emmerich, M. T. M., Deutz, A. H., and Klinkenberg, J. W., "Hypervolume-based expected improvement: Monotonicity properties and exact computation," *2011 IEEE Congress of Evolutionary Computation (CEC)*, 2011, pp. 2147–2154. doi:10.1109/CEC.2011.5949880.

[75] Abdolshah, M., Shilton, A., Rana, S., Gupta, S., and Venkatesh, S., "Expected Hypervolume Improvement with Constraints," *2018 24th International Conference on Pattern Recognition (ICPR)*, 2018, pp. 3238–3243. doi:10.1109/ICPR.2018.8545387.

[76] Sutton, R. and Barto, A., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Massachusetts, 2018.

[77] van Otterlo, M. and Wiering, M., "Reinforcement Learning and Markov Decision Processes," *Adaptation, Learning, and Optimization*, Springer Berlin Heidelberg, 2012, pp. 3–42. doi:10.1007/978-3-642-27645-3_1.

[78] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," 2015. doi:10.48550/arxiv.1509.02971.

[79] Grondman, I., Busoniu, L., Lopes, G. A. D., and Babuska, R., "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 42, No. 6, 2012, pp. 1291–1307. doi:10.1109/TSMCC.2012.2218595.

[80] Uhlenbeck, G. E. and Ornstein, L. S., "On the Theory of the Brownian Motion," *Physical Review*, Vol. 36, No. 5, Sept. 1930, pp. 823–841. doi:10.1103/physrev.36.823.

[81] Andersson, S. and Whyte, J., *De Novo Chemical Process Design with Graph Neural Networks and Reinforcement Learning: The Graph Actor-Critic Framework*, Master's thesis, Imperial College London, 2023.

[82] van Moffaert, K., Drugan, M. M., and Nowe, A., "Scalarized multi-objective reinforcement learning: Novel design techniques," *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, IEEE, April 2013, pp. 191–199. doi:10.1109/adprl.2013.6615007.

[83] Ding, Z. and Dong, H., "Challenges of Reinforcement Learning," *Deep Reinforcement Learning*, Springer Singapore, 2020, pp. 249–272. doi:10.1007/978-981-15-4095-0_7.

[84] Kingma, D. P. and Ba, J., "Adam: A Method for Stochastic Optimization," 2014. doi:10.48550/ARXIV.1412.6980.

[85] de Smedt, T., *Aircraft Jet Engine Architecture Modeling: Creating a Benchmark Problem Using a System Architecting Approach with Mixed-Discrete & Multi-Objective Capabilities*, Master's thesis, Delft University of Technology, 2021.

[86] Michalek, J. J. and Papalambros, P. Y., "BB-ATC: Analytical Target Cascading Using Branch and Bound for Mixed-Integer Nonlinear Programming," *Volume 1: 32nd Design Automation Conference, Parts A and B*, ASMEDC, Jan. 2006, pp. 1–7. doi:10.1115/detc2006-99040.

[87] Lee, X. Y., Balu, A., Stoecklein, D., Ganapathysubramanian, B., and Sarkar, S., "A Case Study of Deep Reinforcement Learning for Engineering Design: Application to Microfluidic Devices for Flow Sculpting," *Journal of Mechanical Design*, Vol. 141, No. 11, Sept. 2019. doi:10.1115/1.4044397.

[88] Sajedian, I., Lee, H., and Rho, J., "Double-deep Q-learning to increase the efficiency of metasurface holograms," *Scientific Reports*, Vol. 9, No. 1, July 2019. doi:10.1038/s41598-019-47154-z.

[89] Jaafra, Y., Luc Laurent, J., Deruyver, A., and Saber Naceur, M., "Reinforcement learning for neural architecture search: A review," *Image Vis. Comput.*, Vol. 89, Sept. 2019, pp. 57–66.

[90] Zhong, Z., Yan, J., Wu, W., Shao, J., and Liu, C.-L., "Practical Block-Wise Neural Network Architecture Generation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 6 2018, pp. 2423–2432. doi:10.1109/cvpr.2018.00257.

[91] Picheny, V., Berkeley, J., Moss, H. B., Stojic, H., Granta, U., Ober, S. W., Artemev, A., Ghani, K., Goodall, A., Paleyes, A., Vakili, S., Pascual-Diaz, S., Markou, S., Qing, J., Loka, N. R. B. S., and Couckuyt, I., "Trieste: Efficiently Exploring The Depths of Black-box Functions with TensorFlow," 2023. doi:10.48550/arxiv.2302.08436.

[92] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, Software available from tensorflow.org.

[93] Singh, H., "Deep Deterministic Policy Gradient (DDPG)," Keras code examples, 2020.

[94] Coley, D. A., *An Introduction to Genetic Algorithms for Scientists and Engineers*, WORLD SCIENTIFIC, Jan. 1999. doi:10.1142/3904.

[95] Močkus, J., "On Bayesian methods for seeking the extremum," *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, Springer Berlin Heidelberg, 1975, pp. 400–404. doi:10.1007/3-540-07165-2_55.

[96] Gardner, J., Kusner, M., Xu, E., Weinberger, K., and Cunningham, J., "Bayesian Optimization with Inequality Constraints," *Proceedings of the 31st International Conference on Machine Learning, ICML 2014*, Vol. 3, 06 2014, pp. 1–9.

[97] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, Vol. 22, No. 268, 2021, pp. 1–8.

[98] Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., and Stone, P., "Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey," 2020. doi:10.48550/ARXIV.2003.04960.

[99] Nikulin, Y., Miettinen, K., and Mäkelä, M. M., "A new achievement scalarizing function based on parameterization in multiobjective optimization," *OR Spectrum*, Vol. 34, No. 1, Aug. 2010, pp. 69–87. doi:10.1007/s00291-010-0224-1.

[100] Zitzler, E., Brockhoff, D., and Thiele, L., "The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration," *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2007, pp. 862–876. doi:10.1007/978-3-540-70928-2_64.

[101] Ishibuchi, H., Masuda, H., Tanigaki, Y., and Nojima, Y., "Modified Distance Calculation in Generational Distance and Inverted Generational Distance," *Lecture Notes in Computer Science*, Springer International Publishing, 2015, pp. 110–125. doi:10.1007/978-3-319-15892-1_8.

[102] Zitzler, E., "Evolutionary Multiobjective Optimization," *Handbook of Natural Computing*, Springer Berlin Heidelberg, 2012, pp. 871–904. doi:10.1007/978-3-540-92910-9_28.

[103] Fonseca, C., Paquete, L., and Lopez-Ibanez, M., "An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator," *2006 IEEE International Conference on Evolutionary Computation*, 2006, pp. 1157–1163. doi:10.1109/CEC.2006.1688440.

[104] Frank, C. P., Marlier, R. A., Pinon-Fischer, O. J., and Mavris, D. N., "Evolutionary multi-objective multi-architecture design space exploration methodology," *Optimization and Engineering*, Vol. 19, No. 2, Jan. 2018, pp. 359–381. doi:10.1007/s11081-018-9373-x.

[105] Dorfman, R., "A Formula for the Gini Coefficient," *The Review of Economics and Statistics*, Vol. 61, No. 1, Feb. 1979, pp. 146. doi:10.2307/1924845.

[106] Damgaard, C., "Gini Coefficient," MathWorld - A Wolfram Web Resource, created by Eric W. Weisstein, 2002.

[107] Castellanos, P. C., *Airfoil Optimisation with Genetic Algorithm*, Master's thesis, Universitat Politècnica de Cataluyna, 2021.

[108] Traisak, O., Dolwichai, T., Srisertpol, J., and Thumthae, C., "Study of airfoil shape optimization by using the evolutionary method," *Proceedings of the 5th International Conference on Mechanical Engineering, Materials and Energy (5th ICMEME2016)*, Atlantis Press, 2016, pp. 109–113. doi:10.2991/icmeme-16.2016.19.

[109] Kulfan, B. M., "Universal Parametric Geometry Representation Method," *Journal of Aircraft*, Vol. 45, No. 1, Jan. 2008, pp. 142–158. doi:10.2514/1.29958.

[110] Deb, K., "Introduction to Evolutionary Multiobjective Optimization," *Multiobjective Optimization*, Springer Berlin Heidelberg, 2008, pp. 59–96. doi:10.1007/978-3-540-88908-3_3.

[111] Pelamatti, J., Brevault, L., Balesdent, M., Talbi, E.-G., and Guerin, Y., "How to Deal with Mixed-Variable Optimization Problems: An Overview of Algorithms and Formulations," *Advances in Structural and Multidisciplinary Optimization*, Springer International Publishing, Dec. 2017, pp. 64–82. doi:10.1007/978-3-319-67988-4_5.

[112] Bussemaker, J., "SBArchOpt: Surrogate-Based Architecture Optimization," 2023.

[113] Sonneveld, J., van den Berg, T., la Rocca, G., Valencia-Ibáñez, S., van Manen, B., Bruggeman, A., and Beijer, B., "Dynamic workflow generation applied to aircraft moveable architecture optimization," *Aerospace Europe Conference 2023 - 10th EUCASS - 9th CEAS*, Council of European Aerospace Societies, 6 2023, pp. 1–16.

[114] van den Berg, T., van der Laan, T., van Manen, B., Ciobotia, I., Bansal, D., and Sonneveld, J., "A multidisciplinary modelling system for aircraft structural components," *Aerospace Europe Conference 2023 - 10th EUCASS - 9th CEAS*, Council of European Aerospace Societies, 6 2023, pp. 1–15.

[115] van den Berg, T. and van der Laan, T., "A multidisciplinary modeling system for structural design applied to aircraft moveables," *AIAA AVIATION 2021 FORUM*, American Institute of Aeronautics and Astronautics, July 2021, pp. 1–12. doi:10.2514/6.2021-3079.

[116] van der Laan, T., Johman, A., van Puffelen, T., Nolet, S., van Manen, B., Daugulis, E., and van den Berg, T., "An open source part cost estimation tool for MDO purposes," *AIAA AVIATION 2021 FORUM*, American Institute of Aeronautics and Astronautics, July 2021, pp. 1–12. doi:10.2514/6.2021-3058.

[117] Weymark, J. A., "Generalized Gini inequality indices," *Mathematical Social Sciences*, Vol. 1, No. 4, Aug. 1981, pp. 409–430. doi:10.1016/0165-4896(81)90018-4.

[118] Brockhoff, D. and Zitzler, E., “Are All Objectives Necessary? On Dimensionality Reduction in Evolutionary Multiobjective Optimization,” *Parallel Problem Solving from Nature - PPSN IX*, Springer Berlin Heidelberg, 2006, pp. 533–542. doi:10.1007/11844297_54.

[119] Wang, Z., Hutter, F., Zoghi, M., Matheson, D., and de Freitas, N., “Bayesian Optimization in a Billion Dimensions via Random Embeddings,” 2013. doi:10.48550/ARXIV.1301.1942.

[120] La Rocca, G., “Knowledge Based Engineering,” *Multidisciplinary Design Optimization Supported by Knowledge Based Engineering*, chap. 9, Wiley-Blackwell, 2015, pp. 208–257.

[121] van Gent, I. and La Rocca, G., “Formulation and integration of MDAO systems for collaborative design: A graph-based methodological approach,” *Aerospace Science and Technology*, Vol. 90, July 2019, pp. 410–433. doi:10.1016/j.ast.2019.04.039.

[122] Papalambros, P. Y. and Wilde, D. J., *Principles of Optimal Design*, Cambridge University Press, 3rd ed., Jan. 2017. doi:10.1017/9781316451038.