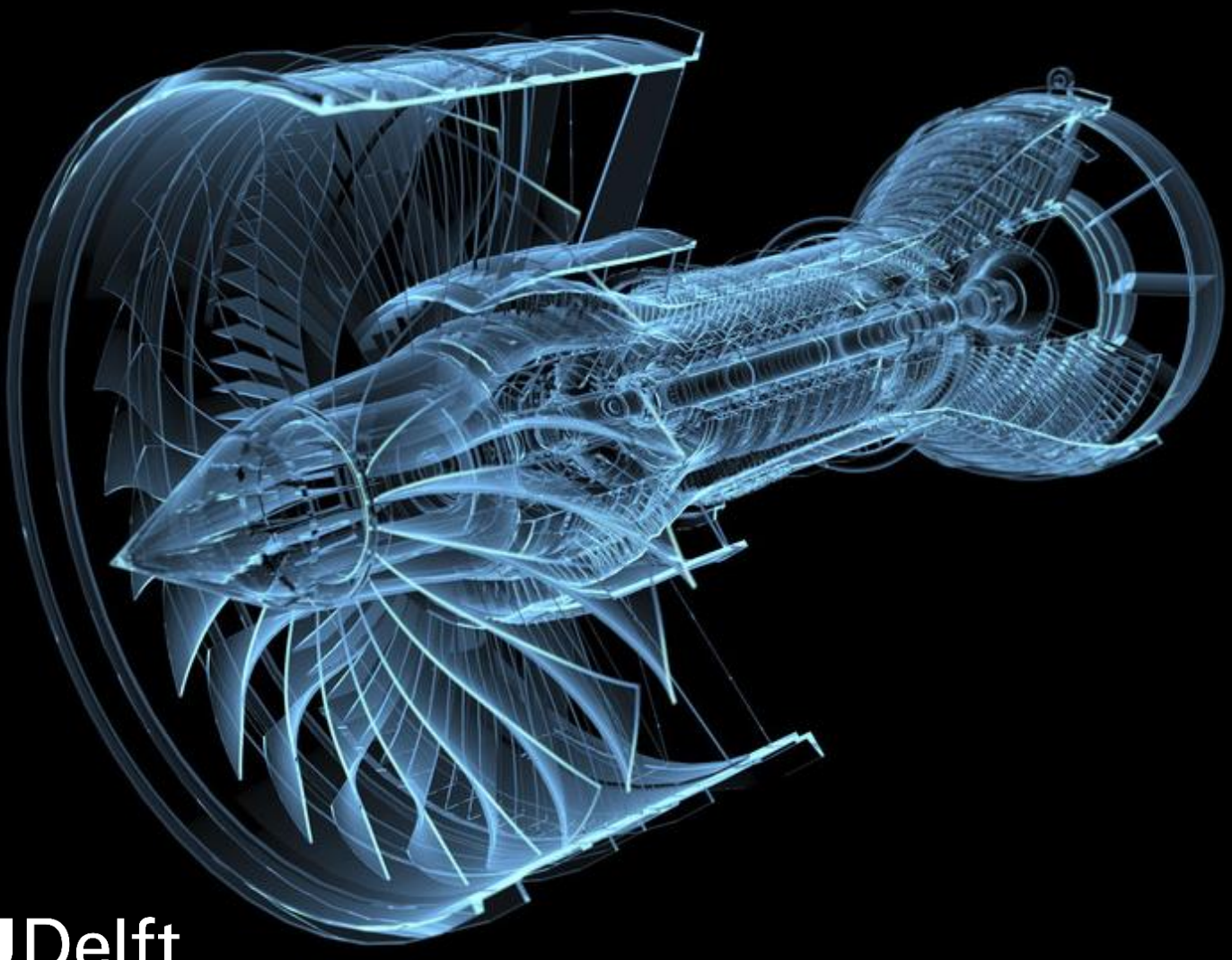
Aircraft Jet Engine Architecture Modeling

Creating a Benchmark Problem using a System Architecting Approach with Mixed-Discrete & Multi-Objective Capabilities

T.S.E.P. De Smedt

Faculty of Aerospace Engineering

Delft University of Technology



TU Delft

Aircraft Jet Engine Architecture Modeling

Creating a Benchmark Problem using a System Architecting Approach with Mixed-Discrete & Multi-Objective Capabilities

by

T.S.E.P. De Smedt

to obtain the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology,
to be defended publicly on Monday September 13, 2021 at 9:00.

Student number:	4556003	
Project duration:	November 2020 - September 2021	
Thesis committee:	Dr.ir. G. La Rocca	TU Delft, FPP, Supervisor
	Dr.ir. M. Pini	TU Delft, FPP
	Dr. F. Yin	TU Delft, C&O
	J.H. Bussemaker	DLR, System Architectures, Supervisor

TU Delft Delft University of Technology

Acknowledgments

This thesis report is the final step in my journey to become an aerospace engineer specialized in aircraft propulsion & power systems. After five intensive but rewarding years at the Aerospace Faculty of the Delft University of Technology, I am ready to bring this chapter to a close and start a new one in my life, both on professional and on personal level.

First of all, I would like to thank Jasper Bussemaker from the DLR Institute of System Architectures in Aeronautics in Hamburg for giving me this amazing opportunity to work on such an intriguing and innovative thesis topic. Next to that, his valuable feedback and trust in me have pushed me to perform better and to contribute to the scientific society. Secondly, I would like to express my gratitude to Gianfranco La Rocca for his critical view on my work and teaching me to take a broad view on the art of research. Next to that, I would like to thank Bieke von den Hoff and Feijia Yin for our conversations on various topics of my work.

Most importantly, I would like to express my sincere gratitude to my family and friends as well as my personal coach for supporting me not only during my thesis, but also during my full five years at the Delft University of Technology and The University of Texas at Austin. Finally, I cannot express how grateful I am to have gotten such amazing opportunities and mental support from my family.

Thibault De Smedt
Merchtem, Belgium, September 2021

Executive Summary

A system can be defined as a set of elements which interact with each other, and of which the resulting functionality is greater than the sum of the separate entity functionalities. The description of these elements and their interactions is called a system architecture. Even though the field of systems engineering has existed for a long time allowing for the design of very complex systems, it still has some challenges that need to be addressed. With a growing number of elements and development & improvement of technologies, systems are becoming increasingly complex. Furthermore, it is difficult to predict the effect of important decisions taken during the early stages of the design on the final system performance. However, system designers are met with the task of making important design choices even without in-depth knowledge of design decision effects. To tackle these challenges, system experts usually weigh in on these decisions with their advice. However, this advice can suffer from bias, subjectivity, conservatism or overconfidence. Systematic exploration of the system design space would offer noteworthy advantages as multiple potential system designs could be generated and their performance analyzed during all stages of the design process.

The architecture optimization process of a system is a black-box, mixed-discrete, multi-objective and hierarchical problem, making it a challenging problem. It is expected that existing optimization algorithms will have difficulties with either solving the problem at all (i.e. finding an optimal architecture), or with solving the problem without needing an excessive number of function evaluations. Therefore, existing and new optimization algorithms need to be developed for and tested on realistic architecture optimization problems. However, from literature and state of the art, it was found that no such benchmark problem currently exists. This means that there is still a need for a system architecture optimization benchmark problem to develop and test optimization algorithms which can deal with design space exploration and to educate system architecture optimization stakeholders. These stakeholders include disciplinary experts, analysis tool developers, system architects, system engineers and project managers.

In the thesis research, such a benchmark problem with an application to aircraft jet engine architecture modeling is created. The reason for this topic choice is the strongly multidisciplinary, mixed-discrete and multi-objective nature of engine design. To tackle the systems engineering challenges of the benchmark problem, a multidisciplinary design optimization tool using a system architecting approach with mixed-discrete and multi-objective capabilities was developed and assessed. The mixed-discrete capabilities enable the implementation of three types of design variables: continuous, integer and categorical. Therefore, the optimization algorithm can make both influential design decisions and sizing decisions, leading to a better overview of the performance of the possible system architectures in the design space. To avoid that the same design vector is evaluated multiple times, result caching is implemented: in case a design vector has to be evaluated an additional time, the results of the first evaluation are simply returned. Next to that, imputation of the design vector is implemented in the tool to deal with decision hierarchy: inactive design variables in the original design vector are replaced by a predefined value leading to the imputed design vector which is then evaluated. Furthermore, with the multi-objective capabilities of the tool, it is possible to take into account the multiple conflicting stakeholder requirements and regulations of the aircraft jet engine design and optimization

process. Finally, the tool uses the extreme barrier approach, meaning that when a result is invalid or the system architecture evaluation did not converge, a value of $+\infty$ or $-\infty$ is returned for all results to represent the design point as a point with inferior performance.

The developed tool uses the engine cycle analysis framework pyCycle, the MDO integration framework OpenMDAO and the optimization framework pymoo. It consists of two main parts:

1. The *Engine Architecting Framework*: this component provides a way to describe the architectural choices, and from there define a design vector fed to the *Engine Architecture Evaluator*, and functionality for translating a design vector into an engine architecture definition. The architectural choices in the tool include the incorporation of a fan at the front of the engine, specification of the number of shafts, implementing a gearbox, consideration of innovative technologies such as a counter-rotating fan or inter-turbine burner, etc. This means that different distinct aircraft engine architectures can be formed such as a conventional and afterburning turbojet, a conventional and geared turbofan, and multi-shaft engines. Therefore, the *Engine Architecting Framework* operates at the level of the architecture design space and optimization problem.

2. The *Engine Architecture Evaluator*: this component provides a way to create and evaluate an engine architecture for specific operating conditions, by translating the architecture definition from the *Engine Architecting Framework* into a pyCycle problem. Due to the modular approach of pyCycle, the aircraft engine components are represented as building blocks with which the engine thermodynamic cycle is built up, after which thermodynamic cycle analysis can be performed. Furthermore, it evaluates the disciplines of the engine design problem: weight, geometry, NOx emissions and noise. Therefore, the *Engine Architecture Evaluator* operates at the level of the individual architecture.

Verification and validation of the aircraft jet engine architecting tool shows that the implemented architectural design choices of the *Engine Architecting Framework* lead to feasible engine architectures and that the discipline results are valid compared to the CFM LEAP-1C turbofan and P&W F100 turbojet engines used on the Comac C919 and F16 aircraft, respectively. Next to that, two aircraft jet engine architecting problems were created: a simple single-objective problem with only feasibility constraints, and a more realistic multi-objective problem with three objectives (TSFC, weight and noise) and multiple constraints (geometry, NOx emissions and feasibility). The same operating conditions were used as for the existing LEAP-1C engine in take-off conditions, and both conventional and innovative engine architectures (such as a counter-rotating fan and/or inter-turbine burner) were generated and evaluated. The genetic optimization algorithm NSGA-II was used to optimize the engine architectures in both problems. The trends of the results obtained by the optimizer were as expected serving as additional validation of the tool. Furthermore, even though non-convergence issues were encountered during the optimization, a Pareto front with non-dominated aircraft engine architectures was successfully formed. Therefore, the aircraft jet engine architecting problem can be used as realistic benchmark problem to support the development, evaluation and comparison of system architecture design space modeling methods and optimization algorithms while simultaneously educating various stakeholders on the relevant aspects of architecture optimization.

The results of these architecting problems showed that the objective of the thesis was achieved: a benchmark problem with an application to aircraft engine architectures using a system architecting approach with mixed-discrete and multi-objective capabilities was created, in order to further research and educate stakeholders on system architecture modeling & optimization.

Contents

List of Figures

List of Tables

Nomenclature

Latin Symbols

A	Area	[m^2]
a	Speed of sound	[m/s]
c	Heat capacity	[J/kgK]
D	Diameter	[m]
F	Force	[N]
l	Length	[m]
$\dot{m}$	Mass flow rate	[kg/s]
P	Pressure	[N/m^2]
p	Power	[W]
q	Heat transfer rate	[W]
R	Distance	[m]
T	Temperature	[K]
t	Time	[s]
U	Overall heat transfer coefficient	[W/m^2K]
V	Velocity	[m/s]
W	Weight	[kg]

Greek Symbols

ϵ	Effectiveness	[-]
η	Efficiency	[-]
κ	Heat capacity ratio	[-]
λ	Bypass ratio	[-]
ω	Angular velocity	[rad/s]
ρ	Density	[kg/m^3]
τ	Torque	[Nm]

Abbreviations

(O)PR	(Overall) Pressure Ratio
(U)HBR	(Ultra-)High Bypass Ratio
(X)DSM	(Extended) Design Structure Matrix
ADORE	Architecture Design and Optimization Reasoning Environment
ADSG	Architecture Design Space Graph
ANOPP	Aircraft Noise Prediction Program
BPR	Bypass Ratio
CC	Combustion Chamber
CFD	Computational Fluid Dynamics
CRTF	Counter-Rotating Turbofan
DBSE	Document-Based System Engineering
DLR	German Aerospace Center
DOE	Design Of Experiments
ECS	Environmental Control System
FAR	Fuel-to-Air Ratio
FEM	Finite Element Modeling
FPR	Fan Pressure Ratio
GA	Genetic Algorithm
GSP	Gas turbine Simulation Program
GTF	Geared Turbofan
HEX	Heat Exchanger
HPC	High-Pressure Compressor
HPT	High-Pressure Turbine
ICAO	International Civil Aviation Organization
IPC	Intermediate-Pressure Compressor
IPT	Intermediate-Pressure Turbine
ISA	International Standard Atmosphere
ITB	Inter-Turbine Burner

KBE	Knowledge-Based Engineering
LHV	Lower Heating Value
LMTD	Log Mean Temperature Difference
LPC	Low-Pressure Compressor
LPT	Low-Pressure Turbine
MBSE	Model-Based System Engineering
MDO	Multidisciplinary Design Optimization
MOEA	Multi-Objective Evolutionary Algorithm
MTOW	Maximum Take-Off Weight
NPSS	Numerical Propulsion System Simulation
NSGA-II	Nondominated Sorting Genetic Algorithm II
NTU	Number of Transfer Units
OASPL	Overall Sound Pressure Level
QOI	Quantity Of Interest
SEP	Systems Engineering Process
SysML	Systems Modeling Language
TIT	Turbine Inlet Temperature
TSFC	Thrust-Specific Fuel Consumption
WATE	Weight Analysis of Turbine Engines

Chapter 1
Introduction

With an increase in number of elements, the diversity of those elements and their connectivity, systems are becoming increasingly complex [1]. In the current engineering climate, important architectural decisions, which could have a significant effect on the performance of the designed system, have to be taken early on in the design phase. Next to that, with the development and improvement of new technologies, the range of options for the system design is further increased. This means that more design decisions have to be made for more complex systems. Without a thorough understanding of the effect of these design decisions on the system, the system performance and capabilities could potentially be limited [2].

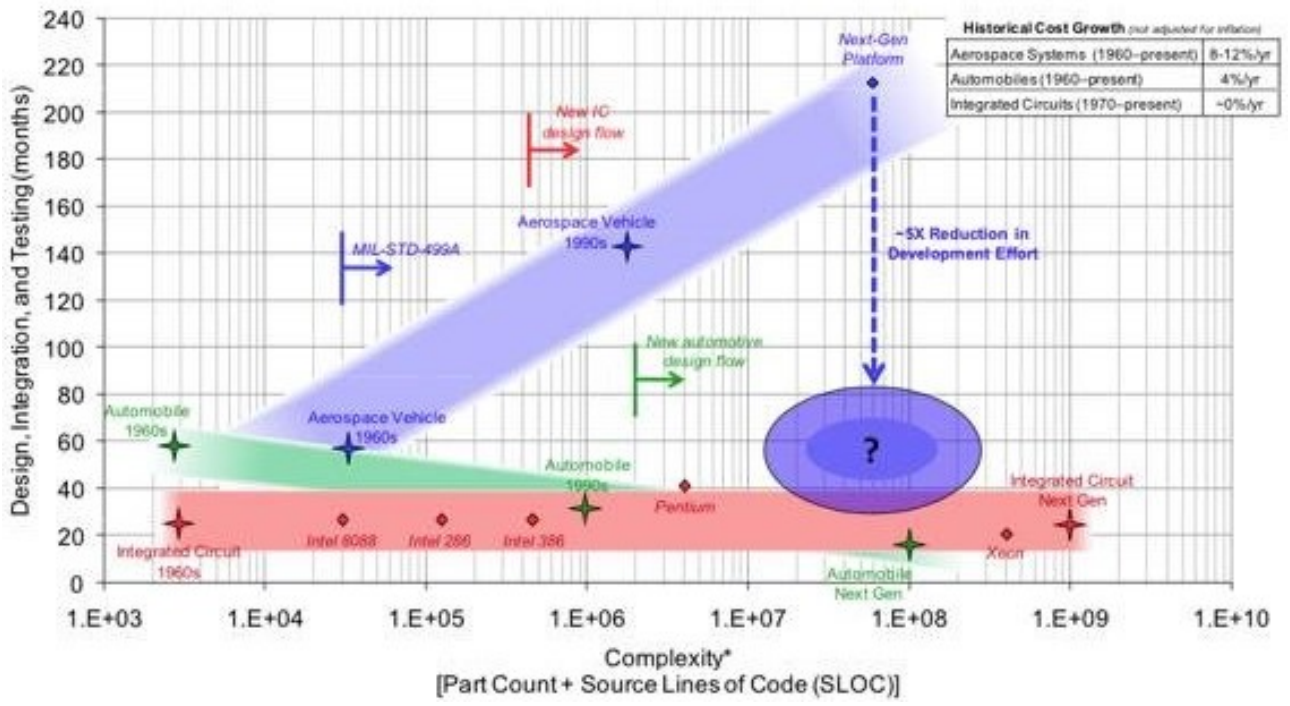


Figure 1.1: Increasing complexity in both parts and lines of code for systems and the result on system development time, especially in the aviation sector [3].

To deal with these challenges, system experts are often consulted to provide their advice on the design choices. Even though these experts can provide some insight into the decision effects, they may suffer from bias, subjectivity, conservatism or overconfidence [4]. To solve these problems, systematic design space exploration techniques need to be applied during the design of a complex system, also at the earlier design stage when the architecture of the system has not been fixed yet [5]. This would not only enable effective exploration of the design space, but also increase the understanding of the impact of important architectural decisions and reduce

reliance on expert judgment. To improve design space exploration techniques, existing and new optimization algorithms should be developed and tested, and system architecture optimization stakeholders should be educated. These stakeholders include disciplinary experts, analysis tool developers, system architects, system engineers, project managers, engineering students, etc. Therefore, a benchmark problem regarding system architecture optimization should be available to perform these tasks on. However, a thorough review of literature and state of the art indicated that no such benchmark problem currently exists.

In this thesis research, a system architecture optimization benchmark problem with an application to aircraft jet engine design is created. There are multiple reasons why aircraft engine design is chosen for this purpose. The combination of conventional engine architectures and innovative technologies, such as a geared turbofan and intercooled cycles, makes that many architectural choices have to be considered. These choices are not only on the component level but also on the complete system level, resulting in different types of design decisions such as how many shafts should be included in the engine architecture. Then, the effect of the decisions has to be analyzed in terms of the many disciplines involved in the design process of an aircraft jet engine, including fuel consumption, weight and emissions. Next to that, there are multiple conflicting stakeholder requirements and regulations that have to be complied with. This leads to a mixed-discrete, multidisciplinary and multi-objective design problem [6].

It must be noted that during the research, only aircraft jet engines will be considered. Other aircraft engine architectures exist such as ramjets, scramjets and even electrical or hydrogen propulsion systems [7–10] but these architectures were not included. The main reason is that major adjustments in the engine architectures and disciplines would have to be implemented when including the additional architectures, leading to a too complex benchmark problem for the purpose.

A system architecting approach is developed which can deal with the systems engineering challenges of the benchmark problem by implementing mixed-discrete and multi-objective capabilities. Due to its mixed-discrete capabilities, it is expected that an increase in design space size and more freedom to the designers of the system will be established, which might lead to the generation of systems with improved performance. To illustrate the improvement mixed-discrete capabilities offer to the optimization process, the example of the Numerical Propulsion System Simulation (NPSS) of NASA can be taken [11]: while in the past only one single aircraft engine architecture such as a turbofan without gearbox could be optimized in the design process [6], the system architecting approach taken in the thesis research allows for optimization of the complete aircraft engine design space. This could improve the chances of finding a better performing aircraft engine architecture which provides more value to the stakeholders. To demonstrate that the developed system architecting approach can effectively be implemented and exploited for real-life engineering optimization problems with multiple disciplines involved, a Multidisciplinary Design Optimization (MDO) tool will be built which incorporates this system architecting approach. The tool will be designed to generate and optimize aircraft jet engines for user-set objectives and constraints.

Based on this background knowledge, the thesis objective can be formulated in section 1.1. Then, the thesis objective will be broken down into research questions in section 1.2. Finally, an outline of the thesis paper will be given in section 1.3.

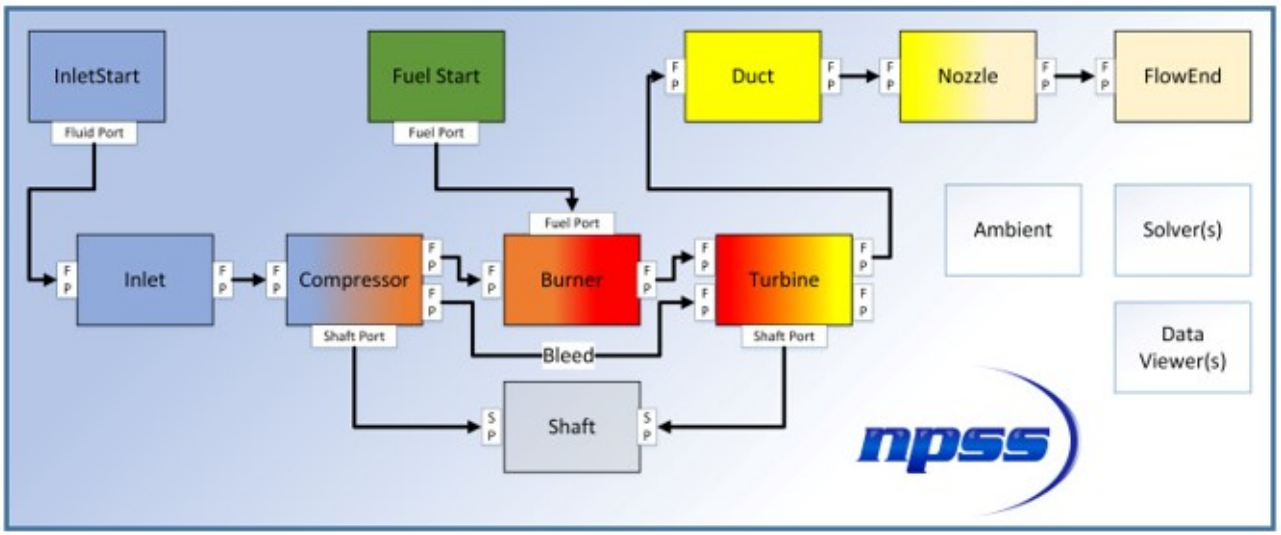


Figure 1.2: Static aircraft engine system architecture setup of NPSS [12].

1.1 Thesis Objective

The thesis objective can be formulated as:

> "To further research and educate stakeholders on system architecture modeling & optimization, by creating a benchmark problem with an application to aircraft engine architectures using a system architecting approach with mixed-discrete & multi-objective capabilities."

1.2 Research Questions

The different research questions that will be addressed during the thesis are listed as follows:

RQ1 How is the system architecting approach implemented in aircraft engine modeling?

- RQ1.1 To what extent can realistic values for the engine performance be obtained?
- RQ1.2 Which limitations does the system architecting approach have?

RQ2 How does the approach address the nature of architecting decisions?

- RQ2.1 Which continuous and discrete architecting decisions can be taken?
- RQ2.2 What is the impact of the discrete architecting decisions on the aircraft engine components and connections?
- RQ2.3 What is the influence of the discrete architecting decisions on the design vector?
- RQ2.4 Can the relevant effects of engine architecting decisions be captured?
- RQ2.5 What is the size of the design space?

RQ3 Does the approach enable the creation of multiple optimal aircraft engine architectures?

- RQ3.1 Which engine objectives and constraints are used?

RQ3.2 Which existing and novel engine architectures are part of the solution space generated by the approach?

RQ3.3 Can a Pareto front be formed with non-dominated engine architectures?

1.3 Thesis Outline

To provide the reader with sufficient background on engineering fields relevant for the thesis research, systems engineering and some of its challenges are first explained in chapter 2. Then, a benchmark problem is created to further research and educate stakeholders on system architecture modeling and optimization in chapter 3.

After that, the MDO tool implementing a system architecting approach with mixed-discrete and multi-objective capabilities used to tackle the benchmark problem is explained in chapter 4. This tool consists of two parts: the *Engine Architecting Framework* and the *Engine Architecture Evaluator*. These will be taken a closer look at in chapter 5 and 6, respectively.

To ensure that the aircraft jet engine architecting tool outputs feasible results, it is verified and validated in chapter 7. Then, a simple single-objective and a more realistic multi-objective aircraft engine design problem will be created and the results of both problems will be investigated in chapter 8. Finally, conclusions and recommendations will be given in chapter 9.

Chapter 2
Systems Engineering

A system can be defined as a set of elements which interact with each other, and of which the resulting functionality is greater than the sum of the separate entity functionalities [2]. The process of creating such a system while taking into account stakeholder requirements and needs is called the Systems Engineering Process (SEP) [13]. Many approaches to SEP currently exist but in general, SEP can be divided into three main steps [14, 15]:

1. Stakeholder needs definition;
2. Technical requirement definition;
3. System architecting.

The focus of this work lies on the third step of SEP: system architecting.

In this chapter, the fields of system architecting and Multidisciplinary Design Optimization (MDO) are explained in section 2.1 and 2.2, respectively. After that, some challenges of these fields in systems engineering are discussed and linked to the scientific contribution of the thesis research in section 2.3.

2.1 System Architecting

Systems are created for a purpose: they offer value to their stakeholders, defined as a certain benefit at a certain cost. These stakeholders are parties which have an effect or are effected by the system, and they determine what the system in general should do to create value to them. What the system does is referred to as the *function* of the system. How the system fulfills this function is called the *form* of the system, and is taken care of by the design engineers or architects. When form is assigned a function, or vice versa, a form-function-mapping is established [5]. Considering that system architecting is a recursive process, the function and form terms not only apply to the system level but also to the subsystem level etc. This shows that many steps are involved in system architecting such as determining the system functions and component options.

2.1.1 Design Decisions

In system architecting, the entities of a system are connected and assigned to their function(s) by making design decisions. Therefore, the system architecting process itself can be seen as a decision-making process for which many design decisions are usually identified, and the best option for each decision needs to be determined. The result of the system architecting process is a system architecture [16]. When assembling all the different design decisions that can or are taken, the architecture design space is created, containing all the possible system architectures generated by the design decisions.

2.1.2 DBSE and MBSE

To perform the systems engineering process, two main approaches can be taken: document-based (DBSE) or model-based (MBSE). The former uses textual resources, whereas the latter uses digital models to represent system architectures. To visualize the models in the model-based approach, modeling languages such as the Systems Modeling Language (SysML) [17] are used which generally show the different system entities and their connections. Whereas both approaches can be used for the entire system life cycle, the model-based approach is currently more often used than the document-based approach due to its advantages such as the convenient re-usability of models, higher design quality and integrity, etc. [17] Furthermore, even though in the initial phases of the system design MBSE actually requires more effort and costs to implement compared to DBSE, in the long run MBSE proves to be more advantageous for both effort and costs. This effect is visualized in Figure 2.1. During the thesis research, the model-based approach is followed.

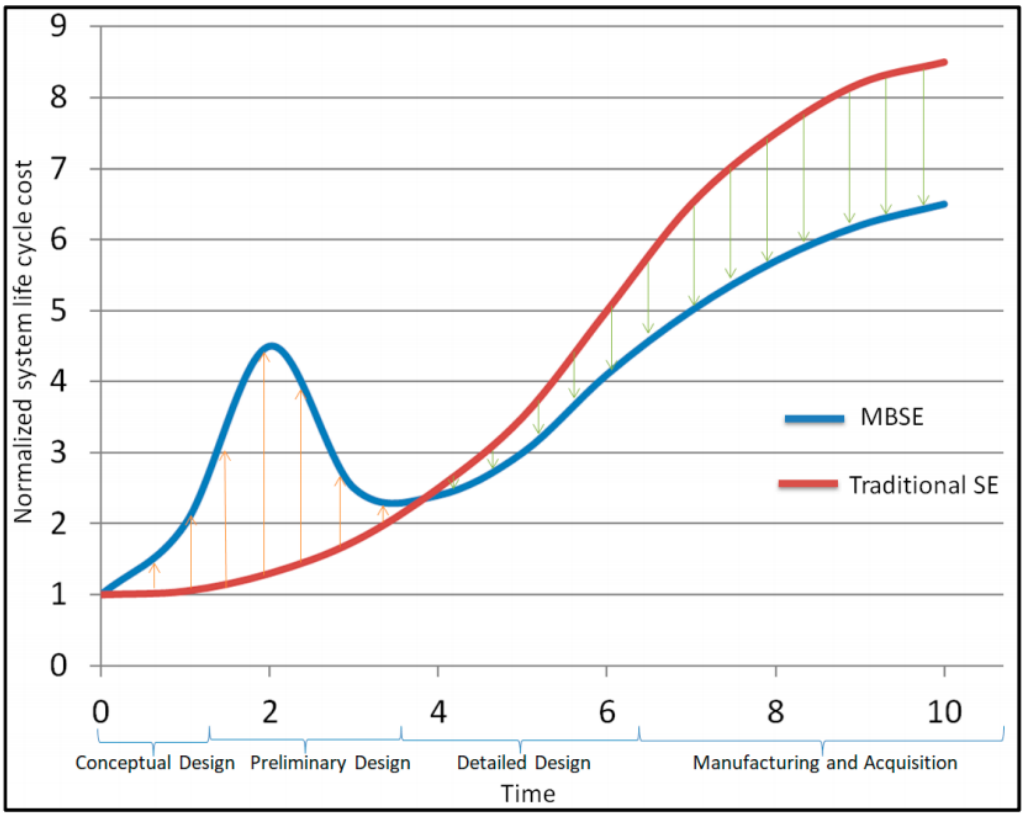


Figure 2.1: System life cycle project cost in time of document- and model-based systems engineering [18].

2.2 Multidisciplinary Design Optimization

In order to find the best performing system architecture for the imposed stakeholder requirements and needs, MDO can be implemented in the SEP. MDO can be defined as the engineering field which focuses on using optimization methods to tackle design problems with multiple disciplines. It involves quantitatively comparing the performance of generated system architectures by defining objectives and constraints that apply to all architectures, regardless of their structure and included functions and components. Additionally, it must be noted that because

of the many stakeholders involved in the design of a system, multiple conflicting goals will arise.

There are several reasons why MDO is often used in current engineering problems [19]:

- To deal with the cooperation and communication when multiple persons are involved in the design of a system, and ensuring the compatibility of in- and outputs when software programs are used.
- To deal with the increasing complexity of the designed system and the design processes themselves, and the explosion of possible architectures when multiple design decision have to be taken.
- To deal with the different disciplines involved in the design process of a system, which could potentially be opposed, meaning that an improvement in one discipline might lead to a worsening of a different discipline.

It must be noted that these are not the only reasons why MDO is used, but these three points were deemed the most important for the thesis research.

2.2.1 Design Space

To formalize the architecting process into an optimization problem, the design space needs to be modeled in such a way that decisions are clearly identified and that each architecture candidate correctly represents the function-form-structure mapping. In that way, architecture candidates can be automatically generated enabling systematic exploration of the design space. In optimization, a distinction can be made between two types of design variables: continuous (differentiable) and discrete (non-differentiable). The discrete design decision type can be subdivided between integer and categorical, and these differ based on ordinality: the order of the numbers matters for the former, while this order does not have a significant meaning for the latter [20]. Examples of these different types of decisions would be the integration of the engine in the airframe (wing- or fuselage mounted) for categorical decisions, the number of engines on an aircraft for integer decisions and the fuselage diameter for continuous decisions.

The possible decisions of an MDO problem are described as design variables, creating a design vector with all the combined variables. Generally, the continuous design variables have bounds indicating its lower and upper limits, forming an *unconstrained* MDO design space. When constraints are imposed on the MDO problem, a *constrained* MDO design space is created containing only valid solutions to the problem. In relatively simple MDO problems, these constraints can be described as an analytical function of (some) design variables. However, in more complex problems, it becomes clear that (some) constraints cannot be simply written as an analytical function of design variables.

2.2.2 Single-Objective Optimization

When the MDO problem only has one single objective, this is referred to as single-objective optimization. This means that only one optimal point can be formed in the MDO design space which leads to the best performance in that objective. The mathematical expression for this kind of MDO problem is [19]:

$$
\begin{aligned}
&\textit{Minimize}: \\
&\qquad \boldsymbol{f(x)} \\
&\textit{Subject to}: \\
&\qquad \boldsymbol{g_j(x)} \leq 0 \quad for \quad j = 1...m \\
&\qquad \boldsymbol{h_j(x)} = 0 \quad for \quad j = 1...p \\
&\qquad \boldsymbol{x^u} \geq \boldsymbol{x_i} \geq \boldsymbol{x^l}
\end{aligned}
\tag{2.1}
$$

In this equation, the design variables are indicated by x_i with lower bound x^l and upper bound x^u. The objective function and constraints are $f(x)$ and $g(x)$ & $h(x)$, respectively. The constraints can be split up into the inequality $g(x)$ and equality $h(x)$ constraints: the former may or may not be active, while the latter is always active. It must be noted that constraints *can* be implemented, while an objective *has* to be used in an MDO problem.

To solve single-objective optimization problems, two main methods can be employed: gradient- and non-gradient-based (also called gradient-free). Both start off with an initial guess or population of initial guesses for the optimal solution of the problem, which can be based on industry expertise or even a random guess. For gradient-based methods, an improved solution is found by using the design variables derivative, forming a function gradient, to guide the search direction of the optimizer [21]. The gradient-based method can be advantageous for problems where a mathematical model of the design space can be formed, however the disadvantage is that the optimizer can get stuck in optimizing local optima instead of finding the global optimum. In contrast, non-gradient-based methods simply evaluate the objective function of the problem at certain points to try to arrive to an improved solution [22]. Many methods exist to determine these points such as the Divided Rectangles Method, Nelder-Mead Simplex and Genetic Algorithms [23]. The non-gradient-based method increases the probability of finding the global optimum, but might be less efficient for problems where a mathematical model of the design space can be created. Furthermore, a combination of the two optimization methods is also possible, which starts with an exploration of the design space to find the area of the best solution with a non-gradient-based method and then transitions to gradient-based method to improve the speed and accuracy of the optimization.

2.2.3 Multi-Objective Optimization

Multi-objective optimization problems deal with optimization problems with multiple conflicting objectives. This means that multiple optimal solutions can be created which perform better in at least one of the objectives than other solutions but worse in others. The set of these optimal (or non-dominated) solutions is called the Pareto front [24]. This means that there will be a trade-off for the system designer based on the included objectives, and it is up to this designer to choose the solution with the desired combination of performances. An example of points on a Pareto front for a bi-objective problem can be seen in Figure 2.2. In this Apollo mission design problem, the system designers will have to make a trade-off between mission probability success and initial mass into low earth orbit as it is only possible to increase the former by increasing the latter [5].

Equation 2.2 describes the mathematical expression for multi-objective optimization problems [19]. Note that this equation is similar to Equation 2.1: the only difference is that the objective function has become a set of objective functions.

$$
\begin{aligned}
&\textit{Minimize}: \\
&\qquad \boldsymbol{f(x)} = \{\boldsymbol{f_1(x)}...\boldsymbol{f_q(x)}\}^T \\
&\textit{Subject to}: \\
&\qquad \boldsymbol{g_j(x)} \leq 0 \quad for \quad j = 1...m \\
&\qquad \boldsymbol{h_j(x)} = 0 \quad for \quad j = 1...p \\
&\qquad \boldsymbol{x^u} \geq \boldsymbol{x_i} \geq \boldsymbol{x^l}
\end{aligned}
\tag{2.2}
$$

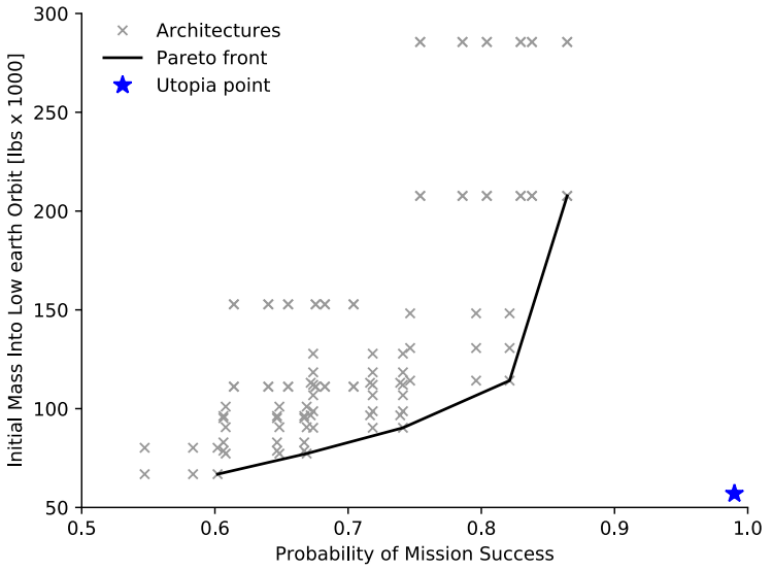


Figure 2.2: Bi-objective problem with multiple generated architectures and a clearly indicated Pareto front, connected by architectures with high performance in at least one of the two objectives. The utopia point indicates where the "best" architectures should be located [5].

2.3 Challenges & Scientific Contribution

With the background knowledge of system architecting and MDO in systems engineering, their challenges can be explained in subsection 2.3.1 and 2.3.2, respectively. Then, the scientific contribution of the thesis research will be laid out in subsection 2.3.3.

2.3.1 System Architecting

Optimization With new technological developments and more complex systems increasing the size of the architecture design space, it is important to be able to perform accurate design and fair comparison of the different system architectures. However, relevant architecting decisions are usually not included in the optimization process as categorical design decisions cannot be made by the optimizer. In current modeling of the system architecture design space, important design decisions are usually taken early on in the design process of a system without in-depth knowledge of its effect(s) on the system, possibly leading to inferior performance. To address this problem, industry experts are employed to estimate the decision effects on the system based on their experience, however these can be biased and subjective [5]. Several factors

contribute to the difficulty of solving system architecture optimization problems. The performance analysis model generally consists of multiple expensive black-box functions. This means that no a-priori information is available about the shape and topology of the design space. Next, the architecture optimization problem is a mixed-discrete, multi-objective, hierarchical problem: mixed-discrete because design decisions might consist of both categorical architecting decisions and integer or continuous sizing parameters, multi-objective because the evaluation of possible architectural choices is generally based on multiple conflicting stakeholder selection criteria, and hierarchical because design variables can be conditionally active based on other design variables [5]. In fact, the hierarchical nature is generally a consequence of the aforementioned categorical variables, as the presence of certain component(s) in the system architecture comes with the necessary design variables to define said component(s).

Benchmark Problem The system architecting characteristics make for a class of challenging optimization problems. It is expected that existing optimization algorithms will have difficulties with either solving the problem at all (i.e. finding an optimal architecture), or with solving the problem without needing an excessive number of function evaluations. In light of this, it is important that existing and new optimization algorithms are developed for and tested on realistic architecture optimization problems. After a thorough review of literature and state of the art, it was concluded that no such benchmark architecture optimization problem currently exists. Therefore, there is a need to fill said gap by the development of a benchmark optimization problem for the specific purposes of:

1. Supporting the development, evaluation, and comparison of optimization algorithms suitable for system architecture optimization;
2. Supporting the development, evaluation, and comparison of system architecture design space modeling methods;
3. Educating researchers, engineers, and other stakeholders on the behavior and characteristics of system architecture optimization problems.

2.3.2 MBSE Process Integration

The MBSE process can be subdivided into two main architecting phases [5]:

- Upstream: this includes the identification of stakeholder requirements and goals, the system architecture design and optimization process, and the trade-off of performance objectives;
- Downstream: this includes the identification of the different system design stages and their respective challenges, and the creation and operation of system design processes to enable architecture design space exploration.

The ultimate goal of these two processes is to make sure that the performance of the designed system meets the stakeholder requirements and needs, i.e. that the system contributes *value* to the stakeholders. In order to achieve this, it is necessary to create a link between the upstream and downstream processes, meaning that the stakeholder requirements upstream can automatically be transformed to objectives and/or constraints downstream while designing the system. This is one of the objectives of the AGILE 4.0 project of the European Commission [25]. It is expected that linking the upstream and downstream processes will make the development process of a complex system more inclusive, enabling trade-offs which will ultimately improve the performance of the system.

2.3.3 Scientific Contribution

Based on the challenges of system architecting, there is a need for a benchmark problem which will help develop and evaluate system architecture design space modeling methods and optimization algorithms. This benchmark problem was developed during the thesis and it is expected that, next to helping to solve the challenges, it will educate stakeholders on system architecture optimization problems. Furthermore, the benchmark problem could be used as a first step in bridging the upstream and downstream phases of the MBSE process by serving as an ad-hoc implementation on which existing and new system architecture design platforms can be tested.

In order to fulfill these scientific needs, a system architecting approach with mixed-discrete and multi-objective capabilities was implemented in a benchmark problem with an application to aircraft jet engine design. The benchmark problem will be discussed in chapter 3 while the approach will be explained in chapter 4.

Chapter 3
Benchmark Problem

To tackle the challenges discussed in chapter 2, a benchmark problem with an application to aircraft jet engine design architecting is created. It is expected that this benchmark problem will support the development and evaluation of architecture optimization algorithms and educate stakeholders such as engineers and researchers in its characteristics and behavior. First, benchmark problems and the rationale behind using them is discussed in section 3.1. Then, the overall requirements for an optimization benchmark problem will be listed in section 3.2. Finally, the reason behind the aircraft jet engine selection for the benchmark problem will be explained in section 3.3.

3.1 Rationale

A benchmark can be defined as a reference point against which the performance of other elements can be compared. It can be used for all kinds of problems and domains, from comparing the performance of management companies to evaluating different types of algorithms in a specific optimization problem. For the thesis research, the benchmark problem is expected to aid research into optimization algorithms capable of solving system architecture optimization problems, research into modeling methods for system architecture design space, and education on system architecture optimization in general. In this process, benchmarking can show both the advantages and disadvantages of using various solutions to a certain problem.

3.2 Requirements

When it comes to optimization benchmark problems, Gray mentions they should represent realistic engineering challenges by appropriately mimicking the design space behavior and dimensionality, they should have multiple difficulty levels ranging from easy familiarization problems to high-fidelity problems, and they should be open-source and based on open-source tools to aid reproducability, research, and education [26]. From the considerations discussed in this section and previous chapters, it can be concluded that to create a realistic system architecture optimization benchmark problem, it should adhere to the following requirements:

BR1 Design variables should be mixed-discrete: categorical, integer and continuous variables must be included in the design vector, so that the optimization problem becomes a mixed-discrete problem.

BR2 Multiple conditionally active design variables should be included: conditionally active design variables are active or not based on some other design variable choice, and thereby introduce a design variable hierarchy. This is a common occurrence in system architecture design spaces and should therefore be represented.

BR3 The design space should have between 10 and 50 dimensions: an appropriately high design space dimensionality should be defined to make the problem realistic enough. For system architecting problems it is expected that up to several tens of design variables can be present.

BR4 There should be multiple objectives: there should be approximately two to ten non-correlated design objectives to choose from, representing conflicting stakeholder needs.

BR5 The performance evaluation function behavior should be black-box: nothing is known a-priori of the shape and behavior of the design space due to the black-box nature of the evaluation function, and in general the design space should be assumed to be nonlinear.

BR6 The difficulty of the problem should be tunable: different difficulty levels serve different user needs: an easy problem serves as a familiarization with the problem and with system architecting in general, more difficult levels serve as realistic test cases for optimizer performance.

BR7 The problem should be open-source itself and be based on open-source tools: this helps in reproducing and verifying study results, and makes it accessible to a wider audience for research and education purposes.

BR8 The problem should be hackable: it should be easy to modify the problem structure and to include additional elements like engineering disciplines, design decisions, objectives and constraints.

3.3 Aircraft Jet Engine Design

The design of an aircraft jet engine consists of multiple disciplines. However, central to its design is the thermodynamic cycle analysis [27]: it relates the engine performance to the design choices, which decide about the structure of the engine architecture and ultimately also the thermodynamic cycle, at certain flight conditions while also taking into account the design limits. This analysis thus shows the effect of linking the individual engine components on the complete engine level performance, to determine the system-level impact of the individual components and the final performance of the engine. Detailed component-level analysis, such as Finite Element Modeling (FEM) or Computational Fluid Dynamics (CFD), can be used to improve the accuracy of component results. In the benchmark problem created during the thesis, only simple analysis will be implemented as it is used as proof of concept for the novel system architecting approach. The goal of the thesis is not to optimize actual aircraft engine performance. However, the fact that a rather complex architecture optimization problem can be implemented and solved, tackling the challenges related to the nature of its design variables, is in itself a noteworthy result with scientific value.

Next to the thermodynamic cycle analysis, other aircraft engine disciplines need to be taken into account when designing the complex system such as weight, nacelle geometry and emissions. These additional disciplines are necessary to ensure that all the stakeholder needs and requirements are fulfilled, while at the same time making sure that the engine can effectively be manufactured and integrated in the airframe of the aircraft. They may form an interesting counterpart to the thermodynamic cycle. For example, an aircraft engine with high Bypass Ratio (BPR) might be more fuel-efficient but also heavier than an engine with a lower BPR.

Furthermore, a high BPR engine could also be more difficult to integrate in the airframe as the space under the wings is limited. Next to that, (inter)national aviation authorities such as the International Civil Aviation Organization (ICAO) impose regulations on amongst others noise and emissions with which the aircraft engines have to comply [28, 29]. This means that the system designer will have to make multiple trade-offs in this multi-objective problem.

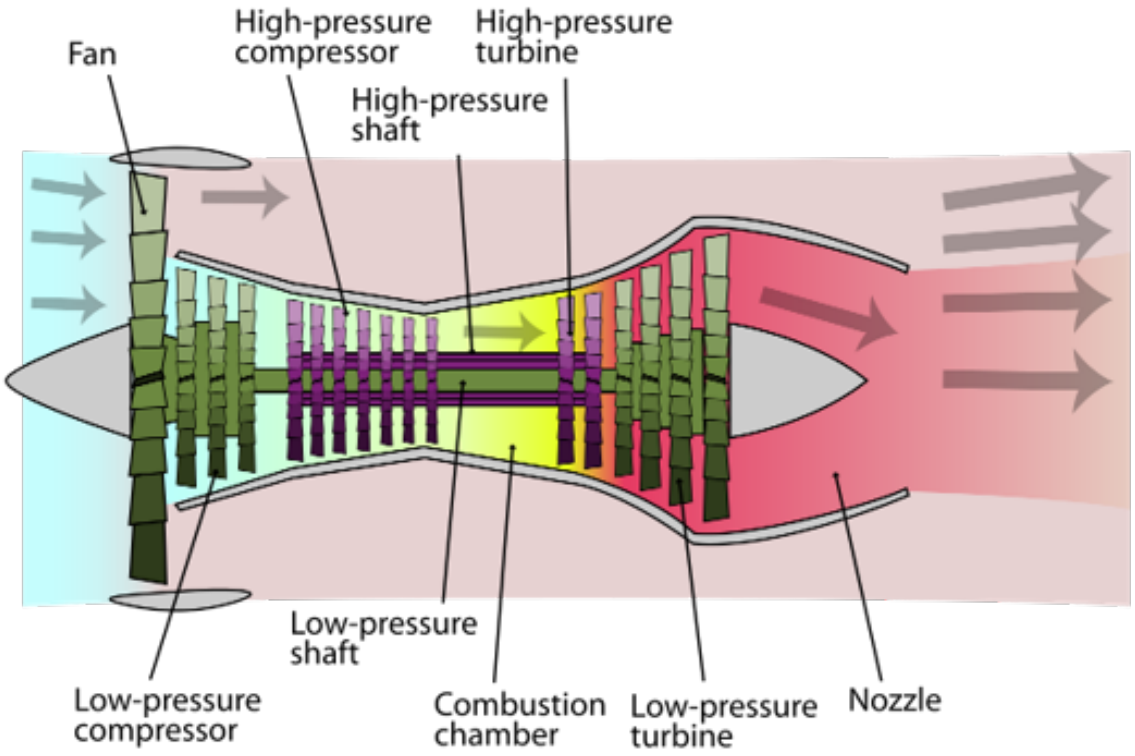
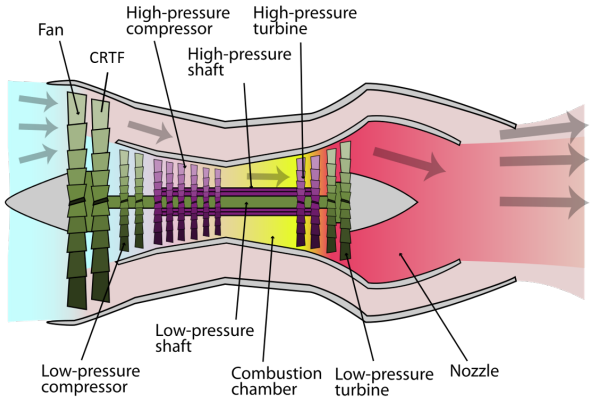
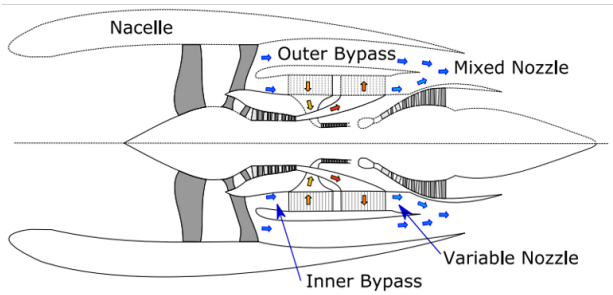


Figure 3.1: Turbofan engine architecture showing the different high-level components [30].

There are many design decisions that have to be taken when designing an aircraft engine, both on component- and on system-level. The most conventional ones are the inclusion of a fan at the front of the engine, the number of shafts, whether a gearbox is used between the fan and low-pressure compressor [31], from which shaft power is extracted to provide the aircraft systems with electricity [32] and from which compressor air is bled off for the environmental control system [33]. Furthermore, innovative configurations might be included in the design process to research their performance and/or environmental benefits as well as to offer more freedom and information to the system designers. These include the use of an intercooler [34–36], a Counter-Rotating Turbofan (CRTF) [37] and an Inter-Turbine Burner (ITB) [38]. A representation of the CRTF and intercooled cycle engine architectures can be found in Figure 3.2. Each of these questions relates to one or more (categorical) architecture decision variables, and can only be answered by systematically exploring the coupled impact of each of these decisions. Furthermore, decision hierarchy is also present as continuous design variables such as BPR are only relevant when their respective components are included in the architecture. Therefore, it is important that the benchmark problem shows what the effects are of the implementation of each of these technologies on the system-level performance and that decision hierarchy is correctly dealt with.



(a) Counter-rotating turbofan engine [37].



(b) Intercooled cycle engine [34].

Figure 3.2: Innovative configurations for aircraft engine design.

To summarize, the design of an aircraft jet engine for a given mission is suitable to implement as a system architecting benchmark problem due to the following reasons:

- The strongly multidisciplinary nature of aircraft engine design (e.g. thermodynamic cycle analysis, weight and geometry considerations, regulations);
- The multiple conflicting stakeholder requirements (e.g. thrust, fuel consumption, noise, emissions) which can be used as nonlinear black-box objectives for the optimization, depending on which engineering disciplines are available;
- The explosion of architecting choices resulting from the different possible aircraft engine architectures: 43 design variables are used in the realistic benchmark problem, as will be discussed in chapter 5, leading to a total of 91 distinct engine architectures and 2.6 million apparent engine design points;
- The inclusion of categorical decisions (e.g. are certain technologies included or not) and integer or continuous parameters (e.g. number of shafts, bypass ratio, pressure ratios), in which the number of decisions can easily be varied while still resulting in realistic engine designs;
- The presence of decision hierarchy (e.g. if there is no fan, there is no bypass ratio variable).

Chapter 4

System Architecting Approach

To tackle the systems engineering challenges of the benchmark problem, a tool implementing a system architecting approach with mixed-discrete and multi-objective capabilities has been developed during the thesis research, which is the main topic of this chapter. First, an overview of the methodological approach to tackling the benchmark problem with the developed aircraft jet engine architecting tool will be provided and linked to the thesis objective and research questions in section 4.1. Then, the capabilities of the approach will be presented in section 4.2 and 4.3, respectively. Finally, the pyCycle software used in the approach for the engine cycle analysis is discussed in section 4.4. The main setup of the approach can be seen in Figure 4.1 and its parts will be explained in the sections of this chapter.

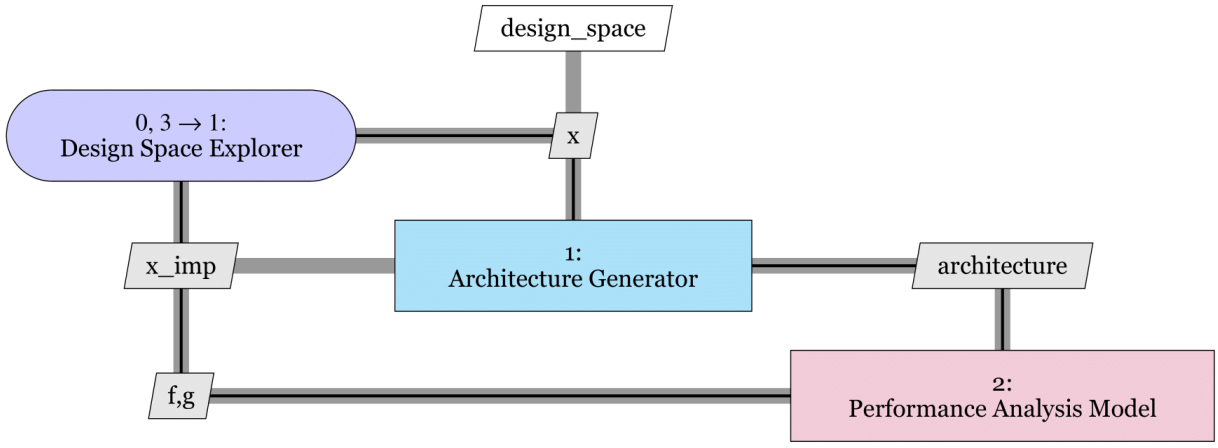


Figure 4.1: Extended Design Structure Matrix (XDSM) view of the coupling between the *Design Space Explorer* and the *Performance Analysis Model* for an architecture optimization problem [5].

The *Architecture Generator* step in Figure 4.1 uses the design space model to interpret the design vector x into an architecture instance, which is then used by the analysis model to estimate the architecture performance. Due to design variable hierarchy and the possibility of infeasible architectures, the design vector can be modified (imputed) to x_{imp} at this stage. It must be noted that the imputed design vector is used to generate the architecture. The imputed design vector x_{imp} can then be fed back to the *Design Space Explorer* for the optimizer to learn which design vectors result in valid architectures, but this is not necessarily required and depends on the optimization algorithm capabilities. Therefore, this potential ability could be analyzed when developing and testing existing and new optimization algorithms.

4.1 Overall Structure

Setting up and executing a system architecting problem such as for the aircraft jet engine consists of several steps. First, there must be some way to model the design space and use this model to define the design vector, objectives and constraints. Then, these need to be communicated to the exploration (i.e. optimization) algorithm. The exploration algorithm will then systematically vary the design vector and ask the performance analysis model to evaluate this design vector. To evaluate, the design vector is then first translated into a system architecture representation, making sure that any decision hierarchy is taken care of, and this architecture is then analyzed by the appropriate multidisciplinary analysis toolchain. Figure 4.1 provides a visualization of the process. The aircraft jet engine architecting tool will consist of two main components:

1. The *Engine Architecting Framework*: this component provides a way to describe the architectural choices, and from there defines a design vector, and functionality for translating a design vector into an engine architecture definition. This component operates at the level of the architecture design space and optimization problem, and is discussed in chapter 5.

2. The *Engine Architecture Evaluator*: this component provides a way to create and evaluate an engine architecture for specific operating conditions, by translating the architecture definition into a pyCycle problem. This component operates at the level of the individual architecture, and is discussed in chapter 6.

This separation is made so that it is clear that the evaluation of individual architectures is distinct from the definition of the architecture design space. Together, these components comprise a standalone engine architecting problem, but both the components can also be replaced for research purposes. In particular, this setup facilitates research into, and comparison of, methods for modeling system architecture design spaces and formulating architecture optimization problems. The complete code of the aircraft engine architecting tool has been published on GitHub as an open-source tool so that researchers can use it as benchmark for system architecting and optimization problems [39].

4.2 Mixed-Discrete Capabilities

As introduced in chapter 2, there are different types of design variables for MDO problems: continuous and discrete, of which the latter can again be subdivided in integer and categorical. The differences between these three types are as follows:

- Continuous: differentiable design variables used to size the system entities.
- Integer: non-differentiable design variables used to either size the entities of the system or to determine the system architecture.
- Categorical: non-differentiable design variables used to determine the system architecture.

Integer and categorical design variables also differ based on the principle of ordinality, stating that the order of the numbers has a meaning. This is the case for integer design variables, while it is not the case for categorical design variables.

In order to perform a systematic exploration of the design space and (possible) identification of improved system designs, these design variable types could be implemented in the MDO process. All three of these design variable types have effectively been implemented in the created system architecting approach leading to a dynamic system architecture generation and a larger design space. This was done by coupling each categorical design variable to a high-level decision in the system architecture. The implementation can be clarified with an example for the aircraft engine propulsion system: jet engines and piston engines could be encoded with the attribution '0' and '1' as categorical design variable, respectively; when the optimizer chooses a '0' for the categorical design variable, the subsequent aircraft architecture will then contain jet engines as propulsion system. By adding the relevant design variables to the design problem and therefore increasing the freedom of the optimizer, a whole system can thus be adapted and improved during the optimization process. With the addition of these relevant architecting decisions, the approach offers advantages to the system designers as they get a more detailed overview of all the different architecture possibilities and their respective performance. Furthermore, the effect of high-level system architecture decisions can be objectively evaluated at every single iteration of the optimization, while before subjective feedback was asked to system experts. This solves the system architecting optimization challenge discussed in section 2.3.

When combining all the different design decisions (i.e. categorical, integer and continuous), a design vector for the system architecture is created which is identified as x in Figure 4.1. This vector is created by the *Design Space Explorer* which tries to improve to performance of the system architectures within the design space. With the knowledge of the design space, the design vector is interpreted and converted to a system architecture by the *Architecture Generator*. Therefore, it is important in this process to model the design space in such a way that these high-level decisions are clearly identified in order to be able to automatically generate the system architecture.

4.2.1 Result Caching

It is possible that two original design vectors lead to an identical system architecture due to design variable hierarchy, discussed in the following subsection. This means that the *Performance Analysis Model* could be requested to evaluate the same exact architecture multiple times, which would of course produce the same results. This would decrease the efficiency and speed of the optimization process. Therefore, a mechanism called result caching is implemented in the approach which stores the results of all imputed design vectors. When this imputed design vector has to be evaluated an additional time, the mechanism catches this and simply returns the same results as the first evaluation, thus without evaluating the same architecture a second time.

4.2.2 Decision Hierarchy

By introducing categorical design variables, a complication might arise regarding the design vector which includes all the design variables. When the optimizer makes a categorical decision, other continuous or discrete design variables might become active or inactive. If, for example, the optimizer chooses a jet engine then the (potential) discrete design variable of number of pistons, used in an aircraft piston engine, becomes meaningless. The pistons number design variable has therefore become inactive. This principle of conditional activity of design variables due to other (categorical) design variables is known as decision hierarchy [5]. Effectively,

decision hierarchy also introduces some order of decisions that have to be taken during the decision process, and it is currently up to the designer of the system to determine this decision order.

The problem of decision hierarchy in optimization processes is that two seemingly different design vectors might result in the same system architecture: an aircraft with jet engines and 6 pistons, which is a possible design vector, is the same architecture as an aircraft with jet engines and 8 pistons. The resulting system architecture is the same, as a jet engine does not use pistons, but the design vector is clearly different. Therefore, both apparent (thus invalid) architectures and double (but valid) architectures should be removed to improve the efficiency of the optimization process.

In general, there are three main ways to cope with this decision hierarchy complication [40]. These are visualized in Figure 4.2.

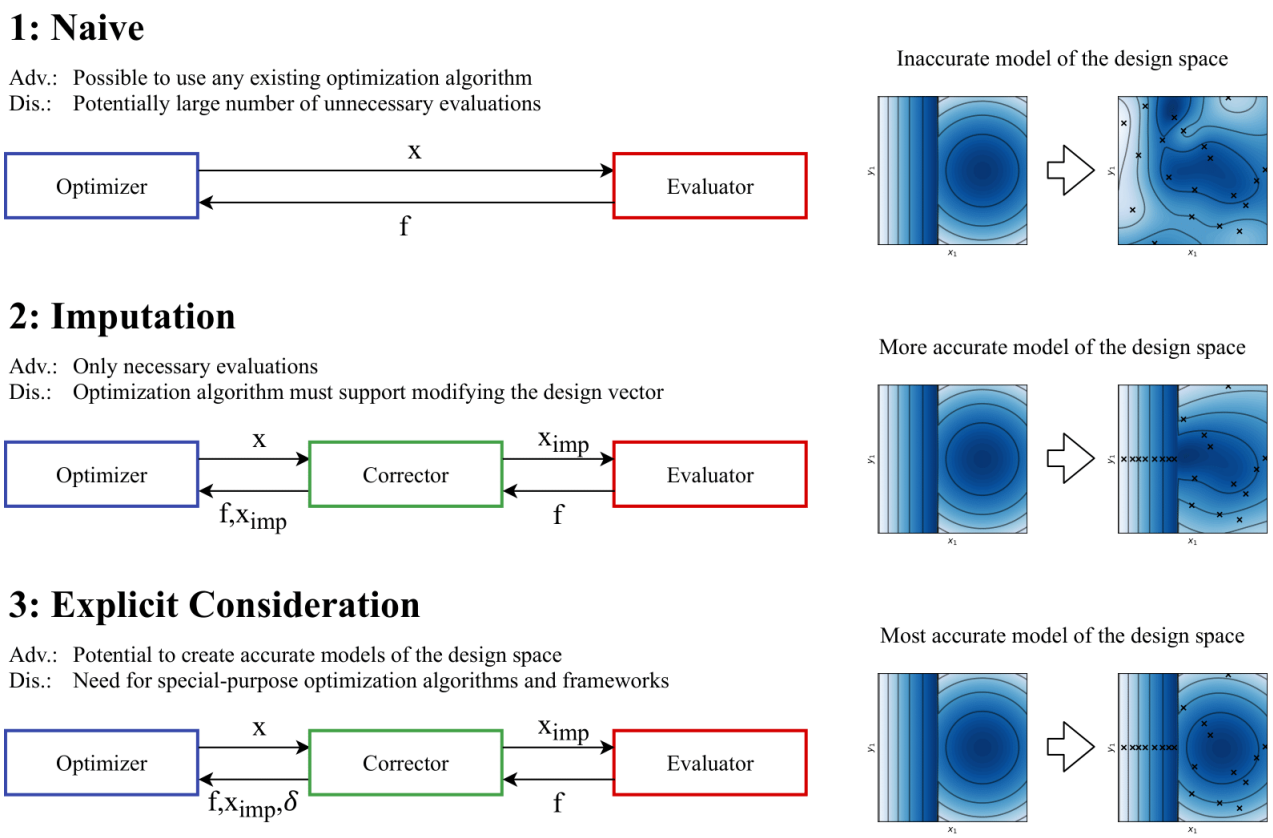


Figure 4.2: Different ways to cope with design variable hierarchy, where x and x_{imp} stands for the original and imputed design vector, respectively [41].

1. Ignoring decision hierarchy: the activity of the design variables is ignored by the optimizer, meaning that different design vectors might lead to the same system architectures which could confuse the optimizer [5]. This is not optimal regarding computation time as the same architecture might be evaluated multiple times.

2. Imputation approach: the original design vector is modified by assignment of a predefined value to the inactive design variables, leading to a new imputed design vector used to generate and evaluate the system architecture. This approach avoids that different design vectors lead to the same system architecture, thus improving the efficiency of the

optimization process. The information of design vector imputation can be fed back to the optimizer for it to learn which variables might become inactive, but this is not required.

3. Explicit consideration: a mathematical model of the design space is created by the optimizer in which the information on design variable activity is implemented. This requires optimization algorithms which can effectively cope with this kind of approach.

For the implemented system architecting approach, the imputation approach is chosen which solves the system architecting optimization challenge regarding decision hierarchy. This can be seen in Figure 4.1, as the *Architecture Generator* outputs an imputed design vector x_{imp} which is fed back to the optimizer. Referring back to the jet/piston engine, this means that a jet engine with 6 pistons will be imputed to a jet engine with 0 pistons, and only the imputed architecture will be evaluated. Benefits of this approach include that the imputation process is fairly simple to implement and that only valid design vectors are evaluated which improves the optimization efficiency. The capacity to inform the optimizer about design vector imputation is implemented in the thesis research system architecting approach. However, it will depend on the optimizer algorithm whether the optimizer is effectively informed about this process: genetic algorithms can deal with the information of imputation performed by the *Architecture Generator* whereas other optimization algorithms cannot. A possible future improvement might be to filter out the original and imputed vector to remove double architectures, and feed this information back to the optimizer. Therefore, the possibility arises to test different optimization algorithms and analyze how they deal with this information.

A side-effect of imputation is that a distinction between the *apparent* and *feasible* architecture design space is created: the former relates to all the possible architectures while the latter refers to only the feasible system architectures. An example from the aviation sector can be taken to clarify this distinction: when choosing whether to use a canard or an aft-fuselage empennage for pitch control of an aircraft while also selecting the aft-fuselage empennage layout (e.g. conventional, T-tail, V-tail), then the aft-fuselage horizontal tail layout choice will be irrelevant when a canard configuration is chosen. Thus, a canard aircraft with a conventional tail will be part of the apparent design space, but not of the feasible design space.

It must be noted that the model of the design space would be most accurate with the explicit consideration approach, but the drawback of needing specialized optimization algorithms offsets this. However, as introduced before, stakeholders including engineers and researchers can use the benchmark problem developed during the thesis research to test existing and new optimization algorithms.

4.3 Multi-Objective Capabilities

As optimization problems can be of multi-objective nature, the implemented system architecting approach should be able to cope with this kind of problems. Referring back to Figure 4.1, the architecture outputted by the *Architecture Generator* is inputted in the *Performance Analysis Model*. The *Performance Analysis Model* then evaluates the architecture for the specified objectives and constraints, defined by the system designer, and outputs them as f and g, respectively. The objectives and constraints are fed back to the *Design Space Explorer* which can then suggest a new design vector to try and improve the system architecture. It is up to the user of the approach to define the different objectives and constraints. Furthermore,

minimization or maximization of the objective should be specified, while for the constraint the limit should be identified. A multi-objective optimization algorithm and framework should then be chosen to meet the objective goals and satisfy the constraints. NSGA-II is chosen as standard optimization algorithm for the benchmark problem and will therefore be discussed later in this section. It must be noted that this algorithm can be changed by the user to test the performance of existing and new optimization algorithm. Next to that, the framework chosen for the thesis research is pymoo and is discussed in detail in Appendix A.

4.3.1 Extreme Barrier Approach

When evaluating an objective or constraint, it is possible that the result is invalid or that the evaluation of a system architecture does not converge, thus not returning a result at all. The inability to return a (valid) result is referred to as hidden constraints of the optimization problem. In that case, the result is assigned the value of $+\infty$ ($-\infty$) when minimization (maximization) is the goal of the optimization. This is called the extreme barrier approach and is used to guide the optimization algorithm towards valid results [42]. However, it must be noted that not all optimization algorithms can actually deal with this approach. Therefore, the ability to deal with the extreme barrier approach must be checked when choosing the optimization algorithm. Furthermore, the extreme barrier approach can also be applied to single-objective optimization problems and is thus not strictly related to multi-objective optimization.

4.3.2 Optimization Algorithm: NSGA-II

The Nondominated Sorting Genetic Algorithm II (NSGA-II) was developed as a successor to the genetic algorithm NSGA to tackle some of the challenges Multi-Objective Evolutionary Algorithms (MOEAs) face such as high computational complexity and the discardment of good solutions. To understand how NSGA-II works, the principles of Genetic Algorithms (GAs) and dominance will be discussed. Then, the overall procedure of NSGA-II will be explained. Finally, some of its most important advantages and disadvantages for the thesis research will be listed.

Genetic Algorithms

GAs are used in many optimization problems and can be compared to Charles Darwin's principle of natural selection. The algorithm starts with the *initialization*: an initial set of designs (*population*) during which it tries out a specified number of possible designs. This could be at random or with techniques such as latin hypercube sampling [43]. Next, an *evaluation* of the design points is carried out after which the performance (*fitness*) of each of the initial designs for the objective function(s) is compared to the other designs in the design space. Then, a *selection* is made within the design set: the designs with the best performance are used to create the new generation as they could pass on their "genetic material" which led to the better performance. This is done by creating variations through *crossover* & *mutation* on sets of parents (i.e. two design points) that are selected from the population, where there is a higher probability for selecting parents with high fitness. The final step is *replacement*: the old population is replaced by the new population. In this process, the complete population could be replaced or only parts of it based on fitness. The iterative process is continued until convergence of the population is achieved, i.e. the iterations do not lead to better performing designs [21].

The process is visualized in Figure 4.3. It must be noted that it is up to the user of the genetic algorithm to determine how many initial designs should be created, more designs in general lead to a better overview of performance over the design space but might increases computational time, and what the termination criteria should be for the algorithm.

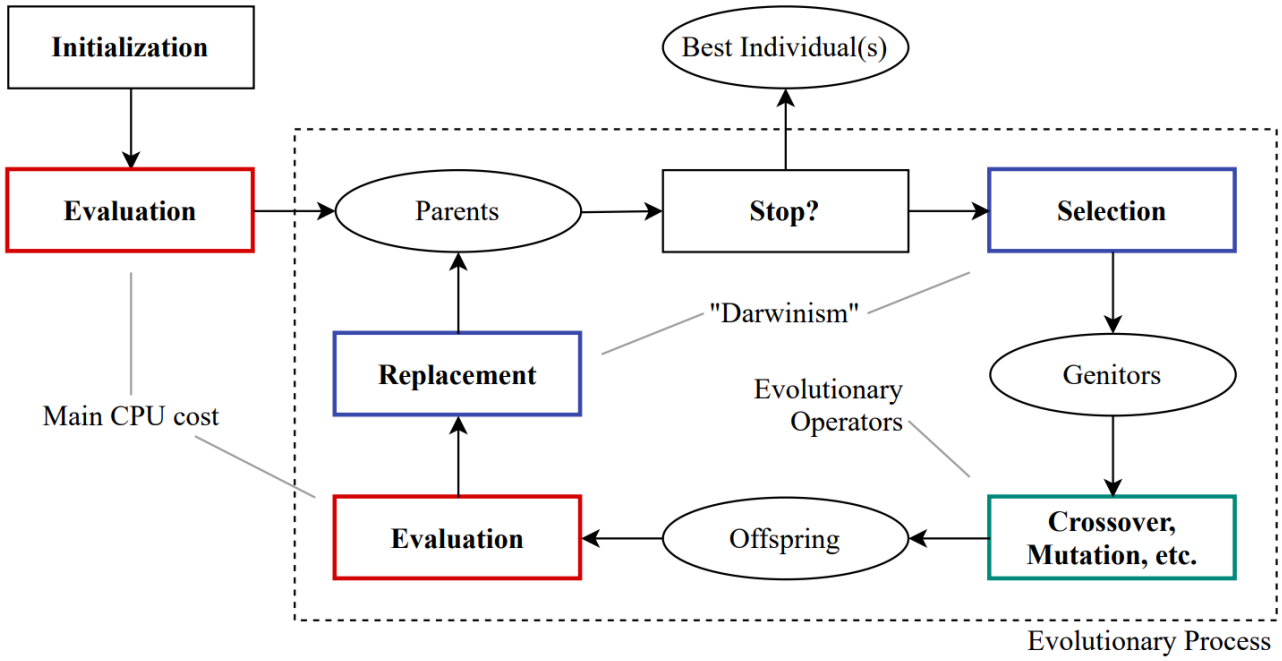


Figure 4.3: Overall procedure of a genetic algorithm [41].

Dominance

The principle of dominance is often used in multi-objective optimization problems. It is necessary to determine which design points are performing better than others, in order to create a sorting of designs. In case of two solutions x_1 and x_2, solution x_1 is said to dominate solution x_2 if [44]:

- x_1 does not perform worse than x_2 in all the objectives of the problem;
- x_1 performs better than x_2 in at least one of the objectives of the problem.

The set of dominating solutions, i.e. solutions that are not dominated, is called the nondominated set.

Overall Procedure

Based on the knowledge gained on genetic algorithms and nondominance, the overall procedure of NSGA-II can be explained [24]. A visualization of the non-dominated sorting process can be seen in Figure 4.4.

1. An initial set of designs for the optimization problem is created and evaluated with the objective function(s). This is called the Design Of Experiments (DOE).
2. The nondominated designs are placed in a list and together form the first nondominated front of the optimization problem, called the Pareto front. The nondominated designs are assigned a *rank* of 0, which offers a measure of how far certain designs are located from

the Pareto front. For the other designs, it is determined by how many designs they are dominated and this is referred to as their *domination count*.

3. In order to create new lists and fronts, the nondominated designs are analyzed. For each design that is dominated by the nondominated design, the domination count is reduced by one. In case this count hits 0, it is placed in a separate list with other dominated designs and assigned a rank of 1. Once this process is finished, the nondominated designs are removed from the set so that the designs in the new front are now considered nondominated designs.

4. The previous two steps are repeated until all front of the problem are found. It must be noted that the rank of the designs increases by 1 with every new front found.

5. The best performing half of the designs advances to the next stage. This is done by first including the highest nondominated lists, i.e. the lists of which the members are the least dominated, until a full list is too large to fill up the remaining spots. In that case, the crowding density of a design within one rank is calculated which is the average distance of that design to the closest higher and lower design result for each objective. The less dense designs are then selected to fill up the remaining spots for the next optimization iteration.

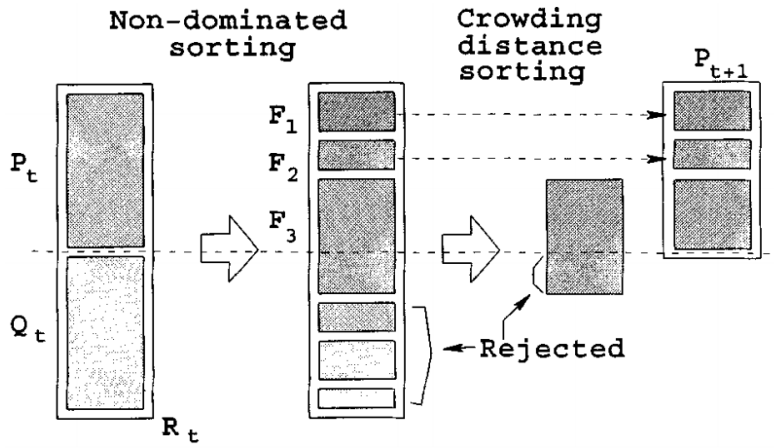


Figure 4.4: Non-dominated sorting procedure of the NSGA-II genetic algorithm [24].

Advantages

NSGA-II offers several advantages to the optimization benchmark problem of aircraft jet engines used in this thesis:

- It is a global optimization method, meaning that it is designed to find the global optimum and to avoid getting trapped in optimizing local optima. The reason for this is that it starts off with a population of design points, rather than starting from one design point and moving from there. The optimizer will be able to generate and analyze multiple specific engine architectures leading to a better overview of the design space.

- NSGA-II is a non-gradient-based method which is required as there are discrete variables used in the engine architecting problems, meaning that the gradient cannot be calculated. Therefore, gradient-based methods were not an option.

- It is a multi-objective genetic algorithms. Therefore, it can deal with the different objectives involved in the design of aircraft jet engines.
- The extreme barrier approach can be effectively implemented with NSGA-II meaning that information on infeasible architectures can be taken into account during the optimization process.

It must be noted that these are not the only advantages of NSGA-II, but these were deemed the most important for the benchmark problem of the thesis research.

Disadvantages

There are also some noteworthy disadvantages of NSGA-II for the thesis research:

- It is computationally expensive due to the high number of function evaluations that have to be performed. In order for the optimizer to understand which designs have a better performance, a relatively high population size and number of iterations is necessary. As a rule of thumb, the population size should be five times the number of design variables, leading to a population size of approximately 200 designs in the case of the aircraft jet engine benchmark problem. With an average analysis computation time of 2 minutes per design, only the DOE can already take more than 6 hours to complete.
- Tuning the parameters of for example population size and number of iterations can have an effect on the results of the NSGA-II optimization. Therefore, sensitivity studies should be performed to ensure that the results are trustworthy. It must be noted that the disadvantage of parameter tuning holds for other (types of) optimization algorithms as well and is thus not exclusive to NSGA-II.

Again, these are not all the disadvantages of NSGA-II, but merely the ones that were deemed most important for the thesis research.

4.4 Engine Cycle Analysis: pyCycle

PyCycle is an open-source engine cycle analysis framework and latest development in a long line of engine analysis platforms, allowing modular definition of engine cycles and easy integration in multidisciplinary analysis and optimization toolchains [45]. Additionally, the analysis code provides analytical derivatives for all parameters, greatly accelerating the design and optimization convergence speed for a single given engine cycle.

The pyCycle code consists of four main components, as can be seen in Figure 4.5 [45]:

- *Cycle*: comprises all the thermodynamic equations needed to create the thermodynamic cycle of the engine, and is therefore the central part of the pyCycle software. It contains the different components of the engine, such as the compressor(s) and turbine(s), which can be connected to form an entire engine.
- *Balance*: contains the design requirements and physical conservation equations of the problem necessary to create a valid engine system, by introducing implicit state variables in residual equations on the complete system level. They must be specified specifically for each developed engine cycle. The balancers implemented during the thesis research

will be explained in detail in chapter 6. In general, the balancer block can be split up into two categories: cycle matchers $\Re_c$ which make sure that conservation equations are satisfied, and design rules $\Re_d$ which ensure that design requirements are complied with. A general equation for a residual equation is:

$$\Re = X - X_{target} = 0 \tag{4.1}$$

- *Solver*: tries to converge the complete system of equations of the engine cycle with the feedback variables, including the residual equations of the *Balance* block if present.
- *Optimizer*: varies the design variables in order to minimize (or maximize) the problem objective and satisfy its constraints.

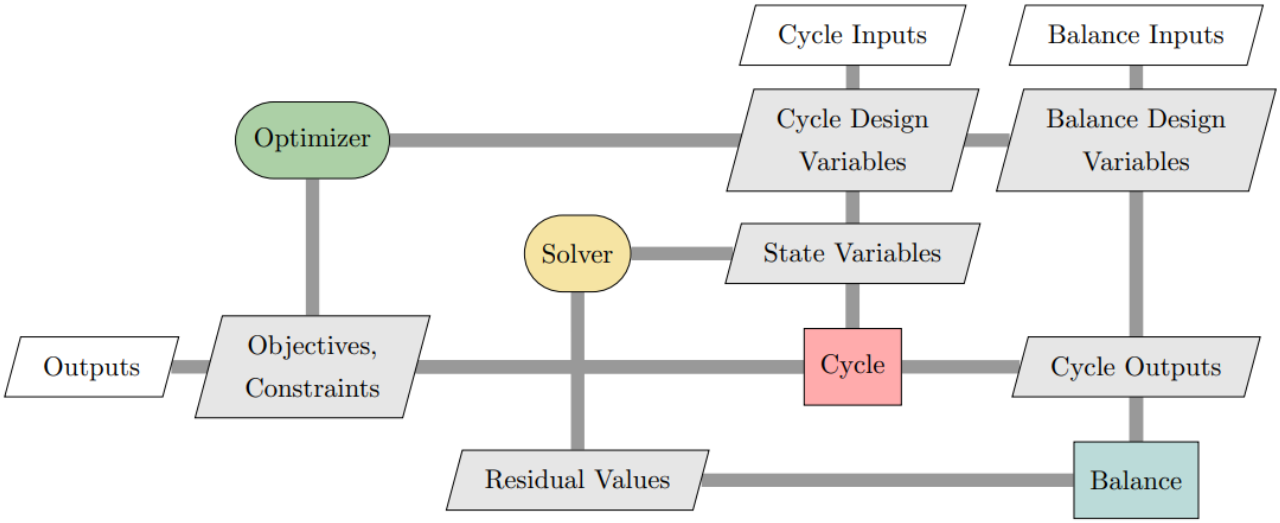


Figure 4.5: Overall setup of the pyCycle engine cycle analysis software [45].

The pyCycle software contains a set of predefined aircraft engine components containing the different thermodynamic variables and equations to model them. As an example, the structure of the compressor component is shown in Figure 4.6. This shows that large engine components are subdivided into smaller blocks to compute the thermodynamic variables of all the different components in the engine cycle. Next to that, subdivision also leads to less derivatives having to be implemented as the MDO framework takes care of the coupled derivatives. An overview of all the different engine components and their inputs will be provided in chapter 6.

Based on this information, it becomes clear that pyCycle plays a vital role in the research of the thesis and the solution of its research questions, as it is necessary to model the different aircraft engine architectures by providing the components required to form a complete engine system. PyCycle employs a modular approach, meaning that the engine components are presented as building blocks with which the engine thermodynamic cycle is built up, improving flexibility. Each engine component contains its own thermodynamic equations and partial derivatives, which are all combined to a complete engine cycle by linking the components. Therefore, a wide range of distinct aircraft engine architectures can be generated through the implementation of categorical design variables. This is important to demonstrate that the system architecting approach can effectively be implemented in complex real-life design problems. Furthermore, the

inputs of the various components determines which continuous and/or discrete design variables can be included in the benchmark problem to size these components. Therefore, the included design variables ultimately decide which distinct engine architectures can be formed and which limitations the system architecting approach has. Finally, the accuracy of the thermodynamic equations and the solver will have an impact on the validity of the engine performance results. It must be noted that other elements such as compressor maps and analysis software also have an influence on the result validity.

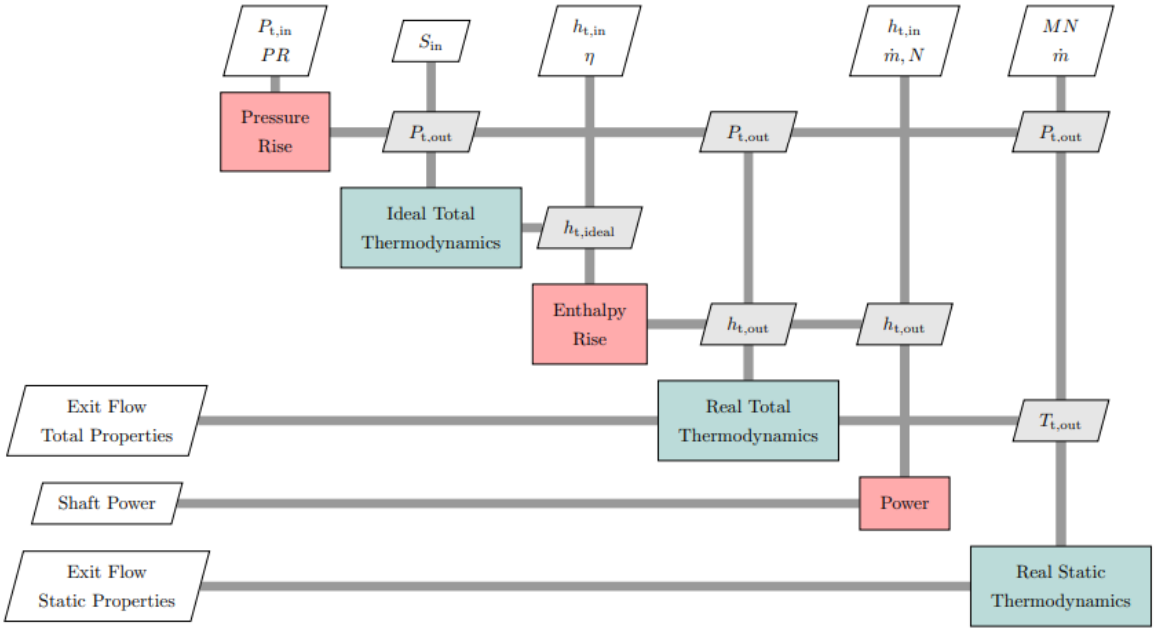


Figure 4.6: Setup of the compressor component in the pyCycle software [45].

In order to combine the four different blocks seen in Figure 4.5 into one engine thermodynamic cycle analysis and optimization software package, an MDO and optimization framework was required [45]. For these purpose, OpenMDAO was selected by the pyCycle developers and pymoo was selected by the author, respectively. These two frameworks are explained in Appendix A in case the reader wants further explanation.

Summary: Research Question Answers

RQ2 How does the approach address the nature of architecting decisions?
RQ2.3 What is the influence of the discrete architecting decision on the design vector:

> The implementation of discrete, and in particular categorical, design variables leads to decision hierarchy: design variable(s) might become (in)active based on the decision taken with the discrete design variables. The result is that two different design vectors could result in the same generated system architecture, confusing the optimizer and reducing the optimization efficiency. To cope with this, an imputation approach was used which modifies the original design vector by assigning a predefined value to inactive design variables. The imputed design vector is then used to generate and assess the system architecture.

Chapter 5
Engine Architecting Framework

The purpose of the *Engine Architecting Framework* is to define the aircraft engine architecting problem and then use the *Engine Architecture Evaluator* to find the best engine architecture. The implementation consists of two parts: the interface for defining the system architecting problem, and the *Architecture Generator* that translates design vectors as generated by the optimization algorithm to *EngineArchitecture* instances that can be evaluated by the *Engine Architecture Evaluator*. This process is visualized in Figure 5.1, in which the *Engine Architecture Evaluator* block will be explained in chapter 6.

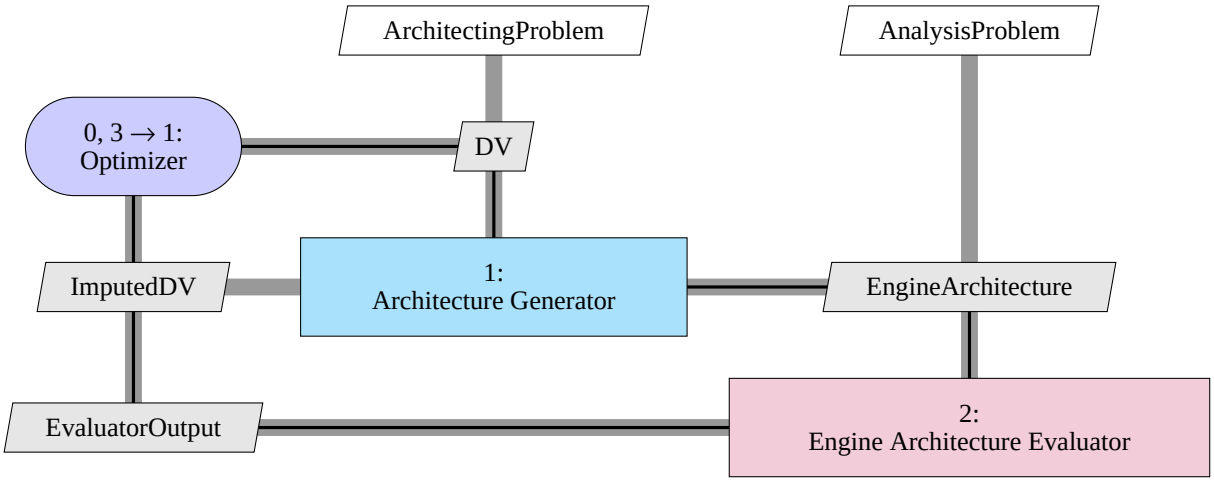


Figure 5.1: XDSM of the *Engine Architecting Framework* component of the tool: it defines an architecture problem, constructs the design vector from this definition, and translates design vectors to an engine architecture definition to be evaluated. The *Engine Architecture Evaluator* block can be replaced by Figure 6.1 to get the XDSM of the entire tool.

In this chapter, the definition of the architecting problem will first be discussed in section 5.1. After that, the generation of the aircraft engine architectures will be analyzed in section 5.2.

5.1 Architecting Problem Definition

The architecting problem defines the architecture choices and design metrics. Operating conditions are also defined here, so that the engine architecture evaluation component can be called directly without needing any additional interaction. The main use case of the architecting problem definition is to formally define an optimization problem for testing optimization algorithms. The engine architecting framework provides interfaces to the Python multi-objective optimization framework pymoo to help with testing algorithms [46].

5.1.1 Choice Definition

Choices semantically define architecting decisions to include in the architecting problem. One choice can map to one or more discrete or continuous design variables, and therefore adding choices increases the size of the architecture design space. If no choices are defined for the architecting problem, the resulting architecture represents a simple turbojet: one shaft, compressor, turbine, and burner. This simple engine architecture produces all of the required thrust. Choices can be included as free (design variable) or as fixed (design parameter): the latter option might be used to modify this turbojet default.

The implemented architecting choices and their design variable mappings can be found in the list below, in order of execution. Regarding the execution order, only the first three choices (Fan, CRTF and Shaft) should be executed first in a fixed order as some architecting choices are dependent on decisions made in previous architecting choices; the other choices can be interchanged as they are not dependent on each other but are on the first three choices. This clearly shows that decision hierarchy is present which is solved by sorting the design decisions, i.e. applying ordinality. In case the order of execution is changed without adjusting the source code accordingly, an incorrect engine architecture might follow from the design vector. For example: when the CRTF choice is executed before the fan choice, the counter-rotating fan will always be inactive (even when the design vector marks it as present) as it is dependent on the presence of a fan, which is decided after the CRTF choice with the new order. It must be noted that manually deciding on the execution order is possible in the tool as the total number of decisions is fairly limited. When the number of decisions becomes very large, it would be more advantageous to automate the ordering of the decisions in the architecture generation process. This could be done by tracking dependencies in design decisions on other decisions and creating a execution order based on that information, however this should be investigated further.

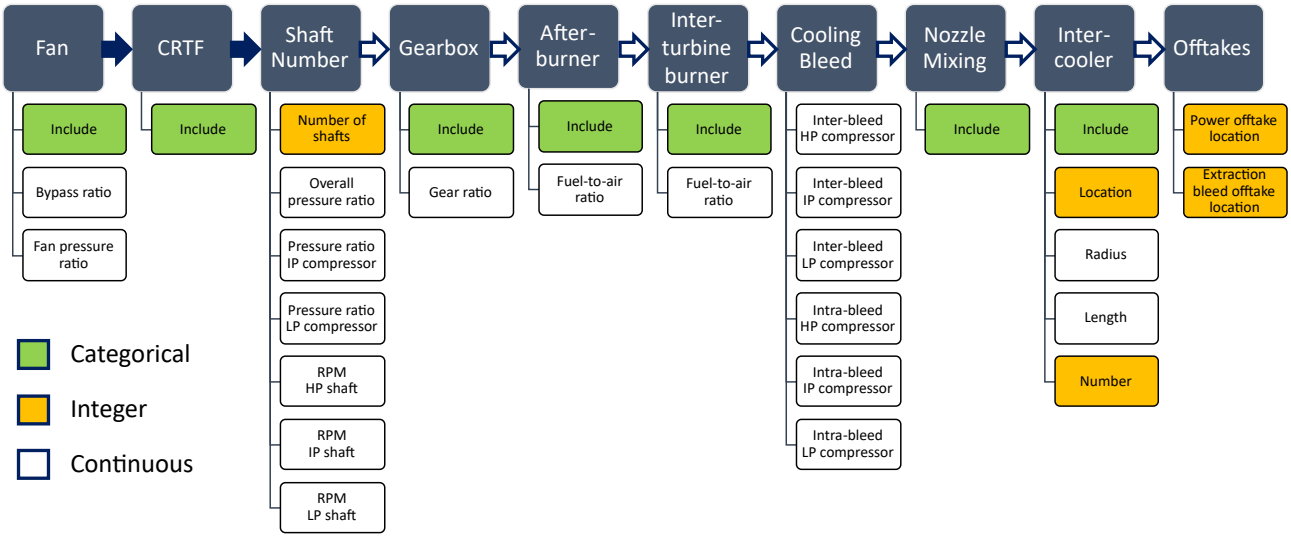
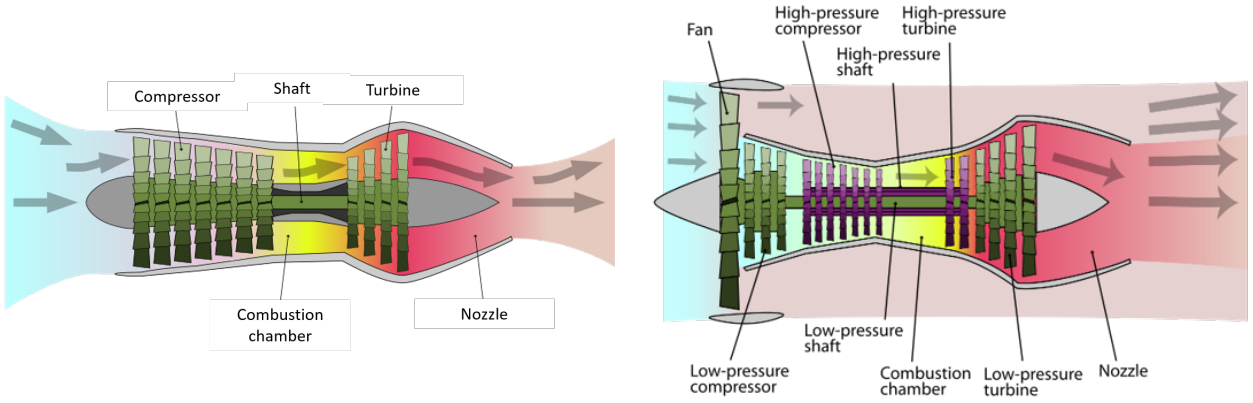


Figure 5.2: Visual representation of the engine architecting decisions determining the final engine architecture, including the different design variables which can be categorical, integer or continuous in nature. The arrows indicate the order of execution which is important as some architecting choices are dependent on the decisions of other architecting choices. Filled arrows indicate a fixed order of execution whereas unfilled arrows indicate an interchangeable order of execution.

The following subsections will discuss which architecting choice can be taken by the optimizer, while figures will be used to show which aircraft jet engine architectures could result from those choices and their influence on the engine architecture. A general overview of all the choices and their respective design variables can be seen in Figure 5.2. Furthermore, a complete overview of the design variables of the architectural choices including bounds can be found in Appendix B. In total, the design space includes 43 design variables: 7 categorical, 5 integer and 31 continuous. This seems less in the figure, however each white block in the cooling bleed vertical line actually consists of 3 design variables to regulate the amount of cooling bleed flowing to its turbine targets. Only the discrete variables already account for approximately 2.6 million apparent engine design points, which was found by taking the product of the number of options for all the different discrete design variables. The total number of distinct engine architectures that can be generated equals 91.

1. Fan Choice

The first architecting choice is the inclusion of a fan at the front of the engine. In case a fan is not included, the resulting engine architecture is a turbojet. Otherwise, the architecture is identified as a turbofan. For a turbofan architecture, two additional design variables are activated: the BPR, with bounds of 2 to 12.5, and the Fan Pressure Ratio (FPR), with bounds of 1.1 to 1.8. Figure 5.3 shows what the two different resulting engine architectures are that can arise from this *Fan Choice*: a turbojet or a turbofan.

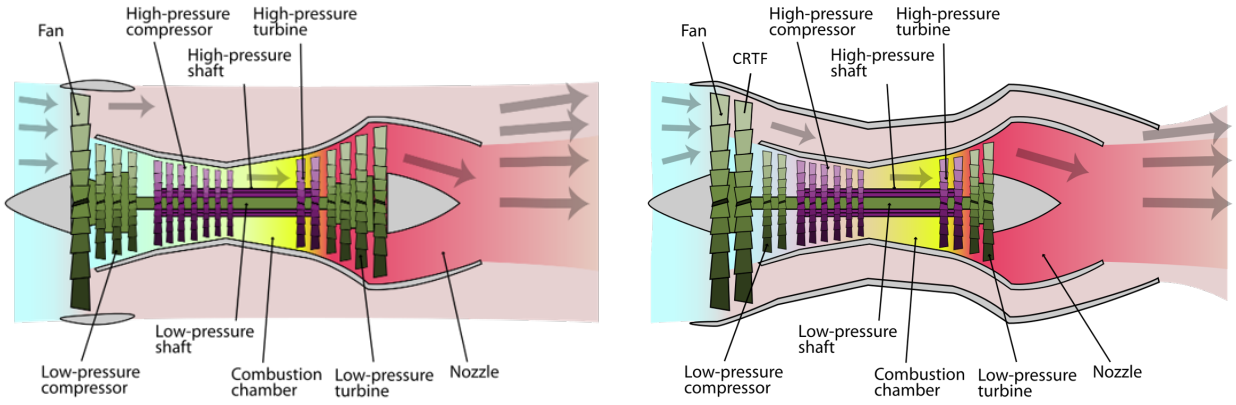


(a) Turbojet [47]. (b) Turbofan [30].

Figure 5.3: Resulting engine architectures from the *Fan Choice*.

2. CRTF Choice

If the result of the first architecting choice is to include a fan, the option arises to include a second fan at the front of the engine which rotates at the same speed but the opposite direction as the main fan. This concept is called a Counter-Rotating Turbofan (CRTF) and could result in a decrease of 5 EPNdB in noise, at the cost of 10% weight and 5.74% TSFC increase at cruise compared to an Ultra-High Bypass Ratio (UHBR) engine according to the COBRA project results of the European Commission [48]. Furthermore, the efficiency of the fan in front of the main fan was found to be 5% higher than the main fan. However, in case the fan efficiency is already very high, the efficiency of the counter-rotating fan was set to a maximum of 95%. Figure 5.4 shows what the two different resulting engine architectures are that can arise from this *CRTF Choice*: a conventional or a counter-rotating turbofan.

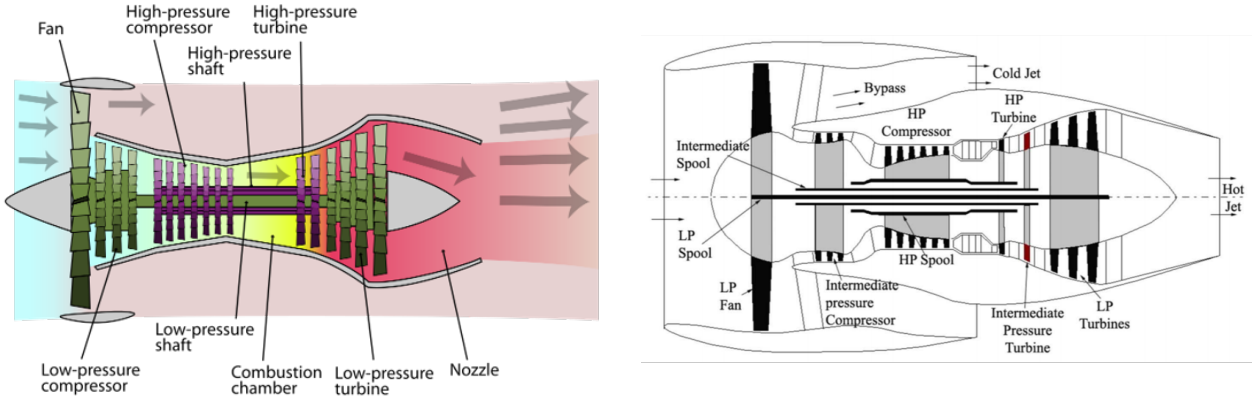


(a) Conventional turbofan [30]. (b) Counter-rotating turbofan [30].

Figure 5.4: Resulting engine architectures from the *CRTF Choice*.

3. Shaft Number Choice

Next, the number of shafts is decided: the engine architecture will have either one, two or three core shafts. These can be referred to as the low-, intermediate- and high-pressure shafts. Note that the fan shaft, which will be explained in the *Gearbox Choice*, is not included in the *Shaft Number Choice*. The advantage of multiple shafts is that the compressors and turbines can rotate at optimal speeds leading to a better engine performance. However, it also results in an increase in weight and complexity of the system. For each shaft, the rotational speed is a design variable between 1,000 and 20,000 RPM. Other design variables include the OPR of the complete engine, between 1.1 and 60, as well as the percentage of the OPR that each compressor generates, ranging from 10% to 90%. The OPR percentage does not have to be specified for the high-pressure compressor as this can be derived from the percentages of the other shafts. The OPR design variable is part of the shaft number choice as the maximum OPR is constrained by the number of compressors each having their own maximum pressure ratio. It must be noted that two always-active constraints were introduced: the maximum pressure ratio of each individual compressor is set to 15, while the sum of the OPR percentages of the LPC and IPC is set to maximum 90% to allow at least 10% of the OPR to the HPC. Figure 5.5 shows what two of the three different resulting engine architectures are that can arise from this *Shaft Number Choice*: a 2- or 3-shaft jet engine.

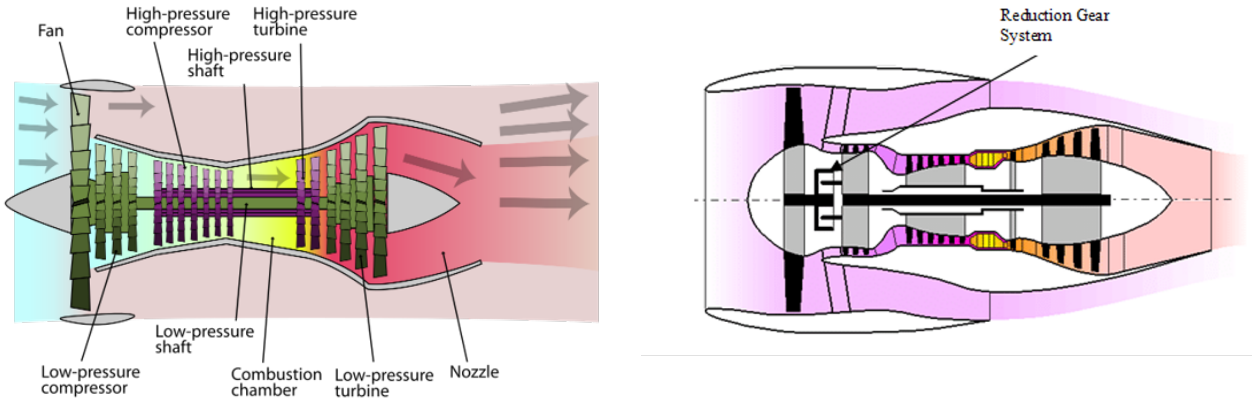


(a) 2-shaft jet engine [30]. (b) 3-shaft jet engine [49].

Figure 5.5: Resulting engine architectures from the *Shaft Number Choice*.

4. Gearbox Choice

To enable the fan to rotate at an optimal speed, a gearbox can be inserted between the low-pressure shaft and the fan shaft. This architecture is referred to as the Geared Turbofan (GTF), which is only active when a fan has been selected in the *Fan Choice*. In case a gearbox is included, the gear ratio is a design variable between 1 and 5. The gearbox is implemented by adding a fan shaft which contains the engine fan and connecting it to the low-pressure shaft of the engine. The rotational speed of this fan shaft is then the rotational speed of the low-pressure shaft divided by the gear ratio. The GTF option increases efficiency and decreases noise, due to the slower rotation speed of the fan, however comes at the price of possibly higher weight. Figure 5.6 shows what the two different resulting engine architectures are that can arise from this *Gearbox Choice*: a conventional or geared turbofan.

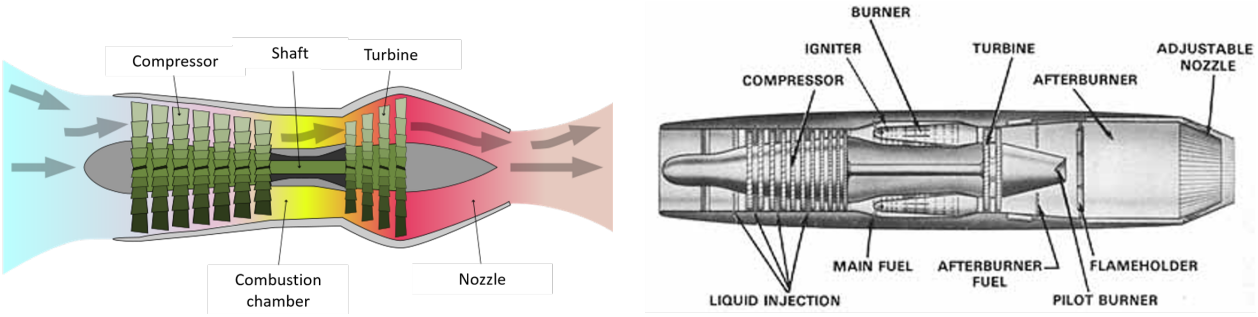


(a) Conventional turbofan [30]. (b) Geared turbofan [50].

Figure 5.6: Resulting engine architectures from the *Gearbox Choice*.

5. Afterburner Choice

An afterburner can be implemented to increase the thrust force of the engine and reach supersonic velocities. This is done by injecting additional fuel in between the most aft turbine and the core nozzle of the engine. Therefore, when selecting an afterburner, the Fuel-to-Air Ratio (FAR) of the afterburner becomes a design variable, which has bounds of 0 to 0.05. The drawback of the technology is that the engine fuel consumption is considerably increased due to incomplete combustion of the afterburner fuel. This is one of the reasons why afterburners are currently only used in military aviation [7]. Furthermore, afterburning is only active when no fan is selected in the *Fan Choice*.



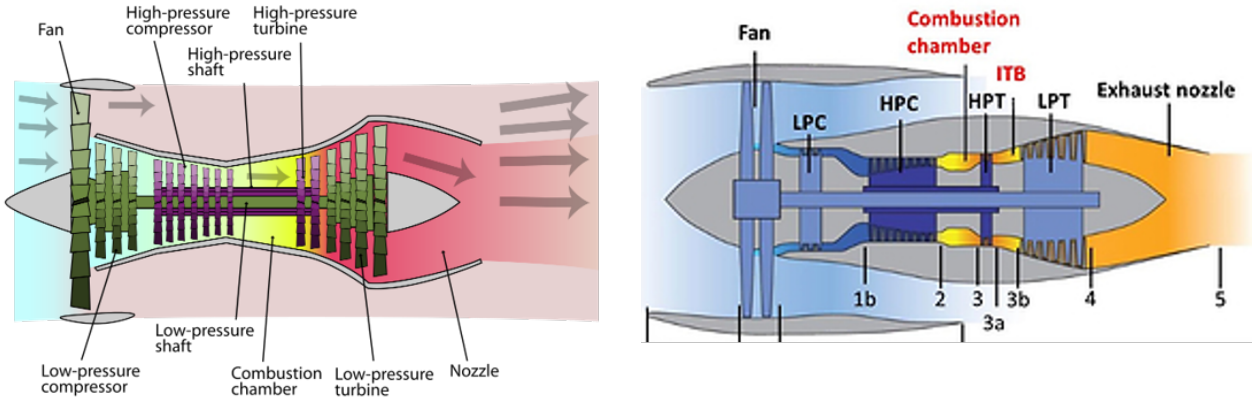
(a) Conventional turbojet [47]. (b) Afterburning turbojet [51].

Figure 5.7: Resulting engine architectures from the *Afterburner Choice*.

Figure 5.7 shows what the two different resulting engine architectures are that can arise from this *Afterburner Choice*: a conventional turbojet or an afterburning turbojet.

6. ITB Choice

It is possible to include a second combustion chamber in the engine architecture, called an Inter-Turbine Burner (ITB). As the name suggest, the ITB is located between two turbines aft of the main combustion chamber. As the combustion process of the ITB occurs at high pressure, its efficiency is higher compared to for example an afterburner, reducing the fuel consumption. In addition, an ITB achieves lower NOx emissions and performs better in off-design conditions compared to a conventional turbofan. However, disadvantages include an increased weight and complexity of the system [38]. When the ITB is selected, its FAR is a design variable with the same bounds as in the *Afterburner Choice*. Figure 5.8 shows what the two different resulting engine architectures are that can arise from this *ITB Choice*: a conventional or inter-turbine burning jet engine.



(a) Conventional jet engine [30].

(b) Inter-turbine burning jet engine [52].

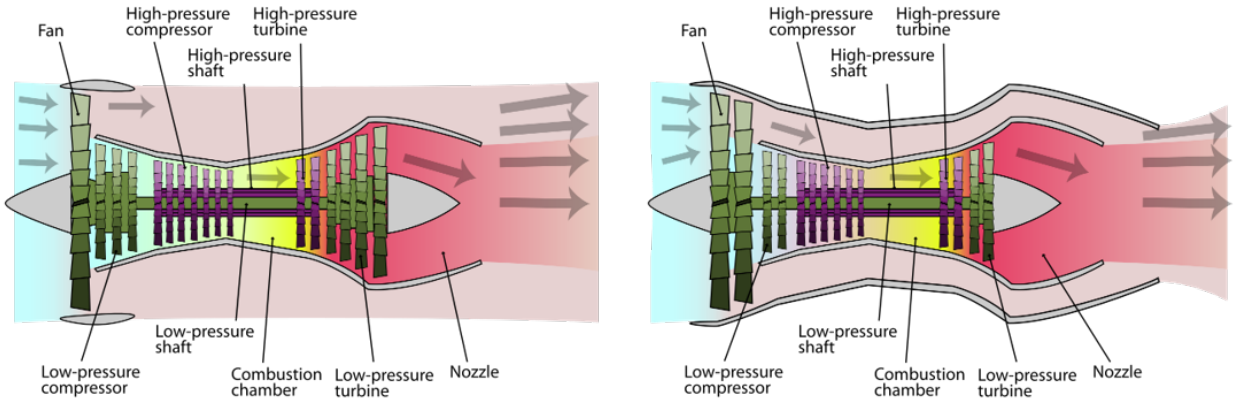
Figure 5.8: Resulting engine architectures from the *ITB Choice*.

7. Cooling Bleed Choice

In order to cool the turbine or to regulate the axial velocity of the compressor gases, air can be bled from the compressors through bleed valves. This air is referred to as cooling bleed and is a design choice, opposed to the extraction bleed which will be explained in the *Offtakes Choice*. For the engine architecting problem, the amount of cooling bleed and its targets (i.e. defined turbines) need to be specified for each individual compressor. The amount of cooling bleed a target receives is encoded as a percentage of the total cooling bleed of a certain compressor. To limit the effect of cooling bleed on the overall engine performance, it is advised to bleed air at early stages in the compressor(s) [33]. The implementation of cooling bleed is split up into intra-bleed and inter-bleed: the former defines a situation where air is bled from within a compressor, whereas for the latter it is bled from in between two compressors. Furthermore, each cooling bleed element can bleed a maximum of 10% of the incoming flow of the component. Next to that, an always-active constraint was set: the sum of the cooling bleed target percentages is maximum 100%.

8. Nozzle Mixing Choice

For turbofans, a choice can be made between a separate or mixed flow nozzle. Therefore, the choice is only active when a fan has been opted for in the *Fan Choice*. In the mixed flow nozzle, the flow of the core and the bypass is mixed at the end of the engine and leaves the engine through one joint nozzle, whereas this is not the case for a separate flow nozzle. A mixed nozzle results in a slightly higher efficiency, but also higher weight and length of the engine, which is why the separate flow nozzle is currently more often used in commercial aviation [33]. Figure 5.9 shows what the two different resulting engine architectures are that can arise from this *Nozzle Mixing Choice*: a separate turbofan or a mixed turbofan.



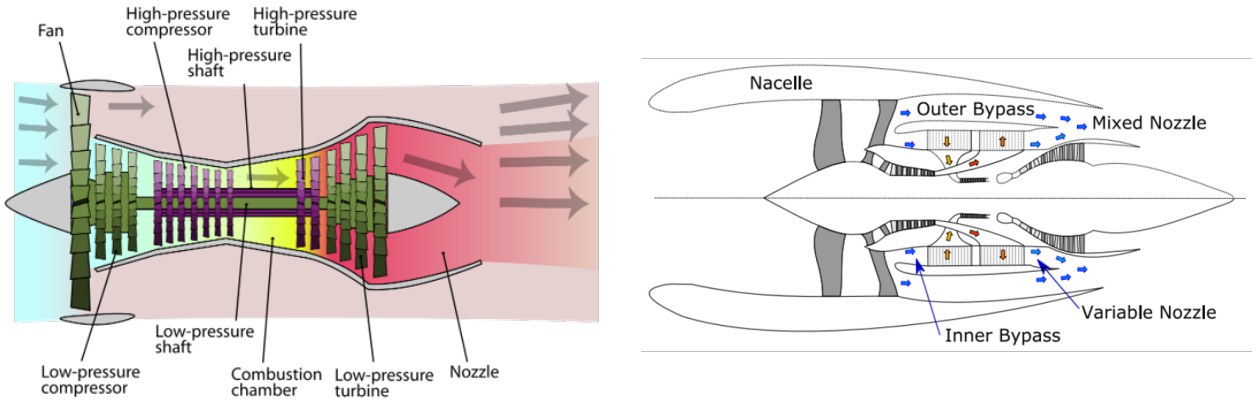
(a) Separate nozzle turbofan [30].

(b) Mixed nozzle turbofan [53].

Figure 5.9: Resulting engine architectures from the *Nozzle Mixing Choice*.

9. Intercooler Choice

On stationary and marine gas turbines, a heat rejection method is sometimes implemented, which is called intercooling. During this process, heat is removed from the thermodynamic cycle using a Heat Exchanger (HEX) resulting in lower fuel consumption and emissions, at the cost of higher engine volume and vibration issues [34]. In the HEX, the core air is the hot fluid while the bypass air is the cold fluid, which is why the intercooler is only active when a fan is implemented in the engine architecture. The HEX component will be further explained in chapter 6. In the architecting problem, an intercooler can be added by specifying its location in the engine and its geometry (radius, length and number of pipes). Regarding the HEX length, an always-active constraint of maximum 50% of the engine radius was set based on the research of Zhao [34]. The heat exchanger will only be used for intercooling and not to produce useful work in the thermodynamic cycle [35, 36]. Furthermore, it is up to the designer of the system to determine the appropriate overall heat transfer coefficient of the HEX. However, this parameter was set to 400 W/m^2K by default as pressurized air from the compressor(s) is air-cooled by the HEX [54]. Figure 5.10 shows what the two different resulting engine architectures are that can arise from this *Intercooler Choice*: a conventional turbofan or an intercooled turbofan.



(a) Conventional turbofan [30].

(b) Intercooled turbofan [34].

Figure 5.10: Resulting engine architectures from the *Intercooler Choice*.

10. Offtakes Choice

Power and extraction bleed offtakes are specified as part of the engine requirements, and are therefore always present in an engine architecture. Power offtakes are satisfied by extracting electrical power from one of the engine shafts in order to power different systems onboard the aircraft. For extraction bleed, air is bled from one of the compressors to support for example the Environment Control System (ECS) or anti-icing systems of the aircraft [33, 55, 56]. The offtake location (both for power and extraction bleed) can be specified with the design variables.

5.1.2 Objective & Constraint Selection

Objectives are metrics to be either maximized or minimized; constraints are metrics where a lower or upper limit is placed on the value. Available metrics to be used as objectives or constrains include the results from the aircraft jet engine disciplines which will be discussed in chapter 6: TSFC, weight, length, diameter, NOx emissions and noise. Additionally, the jet nozzle Mach number can also be set as objective and/or constraint. All of these are minimized if selected as objectives. In addition to these metrics, a number of constraints will always be present in the architecting problem to ensure the feasibility of the generated engine architectures, including counter-rotating fan efficiency and intercooler length as compared to the overall engine radius.

5.2 Architecture Generator

Once the architecting problem has been defined, the optimization loop can be started and the optimizer will start generating design vectors to be evaluated. This can be seen in Figure 5.1. Each design vector first needs to be converted into an architecture definition before it can be evaluated by the *Engine Architecture Evaluator*. The *Architecture Generator* converts the design vector into an *EngineArchitecture* instance using the selected architecture choices, in which each choice implements the logic to construct its corresponding part of the architecture. Furthermore, the *Architecture Generator* also contains the imputation and result caching mechanisms which were discussed in chapter 4.

Summary: Research Question Answers

RQ2 How does the approach address the nature of architecting decisions?
RQ2.1 Which continuous and discrete architecting decisions can be taken?

> Figure 5.2 gives a clear overview of the different architecting decisions that can be taken in the aircraft jet engine architecting tool. Furthermore, the type of each design variable is also indicated in the figure. Next to that, Appendix B gives a complete overview of the design variables for the architecting choices including variable type and bounds.

RQ2.2 What is the impact of the discrete architecting decisions on the aircraft engine components and connections?

> The discrete architecting decisions can be subdivided into categorical and integer decisions. Categorical decisions determine whether or not a component will be included in the engine architecture. Integer decisions determine how many instances of that component type will be included in the engine architecture and what their position in the engine will be. The connections between the components will depend on the components present in the engine architecture.

RQ2.5 What is the size of the design space?

> In total, the design space includes 43 design variables: 7 categorical, 5 integer and 31 continuous. Only the discrete variables already account for approximately 2.6 million apparent engine design points, which was found by taking the product of the number of options for all the different discrete design variables. The total number of distinct engine architectures that can be generated equals 91.

Chapter 6

Engine Architecture Evaluator

The purpose of the *Engine Architecture Evaluator* is to perform design and simulation of a given engine architecture. To do this, it uses two frameworks: the engine cycle analysis framework pyCycle and the MDO framework OpenMDAO explained in chapter 4 and Appendix A, respectively. It does this by building an OpenMDAO problem using pyCycle and any additional disciplines that might be needed to evaluate the requested metrics. This section introduces the engine architecture format that is used to define architectures, the setup of the thermodynamic cycle analysis using pyCycle, and the aircraft engine discipline evaluator. The XDSM of the *Engine Architecture Evaluator* can be seen in Figure 6.1.

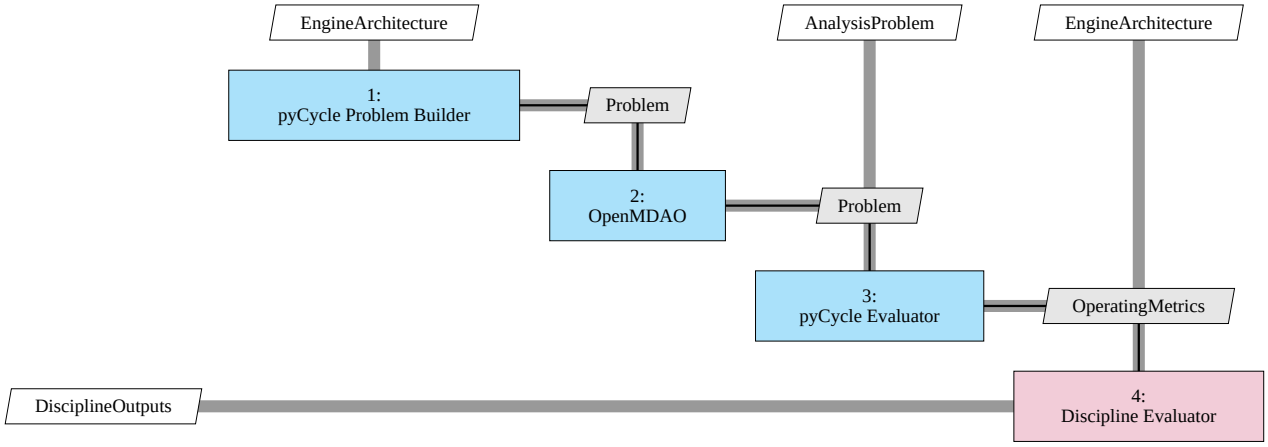


Figure 6.1: XDSM of the *Engine Architecture Evaluator* component of the tool: it defines the architecture definition format, translates architecture definitions into a pyCycle problem, and executes OpenMDAO to extract the desired metrics (objectives and constraints).

First, the definition of the engine architecture components will be analyzed in section 6.1. After that, the thermodynamic cycle analysis will be discussed in section 6.2. Finally, an overview and explanation of the disciplines involved in the aircraft jet engine architecting tool will be provided in section 6.3.

6.1 Engine Architecture Definition

The engine architecture is defined according to the class diagram shown in Figure 6.2. The architecture is defined as instances of Python objects, and it is trivial to develop an additional interface so that these objects are instantiated from the definition in some text file format (e.g. XML, JSON) so that the architecture evaluation tool could be implemented as a standalone tool. The engine architecture is defined by architecture elements that are connected to each

other. The kind of elements defined in this diagram (e.g. inlet, compressor, burner, turbine) are common across most thermodynamic cycle analysis frameworks, and are taken directly from the pyCycle library itself [45]. An overview of the different aircraft engine components and their design variables can be seen in Figure 6.2. The only addition of the author is the Heat Exchanger (HEX) component, to model architectures with an intercooler. In order to determine net thrust, each architecture should have one inlet and at least one nozzle, which are the elements connected to the freestream flow.

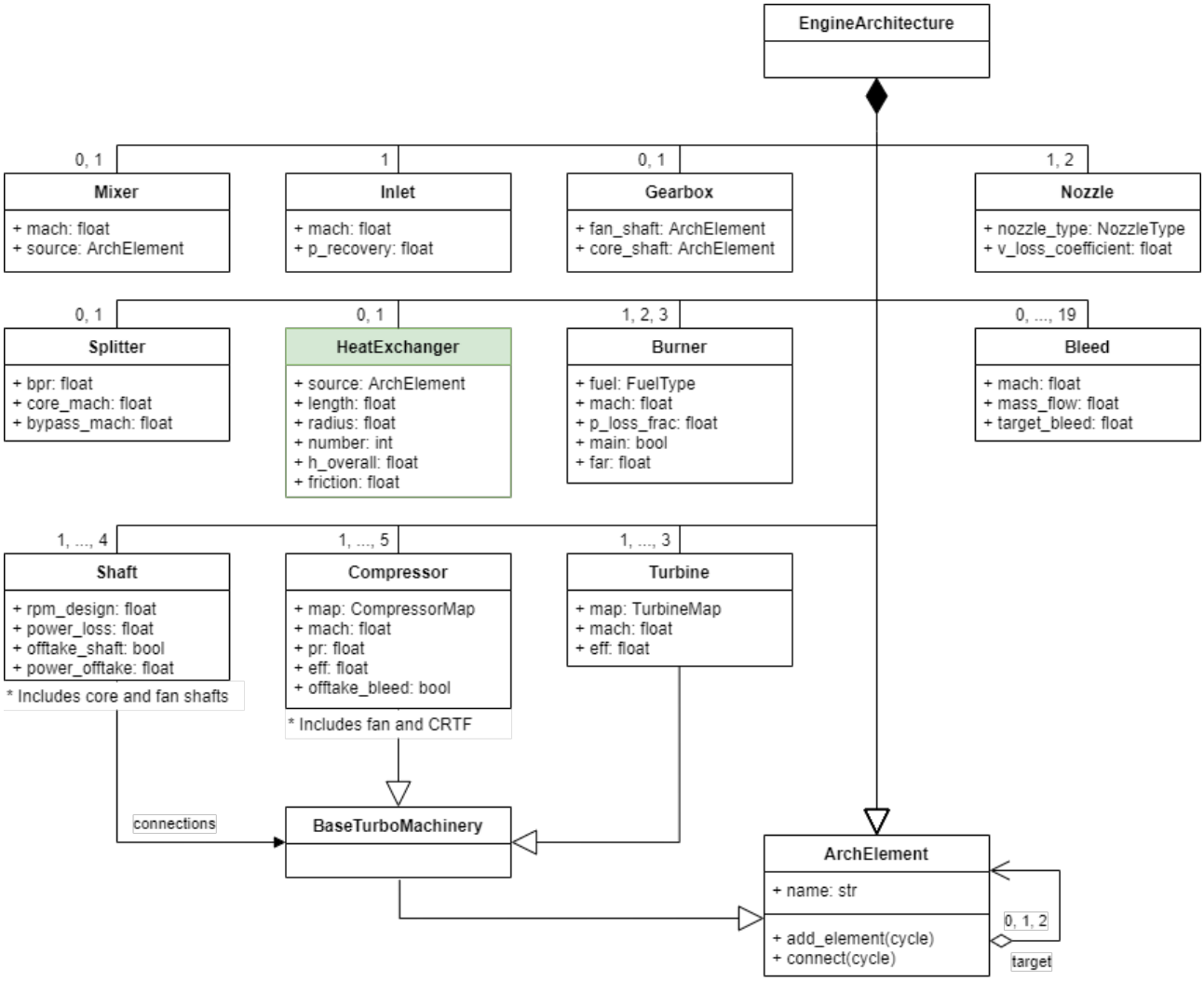


Figure 6.2: Engine architecture class diagram for the aircraft jet engine architecting tool. All components are taken from pyCycle without modification, except the HeatExchanger which was developed by the author.

6.1.1 Heat Exchanger Development

As introduced before, the HEX is the only engine component in Figure 6.2 that has been developed by the author with the purpose of being able to model intercooled aircraft engine architectures. The other components were taken directly from the pyCycle library. Therefore, this subsection discusses how this HEX components has been created. In general, there are two main ways to model a heat exchanger: LMTD and NTU-Effectiveness [57].

LMTD

The Log Mean Temperature Difference (LMTD) approach uses the readily available in- and outlet temperatures of both the cold and hot fluid to calculate the size or heat transfer of the HEX. In that case, the heat transfer rate can be written as [57]:

$$q = UA\Delta T_{lm} = UA\frac{\Delta T_2 - \Delta T_1}{\ln(\Delta T_2/\Delta T_1)} \quad \text{where} \quad \Delta T_1 = T_{h,i} - T_{c,i} \quad \text{and} \quad \Delta T_2 = T_{h,o} - T_{c,o} \tag{6.1}$$

In this equation, U and A represent the overall heat transfer coefficient and heat exchanger area while T_h and T_c refer to the hot and cold fluid, respectively. The subscripts i and o indicate the in- and outlets of the fluids, respectively. The advantage of this technique is that a higher heat transfer accuracy can achieved. However, an additional balancer needs to be implemented in the aircraft engine design problem as the hot and cold fluid outlet temperatures are not readily known.

NTU-Effectiveness

The Number of Transfer Units (NTU)-Effectiveness approach uses only the readily available inlet temperatures of both the cold and hot fluid to calculate the size or heat transfer of the HEX. It then uses NTU-Effectiveness relations to estimate the heat transfer rate and thus also the temperature change of the outlet compared to the inlet. As in the aircraft jet engine architecting tool the area of the HEX will be sized and the outlet temperatures of the hot and cold fluid are not readily known, the NTU-Effectiveness method was opted for to create the HEX component. First, the heat capacity rates have to be calculated [57]:

$$\begin{aligned} C_{max} &= \max(\dot{m}_h c_h, \dot{m}_c c_c) \\ C_{min} &= \min(\dot{m}_h c_h, \dot{m}_c c_c) \\ C_r &= C_{min}/C_{max} \end{aligned} \tag{6.2}$$

In these equations, $\dot{m}$ and c represent the mass flow rate and specific heat of the hot and cold fluids. In case the area of the HEX is known by designing the system, the NTU of the HEX component can be computed [57]:

$$NTU = \frac{UA}{C_{min}} \tag{6.3}$$

When the NTU has been found, NTU-Effectiveness relations can be used to find the effectiveness (ϵ) of the HEX. Using the aircraft engine HEX of Zhao [34], the HEX will be of cross-flow type with the bypass cross-flow as mixed cold fluid and the HEX tube-flow as unmixed hot fluid [58]. This can be seen in Figure 6.3.

The HEX effectiveness can be found with the following equations depending on the heat capacity rates [57]:

$$\begin{aligned} &\text{If the bypass flow is } \mathrm{C_{max}} : \epsilon = \frac{1}{C_r}\left(1 - e^{-C_r\left[1-e^{-NTU}\right]}\right) \\ &\text{If the HEX flow is } \mathrm{C_{max}} : \epsilon = 1 - e^{-C_r^{-1}\left(1-e^{-C_r NTU}\right)} \end{aligned} \tag{6.4}$$

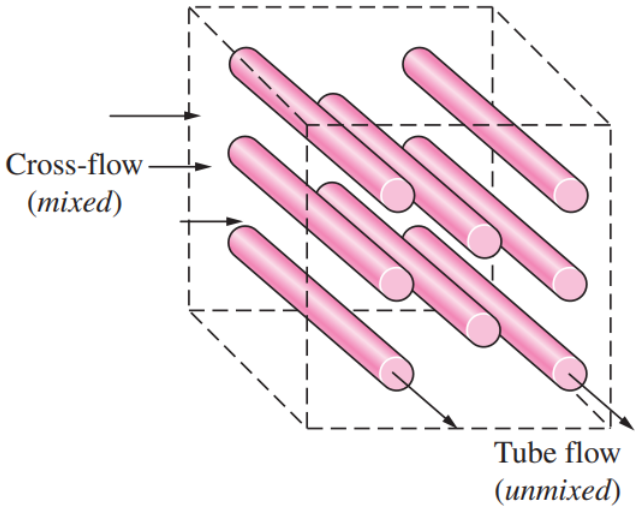


Figure 6.3: Representation of the unmixed hot tube-flow (core flow) and the mixed cold cross-flow (bypass flow) in the cross-flow HEX [58].

With the HEX effectiveness, the actual heat transfer rate of the component can be calculated and related to the temperature change of the hot and cold fluids [57]:

$$\begin{aligned} q_{max} &= C_{min}(T_{h,i} - T_{c,i}) \\ T_{h,o} &= T_{h,i} - \frac{q_{actual}}{C_h} = T_{h,i} - \frac{q_{max}\epsilon}{C_h} \\ T_{c,o} &= T_{c,i} + \frac{q_{actual}}{C_c} = T_{c,i} + \frac{q_{max}\epsilon}{C_c} \end{aligned} \tag{6.5}$$

The advantages of this method are that it is very straightforward to implement and that no balancer is required, thus reducing the complexity of the MDO problem. The main drawback is that this method is less accurate than the LMTD method.

6.1.2 Analysis Problem Definition

To evaluate an engine architecture, operating conditions need to be specified as seen in Figure 6.4: the Mach number, altitude, temperature difference compared to the International Standard Atmosphere (ISA), required thrust, and bleed and power offtake requirements. An engine cycle analysis always contains one set of design conditions: these are the operating conditions for which the engine cycle is sized, for example for a given Turbine Inlet Temperature (TIT). Then, there can be any number of evaluate conditions (or off-design conditions) where the engine is evaluated at, but which do not size any of the components. Additionally, each operating condition has a balancer. Balancers represent extra equations needed for solving the thermodynamic cycle using implicit state variables; this principle is well-known in the thermodynamic cycle analysis literature. These balancers will be explained in the following subsection.

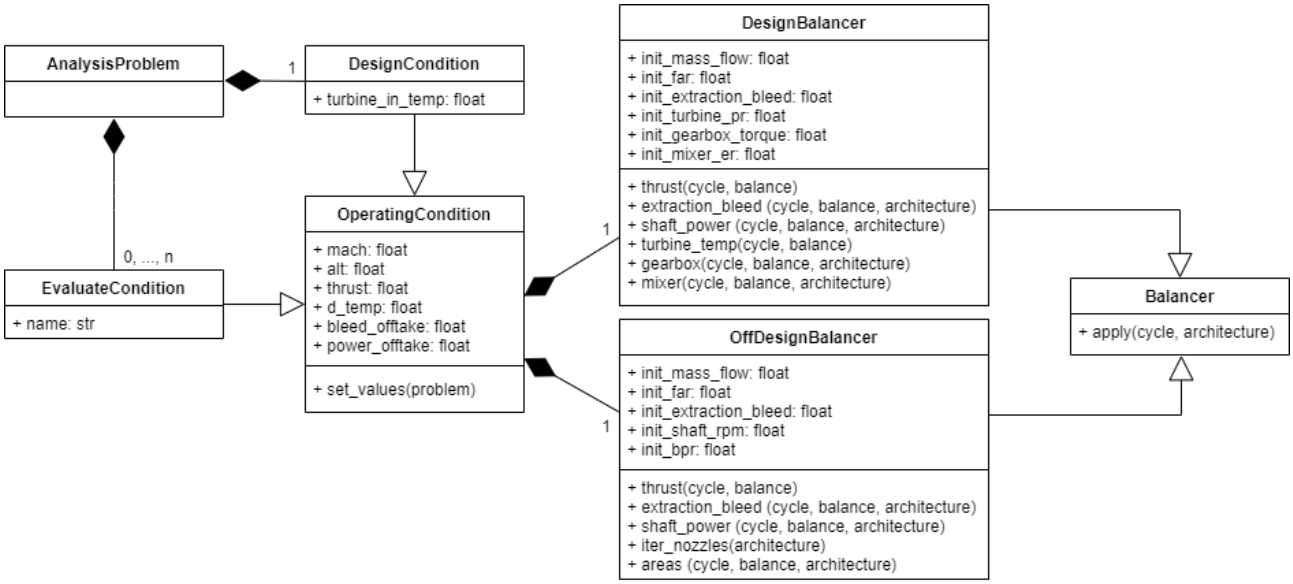


Figure 6.4: Definition of the analysis problem for the aircraft jet engine architecting tool: the operating conditions and the balancers. To clarify, there is no immediate connection between the *AnalysisProblem* and the *EngineArchitecture* as they do not exchange information.

6.1.3 Balancers

For the *Engine Architecture Evaluator*, there are three types of balancers: thermodynamic, mechanical and operating balancers. All three balancer types will be explained.

Thermodynamic Balancers

To balance the thermodynamic calculations within the aircraft engine components, pyCycle varies the fluid static temperature for it to match with a given enthalpy or entropy. Next to that, it also matches the atmospheric conditions with the engine inlet conditions by adjusting the engine inlet total temperature and pressure.

Mechanical Balancer

Next to the thermodynamic balancers, there are also mechanical balancers.

- Considering that an engine compressor and turbine are connected through a shaft, it has to be ensured that the power generated by the turbine matches the power required by the compressor. In the code, this is achieved by varying the turbine pressure ratio in on-design conditions and the shaft rotational speed in off-design conditions. This can be seen in Equation 6.6 and 6.7, respectively [7].

$$T_{turb,in}\eta_{isen}\left(1 - PR^{\frac{\kappa-1}{\kappa}}\right) = \frac{p \cdot t}{\eta_{mech}\dot{m}c_p} \tag{6.6}$$

$$p = \tau \cdot \omega \tag{6.7}$$

- When the rotational speed of the fan shaft is specified through the gearbox, the power equation of Equation 6.7 still has to hold. Therefore, the torque of the gearbox is varied in order to match the power delivered by the shaft with its rotational speed. This holds for both on- and off-design conditions.

- As the component areas are fixed with the on-design conditions, they should stay the same for the off-design conditions. In order to achieve this, the mass flow rate through the components is varied in the off-design components in order to satisfy the law of mass conservation seen in Equation 6.8.

$$\dot{m} = \rho A V \tag{6.8}$$

Operating Balancers

Finally, there are balancers to match the engine performance with the operating conditions.

- To ensure that the aircraft engine meets the thrust requirements, the air mass flow rate taken in by the engine is varied in on-design conditions, while in off-design conditions the Fuel-to-Air Ratio (FAR) of the main combustion chamber is adjusted. The influence of mass flow rate on thrust immediately becomes clear from Equation 6.9, while FAR affects the nozzle exit velocity V_j.

$$F_T = \dot{m}(V_j - V_0) + A_{nozzle}(P_j - P_0) \tag{6.9}$$

- In order to match the TIT with the on-design conditions, the FAR of the main combustion chamber is varied to achieve this. The influence of FAR on TIT is shown in Equation 6.10.

$$\dot{m}_{fuel} = \frac{\dot{m} c_p \Delta T_{cc}}{\eta_{cc} LHV} \tag{6.10}$$

- As the engine has to deliver a certain amount of extraction bleed, the percentage of air bled from a selected compressor is adjusted to match this requirement. This was discussed in more detail in chapter 5. The variation of air bleed percentage holds for both on- and off-design conditions.

6.2 Thermodynamic Cycle Analysis

The core of the engine analysis consists of the thermodynamic cycle analysis done using pyCycle [45]. PyCycle is a library of engine components that can be combined to create an OpenMDAO problem that performs thermodynamic cycle analysis.

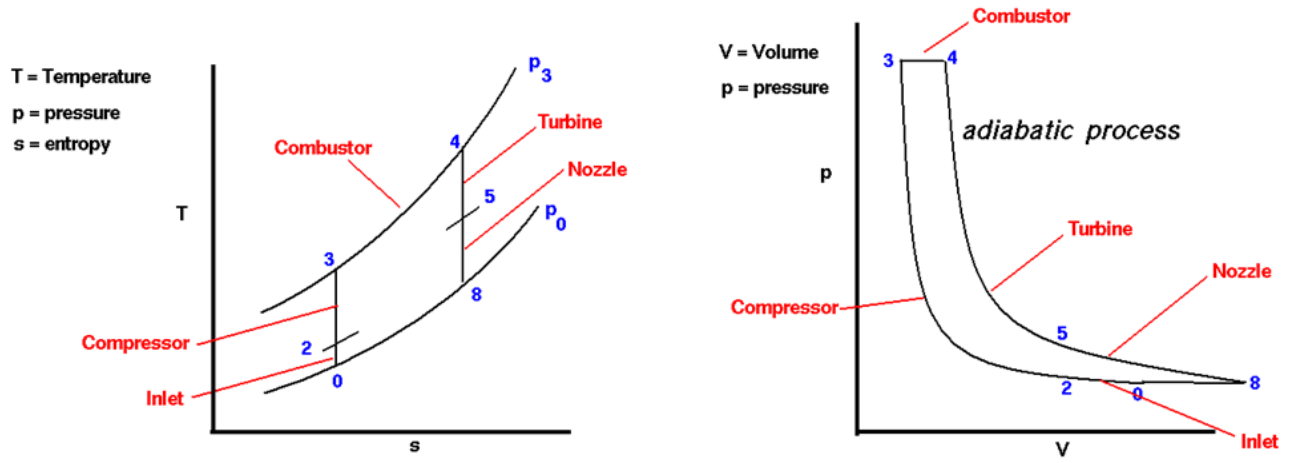


Figure 6.5: General thermodynamic cycle of an aircraft engine [59].

A typical pyCycle problem consists of a multi-point pyCycle component that contains multiple cycle components (one design cycle and multiple off-design cycles), each representing one operating condition. Quantities like sized cross-sectional areas are fed from the design cycle to the off-design cycles to ensure consistency. Each of the cycle components includes the full engine architecture as composed from the elements in the pyCycle library, and a balancer component with multiple balancing equations. As discussed before, these balancers determine how important engine parameters like thrust and turbine inlet temperature are tuned. For each of the operating conditions performance parameters are additionally determined, for example fuel flow, thrust and Overall Pressure Ratio (OPR). The *Engine Architecture Evaluator* constructs a pyCycle problem from an *EngineArchitecture* and an *AnalysisProblem* instance. Each of the architecture elements contains the logic required for adding the element to the cycle, for connecting the element to the next element in the architecture (e.g. the compressor to the burner, or the splitter to the core and bypass flows), and for adjusting specified component parameters (such as efficiency, which is not a design variable).

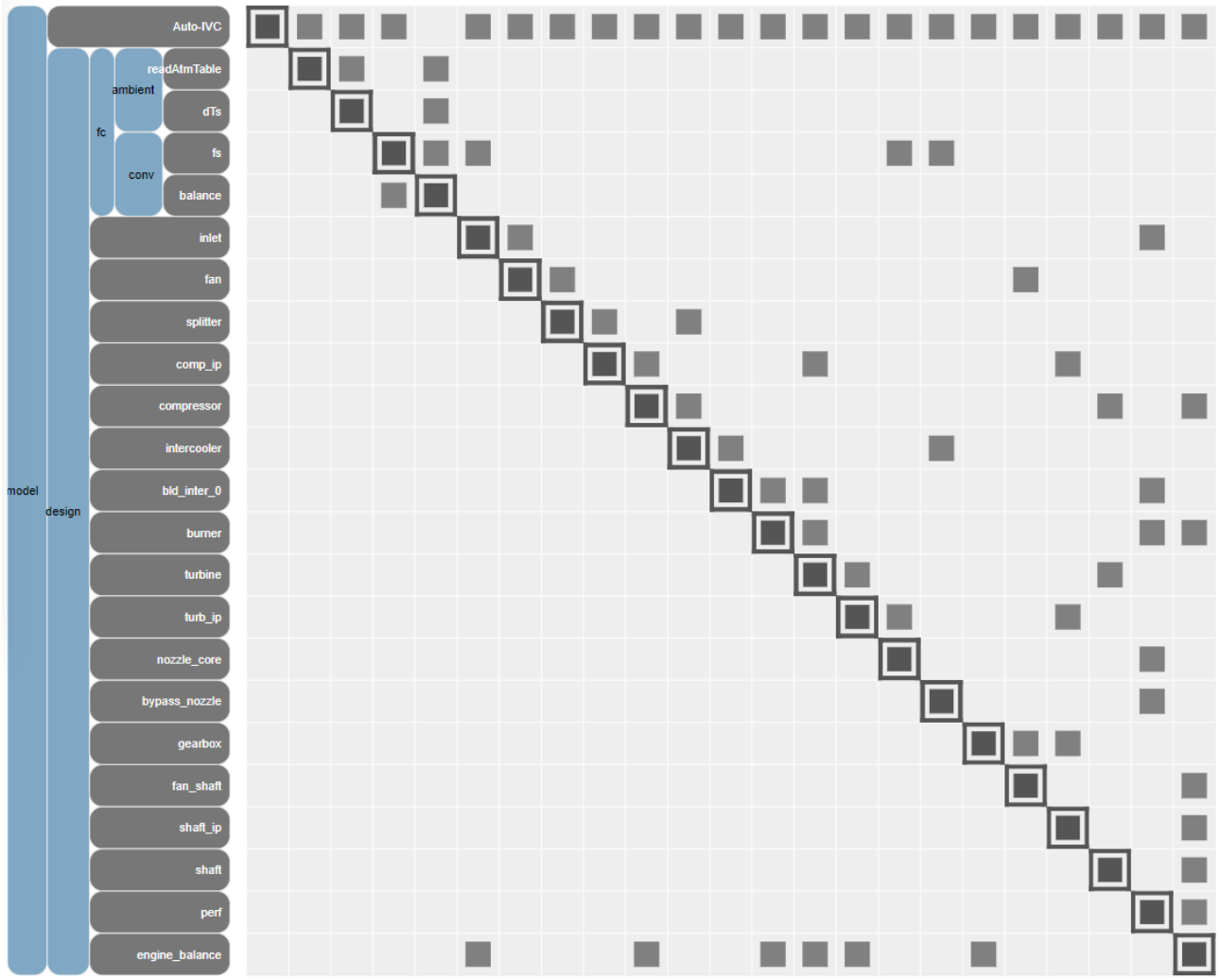


Figure 6.6: Complete DSM diagram of one of a two-shaft geared turbofan with intercooler. The different components of the engines can be seen as well as the connections between the components. Note that there are two balancers: *balance* which are balancers inherent to pyCycle (thermodynamic balancers), and *engine_balance* which were developed by the author (mechanical and operating balancers). Both were discussed in subsection 6.1.3.

Figure 6.6 shows the Design Structure Matrix (DSM) diagram of an exemplary engine thermodynamic cycle, displaying what the different engine components are and what the connections between these components are. Then, the design point is connected to the off-design points, and the operating conditions are set. The result is an OpenMDAO problem that can be executed using the regular OpenMDAO API. The thermodynamic cycle design thereby provides sizing of the engine and common performance parameters like OPR, fuel flow, thrust and Thrust-Specific Fuel Consumption (TSFC). Additionally, flow properties (static and total) are available for all flows (input and output) of all elements. These pyCycle thermodynamic cycle results are also used in the *Discipline Evaluator*.

6.3 Discipline Evaluator

Next to the thermodynamic cycle, there are several other disciplines that need to be taken into account during the design of an aircraft engine such as weight, nacelle geometry, noise and emissions. These extra disciplines are calculated after the thermodynamic cycle analysis and all disciplines, except the NOx calculations, need both the output of the thermodynamic cycle analysis and the architecture definition as inputs. Calculation of NOx emissions only needs thermodynamic cycle calculation output. The XDSM diagram of the *Discipline Evaluator* can be seen in Figure 6.7. The pymoo framework was chosen instead of OpenMDAO due to its focus on multidisciplinary and multi-objective optimization [46]. Both frameworks are explained in detail in Appendix A in case the reader wants more information on them.

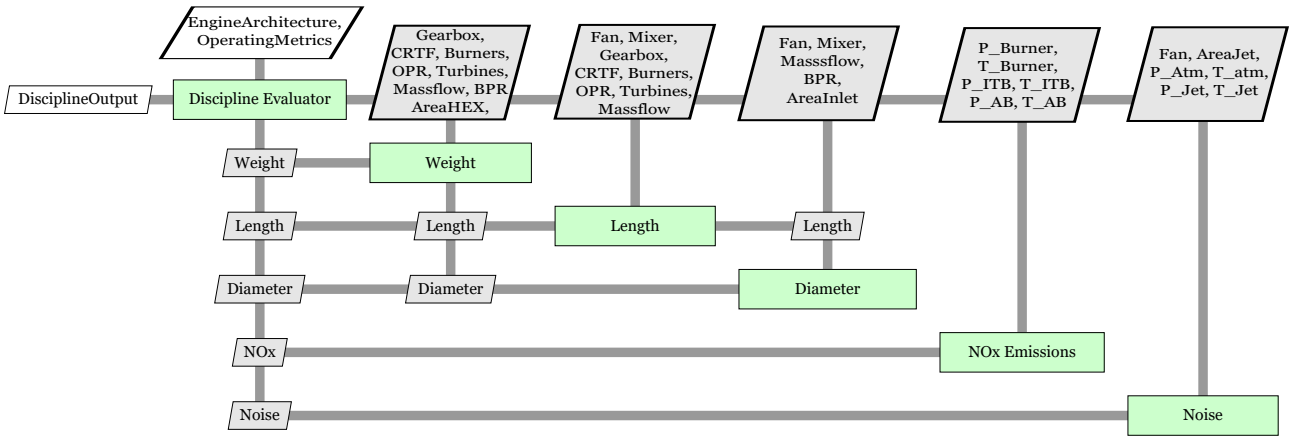


Figure 6.7: XDSM of the *Discipline Evaluator* component of the tool: it takes the engine architecture *EngineArchitecture* and the output of the thermodynamic cycle *OperatingMetrics* to calculate the discipline results of the aircraft engine design problem.

The estimation methods for the disciplines included in the aircraft jet engine architecting tool will be discussed in the following subsections. These are: weight, length, diameter, NOx emissions and noise.

6.3.1 Weight Estimation

To estimate the weight of the entire aircraft engine system, the weight of both the bare engine and the nacelle are calculated and added together. The method to calculate both of these separate weights will be explained. Furthermore, corrections were made for engines with multiple shafts or burners. This discipline adds an interesting extra design objective that normally

opposes TSFC: a more fuel-efficient engine might also be heavier, leading to a trade-off for the designer of the engine. It might also be used as a constraint, which can be interpreted as a weight budget coming from overall aircraft design considerations.

Bare Engine

Weight Analysis of Turbine Engines (WATE) is a closed-source software program developed by NASA to estimate the bare weight of an aircraft engine using thermodynamic values at component flow stations [60, 61]. To compensate for the fact that WATE is not open-source, MIT developed empirical correlations based on the WATE software and results which translate important aircraft engine data to an estimated weight, and are tuned with seven existing aircraft engine weights. Furthermore, a distinction is made between the weight of a geared and conventional engine. Results showed that the MIT correlations were 10% and 11% off compared to existing conventional and geared engines, respectively. The equations to calculate the bare engine weight can be found below in which a, b and c represent the polynomials for Equation 6.11 [62].

$$W_{bare,lbs} = a\left(\frac{\dot{m}_{core}}{100lbs/s}\right)^{b}\left(\frac{OPR}{40}\right)^{c} \tag{6.11}$$

Conventional:

$$a = (1.538 \cdot 10^{1})BPR^2 + (4.011 \cdot 10^{2})BPR + 631.5$$

$$b = (1.057 \cdot 10^{-3})BPR^2 - (3.693 \cdot 10^{-2})BPR + 1.171$$

$$c = (-1.022 \cdot 10^{-2})BPR + 0.232$$

Geared:

$$a = (-6.204 \cdot 10^{-1})BPR^2 + (2.373 \cdot 10^{2})BPR + 1702$$

$$b = (5.845 \cdot 10^{-5})BPR^2 - (5.866 \cdot 10^{-3})BPR + 1.045$$

$$c = (-1.918 \cdot 10^{-3})BPR + 0.0677$$

Nacelle

To find the weight of the nacelle of the engine, a method from Waters & Schairer was opted for. In this method, the aircraft engine is subdivided into a cowling part and a nozzle & thrust reverser part, which are both represented as cylinders. This can be seen in Figure 6.8. Then, the surface area of the cylinders, excluding the left and right circular areas, is calculated and multiplied with an average surface density to estimate the nacelle weight. These surface densities are 18.35 kg/m^2 and 73.2 kg/m^2 for the cowling and nozzle & thrust reverser cylinders, respectively. Then, the weight of both parts are summed forming the complete nacelle weight [63]. Finally, the nacelle weight is added to the bare engine weight to find the complete aircraft engine system weight.

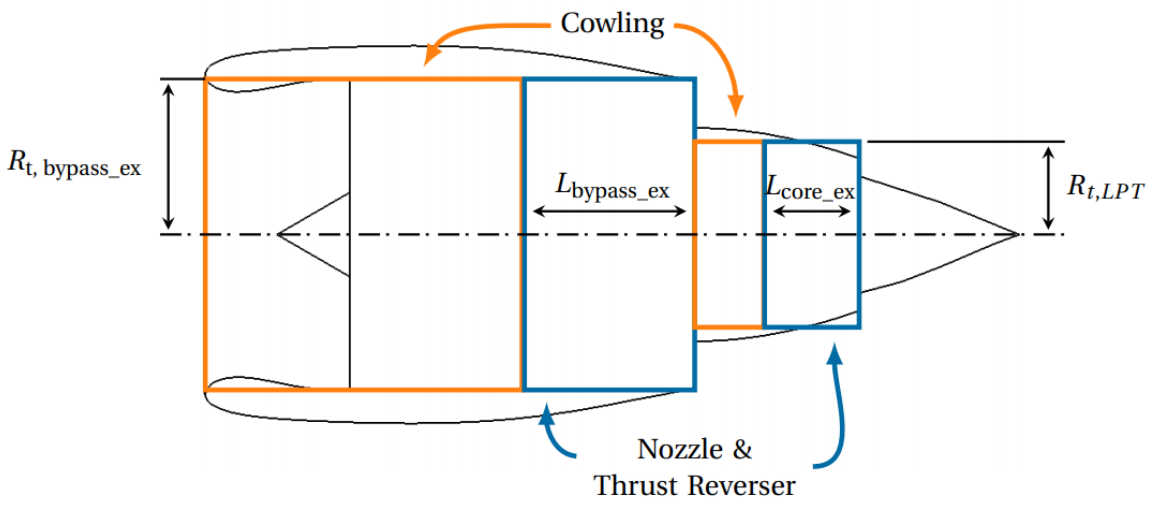


Figure 6.8: Estimation of the nacelle weight by dividing the engine into a cowling and a nozzle & thrust reverser section. The surface area of each section is multiplied with an empirical surface density and summed to find the final nacelle weight [64].

Corrections

As the MIT method does not take into account the change in weight due to multiple shafts or innovative technologies such as an inter-turbine burner or counter-rotating fan, corrections were made to the calculated bare engine weight for these cases. These were primarily based on the weight each components contributes to the entire engine, as specified by WATE simulations for a conventional two-shaft turbofan. The division of these percentages can be seen in Table 6.1 [62]. For example, when an additional burner was added to the aircraft engine architecture by the optimizer, 5% was added to the computed bare engine weight to compensate for this burner addition. Next to that, as no exact information was found on shaft weight, 10% was subtracted for one-shaft engines and 10% was added for three-shaft engines. Furthermore, in case of a counter-rotating fan, 10% was added to the estimated bare engine weight based on research from project COBRA of the European Commission [48].

Table 6.1: Percentual contribution of each of the aircraft engine components to the bare engine weight [62]. Note that these are for a conventional two-shaft turbofan.

Engine component	Fan	LPC	HPC	CC	HPT	LPT	Accessories	Other
Bare weight [%]	26.04	5.92	10.78	4.84	11.4	23.55	15.53	1.94

6.3.2 Length & Diameter Estimation

As the aircraft engine needs to be integrated in the aircraft airframe, the dimensions of the nacelle need to be taken into account to ensure that this does not cause any issues [65]. In the aircraft engine architecting tool, both the length and diameter of the engine can be calculated. This is achieved with a Torenbeek & Berenschot method [66]. Figure 6.9 provides an overview of which dimensions are estimated by the tool. The user of the tool can select which length and diameter to opt for as design objective or constraints, however the length l_n and diameter D_n are returned by default by the aircraft jet engine architecting tool.

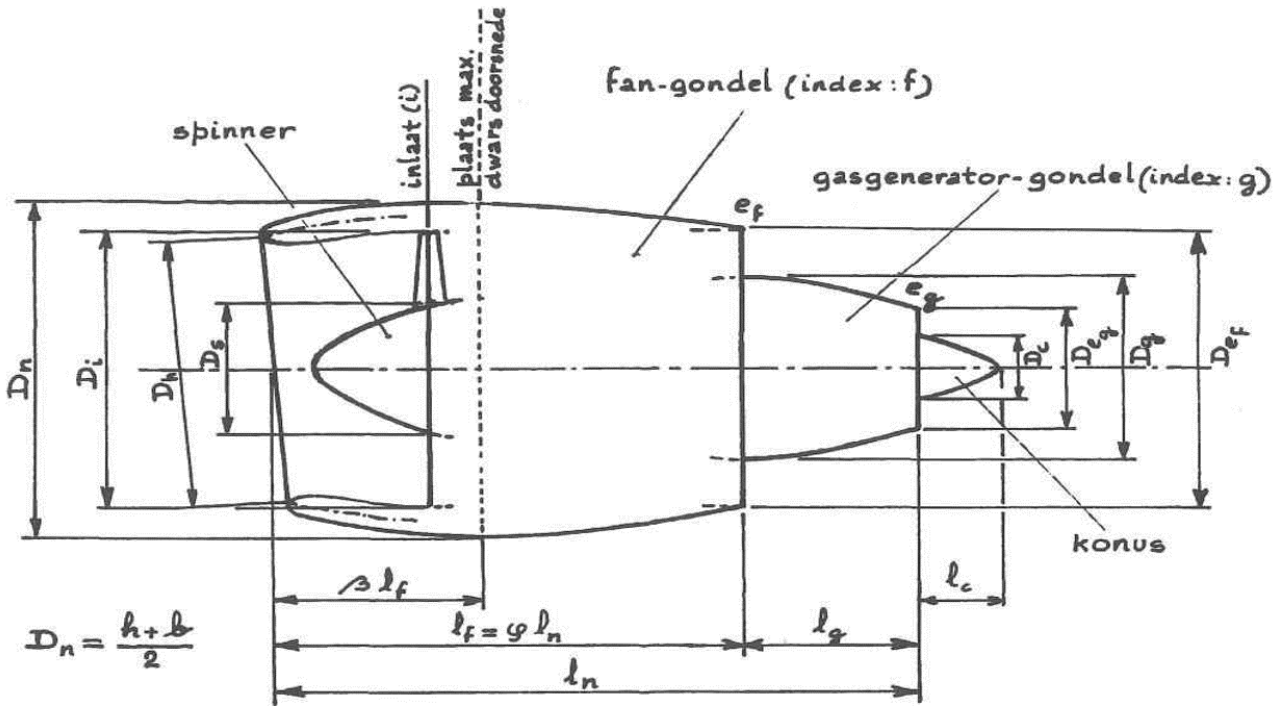


Figure 6.9: Overview of the dimensions (length and diameter) calculated by the aircraft engine architecting tool [66]. The length l_n and diameter D_n are returned by default by the tool.

Equations 6.12 to 6.18 show how the different lengths and diameters of the aircraft engine, including nacelle, can be estimated [66]. The density and speed of sound at sea level are referred to as ρ_0 and a_0, respectively. Furthermore, λ and $\dot{m}$ are the bypass ratio and air mass flow rate taken in by the engine, respectively. The constants l_{ref}, Δl and ϕ are dependent on engine configuration:

- Turbojets: $l_{ref} = 12$, $\Delta l = 0$ m and $\phi = 1$;
- Turbofans:
  - Mixed nozzle: $l_{ref} = 9.8$, $\Delta l = 0.05$ m and $\phi = 1$;
  - Separate nozzle: $l_{ref} = 7.8$, $\Delta l = 0.1$ m and $\phi = 0.625$.

Lengths

$$l_f = \phi l_n = \phi l_{ref} \left(\sqrt{\frac{\dot{m}}{\rho_0 a_0} \frac{1 + 0.2\lambda}{1 + \lambda}} + \Delta l \right) \tag{6.12}$$

$$l_g = (1 - \phi)\, l_n \tag{6.13}$$

Diameters

$$\frac{D_s}{D_i} \approx 0.05 \left[1 + 0.1 \frac{\rho_0 a_0}{\dot{m}} + \frac{3\lambda}{1 + \lambda} \right] \tag{6.14}$$

$$D_i = D_h = 1.65 \sqrt{\frac{\frac{\dot{m}}{\rho_0 a_0} + 0.005}{1 - (D_s/D_i)^2}} \tag{6.15}$$

$$D_n = D_i + 0.06\phi l_n + 0.03 \tag{6.16}$$

$$D_{ef} = D_n \left(1 - \frac{1}{3}\phi^2\right) \tag{6.17}$$

$$D_{eg} \approx 0.55 D_g = 0.55 D_{ef} \left(\frac{0.089\frac{\dot{m}}{\rho_0 a_0}\lambda + 4.5}{0.067\frac{\dot{m}}{\rho_0 a_0}\lambda + 5.8}\right)^2 \tag{6.18}$$

Corrections

The length and diameter estimation method does not take into account a change in length and diameter in case multiple shafts or innovative technologies are used. Therefore, adjustments are implemented in the tool for the length discipline. No exact information was found of the addition of shafts on the engine length, thus the assumption was made that 10% length is subtracted for one-shaft engines and 10% length is added for three-shaft engines compared to a two-shaft engine. Next to that, 10% length is added when an inter-turbine burner is present as well as when a counter-rotating fan is included in the engine architecture. However, these adjustments were not implemented for the diameter discipline as it was assumed that the BPR of the engine is dominant for the diameter calculation, and therefore the addition of for example an inter-turbine burner would not make a (significant) difference.

6.3.3 NOx Emissions Estimation

Considering that aircraft emissions have an impact on the environment, ICAO has created regulations to limit this impact: NOx emissions are limited to 15 g/kg fuel [29]. On top of that, Europe has put the objective to reduce these NOx emissions by 90% by the year 2050 [67]. Therefore, NOx emissions are important to take into account as constraint (or even as objective) when designing an aircraft engine. Other important emissions have not been taken into account in the aircraft jet engine architecting tool, as it was found in literature that they are positively correlated to the fuel consumption, which has already been taken into account in the thermodynamic cycle analysis of section 6.2. By eliminating these correlated engine emissions, the dimension of the multi-objective problem is reduced. The emissions relate to fuel consumption as [68, 69]:

- CO_2 = 3.16 kg/(kg fuel)
- SO_4 = 2.0e-4 kg/(kg fuel)
- H_2O = 1.26 kg/(kg fuel)
- Soot = 4.0e-5 kg/(kg fuel)

To calculate NOx emissions, the P3T3 method is implemented which uses the pressure and temperature at the inlet of the combustion chamber to estimate the NOx emissions, as can be seen in Equation 6.19 [70]. This method has been implemented in other aircraft engine design and analysis software such as GasTurb and Gas turbine Simulation Program (GSP) [71, 72].

$$NO_x = 32 \left(\frac{P_3}{2965kPa}\right)^{0.4} e^{\left(\frac{T_3 - 826K}{194K} + \frac{6.29 - 100war}{53.2}\right)} \tag{6.19}$$

In case an afterburner or inter-turbine burner are present in the aircraft engine architecture, it is assumed that the total NOx emissions are the result of only this afterburner or inter-turbine burner due to reburning of main combustion chamber emissions [73].

6.3.4 Noise Estimation

The noise of an aircraft engine contributes a significant part to the total noise emitted by an aircraft in approach and especially in take-off, as can be seen in Figure 6.10 [74]. Therefore, ICAO has again created regulations regarding aircraft engine noise in three particular flight situations and based on Maximum Take-off Weight (MTOW) [28, 75]:

- Approach: 105 EPNdB for MTOW $\geq$ 280,000 kg reducing by 2.33 EPNdB per MTOW halving, with a minimum of 98 EPNdB for MTOW $<$ 35,000 kg;
- Flyover:
  - 1- or 2-engine aircraft: 101 EPNdB for MTOW $\geq$ 385,000 kg reducing by 4 EPNdB per MTOW halving, with a minimum of 89 EPNdB for MTOW $<$ 35,000 kg;
  - 3-engine aircraft: 104 EPNdB for MTOW $\geq$ 385,000 kg reducing by 4 EPNdB per MTOW halving, with a minimum of 89 EPNdB for MTOW $<$ 35,000 kg;
  - 4-engine aircraft: 106 EPNdB for MTOW $\geq$ 385,000 kg reducing by 4 EPNdB per MTOW halving, with a minimum of 89 EPNdB for MTOW $<$ 35,000 kg;
- Lateral full-power: 103 EPNdB for MTOW $\geq$ 400,000 kg reducing by 2.56 EPNdB per MTOW halving, with a minimum of 94 EPNdB for MTOW $<$ 35,000 kg.

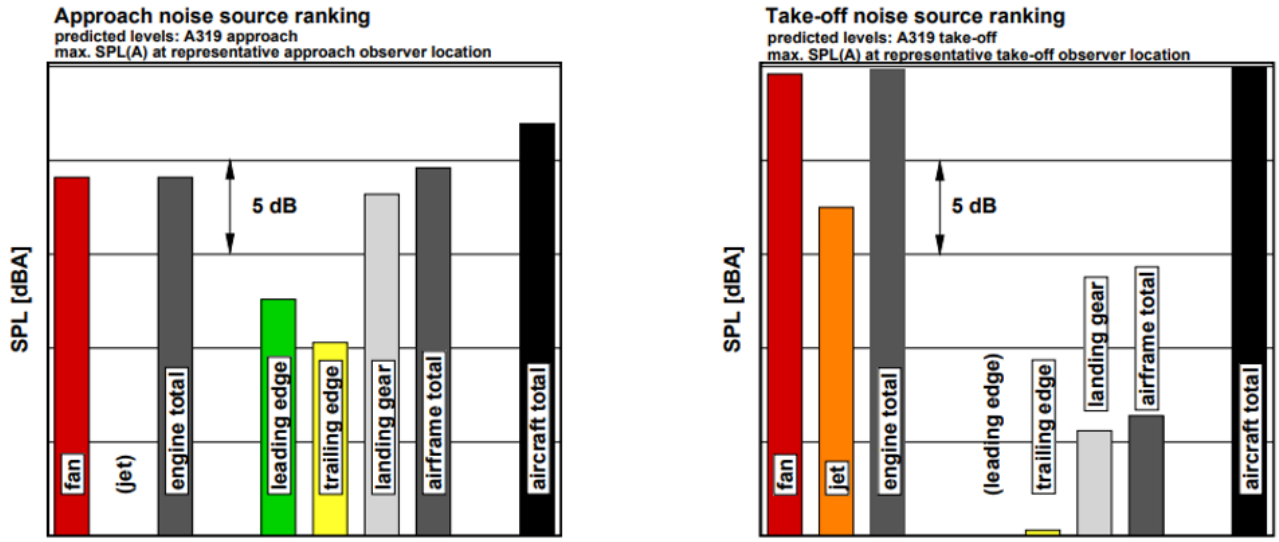


Figure 6.10: Aircraft noise components in both approach and take-off compared to overall aircraft noise, which shows that engine noise contributes a significant part [74].

The noise emitted by the aircraft engine is estimated by a method from Stone [76]. Even though this method is from 1974, noise experts of the TU Delft ANCE group confirmed that this method is still relevant and that closed-source software programs, such as the Aircraft Noise Prediction Program (ANOPP(2)) from NASA [77], use variations of the Stone method. Equation 6.20 shows how the noise is estimated. In this equation, atmospheric (subscript a) and ISA conditions are taken into account by density ρ and speed of sound a to calculate the Overall Sound Pressure Level (OASPL). Furthermore, jet nozzle (subscript j) velocity and area are considered in the method as well. Finally, the distance to the observer R is also included and will depend on the flight stage of the aircraft. Factors such as the fan tone due to thrust

setting influenced by pilot behavior are not taken into account by Equation 6.20, even though they are important according to the TU Delft noise experts. Due to the fact that they are difficult to model and would introduce additional assumptions, influencing the accuracy of the noise estimation significantly, it was decided to not take them into account in the aircraft jet engine architecting tool.

$$\begin{aligned} OASPL_{90^\circ} = 141 + 10\log\left[\left(\frac{A_j}{R^2}\right)\left(\frac{\rho_a}{\rho_{ISA}}\right)^2\left(\frac{a_a}{a_{ISA}}\right)^2\right] + 10\log\left[\frac{(V_j/a_a)^{7.5}}{1+0.01(V_j/a_a)^{4.5}}\right] \\ + 10\left[\frac{3(V_j/a_a)^{3.5}}{0.6+(V_j/a_a)^{3.5}} - 1\right]\log\left(\frac{\rho_j}{\rho_a}\right) \end{aligned} \tag{6.20}$$

Chapter 7
Verification & Validation

To ensure that the design choices made in the aircraft jet engine architecting tool lead to realistic results for the engine disciplines, the tool was verified and validated. This was done with two consecutive steps: pyCycle thermodynamic cycle analysis and real aircraft engine discipline data. Both of these steps will be discussed in section 7.1 and section 7.2, respectively.

7.1 Verification: pyCycle Thermodynamic Cycle Analysis

In order to verify that the created choices in the aircraft jet engine architecting tool lead to the same results as the code of pyCycle, a comparison of the thermodynamic cycle was made for two engine architectures: a simple turbojet with one shaft and a High Bypass Ratio (HBR) turbofan with two shafts. To verify the tool correctly, the same operating conditions, component inputs (such as the OPR) and component efficiencies were used as in the pyCycle examples. All the components and their attributes used to create the verification cases are listed in Appendix C.

7.1.1 Turbojet with One Shaft

First, the thermodynamic cycles of the turbojet with one shaft are compared to each other. The operating conditions were set to take-off: ISA conditions on ground level, with a produced thrust of 52,489 N and TIT of 1,043.5 degC. The OPR of the turbojet was set equal to 13.5 and the rotational speed of the shaft to 8,070 RPM. No extraction bleed or power offtakes were activated.

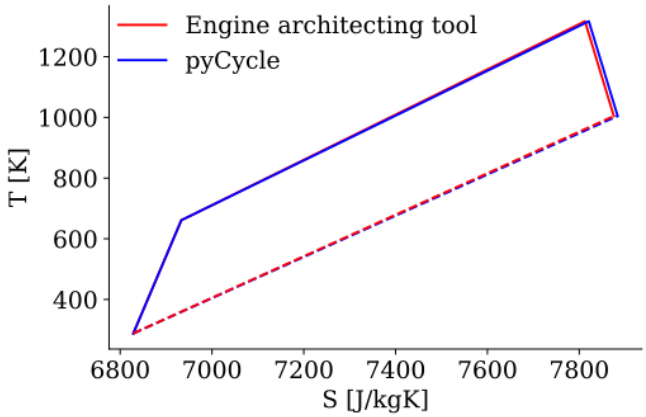


Figure 7.1: Comparison of the thermodynamic cycle computed by pyCycle and the aircraft jet engine architecting tool results for a single-shaft turbojet, to verify whether the implemented design choices in the tool result in the same architecture as a given pyCycle example.

As can be seen in Figure 7.1, the thermodynamic cycles of the engine architecting tool and pyCycle are very similar which verifies that the architecting choices have been implemented correctly for a simple turbojet. Furthermore, when comparing this cycle to the thermodynamic cycle in Figure 6.5 of chapter 6, it can be seen that the results are as expected.

7.1.2 HBR Turbofan with Two Shafts

Next, a more complicated pyCycle example was taken to verify the implemented choices of the engine architecting tool: an HBR turbofan with two shafts. Contrary to the turbojet example, the operating conditions of the engine were set to mid-cruise: ISA conditions at 35,000 ft at a Mach of 0.8, with a produced thrust of 26,244.5 N and TIT of 1,314 degC. In addition, 186,425 W of power was also taken off. The engine has a BPR of 5.105 and FPR of 1.685. Next to that, the OPR of the engine is 30.094 of which 18% and 82% are produced by the LPC and HPC, respectively. The rotational speeds of the corresponding shafts are 4,666.1 RPM and 14,705.7 RPM, respectively. Furthermore, cooling bleed was also included: 7% bleed in the HPC lead to the LPT, and 16.8% bleed from in between the HPC and CC lead to the HPT.

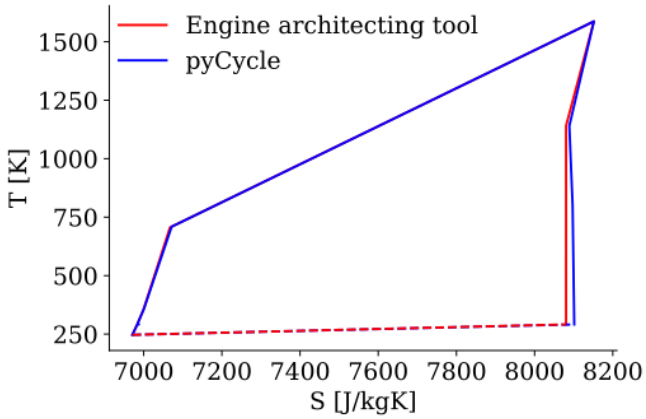


Figure 7.2: Comparison of the thermodynamic cycle computed by pyCycle and the aircraft jet engine architecting tool results for a two-shaft HBR turbofan, to verify whether the implemented design choices in the tool result in the same architecture as a given pyCycle example.

Again, the thermodynamic cycles of the engine architecting tool and pyCycle are very similar. The small discrepancies between both cycles are the result of pressure losses in ducts, which is a pyCycle component that was not included in the tool as it would not contribute any significant value to the optimization problem whilst making it harder for the architecture to converge. Together with the results of the turbojet example, it can be concluded that the engine architecting tool is correctly verified for both examples.

7.2 Validation: Real Aircraft Engine Discipline Data

To validate the results of the tool disciplines, they were compared to existing engine architectures. For the turbojet, the Pratt & Whitney F100 engine [78] used on the F16 is chosen whereas for the turbofan, the CFM LEAP-1C engine [79] used on the Comac C919 is selected. The TSFC of the engines is calculated at the cruise condition (35,000 ft altitude at Mach = 0.8), whereas the other disciplines are calculated at take-off condition (0 ft altitude at Mach $\approx$ 0). The reason for this is that the empirical correlations of the other disciplines are tuned

for take-off conditions, while TSFC data is usually provided for mid-cruise. In order to take into account the technological progress that has been achieved in the design of aircraft engines, correcting factors were applied to the length and diameter disciplines to bring them up to date. The results of both aircraft engine disciplines will be discussed in the following subsections. For more information and visualization of these disciplines, the reader is referred to section 6.3. All the components and their attributes used to create the validation cases are listed in Appendix D.

7.2.1 Turbofan: CFM LEAP-1C

The CFM LEAP-1C engine is analyzed first, as most data of this engine is readily available considering that it is commonly used on commercial aircraft. It can produce up to 133 kN in thrust with an OPR of 40 and BPR of 11 [80]. Unfortunately, other exact data on the engine regarding the operating conditions was not found, so educated guesses were made on for example the TIT. Table 7.1 shows how the results from the aircraft engine architecting tool compare to real aircraft engine data.

Table 7.1: Comparison between properties of the LEAP-1C engine [79] and the high-BPR, two shaft engine validation architecture. The TSFC is calculated at mid-cruise condition whereas the other disciplines are calculated at take-off condition as their empirical correlations were tuned for this operating condition. The weight of the engine includes the bare engine and nacelle, while length and diameter are the maximum dimensions of the engine.

	TSFC [g/kNs] Mid-cruise	Weight [kg] Take-off	Length [m] Take-off
CFM LEAP-1C	14.4	3,932	4.51
Engine tool	16.6	3,910	4.49
	Diameter [m] Take-off	**NOx [g/kg fuel] Take-off**	**Noise [dB] Take-off**
CFM LEAP-1C	2.71	18.77-64.36	Unknown
Engine tool	2.71	61.99	121

As the outcome of the disciplines indicates, the engine architecting tool is able to achieve realistic results for a turbojet engine: almost all the discipline results lie very closely to the real engine data. The TSFC calculated by the tool is a little higher than the real engine, however this could be result of lower component efficiencies used in the tool. Unfortunately, no trustworthy data on efficiencies was found for the real engine making it hard to make a very accurate comparison for TSFC. Next to that, no data was found for the engine noise either. When considering that an aircraft can emit around 140 dB in noise and the engine contributes a large portion of that, as discussed in section 6.3 with Figure 6.10, the noise discipline result seems to be in the correct order of magnitude [81]. Therefore, the aircraft jet engine architecting tool is deemed validated for turbofan applications.

7.2.2 Turbojet: P&W F100

After the turbofan validation, the Pratt & Whitney F100 is analyzed. This engine produces 129.7 kN thrust at an OPR of 31 and BPR of 0.36 [78]. With a BPR of 0.36, the F100 is strictly

speaking not a pure turbojet, but this BPR is deemed low enough to be viewed as a turbojet. Just like for the turbofan, other exact data on the engine regarding the operating conditions was not found, so educated guesses were made on for example the TIT. The comparison between the discipline results of the tool and the real F100 engine can be seen in Table 7.2.

Table 7.2: Comparison between properties of the F100 engine [78] and the low-BPR, two shaft engine validation architecture. The TSFC is calculated at mid-cruise condition whereas the other disciplines are calculated at take-off condition as their empirical correlations were tuned for this operating condition. The weight of the engine includes the bare engine and nacelle, while length and diameter are the maximum dimensions of the engine.

	TSFC [g/kNs] Mid-cruise	Weight [kg] Take-off	Length [m] Take-off
P&W F100	Unknown	1,735	4.85
Engine tool	26.7	1,752	4.82
	Diameter [m] Take-off	**NOx [g/kg fuel] Take-off**	**Noise [dB] Take-off**
P&W F100	1.18	Unknown	Unknown
Engine tool	1.35	44.38	108.29

Not much trustworthy data was found for the F100, which was expected as the F100 is an engine used in the military. However, the weight and geometry could be retrieved on the Pratt & Whitney website. This shows that the weight and length of the tool and real engine are very similar. Next to that, the diameter estimated by the tool is slightly higher but still in a correct order of magnitude. For TSFC, a comparison with the CFM LEAP-1C engine can be made as validation: the tool indicates that the TSFC of a turbojet is significantly higher than the TSFC of a turbofan, which was expected. Furthermore, as the OPR and TIT of the F100 are lower than the LEAP-1C, in theory its NOx emissions should be lower as well which is indeed the case as shown in Table 7.2. Finally, it can be noticed that the noise of the F100 is lower than the LEAP-1C, but this could be due to the fact that the noise is calculated only for the jet nozzle and not for the entire engine. With all these results, the aircraft jet engine architecting tool is deemed validated for turbojet engines as well.

Summary: Research Question Answers

RQ1 How is the system architecting approach implemented in aircraft engine modeling?

RQ1.1 To what extent can realistic values for the engine performance be obtained?

> The aircraft jet engine architecting tool achieves valid results for both turbojet and turbofan engines, both for thermodynamic cycle as well as other disciplines in aircraft engine design. The tool estimation for TSFC, weight and geometry results is very close to real aircraft engine data, while NOx emissions and noise are deemed in the correct order of magnitude.

Chapter 8
Results

In this chapter, two aircraft jet engine architecting problems will be created and evaluated: a simple single-objective one and a more realistic multi-objective one. For these problems, the same operating conditions will be used as for the CFM LEAP-1C engine to show that the tool can produce useful results for the engine design of commercial aircraft. Furthermore, the MDO, mixed-discrete and multi-objective capabilities of the aircraft jet engine architecting tool are demonstrated to show that it can model and compare different aircraft engine architectures.

First, an overview of the overall setup of the architecting problems will be presented in section 8.1. To ensure that the benchmark problem requirements of chapter 3 are satisfied, this will be reviewed in section 8.2. Then, the results of the simple single-objective problem will be discussed in section 8.3. After that, the realistic multi-objective problem results will be analyzed in section 8.4. With the results of the realistic problem, it will be investigated whether a Pareto front has effectively formed in section 8.5. Next, weight and length sensitivity studies were performed on the aircraft jet engine architecting tool in section 8.6. Finally, the results are discussed to see how they meet the thesis objective and answer the research questions in section 8.7. Component attributes that were kept constant for the engine architecting problems can be found in Appendix E.

8.1 Architecting Problems Setup

Before analyzing the results of the simple and realistic engine architecting problems, the general setup of both problems has to be discussed. To demonstrate that the aircraft jet engine architecting tool can be used to model and compare realistic aircraft engines, the same operating conditions will be taken as for the commercial CFM LEAP-1C engine in take-off conditions. Take-off conditions are used as the weight and geometry disciplines are tuned for this condition and other conditions lead to unrealistic results. The following operating conditions are used [32, 79]:

- Mach $\approx$ 0;
- Altitude = 35,000 ft;
- Thrust = 133 kN;
- TIT = 1,450 degC;
- Bleed offtake = 0.5 kg/s;
- Power offtake = 37.5 kW.

As discussed in chapter 4, NSGA-II is used as standard optimization algorithm to tackle the benchmark problems. Furthermore, it must be noted that the implemented architecting choices and objectives & constraints will be discussed in the sections of the two problems as they are vital for the obtained results.

8.2 Requirement Compliance

The engine architecting problems should adhere to the requirements of creating a realistic system architecture optimization benchmark problems specified by Gray [26] and laid out in chapter 3. This will be reviewed for each requirement individually based on the information gathered in the previous chapters of the thesis paper:

BR1 As discussed in section 5.1 and visualized in Figure 5.2, all three types of design variables are present: categorical, integer and continuous. This makes the problem of mixed-discrete nature.

BR2 In section 5.1, it is shown that design variable hierarchy is present. This means that some design variables are conditionally active based on other design variables. An example is the engine BPR based on the fact whether a fan is present in the architecture.

BR3 The design space is made up from a total of 43 design variables: 7 categorical, 5 integer and 31 continuous. This satisfies the constraint of the design space having between 10 and 50 dimensions.

BR4 Through the thermodynamic cycle analysis and the discipline evaluator, a maximum of six objectives can be set for the engine architecting problems: TSFC, weight, length, diameter, NOx emissions and noise. It is important to note that these objectives are not correlated and that they represent conflicting stakeholder needs.

BR5 The shape and behavior of the design space is not known a-priori, showing that the performance evaluation behavior is black-box.

BR6 The user of the aircraft jet engine architecting tool has the possibility to choose the objectives and constraints of the problem, and to fix certain design variables to parameters in the design problem. Therefore, the difficulty of the problem is tunable. This will also be demonstrated through the simple single-objective and realistic multi-objective problem of section 8.3 and 8.4, respectively.

BR7 The complete code of the aircraft jet engine architecting tool has been published on GitHub as an open-source tool so that researchers can use it as benchmark for system architecting and optimization problems [39]. Next to that, it is based on the open-source software pyCycle [45] and OpenMDAO [82]. Therefore, it meets the open-source requirement of benchmark problems.

BR8 Through object-oriented coding and extensive documentation, it should be relatively easy to modify the problem structure and to include additional elements in the code such as design decisions, objectives and constraints. This means that the problem is indeed hackable as was required.

Therefore, it can be concluded that the requirements to create a realistic system architecture optimization benchmark problem have all been met.

8.3 Simple Single-Objective Problem

First, the aircraft jet engine architecting tool was used to optimize a simple architecting problem. This architecting problem features only one objective (TSFC), and only design feasibility constraints. The included design choices are:

- Fan choice;
- Nozzle mixing choice;
- Shaft number choice;
- Offtakes choice;
- Gearbox choice.

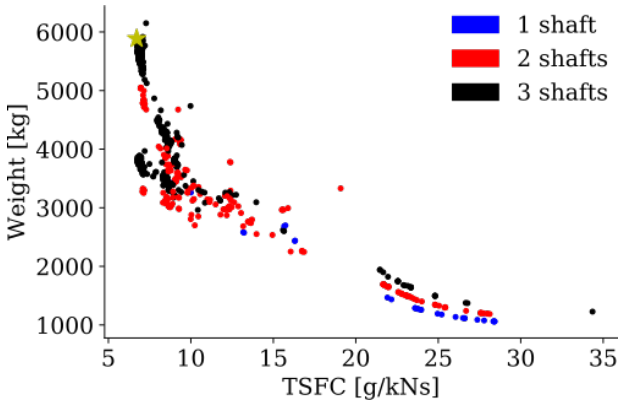
These choices were explained in detail in chapter 5. The design choices result in 15 design variables and the creation of solely conventional engine architectures, i.e. excluding counter-rotating fan, afterburner, inter-turbine burner, cooling bleed and intercooler. In total, 15 distinct engine architectures and 216 apparent engine design points can be generated taking into account the discrete variables. The multi-objective optimization algorithm NSGA-II from the pymoo framework with a population size of 75 and termination criteria of 1000 evaluations was used to run the optimization.

All architectures were analyzed and their results are presented in subsection 8.3.1 and 8.3.2. All feasibility constraints, discussed in chapter 5, are satisfied. Approximately 49% of the generated engine architectures converged during the initial DOE, whereas this increased to 92% for the last iteration. This can be seen as the presence of hidden constraints: constraints that are not known a-priori, and are considered violated whenever the simulation did not converge to a meaningful result.

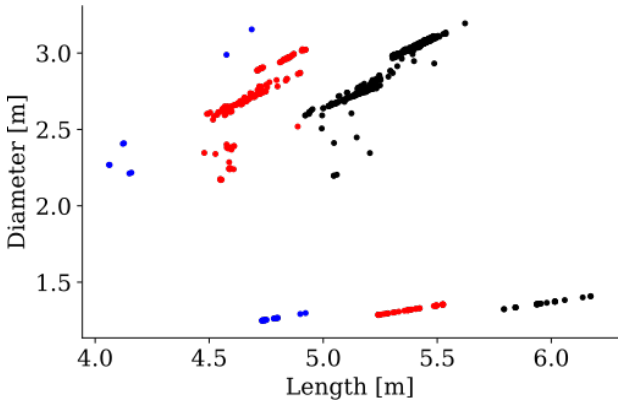
8.3.1 Results for Number of Shafts

The results of the simple architecting problem generated by the aircraft jet engine architecting tool based on number of shafts can be seen in Figure 8.1.

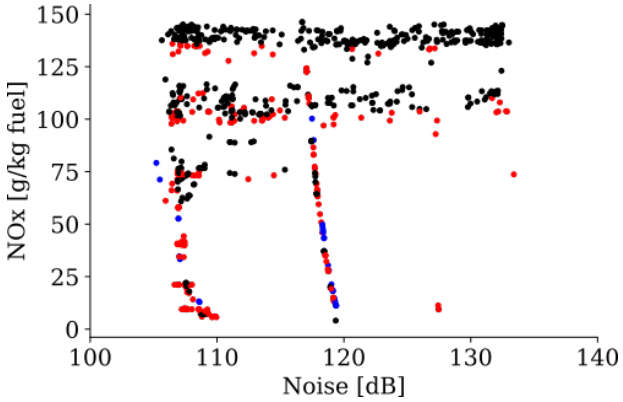
For the number of shafts, it can be deduced that TSFC reduces whereas length and weight increase with increasing number of shafts. This could be explained by the fact that the additional compressors rotate at optimal speeds, leading to a higher efficiency of the engine at the cost of higher length and weight. The effect of this can clearly be seen in Figure 8.1d: towards later iterations, the algorithm starts using more and more 3-shaft architectures in order to optimize TSFC. Regarding NOx emissions, these increase with increasing engine OPR [7]. As a higher OPR can be achieved with a higher number of shafts, NOx emissions will increase with higher number of shafts as well. This can clearly be seen in Figure 8.1c. As noise is dependent on jet velocity and nozzle exit area, no significant deduction can be drawn for noise dependency on number of shafts [76].



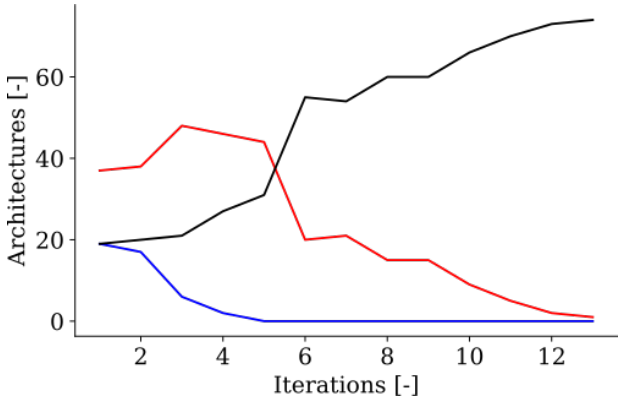
(a) TSFC and weight.



(b) Length and diameter.



(c) Noise and NOx emissions.



(d) Engine architectures over the iterations.

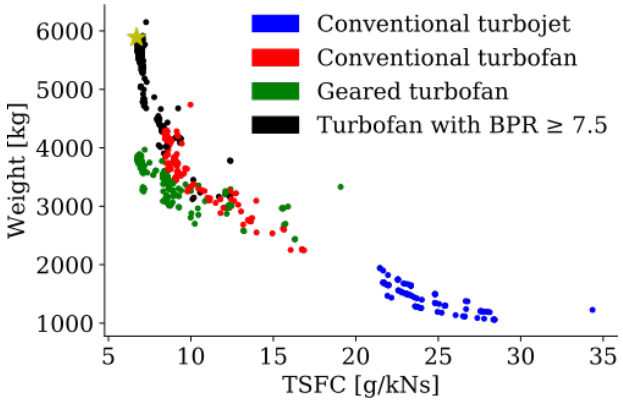
Figure 8.1: Engine results for the simple architecting problem, based on number of shafts. The yellow star indicates the best performing engine architecture as it is a single-objective problem optimized for TSFC.

Based on Figure 8.1, the results of the simple architecting problem are in line with what was physically expected, serving as additional validation of the engine architecting framework of the aircraft jet engine design tool.

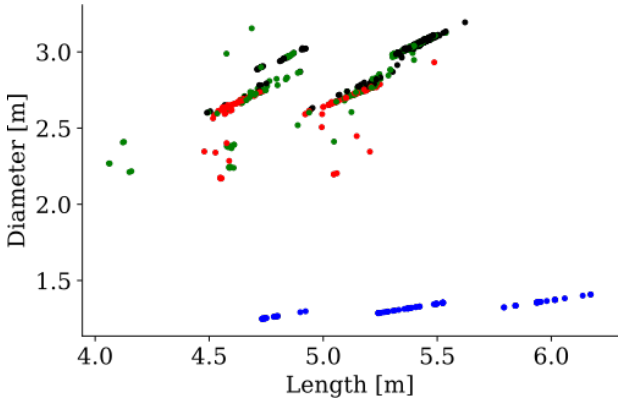
8.3.2 Results for Engine Type

The results of the simple architecting problem generated by the aircraft jet engine architecting tool based on engine type can be seen in Figure 8.2.

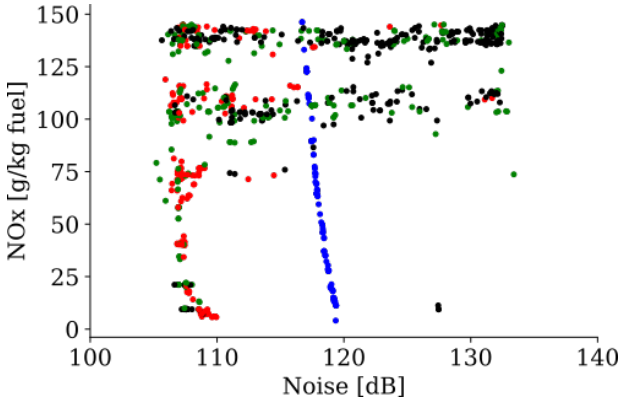
It can be seen that turbojets have higher TSFC and lower weight and lower diameter compared to turbofan engines, which is expected. The same principles applies to the geared turbofans as for the number of shafts: they seem to result in close to the lowest TSFC of all engine architectures, as compressors and fan rotate at optimal speeds, while simultaneously also providing low weight for the turbofan cluster. UHBR turbofans achieve a reduction in TSFC, but this comes at the price of higher weight due to the increased diameter of the engine [65]. It can also be noticed that the optimizer prefers UHBR engines as these specific architectures are generated most often towards later iterations. The lowest TSFC in the entire optimization problem was also achieved by a UHBR turbofan at 6.7 g/kNs, and is indicated in Figure 8.2a by a yellow star. As noise is dependent on jet velocity and nozzle exit area, no significant deduction can be drawn for noise dependency on engine type [76].



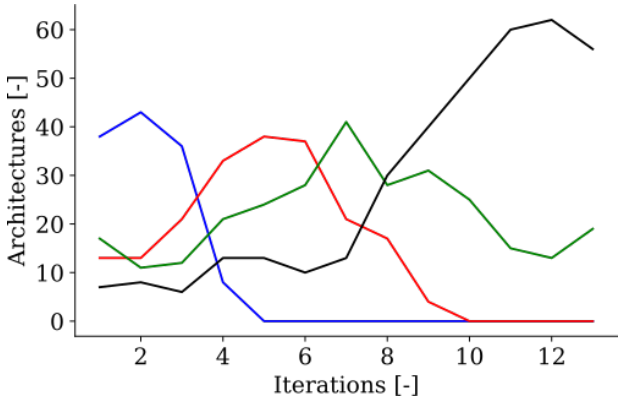
(a) TSFC and weight.



(b) Length and diameter.



(c) Noise and NOx emissions.



(d) Engine architectures over the iterations.

Figure 8.2: Engine results for the simple architecting problem, based on engine type. The yellow star indicates the best performing engine architecture as it is a single-objective problem optimized for TSFC.

Based on Figure 8.2, the results of the simple architecting problem are in line with what was physically expected, serving as additional validation of the engine architecting framework of the aircraft jet engine design tool.

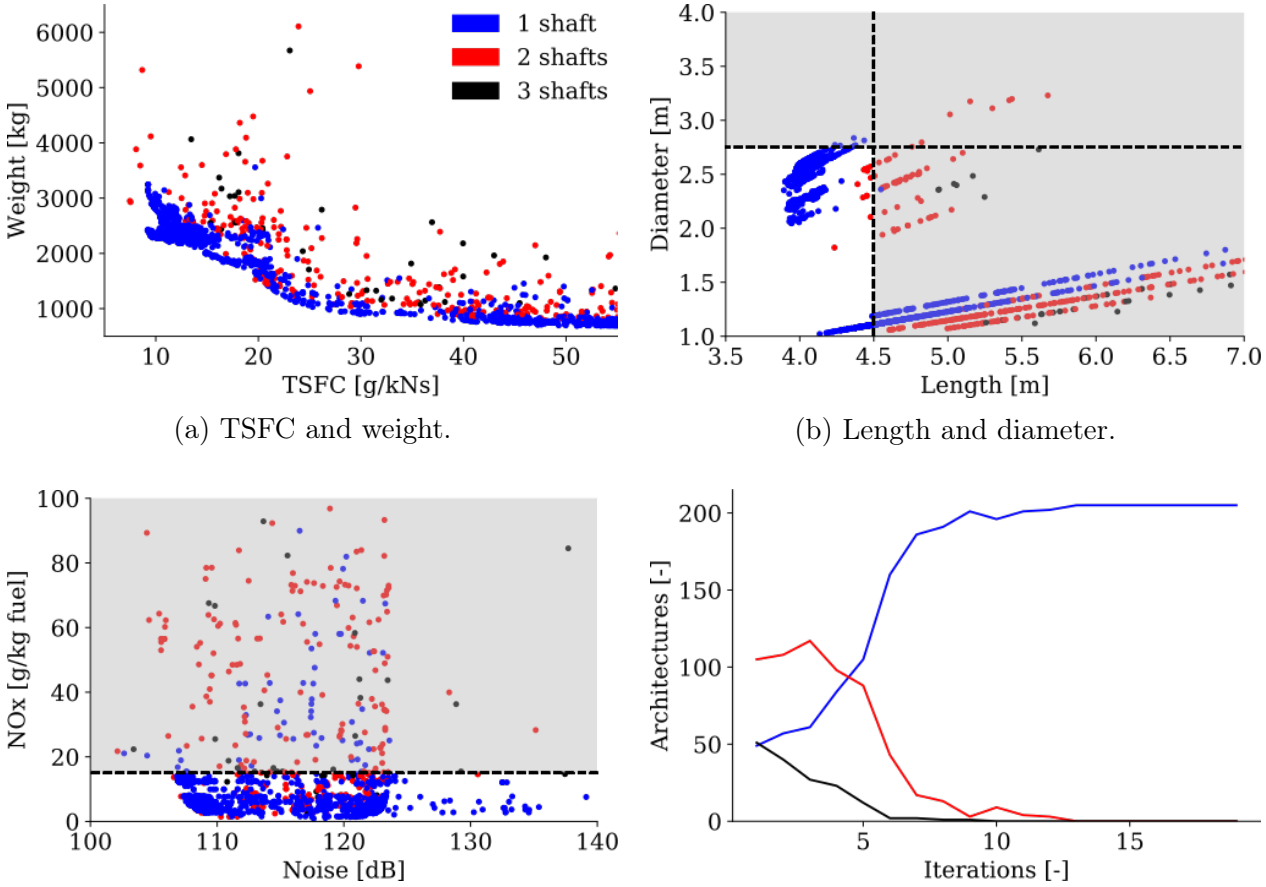
8.4 Realistic Multi-Objective Problem

Finally, a realistic, more complex engine architecting problem is constructed and solved to provide a target Pareto front for future optimization algorithm research. The realistic architecting problem contains three conflicting objectives (TSFC, weight and noise), and multiple constraints (feasibility, geometry, NOx emissions and Mach number of 1). All available decisions discussed in chapter 5 are included in the problem, leading to 43 design variables. This results in the possibility to generate 91 distinct engine architectures and approximately 2.6 million apparent engine design points taking all discrete variables into account. The same design conditions and NSGA-II algorithm are used as for the simple architecting problem. The population size was set to 215 and the termination criteria to 4000 evaluations, to account for the increase in number of design variables.

All architectures were analyzed and their results are presented in subsection 8.4.1, 8.4.2 and 8.4.3. All feasibility constraints, discussed in chapter 5, are satisfied. Approximately 24% of the generated engine architectures converged during the initial DOE, whereas this increased to 98% for the last iteration. This can be seen as the presence of hidden constraints: constraints that are not known a-priori, and are considered violated whenever the simulation did not converge to a meaningful result.

8.4.1 Results for Number of Shafts

The results of the realistic architecting problem generated by the aircraft jet engine architecting tool based on number of shafts can be seen in Figure 8.3.



(a) TSFC and weight.

(b) Length and diameter.

(c) Noise and NOx emissions.

(d) Engine architectures over the iterations.

Figure 8.3: Engine results for the realistic architecting problem, based on number of shafts. The black dotted lines are constraints to be minimized, while the gray areas show infeasible architectures due to imposed constraints.

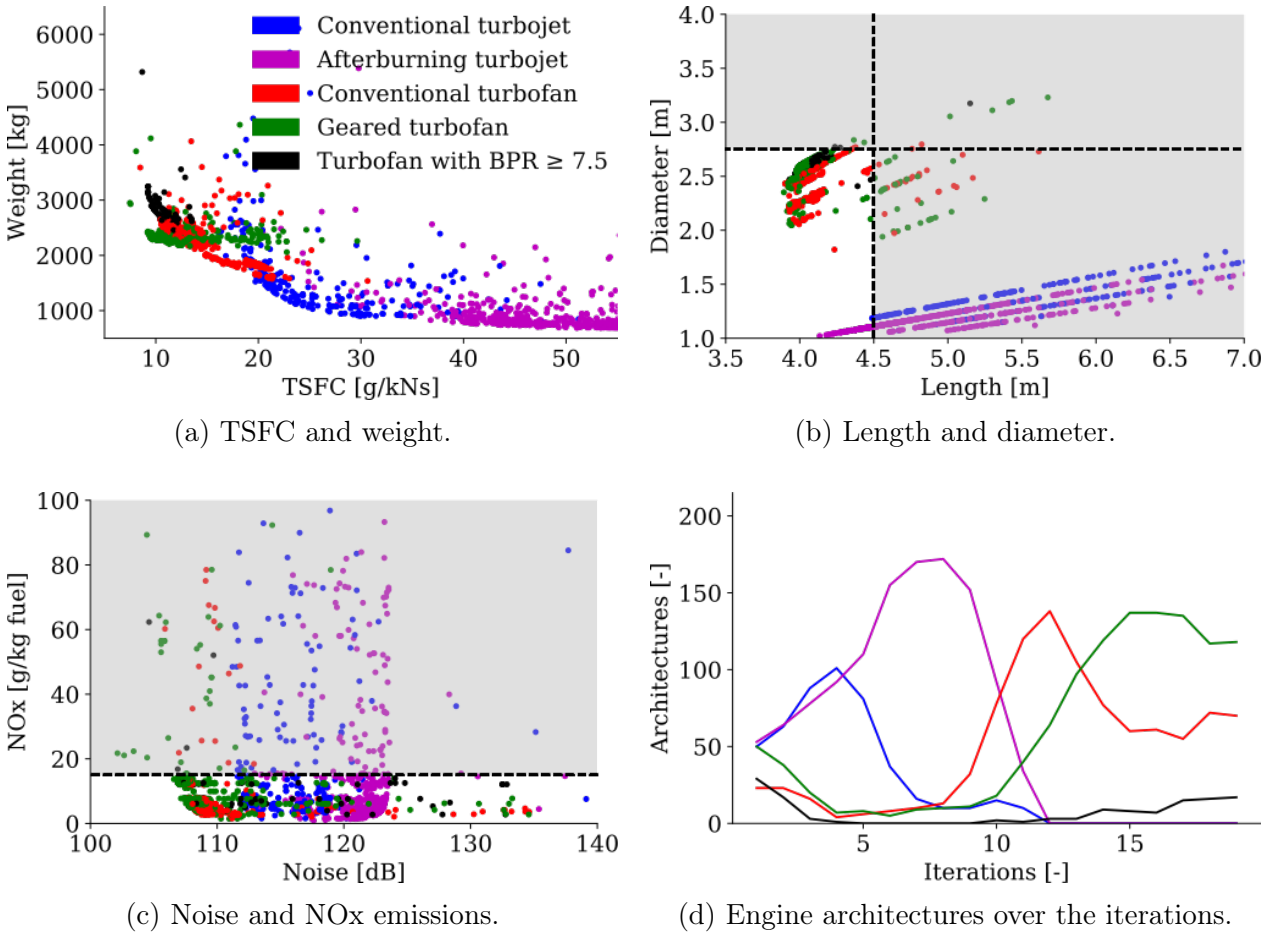
Compared to the results of the simple architecting problem discussed in section 8.3, it can be seen that the discipline results for the aircraft engines are in the same order of magnitude. However, an important difference is that in the realistic problem, the optimizer has a strong preference for single-shaft engine architectures as can be deduced from Figure 8.3d. Three main reasons can be found for this behavior:

- Single-shaft engine architectures in general have a lower length and diameter than double- and triple-shaft architectures, meaning that they have a higher probability of satisfying the geometry constraints imposed on the realistic problem. This effect becomes clear from Figure 8.3b.
- Next to less stringent geometry, single-shaft engine architectures in general also have lower NOx emissions due to a lower OPR, which Figure 8.3c indicates. Therefore, single-shaft engine architectures are more likely to satisfy the NOx constraints.
- Single-shaft engine architectures converge more easily than more complex two- or three-shaft architectures. As non-convergence leads to $+\infty$ results due to the extreme barrier approach, discussed in chapter 4, the optimizer sees these architectures as inferior regarding performance.

However, these results could be caused by assumptions made in the geometry corrections discussed in chapter 6. Therefore, it is advised to perform multiple optimization runs with different corrections and even constraints to verify whether this is actually the case.

8.4.2 Results for Engine Type

The results of the simple architecting problem generated by the aircraft jet engine architecting tool based on engine type can be seen in Figure 8.4.



(a) TSFC and weight.

(b) Length and diameter.

(c) Noise and NOx emissions.

(d) Engine architectures over the iterations.

Figure 8.4: Engine results for the realistic architecting problem, based on engine type. The black dotted lines are constraints to be minimized, while the gray areas show infeasible architectures due to imposed constraints.

When looking at Figure 8.4, some interesting trends can be noticed:

- Most notably, it can be seen in Figure 8.4d that the optimizer starts with a preference for afterburning turbojets making up 80% of the generated engine architectures in iteration 8. The reason for this could be that the afterburning turbojet converged rather easily, which is preferable for the optimizer. However, the afterburning turbojet performs poorly in TSFC and noise performance, which becomes clear from Figure 8.4a and 8.4c, resulting in a decrease of generated afterburning turbojet architectures towards later iterations. From iteration 12 onward, the conventional and afterburning turbojet are not generated anymore by the optimizer.

- At the end of the iterations, the number of generated conventional turbofans is three times higher than HBR turbofans. This was expected as HBR engine result in a higher efficiency and thus lower TSFC at the price of notably higher weight and geometry dimensions. While TSFC and weight are equally important objectives of the realistic engine architecting problem, geometry constraints also play an important role in this case.
- Starting from iteration 13, the geared turbofan engine architecture becomes the most generated architecture by the optimizer. This could have been expected as gearboxes let the fan and LPC rotate at optimal speeds while keeping the weight fairly constant, as discussed in chapter 5. However, the complexity of the system increases but this was not taken into account by the aircraft jet engine architecting tool.

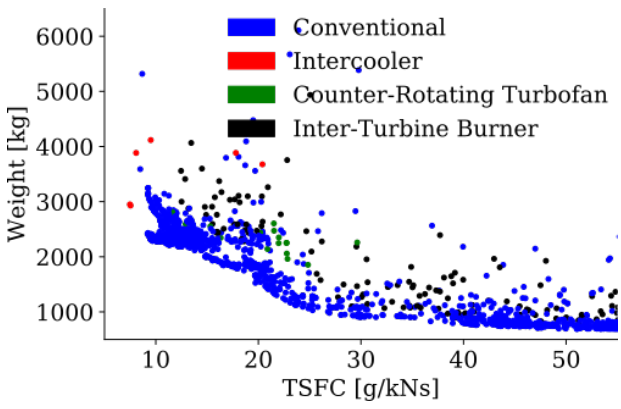
8.4.3 Results for Innovative Engine Configurations

The results of the simple architecting problem generated by the aircraft jet engine architecting tool based on innovative engine configurations can be seen in Figure 8.5. Note that this subsection was not present for the simple architecting problem as innovative engine configurations choices were not activated there by the tool user.

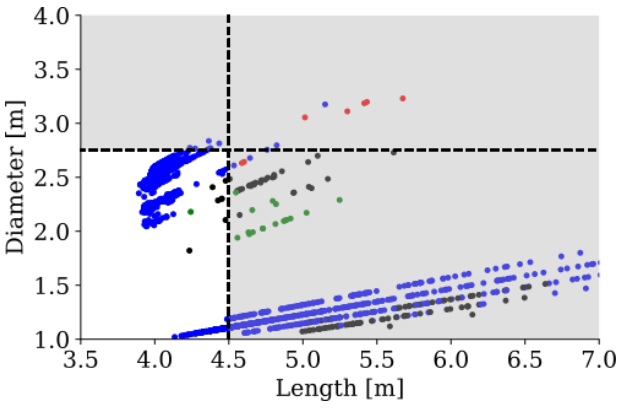
The results indicate that the optimizer has a preference for conventional turbofans which clearly comes forward in Figure 8.5d. However, this is in line with the expectations:

- A counter-rotating fan usually increases TSFC and weight whereas only reducing noise slightly [48];
- An inter-turbine burner reduces TSFC compared to an afterburner, however also increases engine weight significantly [38];
- An intercooler reduces both TSFC and NOx emissions, however intercooler implementation convergence issues were encountered during the optimization [34].

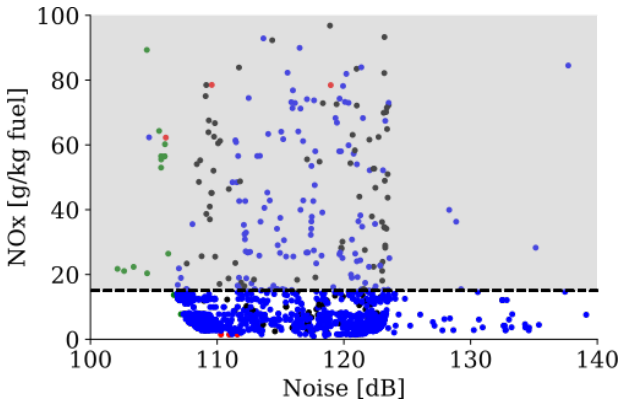
Again, these results have to be interpreted with caution. As innovative technologies are used, it might be that the implementation of these technologies in the aircraft jet engine architecting tool is not optimal, meaning that the tool cannot yet predict their performance accurately. To mitigate this, more in-depth research and high-fidelity analysis should be performed and the knowledge gained should be implemented in the tool.



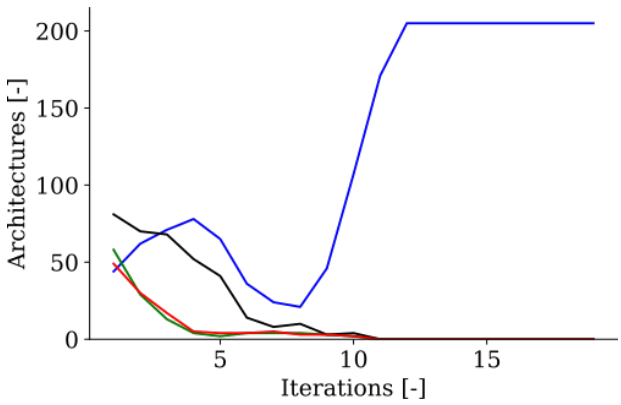
(a) TSFC and weight.



(b) Length and diameter.



(c) Noise and NOx emissions.



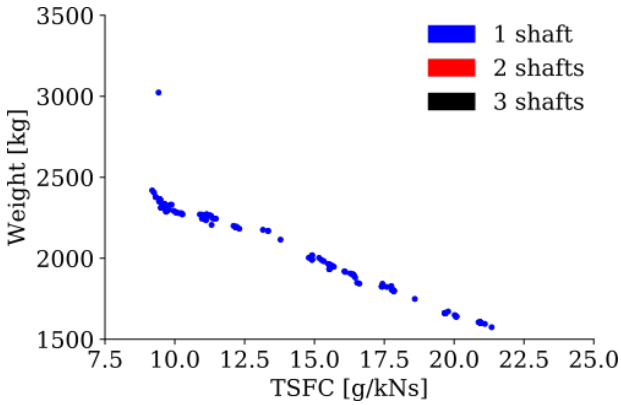
(d) Engine architectures over the iterations.

Figure 8.5: Engine results for the realistic architecting problem, based on innovative configurations. The black dotted lines are constraints to be minimized, while the gray areas show infeasible architectures due to imposed constraints.

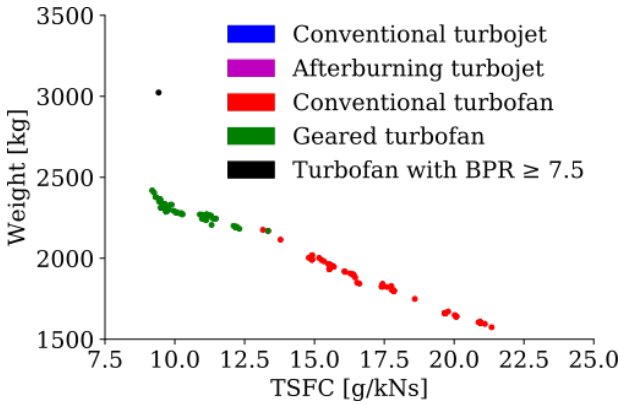
8.5 Pareto Front

With the results of the realistic multi-objective engine architecting problem, the Pareto front with non-dominated engine architectures can be created. These can be seen in Figure 8.6 and are subdivided in the same sections as section 8.4.

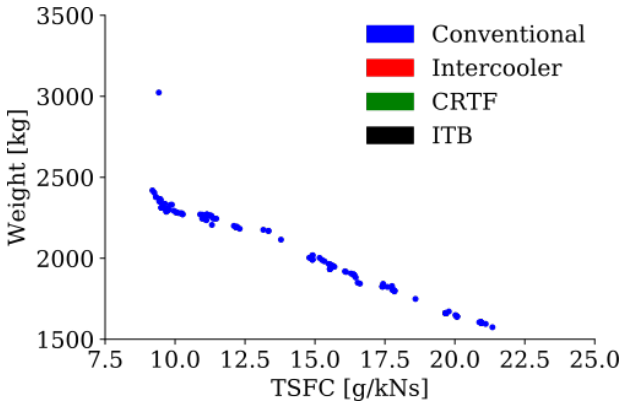
It must be noted that all constraints regarding feasibility, geometry and NOx emissions imposed on the realistic problem have been satisfied for engine architectures part of this Pareto front. As was already concluded in section 8.4, the results indicate that the optimizer has a preference for single-shaft conventional turbofans. Furthermore, it becomes clear that the designer of the aircraft engine system will have to make a trade-off between TSFC and weight as no single best design can be distinguished, i.e. each point has a different combination of optimum TSFC and weight performance. In case low TSFC is most important, a single-shaft conventional geared turbofan architecture seems to be the best choice while this becomes a conventional turbofan when weight becomes more dominant.



(a) Based on number of shafts.



(b) Based on engine type.



(c) Based on innovative engines.

Figure 8.6: Pareto front of TSFC and weight for the realistic engine architecting problem. From this front, it can be concluded that the aircraft jet engine architecting tool has a preference for single-shaft conventional engine architectures.

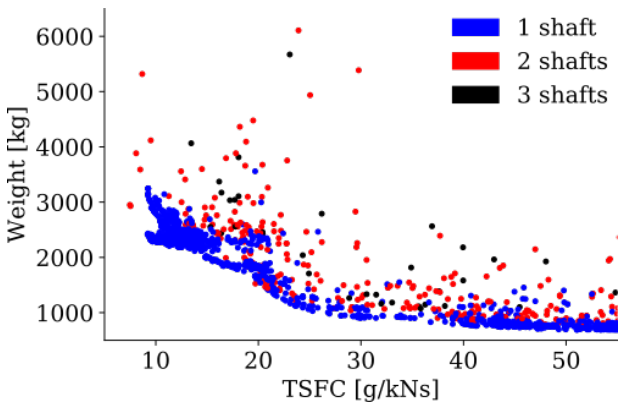
It must be noted that the results could be the result of assumptions in empirical correlation corrections, meaning that further research needs to be performed on the exact influence of shaft number on weight and geometry dimensions. This was done with sensitivity studies which is the topic of the next section.

8.6 Weight & Length Sensitivity Study

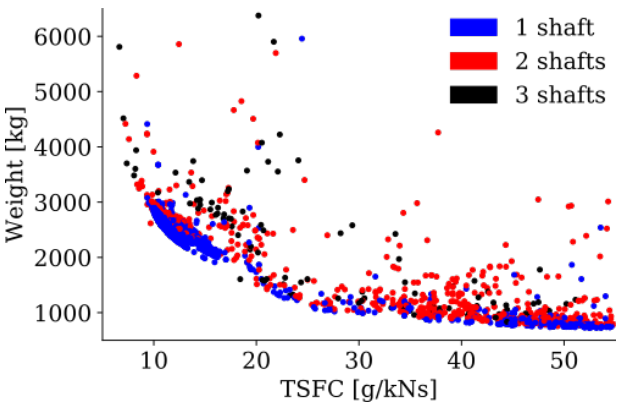
In this section, sensitivity studies will be performed on the weight and geometry assumptions of the original engine architecting problem. As discussed in chapter 6, a correction of 10% was applied for the weight and length disciplines when differing from the standard two-shaft engine architecture. This correction is changed to 5% to see the effects on the optimization process and results. The results will be analyzed based on number of shafts, engine type and innovative engine configurations. Next to that, the Pareto front of both cases will be investigated as well.

8.6.1 Results for Number of Shafts

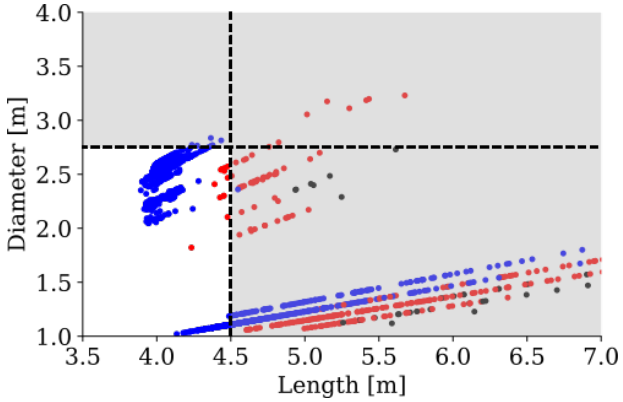
First, the effect on number of shafts is analyzed. This can be seen in Figure 8.7, where the left figures are the results of the original case while the right figures are the results for the adjusted weight and length disciplines.



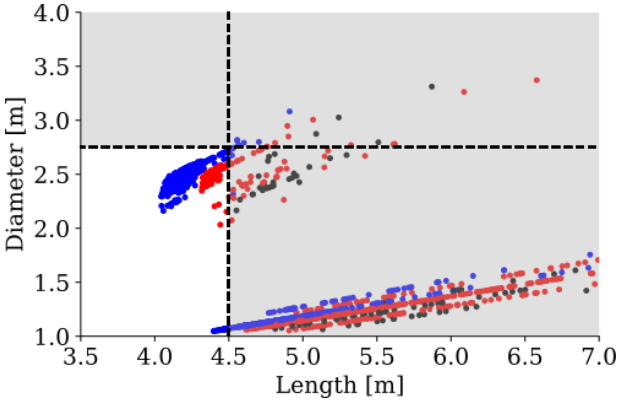
(a) TSFC and weight originally.



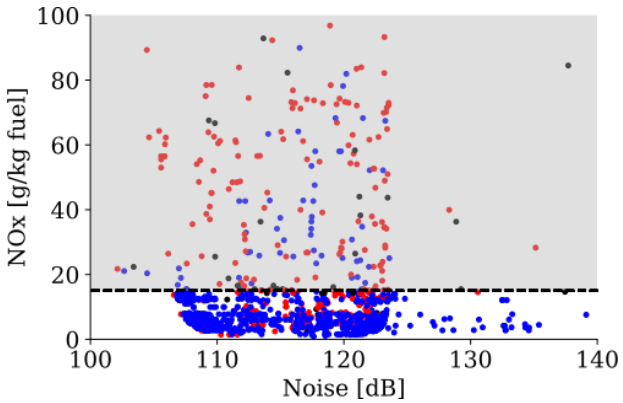
(b) TSFC and weight when adjusted.



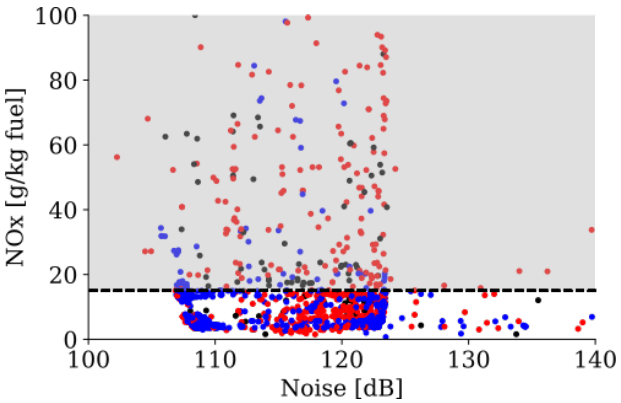
(c) Length and diameter originally.



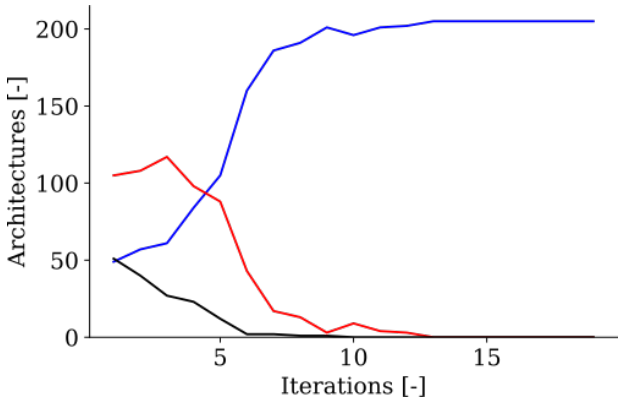
(d) Length and diameter when adjusted.



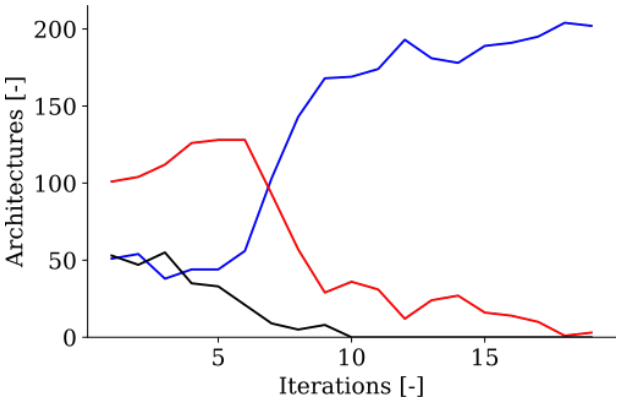
(e) Noise and NOx emissions originally.



(f) Noise and NOx emissions when adjusted.



(g) Architectures over the iterations originally.



(h) Architectures over the iterations when adjusted.

Figure 8.7: Comparison of the results of the original 10% weight and length addition per shaft to the results of the adjusted 5% weight and length addition per shaft based on number of shafts.

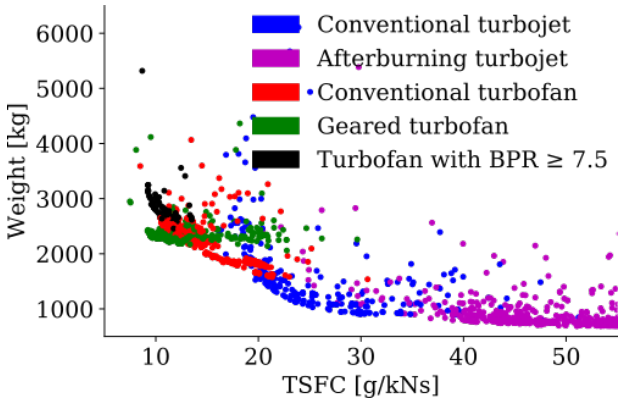
From Figure 8.7g and 8.7h, it can be concluded that approximately the same trends emerge for the adjusted 5% correction for the weight and length disciplines as for the original 10%: the optimizer still has a preference for a single-shaft engine architecture. As discussed before, the reason for this could be that single-shaft architectures tend to satisfy the geometry constraints more often and result in a relatively low weight compared to architectures with two or three shafts. However, two-shaft engine architectures are created more often in the adjusted discipline

case than for the original case proving that the correction does have an effect on the optimization process.

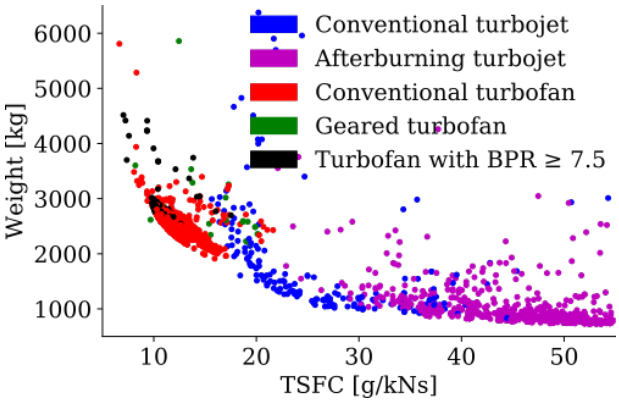
8.6.2 Results for Engine Type

Second, the effect on engine type is analyzed. This can be seen in Figure 8.8, where the left figures are the results of the original case while the right figures are the results for the adjusted weight and length disciplines.

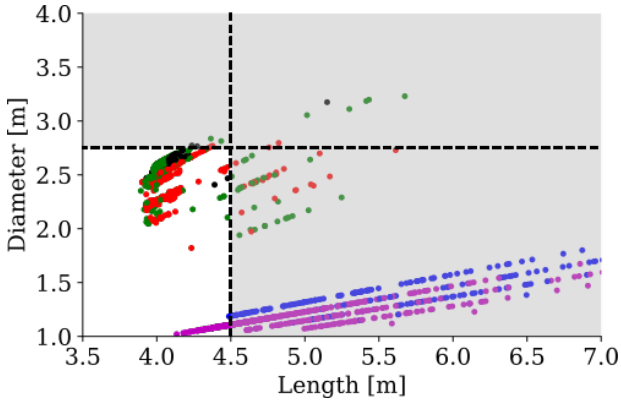
Contrary to the effect on number of shafts, some noteworthy differences can be noticed of the discipline correction adjustment on engine type. Whereas in the original case the optimizer has a preference for geared turbofan architectures, in the adjusted case the optimizer is clearly generating more conventional two-shaft turbofan architectures which are replacing the geared turbofan architectures from iteration 10 onward. This could be due to the fact that discipline results are based on empirical correlations, for which high-fidelity analysis should be performed to confirm or correct the estimations. Next to that, it can be seen that the conventional and afterburning turbojet trends of the adjusted case are relatively similar to the original case, except at iteration 4 where less conventional turbojets were created in the former case.



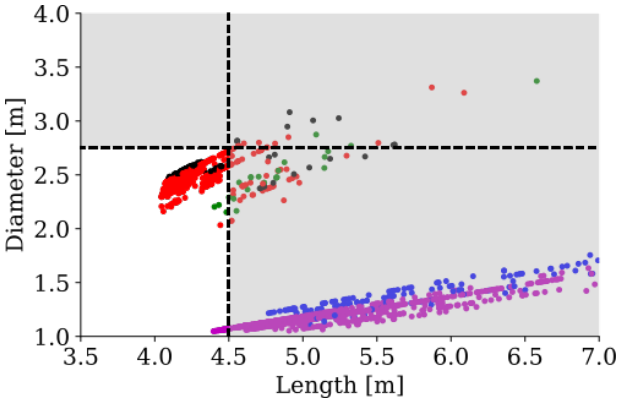
(a) TSFC and weight originally.



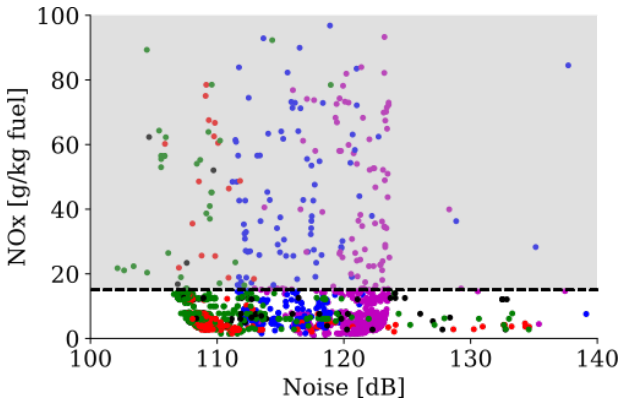
(b) TSFC and weight when adjusted.



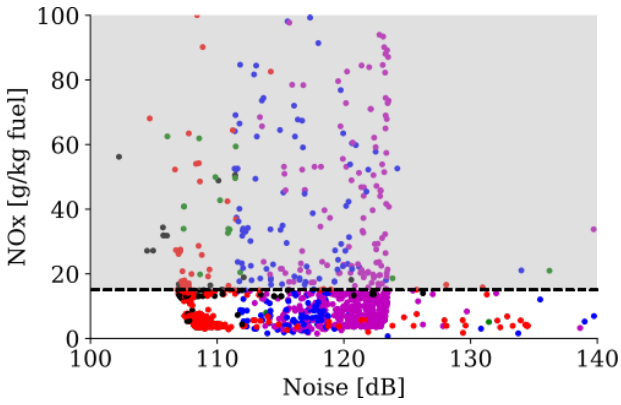
(c) Length and diameter originally.



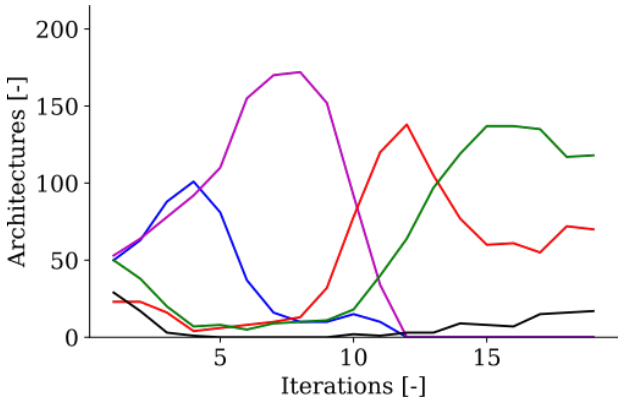
(d) Length and diameter when adjusted.



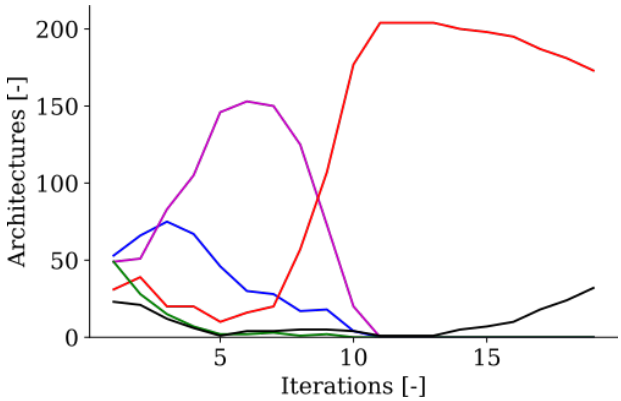
(e) Noise and NOx emissions originally.



(f) Noise and NOx emissions when adjusted.



(g) Architectures over the iterations originally.

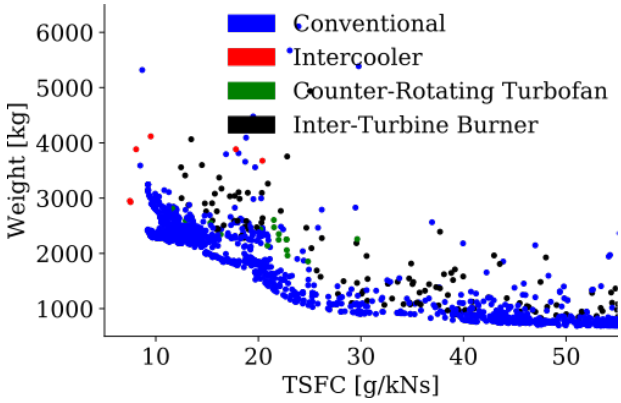


(h) Architectures over the iterations when adjusted.

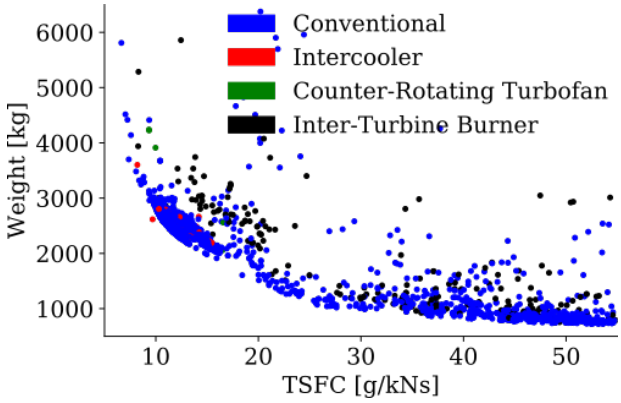
Figure 8.8: Comparison of the results of the original 10% weight and length addition per shaft to the results of the adjusted 5% weight and length addition per shaft based on engine type.

8.6.3 Results for Innovative Engine Configurations

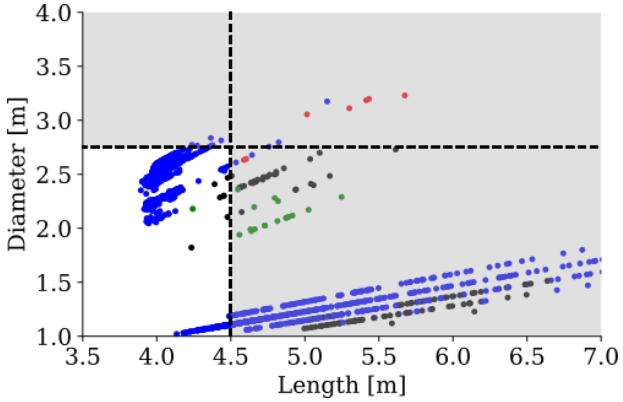
Third, the effect on innovative engine configurations is analyzed. This can be seen in Figure 8.9, where the left figures are the results of the original case while the right figures are the results for the adjusted weight and length disciplines.



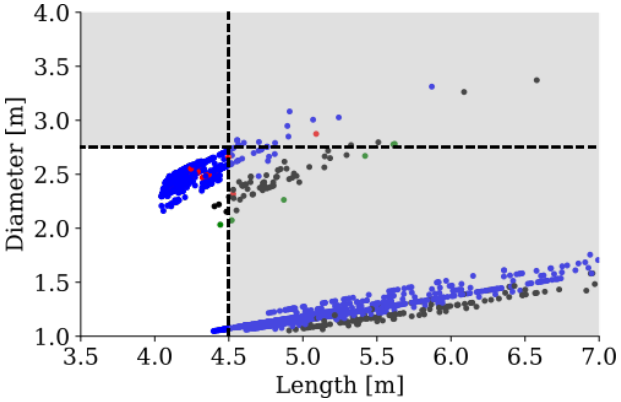
(a) TSFC and weight originally.



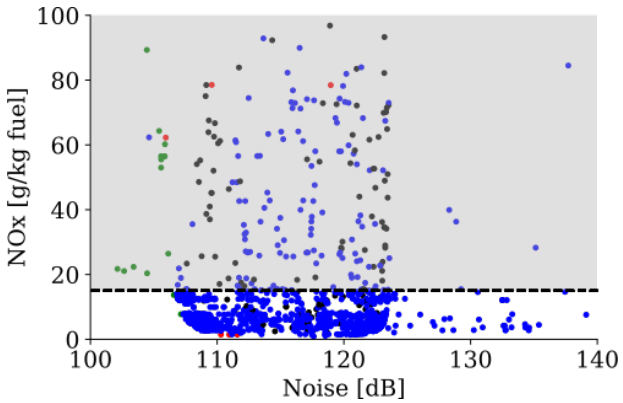
(b) TSFC and weight when adjusted.



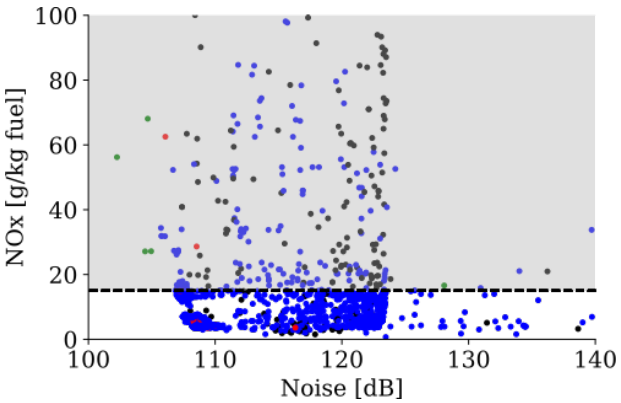
(c) Length and diameter originally.



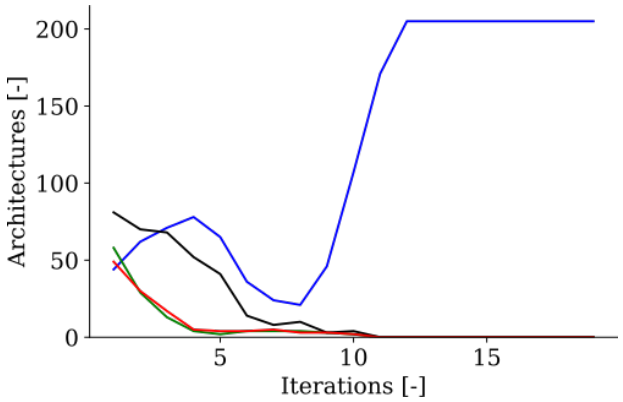
(d) Length and diameter when adjusted.



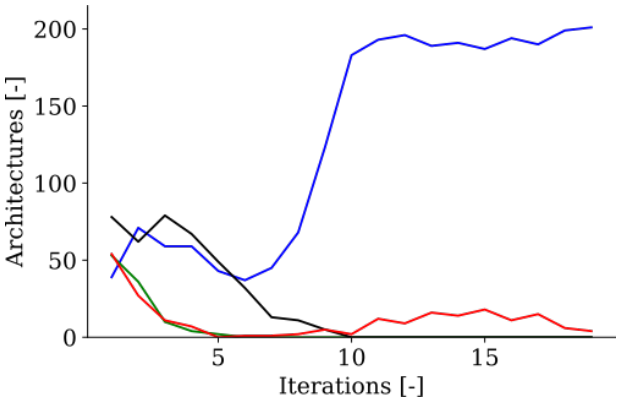
(e) Noise and NOx emissions originally.



(f) Noise and NOx emissions when adjusted.



(g) Architectures over the iterations originally.



(h) Architectures over the iterations when adjusted.

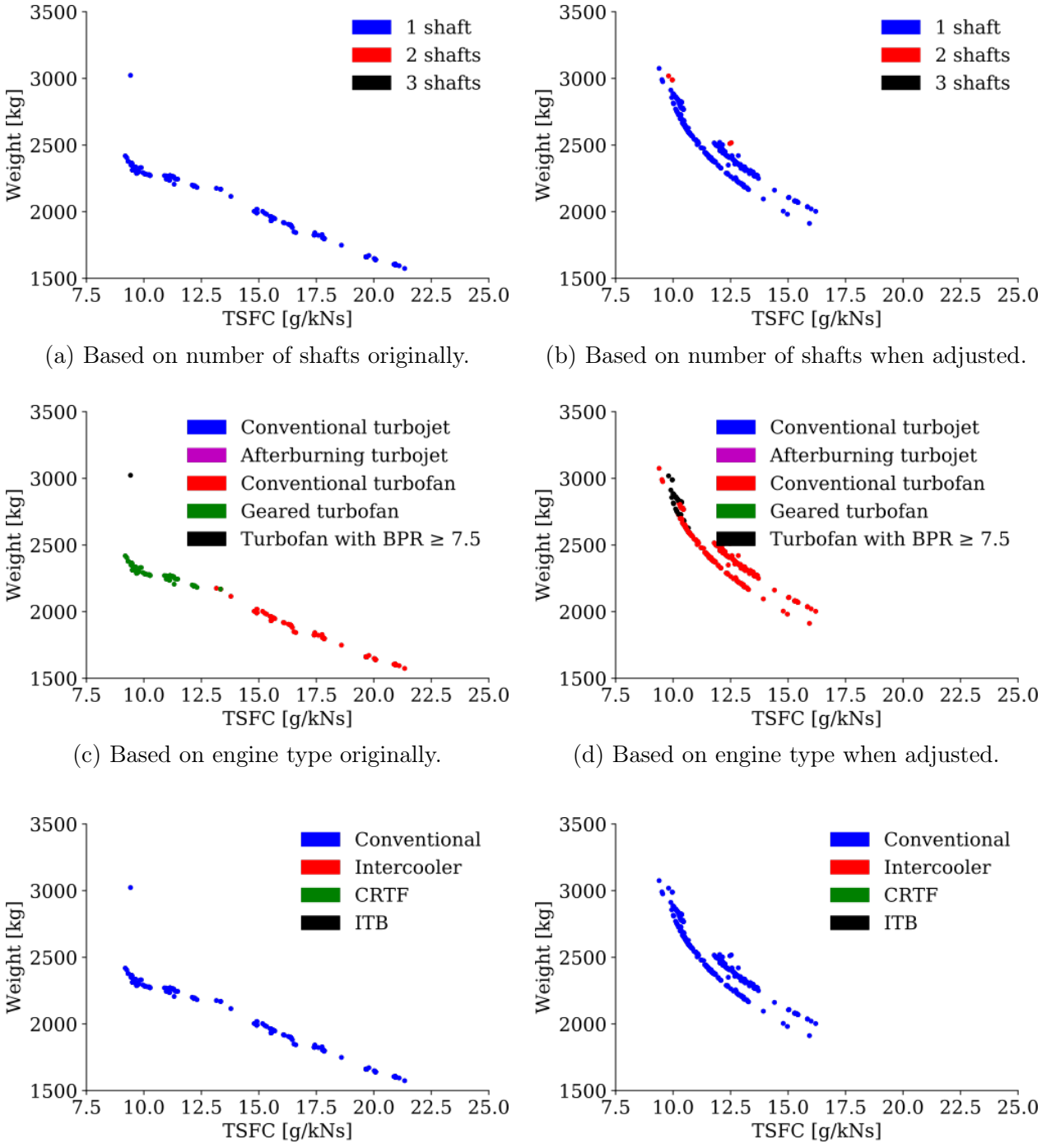
Figure 8.9: Comparison of the results of the original 10% weight and length addition per shaft to the results of the adjusted 5% weight and length addition per shaft based on innovative engine configurations.

Just like for the effects on number of shafts, the difference between the original and adjusted discipline case is minor for the effects on innovative engine configurations: at the start of the iterations the optimizer tries out some innovative engine configurations, but these are rather quickly abandoned for conventional configurations. However, it must be noted that in the adjusted case, the optimizer does try out some more intercooler configurations toward later configurations. Again, as discussed before, these results should be used with caution as the

exact effects of these innovative engine technologies on engine performance might not be entirely correctly estimated with the aircraft jet engine architecting tool.

8.6.4 Pareto Front

Finally, the effect on the Pareto front is analyzed. This can be seen in Figure 8.10, where the left figures are the results of the original case while the right figures are the results for the adjusted weight and length disciplines.



(a) Based on number of shafts originally.

(b) Based on number of shafts when adjusted.

(c) Based on engine type originally.

(d) Based on engine type when adjusted.

(e) Based on innovative engines originally.

(f) Based on innovative engines when adjusted.

Figure 8.10: Comparison of the Pareto front of the original 10% weight and length addition per shaft to the results of the adjusted 5% weight and length addition per shaft.

The results show that the Pareto front of the original and adjusted shows some similarities and differences. The largest part of the Pareto front is made up by single-shaft conventional turbofans in both cases and the lowest TSFC is at approximately 8.5 g/kNs. However, in the adjusted case, a couple two-shaft engine architectures are present in the Pareto front while this was only single-shaft architectures in the original case. Furthermore, the conventional turbofan dominates more while the geared turbofan architecture is not present anymore in the adjusted case than in the original case. Finally, the Pareto front of the adjusted case is much more "concentrated" and higher weights are achieved, indicating that the shape and composition of the front change based on assumptions made in the empirical correlations of the tool disciplines.

8.7 Result Discussion

From the results, it becomes clear that multiple aircraft engine architectures can effectively be modeled and compared by the aircraft jet engine architecting tool incorporating several disciplines of aircraft engine design. This was achieved by the implemented system architecting approach which allows the inclusion of both continuous and discrete architecting decisions leading to a systematic exploration of the entire design space. With the combination of continuous and discrete design variables, a design space was created which contains different distinct engine architectures such as conventional or geared turbofans and even innovative engine architectures with an intercooler or counter-rotating fan. Not only was the tool able to create these architectures, it was also able to obtain realistic values for the different disciplines and to show the relevant effects of engine architecting decisions. A UHBR turbofan, for example, can result in a lower TSFC as desired but this comes at the price of higher weight and geometry dimensions. The system designer will therefore have to make trade-offs when selecting the final architecture for the system. Furthermore, the optimizer managed to create a Pareto front with non-dominated aircraft engine architectures which satisfy all constraints of engine architecting problem.

Therefore, the aircraft jet engine architecting tool based on the created system architecting approach solves several the challenges of system architecting optimization laid out in chapter 2:

- Optimization: mixed-discrete and multi-objective capabilities are included in the architecting framework, while also providing a solution to deal with design variable hierarchy. Next to that, relevant architecting decisions and new technological developments can be included in the optimization process. Next to that, the effect of these important design decision can be assessed early on in the design process of the system. Therefore, different system architectures can be generated and their performance compared fairly.

- Benchmark problem: the aircraft jet engine optimization problem can be used as realistic benchmark problem to support the development, evaluation and comparison of system architecture design space modeling methods and optimization algorithms. Next to that, various stakeholders can be educated on all the relevant aspects of architecture optimization with the engine architecting problem. These stakeholders include disciplinary experts, analysis tool developers, system architects, system engineers, project managers, and engineering students.

From the weight and length sensitivity studies, it was found that assumptions in empirical correlations for the disciplines can have a noteworthy effect on the results of the optimization: the optimizer starts having preferences for different engine architectures and the Pareto front

changes in shape. Although it was not the main objective of the thesis to optimize aircraft engines, it is advised to perform research on these empirical correlations and, if possible, change them by high-fidelity analyses such as CFD or modeling & simulation software to obtain more trustworthy results. Furthermore, it is important to perform in-depth research on the effect of innovative technologies on aircraft engine performance and investigate whether these effects are correctly predicted by the aircraft jet engine architecting tool. A drawback of these high-fidelity analyses, however, is that computational time per engine architecture could increase significantly. Therefore, in this case the system architecture designer will have to make a trade-off between accuracy and computational time, and even in which stages of the design process they want to use the tool.

Summary: Research Question Answers

RQ1 How is the system architecting approach implemented in aircraft engine modeling?
RQ1.2 Which limitations does the system architecting approach have?

> Not all aircraft jet engine architectures can be modeled yet, such as variable area nozzles and dual cycles. Next to that, the accuracy of the engine disciplines is limited to the accuracy of the implemented discipline empirical correlations. This also means that the effects of innovative engine configurations such as a counter-rotating fan and intercooler might not be entirely correctly estimated by the tool. Therefore, additional research is required on these topics. Finally, convergence issues, especially in off-design conditions, are still posing an important problem which should be resolved in future thesis research.

RQ2 How does the approach address the nature of architecting decisions?
RQ2.4 Can the relevant effects of engine architecting decisions be captured?

> Through the combination of continuous and discrete design variables, multiple distinct engine architectures and design points can be created and assessed in the design space. This provides the system designer with important information on the performance of different engine architectures in every stage of the design process. Therefore, the relevant effects of engine architecting decision can be captured.

RQ3 Does the approach enable the creation of multiple optimal aircraft engine architectures?
RQ3.1 Which engine objectives and constraints are used?

> The simple architecting problem only had TSFC as objective and feasibility constraints. The realistic architecting problem had TSFC, weight and noise as objectives. Its constraints were geometry, NOx emissions and a Mach number of 1 next to the feasibility constraints. It must be noted that the user of the system can set these objectives and constraints themselves.

RQ3.2 Which existing and novel engine architectures are part of the solution space generated by the approach?

> Existing engine architectures generated by the approach were a conventional and afterburning turbojet, and a conventional, geared and (U)HBR turbofan. Novel engine architectures were architectures which included an intercooler, counter-rotating fan and/or inter-turbine burner.

RQ3.3 Can a Pareto front be formed with non-dominated engine architectures?

> A Pareto front has clearly formed as can be seen in Figure 8.6 and 8.10. It must be noted that the shape and composition of the Pareto front change based on assumptions made in the empirical correlations of the disciplines included in the aircraft jet engine architecting tool.

Chapter 9
Conclusions & Recommendations

The thesis objective was to further research and educate stakeholders on system architecture modeling & optimization, by creating a benchmark problem with an application to aircraft engine architectures using a system architecting approach with mixed-discrete & multi-objective capabilities. In this final chapter of the thesis paper, the conclusions on this objective are reviewed and presented. Furthermore, recommendations for future research are discussed.

9.1 Conclusions

Even though systems engineering has already been used to design very complex systems and their mission for many years, it still has some challenges that need to be tackled. The architecture optimization of a system is a mixed-discrete, multi-objective and hierarchical problem. Next to that, due to the improvement of existing and the development of new technologies, the range of architectural options keeps increasing. Therefore, it is important to be able to analyze different designs and make a fair trade-off between them in order to make well-considered design choices. However, relevant architecting decisions are usually taken early on in the design process without exact knowledge of their effect on the final design. In order to cope with this issue, system experts are consulted but their advice might be biased or subjective. It becomes clear that these system architecting characteristics make for a class of challenging optimization problems. Therefore, there is a need for a system architecting approach which can generate distinct system architectures and evaluate them, and for optimization algorithms capable of dealing with this kind of optimization problems. In order to do this, a benchmark problem is required to support the development & evaluation of optimization algorithms and the education of system architecture optimization stakeholders. However, a thorough review of literature and state of the art indicated that no such benchmark architecture optimization problem currently exists.

Therefore, a benchmark problem with an application to aircraft jet engine architecture modeling was created during the thesis research which exhibits the black-box, mixed-discrete, multi-objective and hierarchical nature of system architecture optimization. The benchmark problem was tackled by developing an MDO tool using a system architecting approach capable of dealing with these characteristics. The tool was built using the pyCycle and OpenMDAO software in order to generate and analyze the thermodynamic cycles of a wide array of aircraft gas turbine engine architectures. Next to the thermodynamic cycle, other discipline estimation methods were also implemented including engine weight, geometry, NOx emissions and noise. The tool consists of two components: the *Engine Architecture Evaluator* and the *Engine Architecting Framework*. The former analyzes one specific architecture using pyCycle and OpenMDAO, whereas the latter enables the definition of the architecting problem and the translation of design vectors into valid architecture definitions. Furthermore, the architecting problem allows the inclusion of many architecting choices to dynamically tune the difficulty of the architecting

problem to solve such as the inclusion of a fan and multiple shafts. With the different design choices implemented in the aircraft jet engine architecting tool, resulting in 43 design variables, a total of 91 distinct engine architectures and approximately 2.6 million engine apparent design points can be generated taking all discrete variables into account. Furthermore, the user of the tool has the possibility to select certain objectives and constraints to guide the design space exploration.

The combination of the *Engine Architecture Evaluator* and the *Engine Architecting Framework* should give a good introduction to how an architecture optimization problem is implemented in practice. It must be considered how to define and analyze individual system architectures, how to specify the architecture optimization problem (i.e. its design vector, objectives, and constraints), and how to translate between design vector (as generated by the optimization algorithm) and the architecture definition (as needed for analysis). The *Engine Architecture Evaluator* additionally provides a good example for how multidisciplinary analysis toolchains should be implemented for a system architecture optimization. The aircraft jet engine architecting tool could help educate disciplinary experts, analysis tool developers, system architects, system engineers, project managers, and engineering students on all the relevant aspects of architecture optimization problems.

The created tool was verified by comparing engine analysis results with pyCycle example problems, and by running two engine architecting problems of different complexity level. Next to that, validation showed that the discipline results of the tool were very similar to actual engine data of the CFM LEAP-1C and the Pratt & Whitney F100 engines used on the Comac C919 and F16 aircraft, respectively. The code of the tool presented in this paper is available at [39]. A simple single-objective and a more realistic multi-objective engine architecting problem were created to test the optimization capabilities of the tool, and thus also the implemented system architecting approach containing mixed-discrete and multi-objective capabilities.

The optimizer managed to find multiple optimal aircraft engine architectures, ultimately forming a Pareto front, which satisfied all implemented discipline and feasibility constraints. It was found that both the simple and realistic engine architecture problem are subject to non-convergence issues in their design spaces: in the initial DOE only 49% and 33% of design points result in a converged simulation, respectively. However, even with these hidden constraints, the results of both problems showed trends which were in line with what was physically expected, such as decreasing TSFC and increasing weight for a higher number of shafts in the engine, serving as additional validation of the aircraft jet engine architecting tool. In general, the optimizer showed a preference for single-shaft conventional turbofan engine architectures for the realistic engine architecting problem.

It is expected that the aircraft jet engine design problem provides a realistic engineering architecting problem to test optimization algorithms on. The analysis time of one engine architecture should not be prohibitive, such that also relatively inefficient algorithms can be fairly compared to more suitable algorithms. Due to the easy definition of the architecting problem and the release as an open-source package, it is expected that setting up experiments for testing optimization algorithms should be trivial to do. Finally, the Pareto front of the realistic architecting problem can be used to compare results of developed optimization algorithms.

9.2 Recommendations

As discussed throughout the thesis paper, there are still some recommendations for both the system architecting approach and its implementation in the aircraft jet engine benchmark problem. Therefore, the author recommends the following:

1. At the moment, the ordering of the architecting decisions is performed by hand by the designer of the optimization code. This is based on the dependency of certain decisions on other architecting decisions, such as the counter-rotating fan activity based on the fan presence. To make the tool more flexible and user-friendly, it is recommended to implement code which can create the order of these architecting decisions itself. Furthermore, this automatic ordering is expected to reduce the code setup time and human coding mistakes.

2. As discussed in chapter 6, the engine components are created with components from the pyCycle library. However, these components use simplified thermodynamic equations to estimate the performance of the components and connect them to each other, resulting in a complete engine thermodynamic cycle. By reviewing these equations and possibly replacing them by more in-depth relations or high-fidelity analysis tools, the accuracy of the components and thus the entire cycle could be improved.

3. The pyCycle components used to construct the engine thermodynamic cycle only include thermodynamic sizing of engine components; mechanical sizing, such as the determination of the number of compressor and turbine rotors and stators, is not included. By including this mechanical sizing, the tool could become more all-inclusive.

4. Even though the aircraft jet engine architecting tool already includes several innovative technologies such as a counter-rotating fan and inter-turbine burner, other researched technologies such as open rotor [83] and electrical or hydrogen propulsion [8] concepts could be included in the design space to give the system designer a better overview of the complete design space.

5. The boundaries of the design variables should be investigated in detail to help ensure that only feasible engine architectures can be created. Examples of these design variables include the engine BPR, gearbox gear ratio and cooling bleed offtake. Especially for BPR, its upper bound could be significantly increased by making sure that the relevant effects of UHBR engines are modeled correctly.

6. Assumptions were made for the efficiencies of the aircraft engine components. However, with new and improved technologies, these efficiencies should be updated to match the current component efficiencies as this will have a noteworthy impact on the performance of the engine and its disciplines.

7. Additional aircraft engine disciplines should be researched and included in the aircraft jet engine architecting tool to make the design and optimization process more inclusive. Potential disciplines include aerodynamic analysis (nacelle drag) [84], engine development & manufacturing cost [85, 86], technology & system readiness level [87, 88], contrail forming [89], etc.

8. To ensure that the aircraft engine is mechanically feasible and complies with the different engine regulations, additional constraints such as component thermal and vibrational

limits should be added to the benchmark problem to make the optimization process even more realistic. At this point, an engine could be created which is not mechanically feasible due to for example vibrational issues, or does not comply with all regulations including damage tolerance and rotor blade containment [90].

9. The aircraft jet engine architecting tool evaluates the engine disciplines with empirical correlations and correction assumptions, ultimately limiting the accuracy of the results. The P3T3 method implemented to calculate NOx emissions, for example, might not be directly applicable for the CFM LEAP-1C engine used in the aircraft engine benchmark problem [71]. To improve this accuracy and obtain more trustworthy results, it is advised to replace these empirical correlations by either up-to-date empirical correlations or high-fidelity analyses such as CFD or modeling & simulation software such as Dymola [91].

10. Continuing on the previous recommendation, a trade-off should be made between the accuracy of the results and the computational time and resources required to obtain the results. The system design phase and exact purpose of the optimization should be taken into account when making the trade-off.

11. A multi-point design strategy should be correctly implemented. At the moment, the aircraft engine can only be created and analyzed for on-design conditions. The performance assessment for off-design conditions is not yet possible due to non-convergence issues encountered in the analysis process. It was found that this is the result of non-optimal starting values for the convergence; convergence is (almost) only achieved when starting values are chosen which lie very closely to the final values. As these off-design conditions are paramount for the evaluation of engine performance in the entire flight profile, these off-design condition problems should be investigated and resolved. Recently, Diamantidou et al. suggested using a surrogate-assisted multi-point synthesis approach which samples the starting values in order to improve convergence [92].

12. Currently, the provided *Engine Architecting Framework* does not provide any interfaces to upstream MBSE processes, like stakeholder identification and requirements definition. Because of the modular nature of the benchmark problem, the *Engine Architecting Framework* can be replaced by a generally applicable method for modeling architecture design spaces, while still being able to use the engine architecture evaluator to analyze specific engine architecture instances. Next to that, the *Engine Architecture Evaluator* could be replaced by a system architecture design platform called ADORE for which an introduction is provided in Appendix F. Future research into the best way of integrating system architecture optimization with the MBSE process is needed.

13. As the aircraft engine has to be incorporated in the aircraft airframe, the integration effects of this process could be implemented in the aircraft jet engine architecting tool. In that case, the created software might be added to existing MDO and/or Knowledge-Based Engineering (KBE) processes in order to design the entire aircraft.

14. To get an idea of the different parts and overall size of the engine, KBE might be used to create a CAD rendering of the engine. This could then be used to show stakeholders such as engine manufacturing company management how the different aircraft engines compare to each other and to get more accurate weight and geometry estimations. A suitable option is ParaPy which specializes in the automation of these design tasks and enables CFD & FEM analyses on the designs [93]. However, a parametric CAD might also be sufficient for a simplified engine visualization.

Bibliography

Books

[2] Edward F. Crawley, Bruce G. Cameron, and Daniel Selva. *System architecture: strategy and product development for complex systems*. Pearson education Limited, 2016.

[7] Arvind G. Rao, J.P. van Buijtenen, and W.P.J. Visser. *Aero Engine Technology, AE4-238*. Delft University of Technology, Aug. 2018.

[15] Steven R. Hirshorn. *Systems Engineering Handbook*. NASA, 2007.

[17] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufman Publishers, 2012.

[19] Jaroslaw Sobieszczanski-Sobieski et al. *Multidisciplinary design optimization supported by knowledge based engineering*. John Wiley & Sons, 2015.

[21] Mark French. *Fundamentals of Optimization: methods, minimum principles and applications for making things better*. Springer, 2018.

[27] Gordon C. Oates. *The Aerothermodynamics of aircraft gas turbine engines*. National Technical Information Service, 1978.

[28] ICAO. *Annex 16: Environmental Protection: Volume I - Aircraft Noise*. July 2017.

[29] ICAO. *Environmental Technical Manual*. 2014.

[33] *The Jet Engine*. 5th ed. Rolls-Royce, 1986.

[44] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. 1st ed. Wiley, 2001. ISBN: 9780471873396.

[47] *Airplane flying handbook*. United States Department of Transportation, Federal Aviation Administration, Flight Standards Service, 2004.

[55] Jack D. Mattingly. *Elements of Propulsion: Gas Turbines and Rockets*. American Institute of Aeronautics and Astronautics, 2006.

[57] Theodore L. Bergman et al. *Fundamentals of Heat and Mass Transfer*. 6th ed. John Wiley & Sons, 2007.

[58] Çengel Yunus A. *Heat Transfer: a Practical Approach*. 2nd ed. McGraw-Hill, 2006.

[62] E.M. Greitzer and H.N. Slater. *Design Methodologies for Aerodynamics, Structures, Weight, and Thermodynamic Cycles*. MIT, 2010.

[63] Mark H. Waters and Edward T. Schairer. *Analysis of Turbofan Propulsion System Weight and Dimensions*. NASA, 1977.

[70] *Aeronautical Technologies for the Twenty-First Century*. National Academy Press, 1992.

[72] W.P.J. Visser and M.J. Broomhead. *GSP: A generic object-oriented gas turbine simulation environment*. NLR, 2000.

[76] James R. Stone. *Interim Prediction Method for Jet Noise*. NASA, 1974.

[77] William E. Zorumski. *Aircraft Noise Prediction Program Theoretical Manual*. NASA, Feb. 1982.

Theses

[6] Konstantinos G. Kyprianidis. "Multi-disciplinary conceptual design of future jet engine systems". PhD. Apr. 2010.

[34] Xin Zhao. "Aero Engine Intercooling: Conceptual design and experimental validation of an aero engine intercooler". PhD. 2016.

[56] Anis Faidi. "Effect of accessory power take-off variation on a turbofan engine performance". MSc. 2012.

[64] P. Proesmans. "Preliminary Propulsion System Design and Integration for a Box-Wing Aircraft Configuration". MSc. 2019.

[68] Emily S. Dallara. "Aircraft Design for Reduced Climate Impact". PhD. 2011.

[74] Eberhard-Lothar Bertsch. "Noise Prediction within Conceptual Aircraft Design". PhD. 2013.

[85] Jacob Markish. "Valuation Techniques for Commercial Aircraft Program Design". MSc. 2002.

[86] Tomas Grönstedt. "Development of methods for analysis and optimization of complex jet engine systems". PhD. 2000.

Articles

[1] John A. McDermid. "Complexity: concept, causes and control". In: *Proceedings Sixth IEEE International Conference on Engineering of Complex Computer Systems. ICECCS 2000* (Feb. 2000). DOI: 10.1109/iceccs.2000.873923.

[3] Dianne DeTurris and Andrew Palmer. "Perspectives on managing emergent risk due to rising complexity in aerospace systems". In: *INSIGHT* 21.3 (Aug. 2018), pp. 80–86. DOI: 10.1002/inst.12215.

[4] Martijn Roelofs and Roelof Vos. "Correction: Uncertainty-based design optimization and technology evaluation: A review". In: *2018 AIAA Aerospace Sciences Meeting* (Jan. 2018), pp. 1–21. DOI: 10.2514/6.2018-2029.c1.

[5] J.H. Bussemaker, P.D. Ciampa, and B. Nagel. "System Architecture Design Space Exploration: An Approach to Modeling and Optimization". In: *AIAA Aviation 2020 Forum* (June 2020). DOI: 10.2514/6.2020-3172.

[8] Benjamin J. Brelje and Joaquim R.R.A. Martins. "Electric, hybrid, and turboelectric fixed-wing aircraft: A review of concepts, models, and design approaches". In: *Progress in Aerospace Sciences* 104 (2019), pp. 1–19. DOI: 10.1016/j.paerosci.2018.06.004.

[9] A.T. Isikveren et al. "Pre-design strategies and sizing techniques for dual-energy aircraft". In: *Aircraft Engineering and Aerospace Technology* 86.6 (2014), pp. 525–542. DOI: 10.1108/aeat-08-2014-0122.

[11] R.W. Claus et al. "Numerical Propulsion System Simulation". In: *Computing Systems in Engineering* 2.4 (1991), pp. 357–364. DOI: 10.1016/0956-0521(91)90003-n.

[14] Kevin Forsberg and Harold Mooz. "The Relationship of Systems Engineering to the Project Cycle". In: *Engineering Management Journal* 4.3 (1992), pp. 36–43. DOI: 10.1080/10429247.1992.11414684.

[16] Edward Crawley et al. "The Influence of Architecture in Engineering Systems". In: *Engineering Systems Monograph* (Jan. 2004).

[18] Azad Madni and Shatad Purohit. "Economic Analysis of Model-Based Systems Engineering". In: *Systems* 7 (Feb. 2019), p. 12. DOI: 10.3390/systems7010012.

[20] Ian M. Lyons and Sian L. Beilock. "Ordinality and the Nature of Symbolic Numbers". In: *The Journal of Neuroscience* 33.43 (Oct. 2013), pp. 17052–17061. DOI: 10.1523/jneurosci.1775-13.2013.

[22] Warren Hare, Julie Nutini, and Solomon Tesfamariam. "A survey of non-gradient optimization methods in structural engineering". In: *Advances in Engineering Software* 59 (2013), pp. 19–28. DOI: 10.1016/j.advengsoft.2013.03.001.

[24] Kalyanmoy Deb et al. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[31] Alan H. Epstein. "Aeropropulsion for Commercial Aviation in the Twenty-First Century and Research Directions Needed". In: *AIAA Journal* 52.5 (Apr. 2014), pp. 901–911. DOI: 10.2514/1.j052713.

[32] Dieter Scholz et al. "Fuel consumption due to shaft power off-takes from the engine". In: *4th International Workshop on Aircraft System Technologies* (Apr. 2013). DOI: 10.15488/4462.

[35] Olivier Petit et al. "An Outlook for Radical Aero Engine Intercooler Concepts". In: *Proceedings of ASME Turbo Expo 2016: Turbomachinery Technical Conference and Exposition* (June 2016). DOI: 10.1115/gt2016-57920.

[36] A. Rolt and N. Baker. "Intercooled turbofan engine design and technology research in the EU Framework 6 NEWAC programme". In: *International Symposium on Air Breathing Engines* (2009).

[37] Nestor Gonzalez Diez, Arvind G. Rao, and Jos van Buijtenen. "Conceptual Study of Counter-Rotating Turbofan Engines". In: *Proceedings of ASME Turbo Expo 2010: Power for Land, Sea and Air* (June 2010). DOI: 10.1115/gt2010-22770.

[38] F. Yin and Arvind G. Rao. "Performance analysis of an aero engine with inter-stage turbine burner". In: *The Aeronautical Journal* 121 (Sept. 2017), pp. 1605–1626. DOI: 10.1017/aer.2017.93.

[39] J. Bussemaker and T. De Smedt. "jbussemaker/OpenTurbofanArchitecting: Aircraft Jet Engine Architecting Benchmark Problem". In: *Zenodo* (June 2021). DOI: 10.5281/ZENODO.5011359.

[40] Martin Zaefferer and Daniel Horn. "A First Analysis of Kernels for Kriging-Based Optimization in Hierarchical Search Spaces". In: *Parallel Problem Solving from Nature – PPSN XV* 1 (2018), pp. 399–410. DOI: 10.1007/978-3-319-99259-4_32.

[41] J.H. Bussemaker et al. "Effectiveness of Surrogate-Based Optimization Algorithms for System Architecture Optimization". In: *AIAA Aviation Forum 2021* (July 2021). DOI: 10.2514/6.2021-3095.

[42] Juliane Müller and Marcus Day. "Surrogate Optimization of Computationally Expensive Black-Box Problems with Hidden Constraints". In: *INFORMS Journal on Computing* 31.4 (July 2019), pp. 689–702. DOI: 10.1287/ijoc.2018.0864.

[43] Jianyuan Zhai and Fani Boukouvala. "Data-Driven spatial Branch-and-bound algorithms for black-box optimization". In: *Computer Aided Chemical Engineering* 47 (2019), pp. 71–76. DOI: 10.1016/b978-0-12-818597-1.50012-6.

[45] Eric S. Hendricks and Justin S. Gray. "pyCycle: A Tool for Efficient Optimization of Gas Turbine Engine Cycles". In: *Aerospace* 6.8 (Aug. 2019). DOI: 10.3390/aerospace6080087.

[46] J. Blank and K. Deb. "Pymoo: Multi-Objective Optimization in Python". In: *IEEE Access* 8 (2020), pp. 89497–89509. DOI: 10.1109/ACCESS.2020.2990567.

[50] Dipanjay Dewanji, Arvind G. Rao, and Jos van Buijtenen. "Feasibility Study of Some Novel Concepts for High Bypass Ratio Turbofan Engines". In: *ASME Turbo Expo 2009: Power for Land, Sea, and Air* 1 (Feb. 2010), pp. 51–61. DOI: 10.1115/gt2009-59166.

[52] Natalia Marszałek. "The impact of thermodynamics parameters of turbofan engine with ITB on its performance". In: *Combustion Engines* 182.3 (Sept. 2020), pp. 16–22. DOI: 10.19206/ce-2020-303.

[53] Ahmed F. El-sayed, Mohamed S. Emeara, and Mohamed A. El-habet. "Performance Analysis of Cold Sections of High Bypass Ratio Turbofan Aeroengine". In: *International Journal of Development Research* 6.7 (July 2016), pp. 8382–8398. DOI: 10.24218/jrmer.2017.23.

[54] J. Sousa, L. Villafañe, and G. Paniagua. "Thermal analysis and modeling of surface heat exchangers operating in the transonic regime". In: *Energy* 64 (Jan. 2014), pp. 961–969. DOI: 10.1016/j.energy.2013.11.032.

[60] Michael T. Tong and Bret A. Naylor. "An Object-Oriented Computer Code for Aircraft Engine Weight Estimation". In: *Volume 1: Aircraft Engine; Ceramics; Coal, Biomass and Alternative Fuels; Manufacturing, Materials and Metallurgy; Microturbines and Small Turbomachinery* (2008). DOI: 10.1115/gt2008-50062.

[65] Kayla C. Aloyo, Christopher A. Perullo, and Dimitri N. Mavris. "An Assessment of Ultra High Bypass Engine Architecture and Installation Considerations". In: *50th AIAA/ASME/SAE/ASEE Joint Propulsion Conference* (July 2014). DOI: 10.2514/6.2014-3685.

[66] E. Torenbeek and G.H. Berenschot. "De berekening van het omspoeld gondeloppervlak van enkel- en dubbelstroom straalmotoren voor civiele vliegtuigen". In: *Delft University of Technology* (Jan. 1983).

[69] Emily Schwartz Dallara, Ilan M. Kroo, and Ian A. Waitz. "Metric for Comparing Lifetime average Climate Impact of Aircraft". In: *AIAA Journal* 49.8 (Aug. 2011), pp. 1600–1613. DOI: 10.2514/1.j050763.

[73] Feijia Yin and Arvind Gangoli Rao. "A review of gas turbine engine with inter-stage turbine burner". In: *Progress in Aerospace Sciences* 121 (Dec. 2020). DOI: 10.1016/j.paerosci.2020.100695.

[82] Justin S. Gray et al. "OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization". In: *Structural and Multidisciplinary Optimization* 59 (2019), pp. 1075–1104. DOI: 10.1007/s00158-019-02211-z.

[83] Ola Isaksson. "A Collaborative Engineering Design Research Model—An Aerospace Manufacturer's View". In: *Impact of Design Research on Industrial Practice* (July 2015), pp. 363–381. DOI: 10.1007/978-3-319-19449-3_24.

[84] Andreas Hölzer and Martin Sommerfeld. "New simple correlation formula for the drag coefficient of non-spherical particles". In: *Powder Technology* 184.3 (June 2008), pp. 361–365. DOI: 10.1016/j.powtec.2007.08.021.

[88] Brian J. Sauser et al. "From TRL to SRL: The Concept of Systems Readiness Levels". In: *Conference on Systems Engineering Research* (Apr. 2006).

[89] Ulrich Schumann. "Influence of propulsion efficiency on contrail formation". In: *Aerospace Science and Technology* 4.6 (May 2000), pp. 391–401. DOI: 10.1016/s1270-9638(00)01062-2.

[92] Dimitra E. Diamantidou, Panagiotis Tsirikoglou, and Konstantinos G. Kyprianidis. "A Robust Initialization Approach of Multi-Point Synthesis Schemes for Aero-Engine Conceptual Design". In: *AIAA Aviation Forum 2021* (July 2021). DOI: 10.2514/6.2021-3469.

Miscellaneous

[10] Airbus. *Hydrogen fuel cells, explained.* Accessed on 25 July 2021 via https://www.airbus.com/newsroom/news/en/2020/10/hydrogen-fuel-cell-cross-industry-collaboration-potential-for-aviation.html. Oct. 2020.

[12] Tegara. *Numerical Propulsion System Simulation (NPSS) numerical simulation software for propulsion systems.* Accessed on 16 July 2021 via https://www.tegakari.net/en/2019/12/npss/. Dec. 2019.

[13] Luca Boggero. *Systems Engineering Process: Application on a Hybrid-Electric Propulsion System.* Limited document access by DLR. 2019.

[23] Stanford University. *Gradient-Free Optimization.* Accessed on 9 Aug 2021 via http://adl.stanford.edu/aa222/Lecture_Notes_files/chapter6_gradfree.pdf. 2012.

[25] European Commission. *AGILE 4.0: Project.* Accessed on 15 July 2021 via https://www.agile4.eu/project/. 2021.

[26] Justin Gray. *MDO Problem Suite v3: We have the technology, we can rebuild it!* Oral presentation. June 2020.

[30] Wikipedia. *Turbofan.* Accessed on 1 Aug 2021 via https://en.wikipedia.org/wiki/Turbofan. 2021.

[48] Nabil Ben Nasr. *Final Report Summary - COBRA (Innovative counter rotating fan system for high bypass ratio aircraft engine).* Accessed on 23 July 2021 via https://cordis.europa.eu/project/id/605379/reporting. 2018.

[49] Roy Bhaskar and A.M. Pradeep. *Introduction to Aerospace Propulsion.* Accessed on 26 July 2021 via https://nptel.ac.in/content/storage2/courses/101101001/downloads/Intro-Propulsion-Lect-3.pdf.

[51] NASA. *Propulsion primer, performance, parameters and units.* Accessed on 26 July 2021 via https://history.nasa.gov/SP-4404/app-b5.htm.

[59] NASA. *Ideal Brayton Cycle.* Accessed on 22 July 2021 via https://www.grc.nasa.gov/www/k-12/airplane/brayton.html. 2015.

[61] NASA. *Weight Analysis of Turbine Engine - an Object-Oriented Version (WATE)*. Accessed on 23 July 2020 via https://software.nasa.gov/software/LEW-19687-1. 2020.

[67] European Commission. *Flightpath 2050: Europe's Vision for Aviation*. Accessed on 24 July 2021 via https://ec.europa.eu/transport/sites/transport/files/modes/air/doc/flightpath2050.pdf. 2011.

[71] GasTurb GmbH. *GasTurb 13: Design and Off-Design Performance of Gas Turbines*. Accessed on 23 July 2020 via https://www.gasturb.de/. 2018.

[75] Jan Böttcher. *Noise Certification Workshop: Aircraft Noise Certification*. Accessed on 25 July 2021 via https://www.icao.int/Meetings/EnvironmentalWorkshops/Documents/NoiseCertificationWorkshop-2004/BIP_2_2_jb.pdf. Oct. 2004.

[78] Pratt & Whitney. *F100-PW-229 Engine*. Accessed on 27 July 2021 via https://prattwhitney.com/products-and-services/products/military-engines/F100-PW-229-ENGINE. 2021.

[79] EASA. *Type-Certificate Data Sheet For Engine LEAP-1A & LEAP-1C series engines*. Accessed on 27 July 2021 via https://web.archive.org/web/20181013014334/https://www.easa.europa.eu/sites/default/files/dfu/EASA%20E110%20TCDS%20Issue%207%20LEAP-1A-1C.pdf. 2018.

[80] CFM. *LEAP overview*. Accessed on 27 July 2021 via https://www.cfmaeroengines.com/wp-content/uploads/2017/09/Brochure_LEAPfiches_2017.pdf. 2017.

[81] National Institute for Occupational Safety and Health. *Aircrew Safety & Health: Noise/-Hearing Loss*. Accessed on 27 July 2021 via https://www.cdc.gov/niosh/topics/aircrew/noise.html. 2017.

[87] NASA. *Technology Readiness Level*. Accessed on 1 Aug 2021 via https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_accordion1.html. 2017.

[90] FAA. *Engines and Propellers: Regulations & Policies*. Accessed on 1 Aug 2021 via https://www.faa.gov/aircraft/air_cert/design_approvals/engine_prop/engine_prop_regs/. 2020.

[91] Dassault Systemes. *DYMOLA Systems Engineering: Multi-Engineering Modeling and Simulation based on Modelica and FMI*. Accessed on 1 Aug 2021 via https://www.3ds.com/products-services/catia/products/dymola/. 2021.

[93] *ParaPy: Knowledge Based Engineering Platform*. Accessed on 1 Aug 2021 via https://www.parapy.nl/. 2021.

[94] OpenMDAO. *Setting Nonlinear and Linear Solvers*. Accessed on 9 Aug 2021 via https://openmdao.org/newdocs/versions/latest/features/core_features/controlling_solver_behavior/set_solvers.html. 2021.

[95] Jasper Bussemaker. *ADORE Library 0.6.0 Documentation*. Limited document access by DLR. 2021.

Appendix A
Software

The aircraft jet engine architecting tool is implemented in Python 3 and built with three software frameworks: pyCycle, OpenMDAO and pymoo. PyCycle was already explained in chapter 4. The two other packages are explained here.

A.1 MDO Framework: OpenMDAO

The engine cycle framework pyCycle is implemented using OpenMDAO as MDO integration framework. OpenMDAO is an open-source object-oriented MDO framework implemented in Python, allowing the calculation of analytical derivatives for MDO-system-level parameters based on analytical derivatives defined at the discipline level. With Newton-type algorithms and novel hierarchical strategies, problem structures are used and coupled systems are solved which results in an increased computational efficiency. Next to that, OpenMDAO uses a modular modeling structure which was required to provide the modularity characteristic of pyCycle. Similar to the pyCycle framework, OpenMDAO also has four main elements [82]:

- *Component*: contains the calculations that have to be performed by OpenMDAO, and is therefore the most basic element of the software. However, this calculation can range from a simple equation to a complete analysis of a certain discipline with internal or external code. Furthermore, calculations can be implicit or explicit, referred to as the *ImplicitComponent* and *ExplicitComponent* subclasses, respectively.

- *Group*: forms a congregation of *Component* and/or *Group* class instances, resulting in a hierarchy tree. The advantages of this *Group* element is that the code becomes more organized and that hierarchical (non)linear solvers can be used. Linear solvers are required when analytical derivatives have to be computed, while nonlinear solvers have to be implemented when implicit equations are encountered and/or when model components have cyclic dependency [94].

- *Driver*: runs the model once or defines the optimization algorithm and calls the model in an iterative way. For the latter case, the in- and outputs of this model could include the design variables and objectives & constraints, respectively.

- *Problem*: contains all the *Component* and *Group* instances combined with a single *Driver* instance of a certain problem.

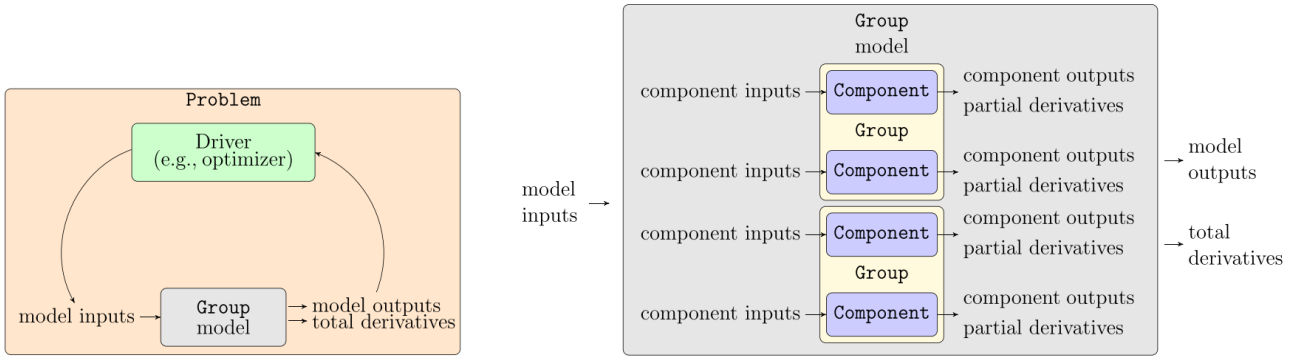


Figure A.1: An OpenMDAO *Problem* element contains one *Driver* and *Group instance*, which can contain multiple *Components* [82].

The combination of pyCycle and OpenMDAO thus allows a modular definition, analysis, and optimization of engine cycles, and the easy integration of additional disciplines such as engine weight and emissions in addition to thermodynamic cycle analysis. As the design of an aircraft engine is a multi-objective and multidisciplinary process, OpenMDAO and pyCycle are thus important to tackle the thesis objective and research questions. However, it must be noted that OpenMDAO is used to converge a single engine thermodynamic cycle whereas non-gradient-based optimization is required for the complete aircraft engine architecting tool. The reason why and the software framework used for this purpose is explained next.

As OpenMDAO focuses on tackling the multidisciplinary nature of problems and more single-objective problems, an optimization framework focusing on multi-objective problems had to be selected. This framework was chosen to be pymoo by the author.

A.2 Optimization: pymoo

To deal with the multi-objective nature of the aircraft engine design process, the multi-objective optimization framework pymoo was selected. Pymoo has the ability to deal with this kind of optimization while simultaneously providing additional tools for the optimization problem itself such as result quality assessment. It must be noted that it is possible to use both single- and multi-objective optimization algorithms in pymoo and that, similar to pyCycle and OpenMDAO, pymoo provides a modular implementation of its framework, increasing its flexibility [46]. The pymoo framework consists of three distinct modules: *Problems*, *Optimization* and *Analytics*. Each of these modules can further be broken down into submodules. A visual overview of pymoo's structure can be seen in Figure A.2. In case the reader is interested in these submodules, they are referred to the article of Blank & Deb [46].

Aircraft engine design optimization is a non-gradient-based optimization as the gradient of the design variables to the objective function cannot simply be calculated. The reason for this is that discrete design variables are used in the engine architecting problems, and thus no gradient can be computed. However, the design variables will definitely influence the outcome of the objective(s). It is clear that the framework will be important in providing the multi-objective capabilities of the aircraft jet engine architecting tool, and in creating the solution space generated by the approach for specified engine objectives and constraints.

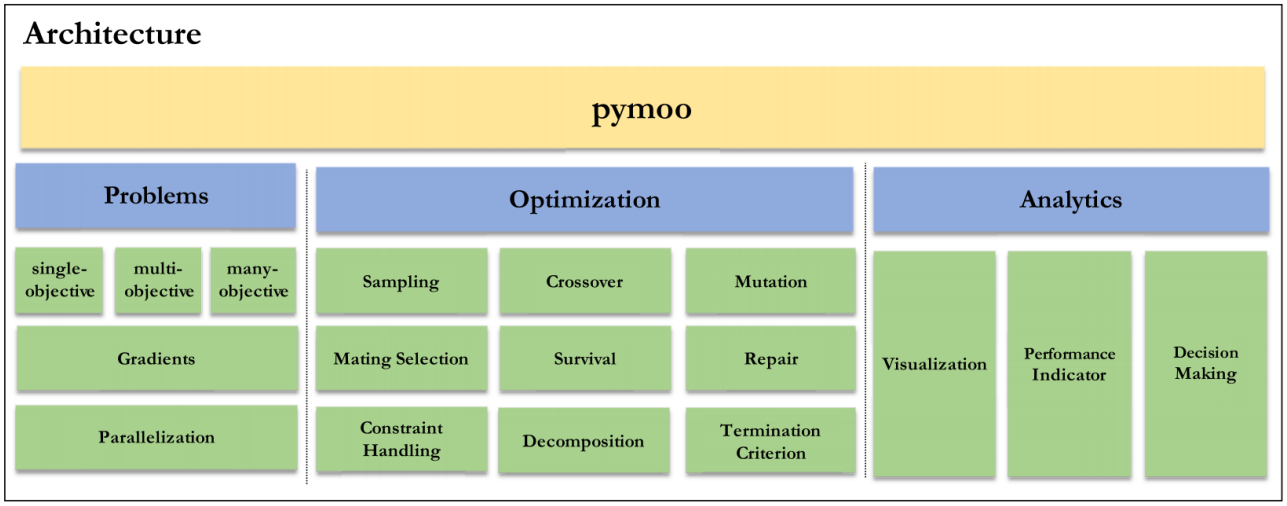


Figure A.2: Organization of the pymoo framework [46].

Appendix B

Architectural Choices Design Variables

Table B.1: Design variables of the architectural design choices with variable type and bounds.

Choice	**Variable**	**Type**	**Lower bound**	**Upper bound**	**Unit**
Fan	Include	Categorical	0 (excluded)	1 (included)	[-]
	Bypass ratio	Continuous	2	12.5	[-]
	Fan pressure ratio	Continuous	1.1	1.8	[-]
CRTF	Include	Categorical	0 (excluded)	1 (included)	[-]
Shaft	Number of shafts	Integer	1	3	[-]
	Overall pressure ratio	Continuous	1.1	60	[-]
	Pressure ratio IPC	Continuous	10	90	[%]
	Pressure ratio LPC	Continuous	10	90	[%]
	RPM HP shaft	Continuous	1,000	20,000	[RPM]
	RPM IP shaft	Continuous	1,000	20,000	[RPM]
	RPM LP shaft	Continuous	1,000	20,000	[RPM]
Gearbox	Include	Categorical	0 (excluded)	1 (included)	[-]
	Gear ratio	Continuous	1	5	[-]
Afterburner	Include	Categorical	0 (excluded)	1 (included)	[-]
	Fuel-to-air ratio	Continuous	0	0.05	[-]
ITB	Include	Categorical	0 (excluded)	1 (included)	[-]
	Fuel-to-air ratio	Continuous	0	0.05	[-]

Cooling bleed	Inter-bleed massflow HPC - CC	Continuous	0	10	[%]
	Above inter-bleed IPT target massflow	Continuous	0	100	[%]
	Above inter-bleed LPT target massflow	Continuous	0	100	[%]
	Inter-bleed massflow IPC - HPC	Continuous	0	10	[%]
	Above inter-bleed IPT target massflow	Continuous	0	100	[%]
	Above inter-bleed LPT target massflow	Continuous	0	100	[%]
	Inter-bleed massflow LPC - IPC	Continuous	0	10	[%]
	Above inter-bleed IPT target massflow	Continuous	0	100	[%]
	Above inter-bleed LPT target massflow	Continuous	0	100	[%]
	Intra-bleed massflow HPC	Continuous	0	10	[%]
	Above intra-bleed IPT target massflow	Continuous	0	100	[%]
	Above intra-bleed LPT target massflow	Continuous	0	100	[%]
	Intra-bleed massflow IPC	Continuous	0	10	[%]
	Above intra-bleed IPT target massflow	Continuous	0	100	[%]
	Above intra-bleed LPT target massflow	Continuous	0	100	[%]
	Intra-bleed massflow LPC	Continuous	0	10	[%]
	Above intra-bleed IPT target massflow	Continuous	0	100	[%]
	Above intra-bleed LPT target massflow	Continuous	0	100	[%]
Nozzle mixing	Include	Categorical	0 (excluded)	1 (included)	[-]
Intercooler	Include	Categorical	0 (excluded)	1 (included)	[-]
	Location	Integer	0 (between HPC & CC)	2 (between LPC & IPC)	[-]
	Radius	Continuous	0.01	0.05	[m]
	Length	Continuous	0.01	0.5	[m]
	Number of pipes	Integer	1	250	[-]
Offtakes	Power offtake location	1 (HP shaft)	3 (LP shaft)		
	Extraction bleed offtake location	1 (HPC)	3 (HPC)		

Cooling bleed clarification: cooling bleed is split up into inter-bleed and intra-bleed. For the former, air is bled from in between two components whereas for the latter, air is bled from within a component. The bled massflow is specified as a percentage of the incoming massflow, with bounds between 0% and 10% bleed. Each cooling bleed component also has target turbines which receive a portion of this bled air, specified by a percentage with bounds between 0% (not receiving any cooling bleed) and 100% (receiving all the cooling bleed).

Appendix C

Verification: Component Attributes

In this appendix, all the important component attributes for the two verification cases are listed: the turbojet and the turbofan.

C.1 Turbojet

Table C.1: Important component attributes for the pyCycle turbojet verification case.

Component	**Attribute**	**Input**	**Unit**
Inlet	Pressure recovery	100	[%]
Compressor	PR	13.5	[-]
	Efficiency	83	[%]
CC	Pressure loss	3	[%]
Turbine	Efficiency	86	[%]
Nozzle	Velocity loss	1	[%]
Shaft	Rotational speed	8,070	[RPM]
	Power loss	0	[%]

C.2 Turbofan

Table C.2: Important component attributes for the pyCycle turbofan verification case.

Component	**Attribute**	**Input**	**Unit**
Inlet	Pressure recovery	99.9	[%]
Fan	FPR	1.685	[-]
	Efficiency	89.48	[%]
Splitter	BPR	5.105	[-]
	Bypass velocity loss	0.61	[%]
LPC	PR	1.98	[-]
	Efficiency	92.43	[%]
HPC	PR	9.02	[-]
	Efficiency	87.07	[%]
CC	Pressure loss	5.4	[%]
HPT	Efficiency	88.88	[%]
LPT	Efficiency	89.96	[%]
Nozzle core	Velocity loss	0.67	[%]
Intrableed HPC	Bleed offtake	7.0982	[%]
	LPT portion	100	[%]
Interbleed HPC-CC	Bleed offtake	16.847	[%]
	HPT portion	100	[%]
LP Shaft	Rotational speed	4,666.1	[RPM]
	Power loss	0	[%]
HP Shaft	Rotational speed	14,705.7	[RPM]
	Power loss	0	[%]

Appendix D
Validation: Component Attributes

In this appendix, all the important component attributes for the two validation cases are listed: the CFM LEAP-1C and the P&W F100 used on the Comac C919 and F16 aircraft, respectively.

D.1 CFM LEAP-1C

Table D.1: Important component attributes for the CFM LEAP-1C validation case.

Component	**Attribute**	**Input**	**Unit**
Inlet	Pressure recovery	100	[%]
Fan	FPR	1.45	[-]
	Efficiency	89	[%]
Splitter	BPR	11	[-]
	Bypass velocity loss	1	[%]
LPC	PR	1.751	[-]
	Efficiency	83	[%]
HPC	PR	15.757	[-]
	Efficiency	83	[%]
CC	Pressure loss	3	[%]
HPT	Efficiency	86	[%]
LPT	Efficiency	86	[%]
Nozzle core	Velocity loss	1	[%]
LP Shaft	Rotational speed	3,894	[RPM]
	Power loss	0	[%]
HP Shaft	Rotational speed	19,391	[RPM]
	Power loss	0	[%]

D.2 P&W F100

Table D.2: Important component attributes for the P&W F100 validation case.

Component	**Attribute**	**Input**	**Unit**
Inlet	Pressure recovery	100	[%]
Fan	FPR	1.45	[-]
	Efficiency	89	[%]
Splitter	BPR	0.36	[-]
	Bypass velocity loss	1	[%]
LPC	PR	4.698	[-]
	Efficiency	83	[%]
HPC	PR	4.698	[-]
	Efficiency	83	[%]
CC	Pressure loss	3	[%]
HPT	Efficiency	86	[%]
LPT	Efficiency	86	[%]
Nozzle core	Velocity loss	1	[%]
LP Shaft	Rotational speed	3,894	[RPM]
	Power loss	0	[%]
HP Shaft	Rotational speed	19,391	[RPM]
	Power loss	0	[%]

Appendix E

Results: Component Attributes

In this appendix, all the important component attributes for the result generation for the simple and realistic architecting problem are listed.

Table E.1: Important component attributes for the simple and realistic architecting problem.

Component	**Attribute**	**Input**	**Unit**
Inlet	Pressure recovery	100	[%]
CRTF	Efficiency	93.45	[%]
Fan	Efficiency	89	[%]
Splitter	Bypass velocity loss	1	[%]
LPC	Efficiency	83	[%]
IPC	Efficiency	83	[%]
HPC	Efficiency	83	[%]
CC	Pressure loss	3	[%]
HPT	Efficiency	86	[%]
IPT	Efficiency	86	[%]
LPT	Efficiency	86	[%]
Nozzle core	Velocity loss	1	[%]
Nozzle joint	Velocity loss	1	[%]
Intercooler	Overall heat transfer coefficient	400	[W/m^2K]
LP Shaft	Power loss	0	[%]
IP Shaft	Power loss	0	[%]
HP Shaft	Power loss	0	[%]

Appendix F
ADORE

As the *Engine Architecting Framework* is meant to be a separate component from the evaluator, this means that it should be replaceable while the *Engine Architecture Evaluator* should still work. Furthermore, in chapter 2, it was introduced that one of the systems engineering challenges is linking the upstream and downstream processes of MBSE. A new system architecture design platform called Architecture Design and Optimization Reasoning Environment (ADORE) was created by the German Aerospace Center (DLR) in order to model and inspect the design spaces of system architectures.

ADORE is based on the Architecture Design Space Graph (ADSG): a graph containing architecture elements represented by nodes which can be connected to form a complete system architecture. These elements could be functions, components, concepts, ports, etc. Next to that, there are three types of design decisions that can be made and represented in the ADSG: option-decisions (mutually-exclusive), permutation-decisions (source-target connection) and additional design variables. For more information, the reader is referred to the paper of Bussemaker et al. [5]

ADORE enables the user to identify the different architecting decisions in the problem by mapping form to function, to define the architecting design problem, to analyze the architecture instance generation and to evaluate the performance of the created architectures. The benefits of ADORE are:

- Its upstream MBSE connection;
- Graphic user interface for defining the architecting problem;
- Implementation of the architecture generation logic, which can take into account and deal with decision hierarchy.

Figure F.1 shows an exemplary setup of ADORE for an aircraft jet engine system architecting design problem. With this platform, complex systems can be modeled and optimized in an MBSE framework [95]. This effectively helps to tackle the challenge of linking the upstream and downstream processes in MBSE processes.

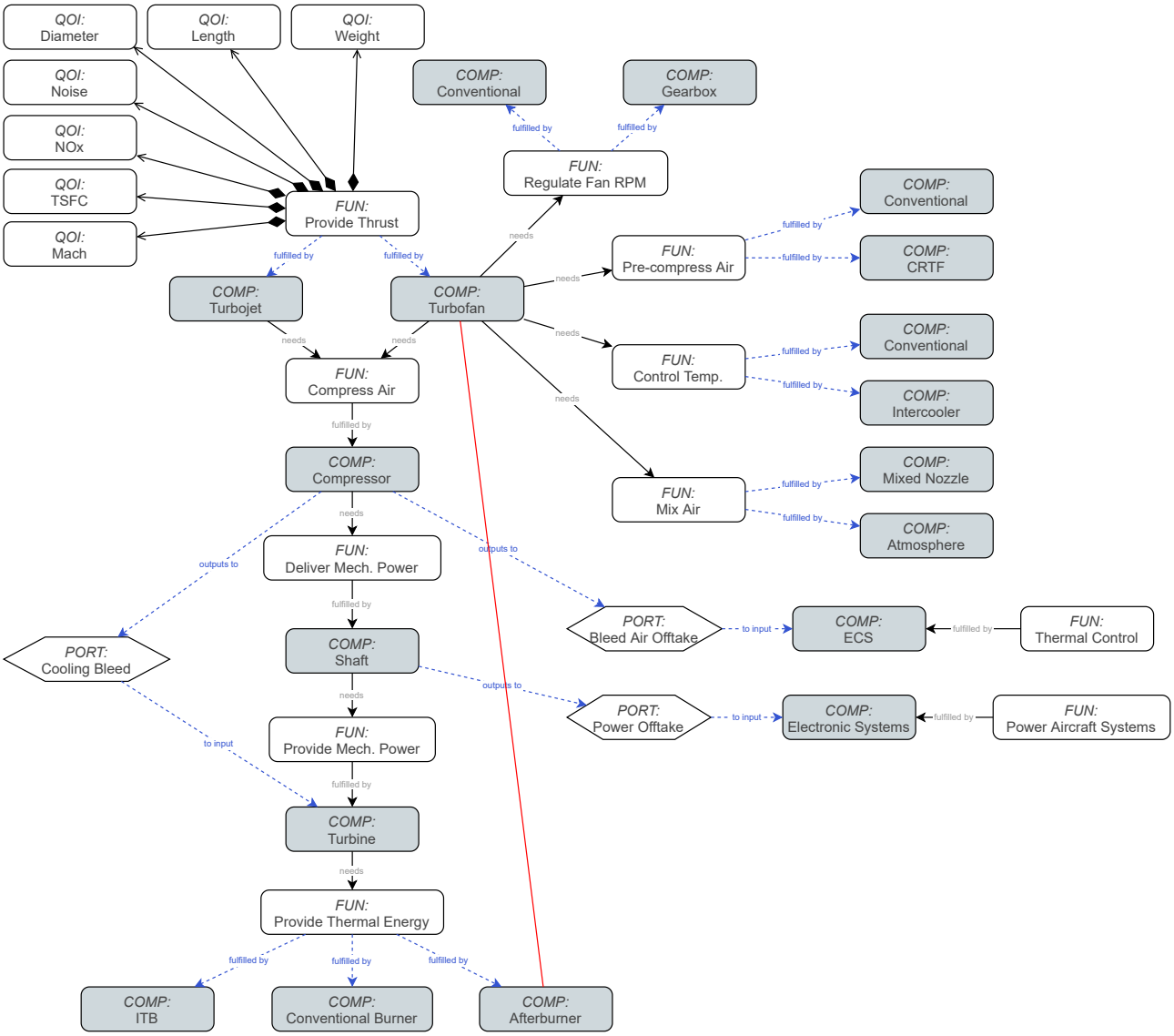


Figure F.1: Setup of an aircraft jet engine system architecting design problem with the ADORE software. This includes elements of *form* (components) which are mapped to *functions* in order to fulfill them. Next to that, *ports* indicate connections between two components. Furthermore, Quantities Of Interest (QOI) can be specified to satisfy the requirements of the system stakeholders.