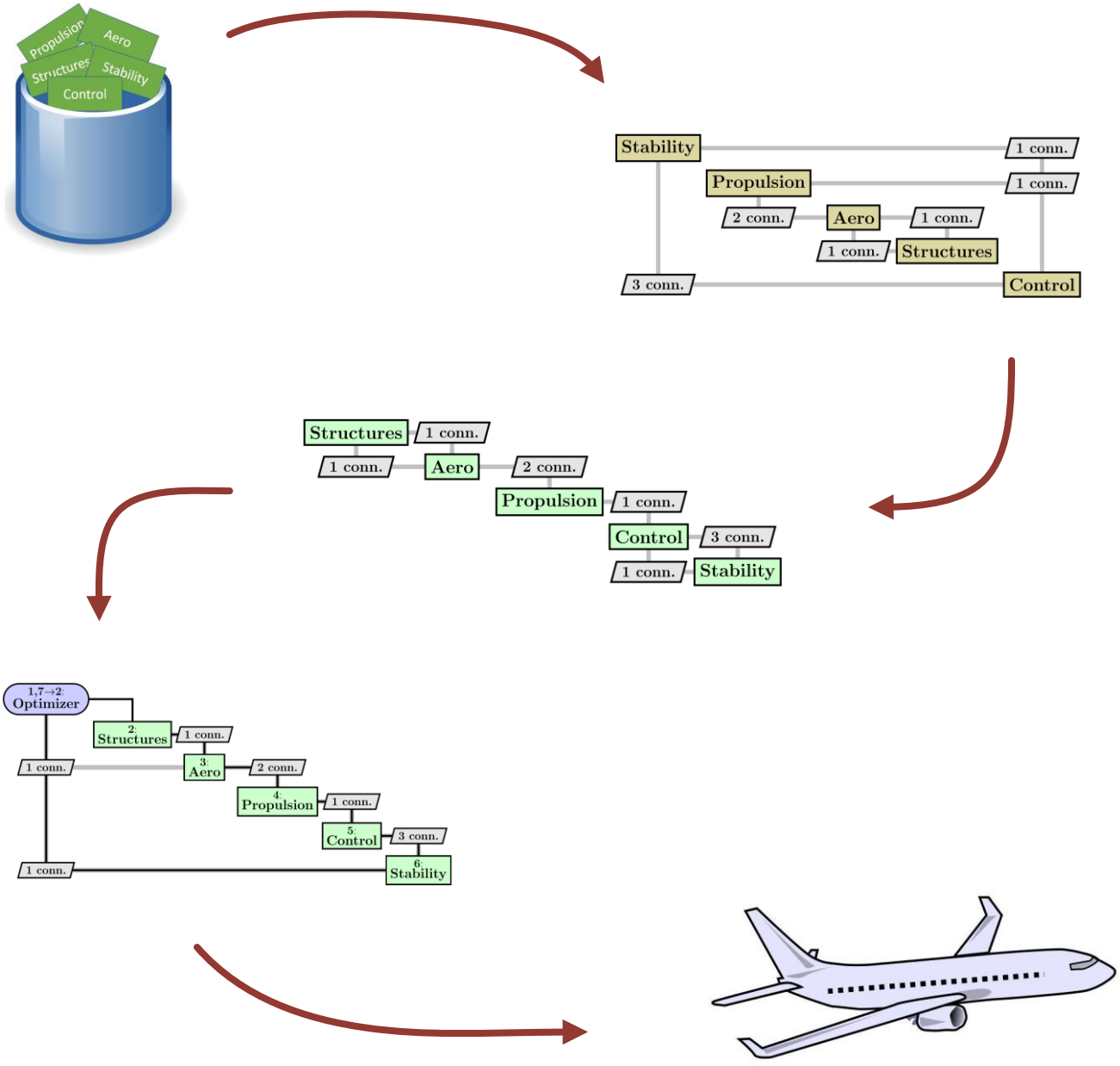
Automated Execution Process Formulation using Sequencing and Decomposition Algorithms for Collaborative MDAO

A.M.R.M. Bruggeman



Delft University of Technology

TU Delft

Automated Execution Process Formulation using Sequencing and Decomposition Algorithms for Collaborative MDAO

by

Anne-Liza M.R.M. Bruggeman

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday January 29, 2019 at 2:00 PM.

Student number:	4223977	
Project duration:	October, 2017 – January, 2019	
Thesis committee:	Prof.dr.ir. L.L.M. Veldhuis,	TU Delft, committee chair
	Dr.ir. G. La Rocca,	TU Delft, supervisor
	Dr.ir. M.B. Zaaijer,	TU Delft
	ir. I. van Gent,	TU Delft, daily supervisor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Summary

With Multidisciplinary Design Analysis and Optimization (MDAO), a fully automated aircraft design analysis is setup and optimization algorithms are used to obtain better designs by balancing the synergy between components. Despites its benefits, MDAO is not yet as widely adapted in industry as one would expect. There are still some technical and non-technical barriers hampering its full implementation.

In the EU project AGILE, a new methodology and framework were developed to make the MDAO approach more accessible to industry. AGILE provides both the blueprint, as well as the tools and processes, necessary for efficient project management and for easy MDAO system formulation and execution in large heterogeneous teams of experts. One of the key components in this framework is the KADMOS package. KADMOS is used to formulate large, heterogeneous MDAO systems and their execution process before they are implemented as executable workflows.

A key step in the formulation of MDAO systems is the execution process formulation. The execution process consists of the sequencing and decomposition of the disciplines within the MDAO system. Sequencing denotes the determination of the execution order, while decomposition refers to the distribution of the disciplines over various partitions. As the partitions have no data dependencies, they can be executed simultaneously, which potentially results in a better use of computational resources (e.g. multiple cores).

A proper sequencing and decomposition can significantly reduce the convergence time of MDAO systems. Before this thesis, sequencing could only be performed manually in KADMOS and the decomposition of the disciplines was not possible at all. Therefore, this thesis focuses on how the setup of the execution process can be fully automated to optimize the workflow and to make better use of the available computational resources.

To achieve this goal, four sequencing algorithms have been developed and implemented in KADMOS. The algorithms primarily minimize the number of feedback connections, while also keeping the execution time of the sequence to a minimum. One algorithm is the branch-and-bound algorithm, which systematically searches the solution space until the global optimum is found. This algorithm provides high-quality execution orders, but can only be applied to small problems due to poor performance scalability. Therefore, three swap algorithms have been implemented as well. Swap algorithms iteratively try to improve a given solution by swapping one or multiple nodes. These algorithms have a lower accuracy but are significantly faster than the branch-and-bound algorithm and therefore more suitable for large MDAO systems.

In addition to the sequencing algorithms, an algorithm to support the decomposition of MDAO systems and the formulation of multiple partitions was added to KADMOS as well: MDK (Metis-based Decomposition of KADMOS graphs). MDK automatically decomposes MDAO systems while account for both the number of variables that need to be converged as well as the execution time of the partitions.

The algorithms were verified and validated on thousands of MDAO systems using a scalable mathematical test case. The verification and validation showed that high-quality partitions and execution orders are obtained in short time periods.

Furthermore, the influence of the decomposition of the disciplines on the total execution time was investigated by solving hundreds of MDAO systems using different solution strategies. The results show that the execution time of MDAO systems can be reduced by using a proper decomposition. Furthermore, the results show that the best solution strategy depends on both the properties of the MDAO problem as well as the number of available computational resources.

The algorithms were not only tested on a theoretical problem, but also on a more realistic aircraft design problem. A novel implementation of the Initiator toolbox was created to test the algorithms on the conceptual design of a conventional aircraft. The results show an improved flexibility of the Initiator. Thanks to KADMOS, the implementation of new modules is straightforward and different analysis or optimization strategies are easily applied. Thanks to the decomposition and sequencing algorithms, the execution order of the modules is automatically determined and multiple partitions can be created to make better use of the available computational resources.

This thesis has made a significant step towards the full automation of the execution process formulation in MDAO systems. Several algorithms have been successfully implemented. The automation of the execution process has created the opportunity to perform large benchmarking studies of MDAO systems. Finally, the

new implementation of the Initiator toolbox has improved both the flexibility as well as the transparency of the toolbox.

Future works involves the optimization of the execution process for the given computational environment and the inclusion of the sensitivities between the input and output of the different disciplinary analyses. However, the current results already showed a significant decrease in the setup and execution time of MDAO systems and therefore, this thesis has made a direct contribution in reducing some of the barriers that still exist for using MDAO today.

Contents

List of Figures

List of Tables

List of Symbols

Symbol	Description
λ	Eigenvalue
ρ_c	Coupling density
A	Aspect ratio
C_D	Drag coefficient
C_L	Lift coefficient
$\mathbf{c}$	Constraint values
D	Diagonal matrix
E	Set of edges in a graph
e	Oswald factor
f	Objective value
G	Graph
L	Laplacian matrix
L/D	Lift over Drag ratio
$N(x)$	Neighborhood of x
n_{ce}	Number of cut edges
n_{cv}	Number of coupling variables per discipline
n_f	Number of feedback couplings
n_{iter}	Number of iteraitons
n_{tc}	Number of test cases
P	Node pair
$\mathbf{R}$	Discipline residuals
s	Standard deviation
T/W	Thrust loading
t	Target partition node weight
t_{iter}	Execution time per iteration
u	Balance factor
v	Eigenvector
W	Weighted adjacency matrix
W/S	Wing loading
w	Partition node weight
x	Metis unbalance factor
$\mathbf{x}$	Design variables
$\mathbf{y}$	Coupling variables
$\bar{\mathbf{y}}$	State variables
$\hat{\mathbf{y}}$	Copies of coupling variables

List of Abbreviations

Abbreviation	Description
AAO	All-At-Once
ADF	AGILE Development Framework
AEO	All Engines Operative
AGILE	Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts
AVL	Athena Vortex Lattice
BKL	Boundary Kernighan-Lin
CMDOWS	Common MDO Workflow Schema
COBYLA	Constraint Optimization By Linear Approximation
CPACS	Common Parametric Aircraft Configuration Schema
DOE	Design of Experiments
DSM	Design Structure Matrix
FF	Fuel Fraction
FM	Fiduccia-Mattheyses
FPG	Fundamental Problem Graph
GA	Genetic Algorithm
GS	Gauss-Seidel
IDF	Individual Discipline Feasible
J	Jacobi
KADMOS	Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System
KL	Kernighan-Lin
MDA	Multidisciplinary Analysis
MD(A)O	Multidisciplinary Design (Analysis and) Optimization
MDF	Multidisciplinary Feasible
MDG	MDAO Data Graph
MDK	Metis-based Decomposition of KADMOS graphs
MLM	Maximum Landing Mass
MPG	MDAO Problem Graph
MTOM	Maximum Take-Off Mass
OEI	One Engine Inoperative
OEM	Operational Empty Mass
P	Partitioned
RCB	Runtime Coupling Balance
RCG	Repository Connectivity Graph
SAND	Simultaneous Analysis and Design
SLSQP	Sequential Least Squares Programming
VCP	Variable Complexity Problem
(X)DSM	(eXtended) Design Structure Matrix
XML	Extensible Markup Language

1 Introduction

Sustainability and environmental requirements are increasingly important in aircraft design. Aircraft must become more efficient and sustainable. One design method that contributes to obtaining better aircraft designs is Multidisciplinary Design Analysis and Optimization (MDAO). MDAO uses optimization techniques to solve aircraft design problems while taking into account different disciplinary analysis.

Using MDAO in aircraft design offers a lot of benefits. Instead of optimizing each discipline individually, MDAO uses the synergy between components to optimize the entire system. Solving the full problem leads to better design results than optimizing each component individually [3]. Furthermore, once an MDAO system is in place, part of the design process has been automated. This creates the opportunity for engineers to focus more on the results and design improvements instead of doing repetitive tasks. By using MDAO, the generation and evaluation of a design go faster and more design iterations can be performed [17].

Because of the benefits, a lot of research and development has been carried out in the field of MDAO over the past few decades [11, 36, 47]. Starting from the first generation in which the entire analysis and optimization process is executed on a local domain and operated by a single small team, MDAO is now evolving towards the third generation in which the design process is distributed over multiple organizations and countries, operated by various large and heterogeneous teams of experts [7].

These developments improve MDAO but also give rise to some challenges. MDAO has several technical and non-technical barriers, which keeps it from being widely used in industry [2]. The setup of an MDAO system can be challenging. For example, due to the increasing size of the design problems an the lack of standards, data management can be difficult [2]. A lack of transparency gives the users a black-box feeling towards the full MDAO systems. Therefore, they have less trust in the results, as it is more difficult to understand how they were obtained [2, 44]. Furthermore, the setup time of MDAO systems is generally long, as it can be difficult to make all the connections between the disciplines [17]. Collaboration between the different stakeholders and teams of experts can be difficult [2]. Finally, once the MDAO problem has been formulated and implemented, it usually lacks the agility to easily change the problem formulation or optimization strategy [44].

In the EU project AGILE (Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts) [8], a new methodology and framework were developed, called the AGILE Paradigm, to make the MDAO approach more accessible to industry and to remove some of these barriers. The main objective is to reduce the aircraft development costs and time to market by developing processes and techniques for efficient collaboration and by reducing the setup and convergence time for MDAO problems. The project advances the state of the art in solving complex and large design problems, ultimately leading to greener and more cost-effective aircraft solutions. Besides the formulation of the methodology, a software framework is developed to apply the methodology: the AGILE Development Framework (ADF).

One of the software packages that is being developed within the ADF is KADMOS. KADMOS (Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System) [51] is used to formulate large, heterogeneous MDAO systems and their execution process before they are automatically implemented as executable workflows.

The MDAO system formulation process in KADMOS starts with a repository in which all available disciplinary tools are present. Each tool is a stand-alone module which only uses an input file as input and

produces an output file as output. These input and output files use a standard data schema, such that KADMOS can easily analyze these files and can generate the input and output connections between the different disciplinary tools. KADMOS then formulates the MDAO problem by only selecting those tools that are necessary to solve the design problem. In the last step, KADMOS wraps the solution strategy around the MDAO problem, such that the full MDAO system formulation is obtained. The MDAO system formulation is then passed to a different software package which translates the formulation into an executable workflow. The MDAO system is now ready to be executed and results can be obtained.

One of the key steps in the problem formulation of MDAO systems is the execution process formulation. The execution process consists of the sequencing and decomposition of the MDAO system. During the sequencing process, the execution order of the disciplinary tools is determined, while the decomposition process divides the tools in multiple partitions which can be executed simultaneously.

The execution process has a major influence on the total convergence time of MDAO systems. A good execution order will reduce the number of feedback variables and thus the number of iterations that are necessary to obtain a consistent solution. Dividing the disciplinary analyses over multiple partitions will reduce the total convergence time due to the parallel execution of the partitions. Furthermore, a good partitioning will make better use of the available computational resources.

Before this thesis, the execution order could only be set manually in KADMOS and decomposition was not possible at all. The disciplinary analyses could only be executed either all in sequence or all in parallel. This leads to sub-optimal processes and an inefficient use of the available computational resources. Furthermore, manually setting the execution order is time-consuming for large MDAO systems and the best order is not always obvious.

Therefore, this thesis focuses on the development of new algorithms to automate the execution process formulation in KADMOS and to add the possibility to decompose the MDAO systems into multiple partitions. More specifically, the thesis tries to find an answer on the following research question:

How can the setup of the execution process of large coupled MDAO systems be fully automated to reduce the formulation time and to optimize the workflow?

The goal is to enable the easy setup of the execution process of complex workflows. Therefore, the research will focus on how the automated process formulation can be achieved and what the benefits of the automated process formulation are. To achieve this goal, several subgoals are formulated:

1. Develop an algorithm to automate the sequencing process
2. Develop an algorithm to automate the decomposition process
3. Verify and validate the working of the new algorithms
4. Test the full MDAO system formulation and execution process on a theoretical mathematical problem
5. Test the full MDAO system formulation and execution process on a more realistic aerospace related design case

The research question and goal can be summarized in the following research objective:

The research objective is to investigate how the setup of the execution process of large coupled MDAO systems can be fully automated to reduce the formulation time and to optimize the workflow by implementing algorithms for the automatic sequencing and decomposition of MDAO systems in KADMOS and testing them both on a simple mathematical test case as well as a more realistic aerospace related design case.

Having the automated execution process formulation in KADMOS will create the opportunity to adapt the solution strategy to the available computational resources. Therefore, the computational resources will be used more efficiently and the total execution time will be reduced. Furthermore, the setup time will reduce as KADMOS will take care of both the sequencing and decomposition as well as their implementation into the executable workflow. This ultimately contributes to the high-level goals of KADMOS and AGILE to reduce the setup and convergence time of MDAO systems and to make it more accessible to the industry.

The thesis is divided into three parts: State of the Art, Algorithms and Test Cases. The first part focuses on the state of the art of both KADMOS and the sequencing and decomposition problem. The part starts with a short overview of the MDAO development process in Chapter 2. This chapter gives a short overview

of the AGILE project and introduces the automated MDAO formulation process which is supported by KADMOS. The chapter introduces the sequencing and decompostion problem and explains its interaction with the MDAO solution strategy. Sequencing and decomposition has been performed before. Therefore, a short overview of the different algorithms that can be used for the sequencing and decomposition problem is given in Chapter 3.

The second part of the thesis focuses on the different algorithms that have been developed to automate the execution process formulation in KADMOS to achieve research goals 1 and 2. Several sequencing algorithms have been developed, which all vary in accuracy and speed. Therefore, the different algorithms are suitable for different MDAO systems. Furthermore, one decomposition algorithm is developed to automate the decomposition process in KADMOS. The implementation of the different algorithms is explained in detail in Chapter 4. The algorithms are verified and validated on thousands of MDAO systems using a scalable mathematical problem in Chapter 5.

The last part, Test Cases, focuses on research goals 5 and 6. The full MDAO system formulation including the automated execution process is tested on a simple mathematical test case in Chapter 6. Hundreds of MDAO systems are formulated and executed to examine the effect of the decomposition and solution strategy on the total execution time. In the final chapter, the automated execution process is tested on a more realistic design case. The conceptual design of a conventional aircraft is performed using a novel implementation of the Initiator toolbox in KADMOS. This chapter shows the easiness with which the MDAO system can now be rearranged and reformulated thanks to the fully automated process formulation.

I

State of the Art

2

MDAO Development Process

The sequencing and decomposition process is part of the bigger MDAO Development Process in AGILE. Therefore, a short overview of the AGILE Paradigm is given in Section 2.1. Part of the AGILE Paradigm is the MDAO system formulation, which is fully supported by the software package KADMOS. An overview of how KADMOS formulates the MDAO system based on a repository of tools is given in Section 2.2. Several solution strategies or coordination strategies can be used to solve the formulated MDAO problem. These solution strategies are shortly explained in Section 2.3. Finally, the sequencing and decomposition problem is explained in more detail in Section 2.4. This section also explains the interactions between the coordination and decomposition of an MDAO system.

2.1. AGILE

An overview of the Agile Paradigm [8] is shown in Figure 2.1. The Agile Paradigm indicates how the development process can be structured in multiple layers and defines the role of each stakeholder and their interactions. The top of Figure 2.1 shows the expected impact of the AGILE Paradigm on the design and optimization process and its goal to significantly reduce the setup time for MDAO systems and the convergence time needed to obtain an optimal solution. The AGILE Paradigm consists of the Knowledge Architecture and Collaborative Architecture, shown at the bottom of Figure 2.1. The Knowledge Architecture is shown at the left. The Knowledge Architecture provides a blueprint, together with the necessary tools and processes, for efficient project management and for easy MDAO problem formulation and execution in large heterogeneous teams of experts. However, working and collaborating in large heterogeneous teams can be a difficult and challenging task, as stakeholders can be working in different companies, in different countries and on different servers. Therefore, the Collaborative Architecture provides methods and software tools for efficient and safe collaboration between the various participant while respecting their different security requirements and intellectual property. Each of the architectures will be shortly discussed below.

2.1.1. Knowledge Architecture

The Knowledge Architecture [49] support the project management and MDAO system formulation and execution. The architecture consists of four layers as shown on the left side in Figure 2.1: the Development Process Layer, the Automated Design Layer, the Design Competence Layer and the Data & Schemas Layer.

The Development Process Layer formulates the business process and supports the project management at the highest level. All activities that need to be executed, both manually or automated, are combined in this layer. The development process starts from the requirements and design case definition, through the problem setup and execution and finishes with the design solution. The goal is to have an optimal design at the end of this process.

In the Automated Design Layer, a specific design analysis or optimization, as specified in the Development Process Layer, is constructed and executed. Different design competences are collected and executed according to an architecture specified by the Development Process Layer. The blue arrows between the Automated Design Layer and Development Process Layer indicates the interface between the two layers. In the downward direction, the Development Process Layer controls the settings of the Automated Design, while in the upward direction, the formalization of the design process is brought into the Development Process Layer.

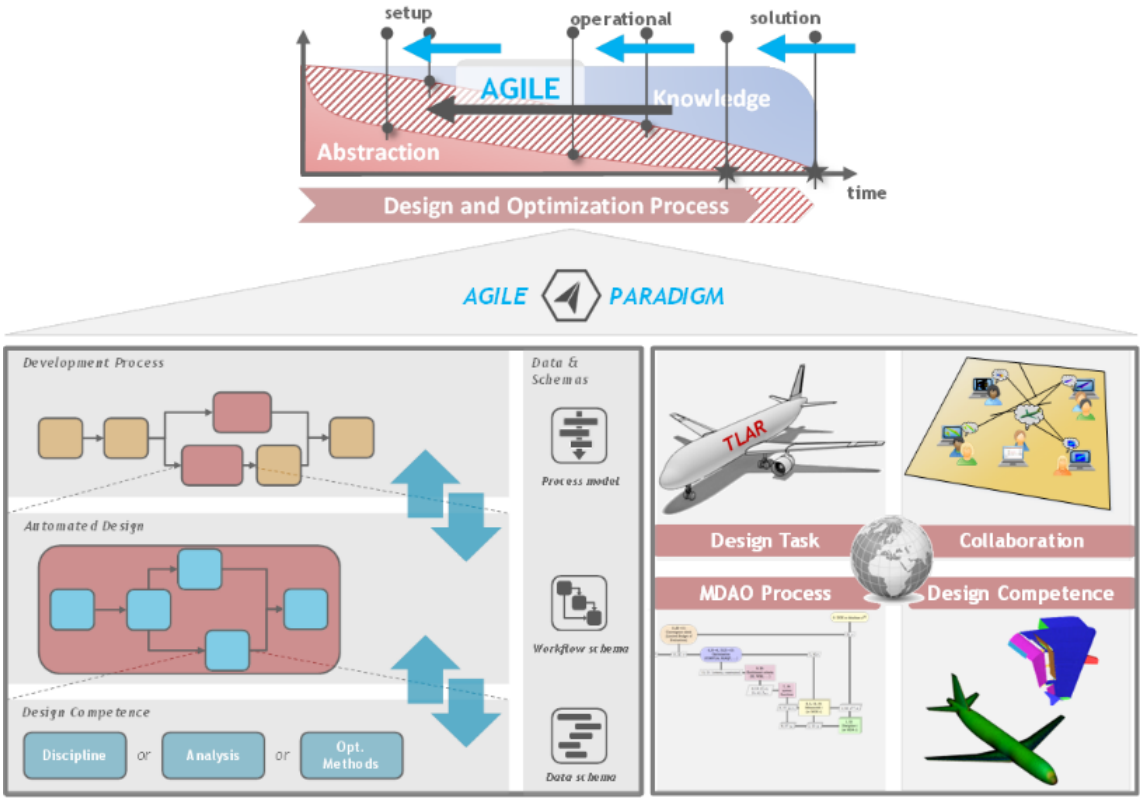


Figure 2.1: Overview of the AGILE Paradigm with on the left side the Knowledge Architecture and on the right side the Collaborative Architecture [49]

The third layer is the Design Competence Layer. This layer includes all the available design competences provided as services. A design competence is a specific capability needed for the design and optimization process, such as a simulation tool or optimization service. Each design team can develop their competence the way they prefer, using their choice of software. However, it should be wrapped around a central data schema, such that the different competences can be easily connected in the other layers. Again, there is an interface between Automated Design Layer and the Design Competence Layer. The design competences are called by the Automated Design Layer, while the information about the design competences should be made available to the Automated Design Layer.

The last layer is the Data & Schemas Layer which provides the standard schemas to be used in all other layers. Two standard schemas are defined in this layer. The first schema is the central data schema which is used in the Design Competence Layer. In the ADF, the Common Parametric Aircraft Configuration Schema (CPACS) is used as central data schema. CPACS is an XML schema data definition for the description of aircraft and its properties. The second data schema is a workflow schema to describe the workflows in the Automated Design and Development Process Layer. The Common MDO Workflow Schema (CMDOWS) is used for this in the ADF. CMDOWS [50] is an XML schema and stores the MDO workflow in a neutral format such that it can be used by the different applications. The usage of standard data schemas reduces the number of interfaces between different tools and applications to a minimum, as shown in Figure 2.2. The central data schemas make the integration and exchangeability of the different tools and applications in the design process easier and thus increases the scalability and agility of the design process.

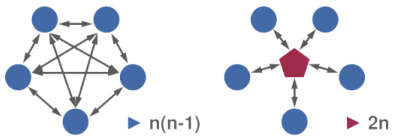


Figure 2.2: Number of interfaces between tools with and without central data schema [51]

Using the structured approach of the Knowledge architecture supports the design process and will lead to a reduction in setup time, easier inspection and debugging, and easy manipulation of MDAO problems [52].

2.1.2. Collaborative Architecture

The Collaborative Architecture [9] provides the methods and software tools for efficient and safe collaboration between the various participants respecting their different security requirements and intellectual properties. The Collaborative Architecture is a service-oriented architecture. This means that the design competences are offered as services to the users.

In a product development process, the MDO integrator is responsible for the deployment and management of the design and optimization process. He requests the execution of several services based on a workflow as can be seen on the left-hand side of Figure 2.3. Some of the services can be located on the integrator's server, but they can also be located on the servers of other organizations. When the execution of a service located on a different server is needed, a request for the service is made and the input file for the service is uploaded to a central data server. The specialist is notified of the request. Only when the specialist accepts, the input file is downloaded and the service is executed on the specialist's server. After execution, the output file is uploaded again to the central data server. The output file is downloaded by the integrator and the process as defined in the workflow can continue. The input and output files uses a standard data schema, which makes communication between different tools easy and straightforward.

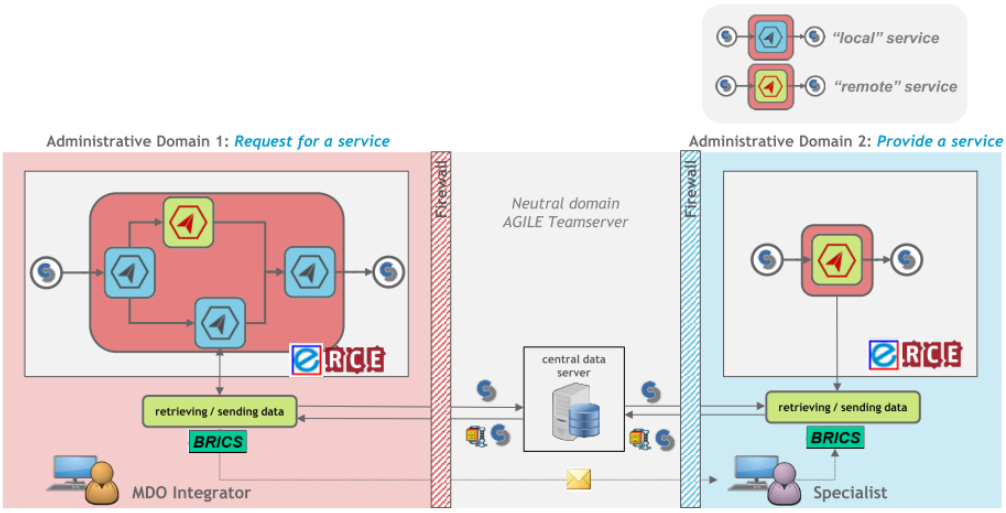


Figure 2.3: Overview of the Service Oriented Architecture in the Collaborative Architecture [9]

Using the service-oriented architecture, a lot of different services located on different servers can be coupled together in a straightforward fashion and large complex problems are easily executed. Furthermore, services are easily reused when the problem formulation changes or new design problems are formulated.

2.2. KADMOS

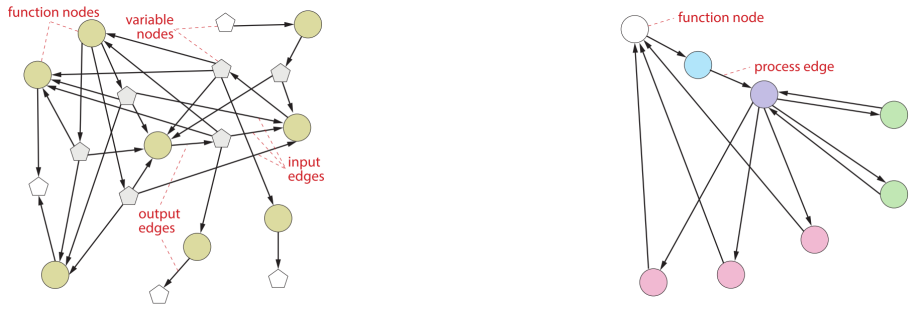
As mentioned in the previous section, the Development Process Layer of the Knowledge Architecture collects all the activities that need to be executed in the design process. More specifically, the Development Process Layer consists of five steps [49]:

I. Define design case and requirements
II. Specify complete and consistent data model and competences
III. Formulate design optimization problem and solution strategy
IV. Implement and verify collaborative workflow
V. Execute collaborative workflow and select design solution

KADMOS [51] is one of the software applications developed within the Knowledge Architecture. KADMOS supports step III in the Development Process Layer and provides a complete formulation of the MDAO system before it is translate into an executable workflow. The software package automates the problem formulation, starting from a repository of tools, to the definition of the problem, until the wrapping of the solution strategy, by making use of graphs and graph manipulation techniques.

A graph is an ordered pair $G = (V, E)$ with vertices or nodes V and edges E. An edge represents a connection between two nodes. Two nodes are neighbors in case an edge $\{u, v\} \in E$ exists. Graphs can be directed or undirected. In directed graphs, or digraphs, the direction of the edge is specified and $\{u, v\}$ is an ordered pair in which u is the start and v the end of the edge. In undirected graphs, the edges indicate connections between the nodes but the start and end points of the edges are not defined. Within a graph, each edge and node can have one or multiple weights assigned to it. Each weight indicates a specific property of the edge or node. For example, when applied to MDAO, the weight of an edge could stand for the number of variables that are passed between two nodes, while a node weight could stand for the execution time of the node.

KADMOS uses two types of digraphs, which can be seen in Figure 2.4. Figure 2.4a shows a data graph, which contains two types of nodes: function and variable nodes. The function nodes represent the executable tools while the variable nodes represent the data that is communicated between the tools. Furthermore, the directed edges indicate the input and output relations between the tools and the variables. The second type, the process graph, is shown in Figure 2.4b. The process graph only contains function nodes and no variable nodes. The edges indicate the execution order of the disciplinary tools.



(a) Data graph

(b) Process graph

Figure 2.4: Examples of KADMOS graphs [51]

Combined the two graphs can be visualized using an eXtended Design Structure Matrix (XDSM) [33]. An example of an XDSM can be seen in Figure 2.5. The executable tools are placed on the diagonal and the variables off-diagonal. Furthermore, the input for each discipline is placed in the corresponding column and the output in the corresponding row. The thick grey lines indicate the data flows, while the thin black lines indicate the process flows. Finally, the number inside the diagonal block indicates the order of the process. In case two blocks have the same number, they can be executed in parallel.

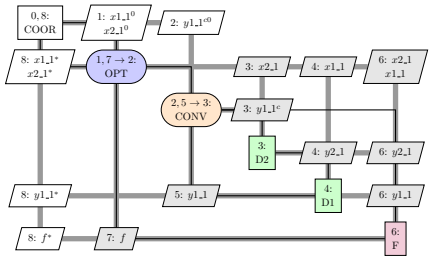


Figure 2.5: Example of an XDSM diagram generated using KADMOS

As mentioned earlier, KADMOS provides a complete MDAO system formulation before it is translated into an executable workflow. A top-level overview of the MDAO development process in KADMOS is visualized in the top part of Figure 2.6. The input is a repository or database containing all the available executable tools, as defined in step II of the Development Process Layer. For each tool an input and output file are formulated according to a standard XML data schema. As explained in the previous section, CPACS is used as standard XML data schema within the ADF. However, it is not necessary to use CPACS. The schema can vary per database, but all the tools in one database must be using the same one. The tools are stand-alone modules, which means that they do not depend on or interact with each other.

Once the database is created, it is loaded into KADMOS using a CMDOWS file. KADMOS evaluates the input and output files to make the connections in the initial graph: the Repository Connectivity Graph (RCG). The RCG (visualized using an XDSM in step 1 in Figure 2.6) is a data graph which simply visualizes the database. It represents all the available tools and shows the input and output connections between these tools.

From the RCG, the Fundamental Problem Graph (FPG) is formulated in the second step. The FPG represent the actual problem that needs to be solved. Different steps are taken to formulate the FPG. These steps are shown in detail in the bottom part of Figure 2.6. The FPG is the smallest graph possible to define the design problem. This means that only those tools are included which are actually needed to solve the problem.

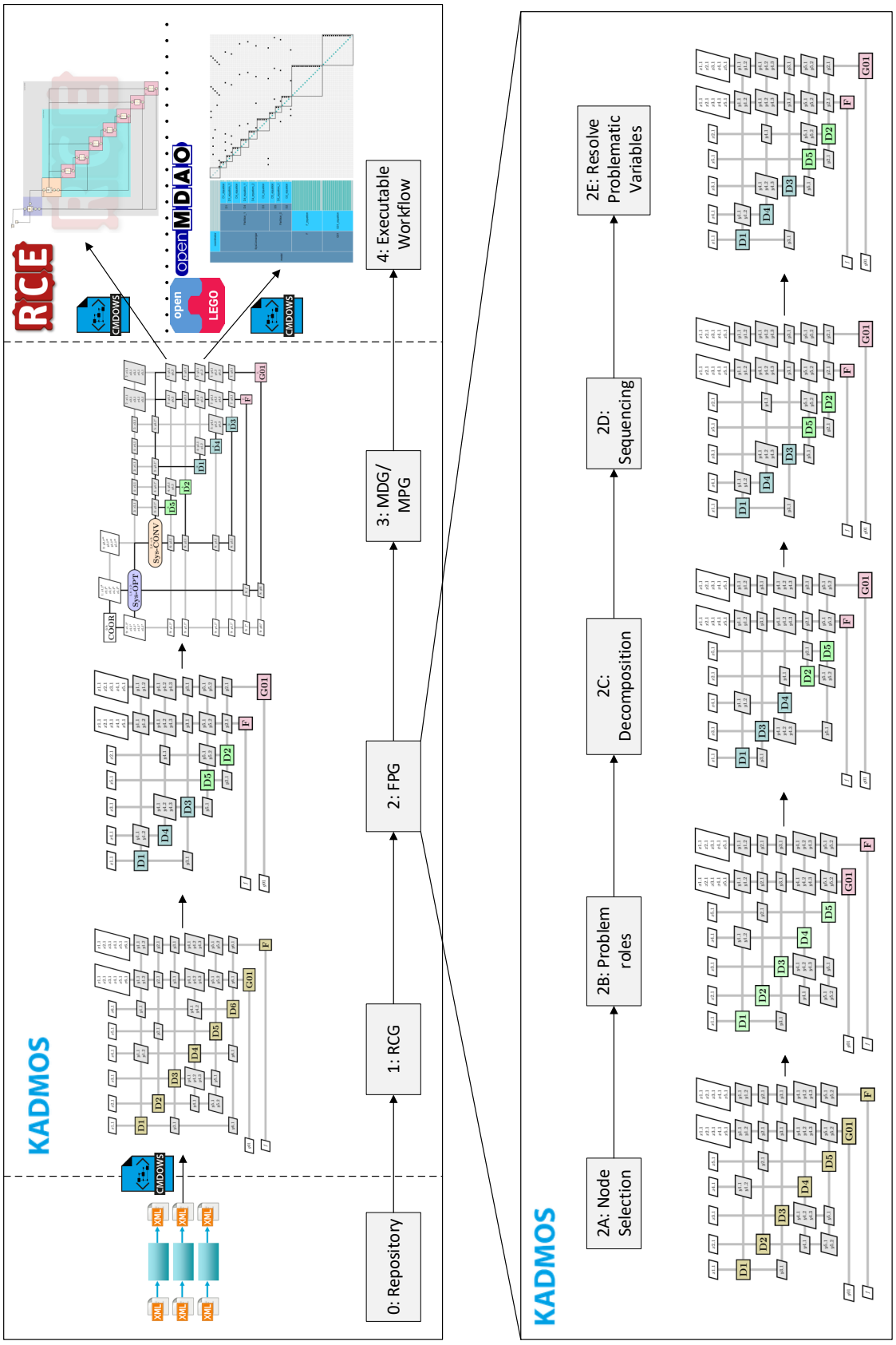


Figure 2.6: Overview of the full MDAO development process, going from a repository of tools to a complete MDAO system formulation

Therefore, the redundant tools are removed in the first step of formulating the FPG (2A).

Each variable node gets a problem role assigned in the next step. The possible problem roles are: objective, constraint, design or state variable. These problem roles are needed to impose the solution strategy in step 3. Furthermore, the function nodes are divided into pre-coupling, coupled and post-coupling functions. An example of the three function node types is shown in Figure 2.7. Coupled functions, visualized in green, are the disciplines which have circular dependencies. The pre-coupling and post-coupling functions do not have circular dependencies. The pre-coupling functions, shown in blue, are the disciplines which only supply (and not receive) input to the coupled functions. The post-coupled functions, indicated in red, are the disciplines which only require outputs from the coupled functions.

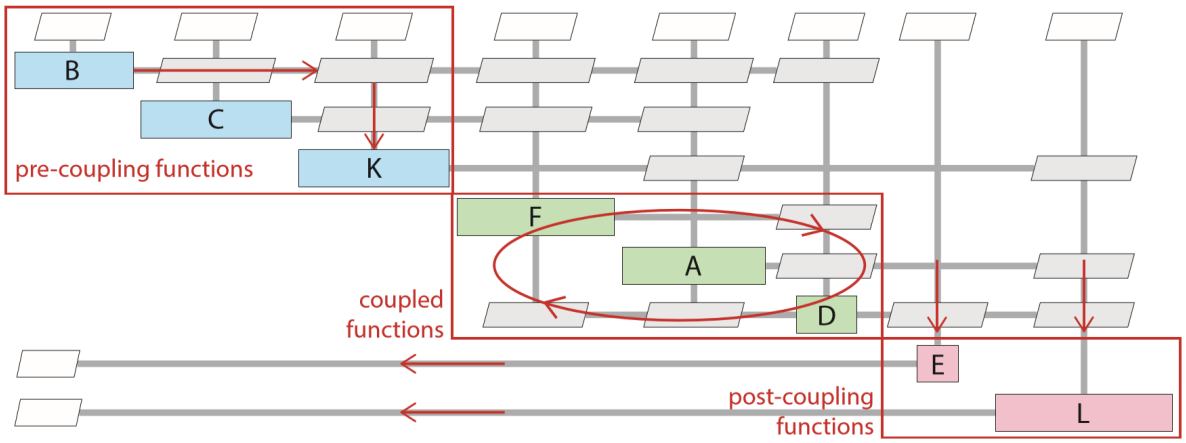


Figure 2.7: Example showing the three different function roles [51]

Once the problem roles are assigned, the execution process of the disciplines is formulated, starting with the decomposition process in step 2C. As explained in the Introduction, decomposition means that the disciplines are divided into several partitions. Decomposition is an important step in the problem formulation as it can reduce the convergence time of MDAO systems significantly. The reason for this is that the partitions have no data dependencies and therefore they can be executed in parallel. Furthermore, better use of the available computational resources can be made. Before this thesis, step 2C was completely missing in the formulation of the FPG. KADMOS did not support the creation of partitions in the MDAO problem formulation. During this thesis, the support for creating multiple parallel partitions has been developed. Furthermore, as will be explained in Chapter 4, a decomposition algorithm was developed to automatically determine the distribution of the disciplines over the various partitions.

The next step in the execution process formulation is the determination of the execution order of the disciplines during the sequencing process in step 2D. Just as for the decomposition, a proper execution order is important to reduce the convergence time of MDAO systems. A good execution order minimizes the number of feedback variables and therefore minimizes the number of iterations necessary to obtain a consistent solution. The sequencing process could only be performed manually in KADMOS. This is a time-consuming process for large MDAO systems. Furthermore, the best sequence is not always obvious. Therefore, multiple algorithms were implemented in KADMOS to automatize the determination of the execution order. The different algorithms will be explained in Chapter 4.

In the last step of formulating the FPG, step 2E, the validity of each node is checked, because the nodes in the FPG must comply with strict conditions on their connectivity. The graph is manipulated to resolve issues when necessary. Variables that do not apply to the conditions are called problematic variables. Problematic variables are resolved by the removal of nodes and edges or by splitting nodes and creating several node instances. Two examples of problematic variables and their solution are visualized in Figure 2.8. The first issue is a collision, where multiple disciplines share the same output variable. The second issue is a circular coupling, where a discipline has the same variable both as input and output. Both issues are resolved by splitting the variable and creating two instances. A full list with the different types of problematic

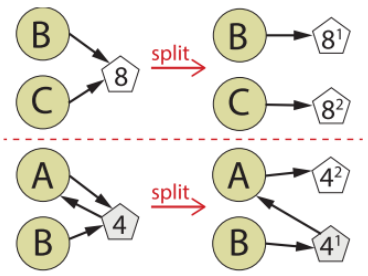


Figure 2.8: Example of splitting problematic variables in the FPG [51]

variables can be found in Van Gent et al. [52]

The last step in the MDAO system formulation is the wrapping of the solution strategy around the formulated problem in step 3. A solution strategy is necessary as the feedback connections in the MDAO system need to be converged by iterating over the MDAO problem until a consistent solution is obtained. The solution strategy that is used to obtain a consistent and optimal solution is called the MDAO architecture or coordination architecture. Examples of different MDAO architectures are Multidisciplinary Feasible or Individual Discipline Feasible. An explanation of these architectures and their differences will be given in Section 2.3. According to the selected architecture, MDAO architectural elements (e.g. optimizers, convergers, DOE-blocks, etc.) are added and two graphs are formed: the MDAO Data Graph (MDG) and the MDAO Process Graph (MPG). The MDG shows the data flows between the different tools and architectural elements, while the MPG presents the execution order of the optimization problem. Together, these two graphs contain all information to formulate a complete XDSM, as shown in the third step of Figure 2.6.

Once the MDG and MPG are created, the process in KADMOS is finished. The resulting problem formulation (saved in a CMDOWS file) is then communicated to a workflow software, which automatically translate the CMDOWS file into an executable model as shown in step 4 of Figure 2.6. Two examples of a workflow software are RCE[1] and OpenMDAO [20]. RCE translates the CMDOWS file through a native plugin, while OpenLEGO [10] is used to translate the file into an OpenMDAO model. Translating the MDAO system formulation into an executable workflow corresponds with step IV in the Development Process Layer of the Knowledge Architecture. Once the executable workflow has been set up, it is executed in step V and the results are obtained.

2.3. MDAO Architectures

As explained in the previous section, an MDAO architecture or coordination architecture is responsible for managing the data in an optimization problem and finding an optimal and consistent solution. The most general form of a coordination architecture as described in Martins and Lambe [36] is shown in equation 2.1 and in XDSM form in figure 2.9. All other architectures can be derived from this general form.

$$
\begin{aligned}
\text{minimize} \quad & f_0(\mathbf{x},\mathbf{y}) + \sum_{i=1}^{N} f_i(\mathbf{x}_0,\mathbf{x}_i,\mathbf{y}_i) \\
\text{with respect to} \quad & \mathbf{x},\hat{\mathbf{y}},\mathbf{y},\bar{\mathbf{y}} \\
\text{subject to} \quad & \mathbf{c}_0(\mathbf{x},\mathbf{y}) \geq 0 \\
& \mathbf{c}_i(\mathbf{x}_0,\mathbf{x}_i,\mathbf{y}_i) \geq 0 \quad \text{for} \quad i = 1,...,N \\
& \mathbf{c}_i^c = \hat{\mathbf{y}}_i - \mathbf{y}_i = 0 \quad \text{for} \quad i = 1,...,N \\
& \mathbf{R}_i(\mathbf{x}_0,\mathbf{x}_i,\hat{\mathbf{y}}_{j\neq i},\bar{\mathbf{y}}_i,\mathbf{y}_i) = 0 \quad \text{for} \quad i = 1,...,N
\end{aligned}
\tag{2.1}
$$

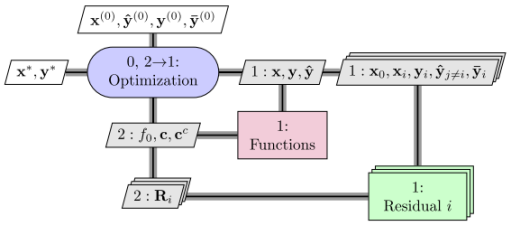


Figure 2.9: XDSM for the All-At-Once architecture [36]

In this formulation, $\mathbf{x}$ are the design variables. The optimizer must find the values of $\mathbf{x}$ for which the objective function f is minimized. A subscript i means that the variable or function only applies to discipline i. The subscript 0 means that the variable or function is shared by multiple disciplines. $\bar{\mathbf{y}}$ are the state variables. State variables are the output or response from the disciplines and give information about the state of the system. $\mathbf{y}$ are the coupling variables. These variables are the variables that are the output of one discipline and used as input for another discipline. $\hat{\mathbf{y}}$ are copies of the coupling variables. Coupling variable copies allow two coupled disciplines to run in parallel. As the system has to be consistent coupling constraints $\mathbf{c}^c$ are added to ensure that the copies of the coupling variables are equal to the original coupling variables. $\mathbf{R}$ are the residuals of the disciplines. If $\mathbf{R} = 0$, then $\bar{\mathbf{y}}$ is the solution of the discipline with respect to $\mathbf{x}$. Finally, $\mathbf{c}$ are the design constraints.

The architecture described in equation 2.1 can be referred to as the All-At-Once (AAO) architecture because the optimizer must find the optimal value for all variables (design, state, coupling and coupling variable copies) such that the objective function is minimized. Furthermore, the optimizer is responsible for meeting all constraints (residual, equality and design constraints).

As stated before, all other architectures can be derived from the AAO. The architectures can be divided into two categories: monolithic and distributed architectures. The monolithic architectures solve a single optimization problem. The distributed architectures partition the problem into several subproblems, opti-

[1] `http://rcenvironment.de/`, accessed: January 11th 2019

mizing both the subproblems and the overall problem. A full explanation of the distributed architectures is out of the scope for this thesis, as only monolithic architectures are considered. However, a full overview of the distributed architectures and their explanation can be found in Martins and Lambe [36].

By eliminating different parts of the AAO architecture, three monolithic architectures can be obtained: Simultaneous Analysis and Design (SAND), Individual Discipline Feasible (IDF) and Multidisciplinary Feasible (MDF) [33]. Regarding the monolithic architectures, only the MDF and IDF are available in KADMOS and used in this thesis. Therefore a short explanation of these two architectures, together with their main benefits and disadvantages is given below.

2.3.1. Individual Discipline Feasible

The IDF architecture is obtained by eliminating the discipline residuals **R** from equation 2.1. The architecture is described in equation 2.2 [36] and shown in Figure 2.10. In the IDF architecture, a discipline coordinator is added to each discipline, such that the disciplines themselves are responsible for driving the residuals to zero. The optimizer gives the design variables, $\mathbf{x}$, and coupling variable copies, $\hat{\mathbf{y}}$, as input and the disciplines calculate the corresponding coupling $\mathbf{y}$ and state variables $\bar{\mathbf{y}}$.

$$
\begin{aligned}
\text{minimize} \quad & f_0(\mathbf{x}, \mathbf{y}(\mathbf{x}, \hat{\mathbf{y}})) \\
\text{with respect to} \quad & \mathbf{x}, \hat{\mathbf{y}} \\
\text{subject to} \quad & \mathbf{c}_0(\mathbf{x}, \mathbf{y}(\mathbf{x}, \hat{\mathbf{y}})) \geq 0 \\
& \mathbf{c}_i(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_i(\mathbf{x}_0, \mathbf{x}_i, \hat{\mathbf{y}}_{j \neq i})) \geq 0 \quad \text{for} \quad i = 1, ..., N \\
& \mathbf{c}_i^c = \hat{\mathbf{y}}_i - \mathbf{y}_i(\mathbf{x}_0, \mathbf{x}_i, \hat{\mathbf{y}}_{j \neq i}) = 0 \quad \text{for} \quad i = 1, ..., N
\end{aligned} \tag{2.2}
$$

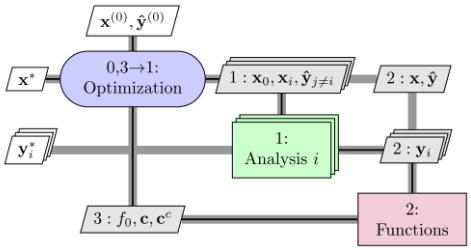


Figure 2.10: XDSM for the IDF architecture [36]

A benefit of the IDF architecture is that all the disciplines can be run in parallel. The main disadvantage is that no feasible system solution is obtained when the optimization is terminated prematurely as the optimizer is responsible for finding both an optimal and consistent solution. [36]

2.3.2. Multidisciplinary Feasible

When both the discipline residuals **R** and the coupling variable copies $\hat{\mathbf{y}}$ are eliminated the Multidisciplinary Feasible (MDF) architecture is obtained, see equation 2.3 [36] and figure 2.11. In this architecture, each discipline has a discipline coordinator to drive the residuals to zero. A system coordinator is added to ensure system consistency. The optimizer communicates the design variables $\mathbf{x}$ to the disciplines. The system coordinator performs iterations with the disciplines to obtain a feasible set of coupling variables $\mathbf{y}$ and state variables $\bar{\mathbf{y}}$. After a consistent set of $\mathbf{y}$ and $\bar{\mathbf{y}}$ is found, the corresponding values of the objective function and design constraints are communicated back to the optimizer.

$$
\begin{aligned}
\text{minimize} \quad & f_0(\mathbf{x}, \mathbf{y}(\mathbf{x}, \mathbf{y})) \\
\text{with respect to} \quad & \mathbf{x} \\
\text{subject to} \quad & \mathbf{c}_0(\mathbf{x}, \mathbf{y}(\mathbf{x}, \mathbf{y})) \geq 0 \\
& \mathbf{c}_i(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_i(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_{j \neq i})) \geq 0 \quad \text{for} \quad i = 1, ..., N
\end{aligned} \tag{2.3}
$$

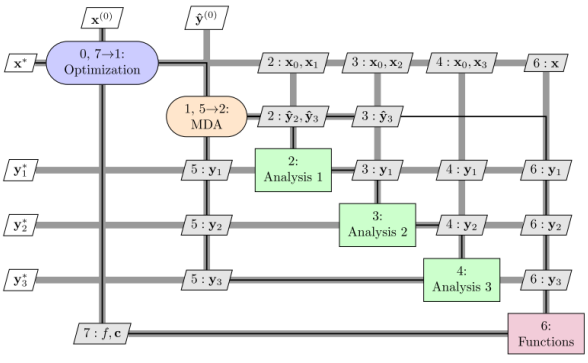


Figure 2.11: XDSM for the MDF architecture [36]

As both discipline and system feasibility are ensured for each optimizer iteration, a feasible solution is obtained when the optimization is terminated prematurely. However, it is not guaranteed that the design constraints are satisfied. As the optimizer is only responsible for finding the optimal values of $\mathbf{x}$, it needs to perform less iterations to find the optimal solution. One iteration will take more time though as each discipline has to be executed multiple times per iteration. [36]

Whether or not the disciplines can be executed in parallel depends on the strategy that is chosen to obtain a converged MDA (Multidisciplinary Analysis). In case the Gauss-Seidel iteration is chosen, all disciplines are run sequentially. In case the Jacobi iteration is chosen, the disciplines can be run in parallel.

2.4. Sequencing, Decomposition and Coordination Interactions

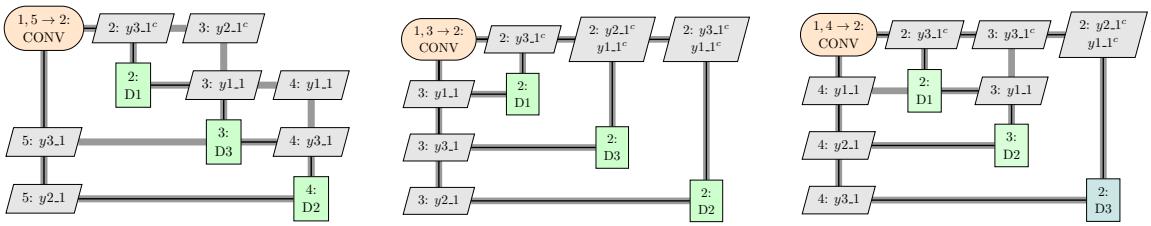
As mentioned in Section 2.2, the decomposition and sequencing of the MDAO system is an important part of formulating the FPG. The execution order of the disciplines is determined during the sequencing process. The execution order has a significant influence on the computational time needed to obtain a solution, as the order will determine the number of feedback connections in the MDAO system. As discussed in the previous section, feedback connections need to be converged by a converger or optimizer, which iterates over the MDAO system until a consistent solution is obtained. The complexity of the coordination process is determined by the number of couplings that need to be converged. When the number of feedback connections increases, the coordination complexity increases and thus the number of iterations will increase. This leads to an increase in execution time. Therefore, in general, the objective of the sequencing process is to minimize the number of feedback connections. This will minimize the number of iterations and thus reduces the execution time to solve the MDAO system.

However, the execution time of an MDAO system is not only determined by the number of iterations. The total execution time is the product of the number of iterations and the execution time of one iteration:

$$t_{tot} = n_{iter} \cdot t_{iter} \tag{2.4}$$

In which n_{iter} is the number of iterations, t_{iter} the execution time of one iteration and t_{tot} the total computational time needed to obtain the solution. n_{iter} can be reduced by minimizing the number of feedback connections, while t_{iter} can be reduced by decomposing the problem. As explained in Section 2.2, the decomposition process divides the disciplinary tools into several partitions. As the partitions have no data dependencies, they can be executed simultaneously. Therefore, the execution time of one iteration is reduced.

As discussed by Allison et al. [1] and Jung et al. [26], the decomposition and coordination process of an MDAO system are closely related. Increasing the number of partitions will decrease the execution time per iteration, but it will increase the coordination complexity as more connections need to be converged. This will cause an increase in the number of iterations. Therefore, a trade-off must be made between the number of partitions and the coordination complexity, such that the total execution time is minimized. This is an important concept that will determine the quality of the partitions in the decomposition algorithm discussed in Section 4.2. A small example explaining this concept is visualized in Figure 2.12.



(a) Gauss-Seidel convergence

(b) Jacobi convergence

(c) Convergence using two partitions. Partition 1 (D1, D2) is indicated in green and partition 2 (D3) in blue.

Figure 2.12: Different convergence schemas showing the interaction between the decomposition and coordination process

Figure 2.12a shows an MDA which is solved using a Gauss-Seidel convergence. This can be regarded as one partition in which all the disciplines are executed sequentially. If the disciplines have an execution time of 5, 6 and 10 seconds for disciplines 1, 2, and 3 respectively, the total computational time per iteration is 21 seconds. Figure 2.12b shows an MDAO which is solved using a Jacobi convergence. This can be seen as three partitions which are executed in parallel. The computational time per iteration will be 10 seconds. However, it can be expected that more iterations are needed to find a converged solution than for the Gauss-Seidel convergence. Lastly, two partitions are formed in Figure 2.12c. Disciplines 1 and 2 form the first partition, while discipline 3 forms the second partition. The computational time per iteration will be 11 seconds. This is only slightly more than for the Jacobi convergence. However, as discipline 2 uses the direct output from

discipline 1, it is possible that fewer iterations are needed to find the converged solution and that the total execution time is less.

This example clearly shows the importance of having a proper decomposition and sequencing. Too many partitions or feedback connections lead to a high computational complexity and therefore a high number of iterations. On the other hand, having too few partitions can lead to an inefficient use of the available computational resources and thus a longer execution time per iteration. Both situations will lead to a longer execution time than necessary. Therefore, it is important to take this concept into account during the formulation of the sequencing and decomposition algorithms in Chapter 4.

Sequencing and decomposition of MDAO systems have been performed before [35, 43]. Therefore, an overview of the different algorithms that can be used for both the sequencing and decomposition problem will be given in the next chapter.

3

Literature Review

Both the decomposition and sequencing problem, step 2C and 2D of Figure 2.6, are a combinatorial optimization problem. This means that the problem has a finite set of solutions. However, due to the large number of possible solutions, it is almost always impossible to try every combination in order to find the best one. Therefore, different algorithms exist for finding either the exact or an approximation of the best solution. This chapter will give an overview of the different algorithms that can be applied to the sequencing and decomposition problem together with their main advantages and disadvantages. Some of the algorithms that are discussed in this chapter, formed the start point for the algorithms that were developed in KADMOS, which will be discussed the next chapter.

The algorithms can be divided into four categories: exact algorithms, iterative improvement methods, metaheuristics and graph partitioning algorithms. The exact algorithms, described in Section 3.1, are characterized by the fact that they are guaranteed to find the global optimum of the optimization problem. However, these algorithms have a bad scalability as their runtime increases fast with increasing problem size. Therefore, the iterative improvement methods, explained in Section 3.2, approximate the optimal solution by iteratively improving the solution. These methods have a high probability of getting stuck in a local optimum. The metaheuristics from Section 3.3 contain methods to get out of the local optima. They therefore have an increased chance of finding the global optimum. The last type of algorithms, the graph partitioning algorithms presented in Section 3.4, use graph theory as the basis for their algorithms. The advantages and disadvantages of each algorithm are summarized and compared in Section 3.5. Finally, a short discussion on how the sensitivity information can be used to improve the sequencing and decomposition solutions is given in Section 3.6.

3.1. Exact Algorithms

Although it is generally impossible to try every decomposition and sequencing combination for large problems, there are some algorithms that are guaranteed to find the global optimum by searching through the solution space in a systematic manner. These algorithms are often a variation on the branch-and-bound algorithm [4].

Branch-and-bound algorithms search the solution space in a systematic manner by representing all possible solutions in a tree structure. Each node, also called a branch, represents a subproblem or a part of the solution space. Every iteration, the algorithm searches through the branches and decides which branch is best to explore next. This decision is made by calculating a bound for each branch. In case of a minimization problem, the branch with the lowest bound is most promising to explore. The algorithm can either continue until the entire solution space is explored, but can also terminate when it is guaranteed that the global optimum has been found. A branch is guaranteed a global optimum if it has the lowest bound of all branches and represents a full solution. [38]

Variations of the branch-and-bound algorithm differ in the subspace definition, branch evaluation and search strategies. They have been applied various times on the decomposition problem. For example, Hager et al. [21] used the branch-and-bound algorithm to find an optimal graph decomposition and Kusiak and Wang [32] used a variation of the algorithm for decomposition in MDAO problems.

The computational cost of the branch-and-bound algorithm is $O(Mb^d)$, in which M is the time needed

to explore a subproblem, b the number of children that can be derived from a branch, and d the three-depth [38]. Therefore, it can be concluded that the computational cost depends heavily on the algorithm formulation and the quality of the generated tree. Furthermore, the algorithms have a bad scalability as the runtime increases significantly with increasing b and d and thus problem size.

3.2. Iterative Improvement Methods

Iterative improvement methods, also called local search methods, try to improve a given solution x by searching the neighborhood for a better solution y [55]. The solution is only accepted if:

$$f(y) < f(x) \quad y \in N(x) \tag{3.1}$$

In which f is the objective function that must be minimized and $N(x)$ the neighborhood of x. The neighborhood of x is defined as all the solutions that can be obtained by one permutation of x. The algorithm terminates when no improvement can be found and the following condition is met:

$$f(x) \leq f(y) \quad \forall y \in N(x) \tag{3.2}$$

Different strategies can be used for moving to a better solution y. For example, the node swapping algorithms, described in section 3.2.1, move to the first improved solution they find, while the Kernighan-Lin and Fiduccia-Mattheyses algorithms, described in Sections 3.2.2 and 3.2.3 respectively, search the entire neighborhood before moving to the best solution.

The iterative improvement methods are based on trial-and-error. Therefore, they have a high probability of getting trapped in a local optimum. Once they are trapped, no techniques are present to escape the local optimum, so the algorithm is terminated. This in contrary to the metaheuristics as described in Section 3.3. To further improve the solution, the algorithm can be run multiple times with different starting points to obtain different local optima. The best optimum will then be chosen as the final solution. However, running the algorithm multiple times leads to an increased computational cost.

3.2.1. Node-Swapping Algorithms

The simplest method of the iterative improvement methods that can be applied to both sequencing as well as decomposition is the node swapping algorithms [34]. Swap algorithms are commonly used to solve the Traveling Salesman Problem [6, 23, 34, 41]. In the Traveling Salesman Problem, one tries to find the shortest route between a given number of cities, while each city may only be visited once. When a swap algorithm is used to solve this problem, the edges between the cities are swapped to reduce the total traveling distance.

The performance and accuracy of the swap algorithms depend on the number of edges that are swapped. Increasing the number of swapped edges per iteration will increase the number of possibilities with which they can be reconnected and thus the computational complexity increases. However, as the neighborhood of x is also larger, better solutions will be obtained, so the accuracy improves. The computational complexity of the algorithm is in the order of $O(n^k)$, in which n is the number of edges and k the number of edges swapped per iteration [41].

Even though the swap algorithms are generally applied to the Travelings Salesman Problem, they can easily be adapted to be used for the sequencing and decomposition in MDAO problems. Instead of swapping the edges, the nodes can be swapped to improve the sequence or decomposition. For example, Lu and Martins [35] improved the sequence by searching for a better position for each node. The algorithm did not continue until a local optimum was found, instead the algorithm terminated when a maximum number of iterations was reached. This algorithm can be seen as a variation on the single swap algorithm.

3.2.2. Kernighan-Lin Algorithm

The Kernighan-Lin Algorithm [30] is a variation of the node swapping algorithm. It is used for decomposition into two partitions. The mathematical representation of the decomposition problem is defined in the following equation [4]:

$$\begin{aligned} &V_1 \cup ... \cup V_k = G \\ &V_i \cap V_j = \emptyset \quad \forall i \neq j \end{aligned} \tag{3.3}$$

In the first line, $V_1 \cup ... \cup V_k = G$ represents the k partitions in which graph G is decomposed. The second line indicates that a node cannot be part of more than one partition. Edges between the partitions are called cut edges and defined as:

$$E_{ij} = \{\{u, v\} \in E : u \in V_i, v \in V_j, i \neq j\} \tag{3.4}$$

The Kernighan-Lin algorithm has the objective to minimize the total cut edge edge weight. The algorithm is based on the idea that every node in a partition has an inner and outer cost. The inner cost of a node is the sum of the edge weights with neighbors in the same partition. The outer cost of a node is the sum of the cut edge weights the node has with neighbors in the other partition.

The Kernighan-Lin algorithm applied to a small example case can be seen in Figure 3.1. In this example, the best partitioning is obtained when the edge between nodes 3 and 6 is cut. The algorithm starts with a random initial partition. The inner and outer cost for each node is calculated. For each node pair $P = \{\{u, v\} : u \in V_i, v \in V_j, i \neq j\}$, the gain of switching these two nodes is calculated. A positive gain indicates a decrease in total cut edge weight, while a negative gain indicates an increase. The node pair with the highest gain is switched and fixed. In this case, nodes 3 and 5 are switched. Switching these nodes reduces the number of cut edges from nine to six. Therefore, the gain is three. The nodes are switched and fixed, such that they cannot be switched again. In the second step, the new inner and outer cost for each node is calculated as well as the new gain for each node pair. Again, the node pair with the highest gain is selected, switched and fixed. In the example, nodes 6 and 2 are switched, obtaining a gain of five. The steps are repeated until all the nodes are switched and fixed and the original partition is obtained again. The partition with the lowest cut edge found so far, in this case the middle one, is chosen and the algorithm starts again with this partition as the next starting point. The algorithm terminates when no improved partitioning can be found. As there is no partition that will have a total cut edge weight lower than one, the algorithm will terminate after the second iteration.

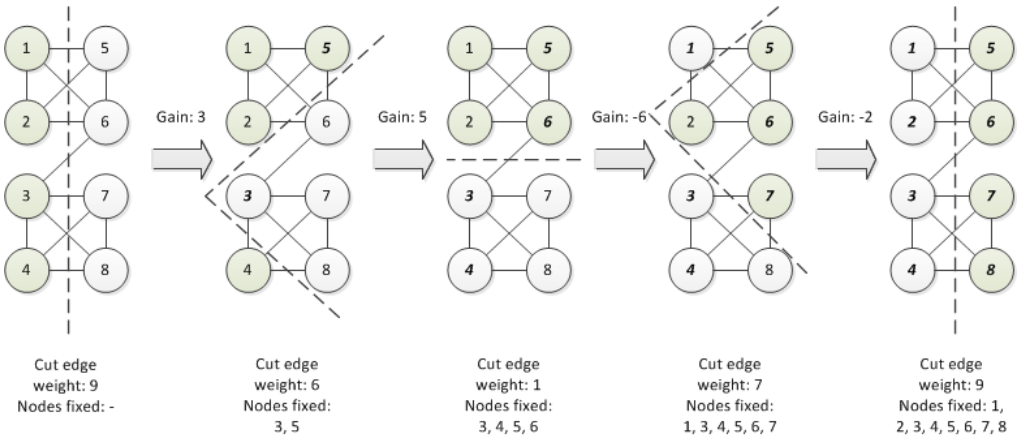


Figure 3.1: Kernighan-Lin Algorithm applied to a small example case [27]

In general, the KL algorithm performs better than the two-node swapping algorithm for only a slight increase in computational cost. Because the algorithm searches the entire neighborhood of x and allows negative gains, this algorithm can escape from the local optimum in which the two-node swapping algorithm will get trapped [16]. However, as it searches only the neighborhood of x, the KL algorithm will get stuck in another local optimum. The computational cost of the KL algorithm is $O(n^2 log(n))$ [4].

Karypis and Kumar [29] adapted the algorithm to decrease its computational cost. In their Boundary-Kernighan-Lin (BKL) algorithm, only boundary nodes are swapped. A boundary node is a node that has one or more connections with other partitions. Non-boundary nodes only have connections with other nodes within the partition. Therefore, they do not have to be considered by the algorithm.

3.2.3. Fiduccia-Mattheyses Algorithm

An important improvement of the KL algorithm is the Fiduccia-Mattheyses (FM) algorithm [16]. This algorithm uses the same principles as the KL algorithm. The difference is that instead of switching two nodes, only one node is moved at each step. To maintain balance between the partitions, the node with the highest

gain from the partition with the highest load must be moved. After the node has been moved, it is fixed such that it cannot be moved again during this iteration. After all the nodes are moved, or in case there is no free node in the partition with the highest load, the iteration is ended and the best partition found so far is the starting point for the next iteration.

The main benefit of the FM algorithm is the improved computational cost of nearly linear time: $O(n)$ [16]. This is a significant improvement compared to the KL algorithm. The quality of the solution is comparable to the KL algorithm as it searches the entire neighborhood and accepts negative gains just like the KL algorithm.

3.3. Metaheuristics

As mentioned in the previous section, the main disadvantage of the iterative improvement methods is that they get trapped in a local minimum and do not have any methods to get out of it. Metaheuristics is a class of methods that either use local information (the neighborhood, $N(x)$) but have active methods to get out of a local optimum or use global information to prevent getting trapped. An example of the first category is Tabu search, explained in Section 3.3.1. Examples of the latter are simulated annealing and genetic algorithms, described in Sections 3.3.2 and 3.3.3, respectively.

3.3.1. Tabu Search

Tabu search [19] is inspired by the human memory. It combines local search algorithms with a short-term memory list of previously visited solutions. It starts by searching the entire neighborhood $N(x)$ of the current solution in order to find the best neighbor. When the solution is a local optimum, the algorithm moves to the best neighbor even though this is a worse solution. To prevent the algorithm of going back to the previously visited optimum, the solution is placed in a tabu list. Solutions in the tabu list may not be visited again. This way the algorithm can climb out of local optima. No obvious stopping criteria exist for the algorithm, so it is terminated when the solution has not improved for the last n iterations, the value of the objective function is below a certain threshold or when a maximum number of iterations is reached.

Because storing all visited solution would require a lot of memory, the permutation used to climb out of the optimum can be placed in the tabu list instead. For the sequencing problem this could mean that two swapped nodes cannot be swapped back [19] and for the decomposition problem it could mean that a node is fixed in a certain partition [4]. Only a maximum number of tabus can be memorized, so after a certain amount of time, tabus are forgotten and the permutation is again allowed. When using permutations as tabus, not only previously visited solutions but also other solutions can be tabu even though they may contain a better solution. Therefore aspiration criteria are used to allow certain tabus to be overwritten. For example, if a neighbor is found that has a better objective value than the best solution found so far, it has not been visited for sure and the algorithm will move to that solution, whether the required permutation is allowed or not.

The benefit of the tabu search is that it is easily implemented and has the potential of finding an optimum solution. The computational time will be more than for a local search method, but less than for the other metaheuristic algorithms, which are explained hereafter [14]. The quality of the obtained solution depends on the formulation of the tabu and the number of tabus in the tabu list. A tabu definition can be strict or loose. Strict means that a lot of solutions are prohibited due to one tabu, while a loose tabu means only a few solutions are prohibited. A tabu definition that is too strict or a list that is too long can prevent the solution from going into promising solution areas. If the tabu definition is too loose or the list of tabus is too short, circularity can occur in which the algorithm keeps returning to previously visited solutions. [19]

3.3.2. Simulated Annealing

Simulated Annealing [58] is inspired by metallurgy and based on a physical analogy of cooling crystal structures. It starts by evaluating an initial solution and setting an initial temperature. Then another solution in the neighborhood of the current solution is picked randomly. Note that in contrary to the other algorithms, the neighborhood in this algorithm is not limited to the solutions obtained with one permutation, but to a predefined maximum number of permutations. In case the new solution is better, it is always accepted. In case the new solution is worse it depends on the current temperature what the probability of acceptance is. The higher the temperature, the higher the probability of accepting worse solutions. After several iterations with the same temperature, the temperature is decreased according to a cooling scheme. The algorithm terminates as soon as the temperature is cooled down until a final temperature of choice.

The algorithm uses the benefits of both global and local search. When the algorithm has a high temperature, it can be compared with a global random search. Therefore, it can easily escape local optima to search

for better regions. When the temperature is cooled down, the algorithm behaves like a local search algorithm to find the optima in the best region it has identified. [58]

The benefit of simulated annealing is that it can escape local optima by regularly accepting worse solutions. Furthermore, it is straightforward and easy to implement. The main disadvantage is that both the quality of the solution and the computational time needed to obtain the solution are strongly dependent on the settings used for the initial temperature, iterations per temperature, cooling scheme and stopping criteria. [48] For example, the initial temperature must be high enough such that the algorithm can escape from local optima. However, a higher initial temperature means also that more iterations need to be performed before the temperature has cooled down.

3.3.3. Genetic Algorithm

Genetic algorithms are derived from the principles of DNA and evolution theory. Using chromosomes, crossovers and mutations, different solutions are generated and evaluated on their fitness, in which better solutions have a higher fitness. Based on the survival of the fittest principle, solutions with a higher fitness have a higher probability of producing offspring. [31]

Genetic algorithms are commonly used in MDO for the decomposition and sequencing problem. Rogers [43] used a genetic algorithm for tool sequencing in his design tool DEMAID in order to minimize the number of feedback loops. McCulley and Bloebaum [37] extended the objective function to minimize not only the number of feedback loops but also the number of crossovers. A crossover is an intersection of two feedback loops in the DSM, without exchanging information at the intersection. This can be seen in figure 3.2 in which the DSM on the left-hand side contains crossovers and the DSM on the right-hand side has the crossovers eliminated. Park et al. [40] further improved the objective function by not only determining the optimal sequence of the tools but also dividing them over a fixed number of partitions. Finally, Jung et al. [26] included the number of partitions in the objective function to obtain the optimal number of partitions by allowing chromosomes to have a variable length.

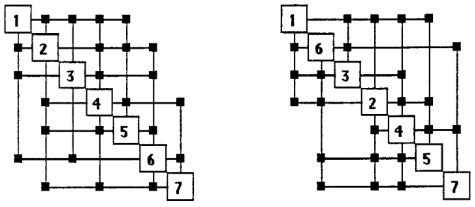


Figure 3.2: Example of a DSM with crossovers on the left side and without crossovers on the right side [37]

Genetic algorithms optimize the solution for all variables (e.g. number of partitions, tool sequence, etc.) at once. Furthermore, they use global information to obtain an improved solution and therefore the chance of obtaining a local optimum is reduced. Finding the global optimum is not guaranteed [31]. Its downside is that these algorithms generally have a high computational cost and are more complex to implement.

3.4. Graph Partitioning

Graph partitioning methods are algorithms that use the specific characteristics of a graph to find a good decomposition. Three different classes of graph partitioning methods will be described, namely the graph growing and bubble framework algorithms in Section 3.4.1, spectral partitioning in Section 3.4.2 and multilevel graph partitioning algorithms in Section 3.4.3.

3.4.1. Graph Growing and Bubble Framework

Graph growing algorithms [4, 29] are used to decompose graphs into two partitions. The algorithm starts a partition at a random node and adds nodes to this partition according to a breadth-first search. When half of the total node weight is assigned to the partition, the algorithm terminates. The remaining nodes form the second partition. The benefit of this algorithm is that it is simple and easy to implement. The main disadvantage is that the quality of the obtained solution depends on the chosen start node. In order to obtain a good solution, multiple runs of the algorithm are needed. The solutions can also be improved by combining it with an iterative improvement method from section 3.2. [4]

Figure 3.3 gives an example of this algorithm. In this example, all nodes have a weight of 1. Node 2 is

randomly chosen as the start point. The algorithm first adds all nodes to the partition which are connected to node 2, before moving to the next node in this case node 8. After node 9 is added to the partition, the algorithm terminates. The resulting partitions are: $V_1 = \{1,2,3,8,9\}$ and $V_2 = \{4,5,6,7\}$.

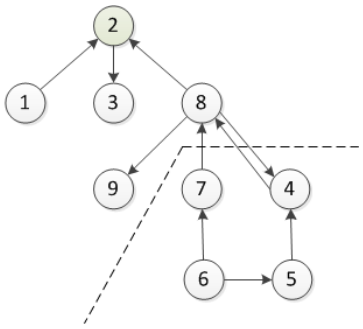


Figure 3.3: Graph partitioning using the graph growing algorithm

A variation of this method is the greedy graph growing algorithm [29]. Instead of using the breadth-first search, nodes are added to the partition based on their gains as defined in the KL algorithm. The node with the highest gain (= largest decrease or smallest increase in cut edge weight) is the preferred node to be added to the partition. Karypis and Kumar [29] found that the quality of the partitions found by this algorithm is less dependent on the starting node than for the graph growing algorithm. Therefore, less reruns are needed and the algorithm is faster. Furthermore, the obtained solutions are of better quality.

In case more than two partitions are needed, the bubble framework can be used [4]. The bubble framework is similar to the graph growing algorithm. For k partitions, it starts with k nodes and let the partitions grow according to k breadth-first searches, see figure 3.4. The obtained partitions are iteratively improved by selecting a new starting point in each partition and repeating the process.

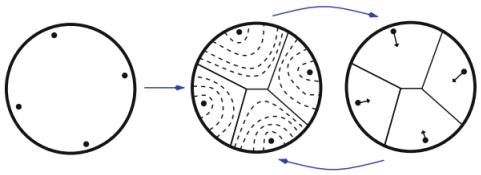


Figure 3.4: Graph partitioning using the bubble framework [4]

3.4.2. Spectral Partitioning

Spectral partitioning [54] uses algebraic graph theory to find a graph decomposition. Only undirected information is used, so digraphs must be transformed to their undirected variant first. From this undirected graph, the Laplacian is calculated according to equation 3.5.

$$L = D - W \tag{3.5}$$

In which L is the Laplacian matrix, D is a diagonal matrix with the degrees of each node and W is the weighted adjacency matrix. An adjacency matrix represents the connections between different nodes in matrix form. The adjacency matrix is a square matrix of size n x n in which n is the number of nodes in the graph. The entry a_{ij} of the adjacency matrix equals 1 if an edge exists from node i to node j, and 0 of no edge exists [13]. In case of a weighted adjacency matrix, the ones are replaced by the edge weights. From the Laplacian the eigenvalues are calculated and sorted such that:

$$\begin{aligned} Lv &= \lambda v \\ \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n & \end{aligned} \tag{3.6}$$

The eigenvector v_2 corresponding to the second smallest eigenvalue λ_2, also called the Fiedler vector, is used to determine the partition. All nodes with a value in v_2 lower than the median of v_2 form the first partition, while the other nodes form the second partition. [54]

To decompose the graph in more than two partitions, two methods can be used. The first is recursive spectral partitioning. After the graph is decomposed in two partitions, the algorithm repeats itself on the two partitions, obtaining four partitions. The number of partitions obtained with this method will always be 2^n. The benefit of this method is that it is easy to implement. The disadvantage is that it leads to a high computational cost. The second method is the k-way spectral partitioning method. This method uses k eigenvectors to decompose the graph into k partitions. It is computationally less expensive than the recursive spectral partitioning and results in better partitions. The main challenge in this algorithm is the determination of the eigenvalues and eigenvectors to be used. [25]

In general, spectral partitioning leads to a good quality of partitions, because it uses global information of the entire graph, instead of local information like the iterative improvement methods. The disadvantage is that it has a high computational cost because of the eigenvalue calculations. [29, 53]

3.4.3. Multilevel Graph Partitioning

Multilevel graph partitioning [24, 29] uses different sizes (or levels) of a graph to obtain a decomposition, as shown in Figure 3.5. A multilevel graph partitioning algorithm consists of three phases: the coarsening phase, initial partitioning phase and uncoarsening phase. During the coarsening phase, the size of the graph is reduced in several steps. The smallest graph is then partitioned in the initial partitioning phase and the result is projected back on the bigger graphs during the uncoarsening phase.

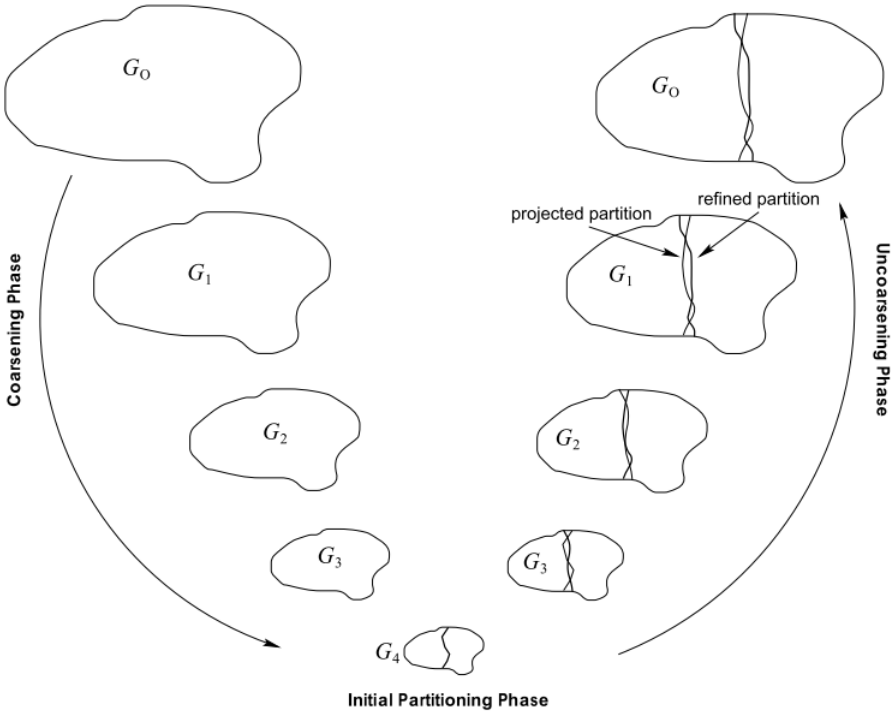


Figure 3.5: Overview of the different steps in multilevel graph partitioning [29]

Coarsening Phase

The first step is to reduce the size of the graph. In each coarsening step, multiple nodes are merged into multinodes, using maximal matchings [29]. A matching is a set of edges without having nodes in common. A maximal matching means that there is no edge in the graph that can be added to the matching. An example of a maximal matching is shown on the left-hand side of Figure 3.6. The edges that are part of the matching are indicated in green. No other edge can be added to this matching and therefore it is maximal. Note that a maximal matching is not unique and there could be multiple equally valid solutions. Coarsening of the graph

is done by collapsing the nodes of the matching edges into multinodes. This is shown on the right-hand side of Figure 3.6 in which the three edges are collapsed and three multinodes are created.

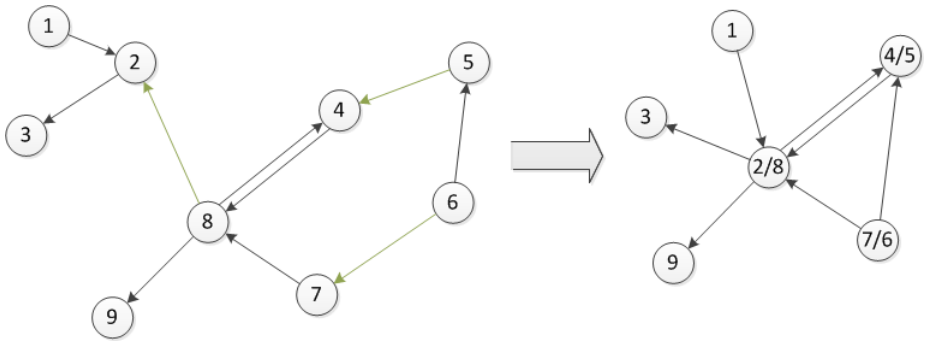


Figure 3.6: Maximal matching to coarsen a graph. The green edges indicate the matching set

There are several strategies to find a maximal matching. With random matching, a random node u is chosen as the start point. If this node has a neighbor v that is not matched yet, the edge $\{u, v\}$ becomes part of the matching. This is repeated until no edges can be found that can be added to the matching. In heavy edge matching [29] a random node u is also chosen as the start point. However, from all unmatched neighbors of u, the one with the highest edge weight is chosen to be part of the matching. The idea behind this is that the smallest graph will only have low edge weights and therefore the cut edge weight after partitioning will be low as well.

Initial Partitioning Phase

The smallest graph obtained during the coarsening phase is partitioned in the initial partitioning phase. This can be done using one of the partition techniques described in the other sections. Karypis and Kumar [29] suggested to use a spectral bipartition, the KL algorithm or a (greedy) graph growing algorithm. Hendrickson and Leland [24] used a spectral partitioning scheme followed by a KL algorithm.

Uncoarsening Phase

During the uncoarsening phase, the graph is uncoarsened by unmerging the multinodes to their original nodes. This is done using the reversed steps of the coarsening phase, as shown in Figure 3.5. Each time a graph is refined, the partitioning from the coarser graph is projected onto the finer graph. As the finer graph has more degrees of freedom than the coarser graph, an algorithm can be sued to further improve the partitioning. Algorithms commonly used for the partition improvements are the KL algorithm [24, 29, 56], the FM algorithm [24] or the BKL algorithm [29].

Even though multilevel graph partitioning consists of three phases, they are generally fast [29] as the decomposition of a small graph is easier and faster than the decomposition of a large graph. During the uncoarsening phase, the improvement algorithm starts already with a good estimation of the partitions. Therefore, only a few iterations are needed to find the local optimum. The disadvantage is that they are more difficult to implement. This difficulty can be avoided by using one of the available software packages, like for example Metis [28].

3.5. Algorithm Comparison

As can be concluded from the previous sections, different algorithms can be used to solve the decomposition and sequencing problem. Table 3.1 gives a summary of the advantages and disadvantages of each category as mentioned in the different sections. Table 3.2 gives a comparison of the characteristics of the different algorithms.

The exact algorithms are the only algorithms that are guaranteed to find the global optimum. However, due to their bad scalability, they will probably take to much time to find the optimum solution and are therefore only suitable for small MDAO systems only.

The iterative improvements methods are simple, fast and easy to implement algorithms and therefore suitable for larger MDAO systems. They do not find the global optimum, but will still give reasonable results. Therefore, these algorithms are most suitable to be implemented in KADMOS.

Table 3.1: Sequencing and decomposition methods and their main advantages and disadvantages

Method	Advantages	Disadvantages
Exact Algorithms	- Global optimum guaranteed	- Bad scalability
Iterative Improvement Methods	- Easy to implement - Computationally less expensive	- Sub-optimal solution
Metaheuristics	- Contains strategies to overcome local optima	- High computational cost
Graph Partitioning	- Same data representation as used in KADMOS	- More difficult to implement

Table 3.2: Overview of the characteristics of the decomposition and sequencing algorithms

Algorithm	*Supports:* Decomposition	Sequencing	Type of Optimum	Computational Time	Order
Branch-and-Bound	✓	✓	Global	High	$O(Mb^d)$
Node Swapping	✓	✓	Local	Low	$O(n^k)$
Kernighan-Lin Algorithm	✓	×	Local	Low	$O(n^2 log(n))$
Fiduccia-Mattheyses Algorithm	✓	×	Local	Low	$O(n)$
Tabu Search	✓	✓	Local	Medium	-
Simulated Annealing	✓	✓	Local	High	-
Genetic Algorithms	✓	✓	Local	High	-
Graph Growing and Bubble Framework	✓	×	Local	Low	- -
Spectral Partitioning	✓	×	Local	High	-
Multilevel Graph Partitioning	✓	×	Local	Low	-

The metaheuristics will escape from the local optima in which the iterative improvement methods get stuck and will therefore return better solutions. However, the metaheurstics have in general a high computational time. Therefore, they are less suitable for KADMOS.

The main benefit of the graph partitioning algorithms is that they use the same data representation as KADMOS, namely graphs. Therefore, they would provide good quality of partitions. However, they are also more difficult to implement. This can be overcome by using a an already available software package.

In conclusion, several algorithms can be used to decompose and sequence the FPG in step 2C and 2D of Figure 2.6. A selection of algorithms were chosen which formed the basis for the algorithms in KADMOS. The algorithms were further developed to meet the specific requirements for sequencing and decomposition of KADMOS graphs as will be explained in the next chapter.

The next section will give a short explanation of how the sequencing and decomposition solutions can be improved using sensitivity analysis. Sensitivity information is not available in KADMOS and therefore not used in the algorithms in KADMOS. However, due to their impact on the decomposition and sequencing solutions it is still shortly explained below.

3.6. Sensitivity Analysis

Sensitivity analysis can support both the sequencing and decomposition process and can be used in combination with all the different algorithms discussed in the previous sections. Using sensitivity information will improve the quality of the obtained solutions as an improved objective function can be formulated, as will be explained below.

Shaja and Sudhakar [45] used the sensitivities to make a prediction of the number of iterations that are needed to obtain a converged solution. Combined with the computational time of each tool, a prediction can be made of the total computational time of a given solution. Using the total computational time as the objective function for the sequencing will most likely give better solutions than using the number of feedback loops. For example, a sequence with two feedback loops with low sensitivity can have a lower total computational time than a sequence with one feedback with a high sensitivity. Indeed, Shaja and Sudhakar [45] used

this concept to evaluate the computational time of several sequences and selecting the most promising one.

Sensitivities also provide useful information in the decomposition process. Qiu et al. [42] and Hajikolaei et al [22] collected nodes with strong sensitivities in the same partition, while placing nodes with low sensitivities in different partitions. This way the dependency between different partitions is minimized. So, by using sensitivities as edge weights in the graph, better partitions can be found than using the amount of data transfer as edge weight. Furthermore, the partitions can be better balanced as the number of iterations within the partition can be estimated and therefore a better prediction of the total computational time for each partition is obtained.

In conclusion, the decomposition and sequencing problem can greatly benefit from sensitivity analysis. However, determining the sensitivities between all inputs and outputs in large systems is a difficult task [5]. Furthermore, all the disciplinary tools have to be executed at least twice to determine the sensitivities. Depending on the tools that are used, this can take a lot of time. Therefore, the sensitivities are not taken into account in the sequencing and decomposition algorithms that were developed for KADMOS, which will be discussed in the next chapter.

II

Algorithms

4

Methodology

One of the subgoals of this thesis is to develop algorithms for the automatic sequencing and decomposition of MDAO systems. The literature survey in the previous chapter showed that various methods exist for the sequencing and decomposition problem. However, not all of these methods are equally suitable to be applied in KADMOS. One of the goals of the AGILE project is to make MDAO more accessible to industry. Therefore, an important requirement for the sequencing and decomposition algorithms is not only accuracy but also user-friendliness. This means that the different algorithms must provide good quality of partitions and execution orders in a short time frame. Furthermore, the algorithms must be easy to use and therefore have a minimal number of settings.

In order to meet these requirements, multiple sequencing algorithms were developed in KADMOS, as will be explained in Section 4.1. These algorithms differ in accuracy and speed and are therefore suitable for different types of MDAO systems. Furthermore, one decomposition algorithm was developed and implemented in KADMOS as discussed in Section 4.2. The different algorithms are verified and validated in Chapter 5.

4.1. Sequencing Algorithms

The objective of the sequencing algorithms is to find the best execution order of the disciplinary tools. In this work, the best execution order is defined as the order which has the minimum number of feedback connections. As explained in Section 2.4, minimizing the number of feedback connections will minimize the coordination complexity and thus the number of iterations that are necessary to obtain a converged solution. The number of feedback connections is defined as the total number of feedback couplings and not the number of unique feedback variables. This means that if two different feedback couplings contain the same variable, this is counted as two feedback connections. The reasoning behind this that for example one output variable giving feedback to five different disciplines will have a bigger impact on the convergence rate than two output variables giving feedback to one disciplines. Therefore, the latter is regarded as a better execution order.

In addition to the feedback connections, the execution time of the sequence needs to be taken into account as well. In case two different orders have the same number of feedback connections, the order with the lowest execution time is preferred. The calculation of the execution time of a sequence is based on the assumption that the tools are executed as soon as all their input values are available. This means that in one sequence, multiple tools can be executed in parallel if they have no data dependencies. In practice, the maximum amount of parallel executions depends on the number of available CPUs, as will be shown in Section 6.5. Unless stated otherwise, it is assumed that there is no limit on the number of available CPUs as the maximum problem size considered during the research is fifty disciplinary tools. Running fifty tools in parallel on a single computer is not possible. However, running fifty tools in parallel on a cluster is no problem at all.

As mentioned above, multiple sequencing algorithms have been developed to sequence the disciplines in KADMOS. One algorithm belongs the exact algorithms (see Section 3.1) and will be explained in Section 4.1.1. The other algorithms belong to the iterative improvement methods (see Section 3.2) and will be discussed in Section 4.1.2. Each algorithm will be explained using a small example case which is shown in Figure 4.1. The example case consists of four disciplines. Each discipline has one output variable and an execution time of one second.

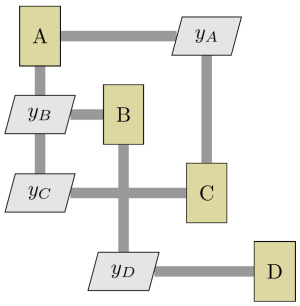


Figure 4.1: Example problem used to explain the sequencing algorithms. Each discipline has an execution time of one second

4.1.1. Exact algorithm

The exact algorithm that has been developed for KADMOS is a new implementation of the branch-and-bound algorithm. As explained in Section 3.1, a branch-and-bound algorithm searches the solution space in a systematic manner by repeatedly dividing the solution space into smaller subspaces using a tree structure. Each iteration, the algorithm searches through the subspaces or branches and decides which branch is most promising to explore next by calculating a bound. In case of a minimization problem, the branch with the lowest bound is most promising. The algorithm terminates when the global optimum has been found. [38]

The new implementation of the branch-and-bound algorithm was developed in KADMOS such that it could be applied to the sequencing problem. In this implementation, the solution space is divided into subspaces by repeatedly fixing one node in the sequence. As explained in the previous section, the objective is to minimize the number of feedback connections, while also minimizing the execution time of the sequence. Therefore, the bound of each branch is defined as the minimum number of feedback connections that are guaranteed to occur in the solution subspace as well as the execution time of the sequence that is fixed so far. The best branch is the branch which has the lowest number of feedback connections. In case multiple branches have the same number of feedback connections, the branch with the lowest execution time is explored first. When multiple nodes have the same bound for both the number of feedback connections and the execution time, a depth-first search is used. This means that the branch with the highest number of fixed nodes is explored first.

In Figure 4.2, the branch-and-bound algorithm is visualized for the small example problem from Figure 4.1. The entire tree is visualized in gray and the steps taken by the branch-and-bound algorithm are shown in black. Note that the full tree represents all possible solutions. In the first step, the solution space is divided into four branches by fixing the first node in the sequence. For each branch, the bound is calculated by determining the number of feedback connections and the execution time. For example, discipline A receives input from disciplines B and C. Therefore, independent of the remaining sequencing, at least two feedback connections will exist when discipline A is placed first. The execution time of discipline A is one second, so the bound is (2, 1). After calculating the bounds, it is found that branch 4 is the most promising, because this branch has the lowest bound for the number of feedback connections. Therefore, branch 4 is explored further by fixing the second node in the sequence. The bounds are again calculated and the most promising branch (1, 2, 3, 5, 6 or 7) is selected for further exploration. In this case, branch 6 has the lowest bound for the number of feedback connections and therefore, this branch is further explored.

The algorithm continues to explore the tree until the global optimum has been obtained. Note that branch 7 is explored before branch 9, because the execution time of branch 7 is lower. Furthermore, branch 14 is explored before branch 3, because branch 14 has more fixed nodes. Finally, note that the algorithm did not terminate when branch 22 was found, even though this is a full solution. The reason for this is that both branch 11 and 21 have a lower bound. Therefore, it is not guaranteed that branch 22 is a global optimum. Once branches 11 and 21 have been explored, it can be concluded that three optimum solutions are found (branches 22, 23 and 24). These three solutions are guaranteed global optima as all the other branches have the same or a higher bound. As all three solutions are equally valid, the algorithm randomly picks one out of the three and returns this one as the final solution. In this case, branch 22 is chosen and the final solution is [D, C, B, A] as shown in Figure 4.3. The solution has one feedback connection and node C and D can be executed in parallel. Therefore, the execution time of this sequence is three seconds.

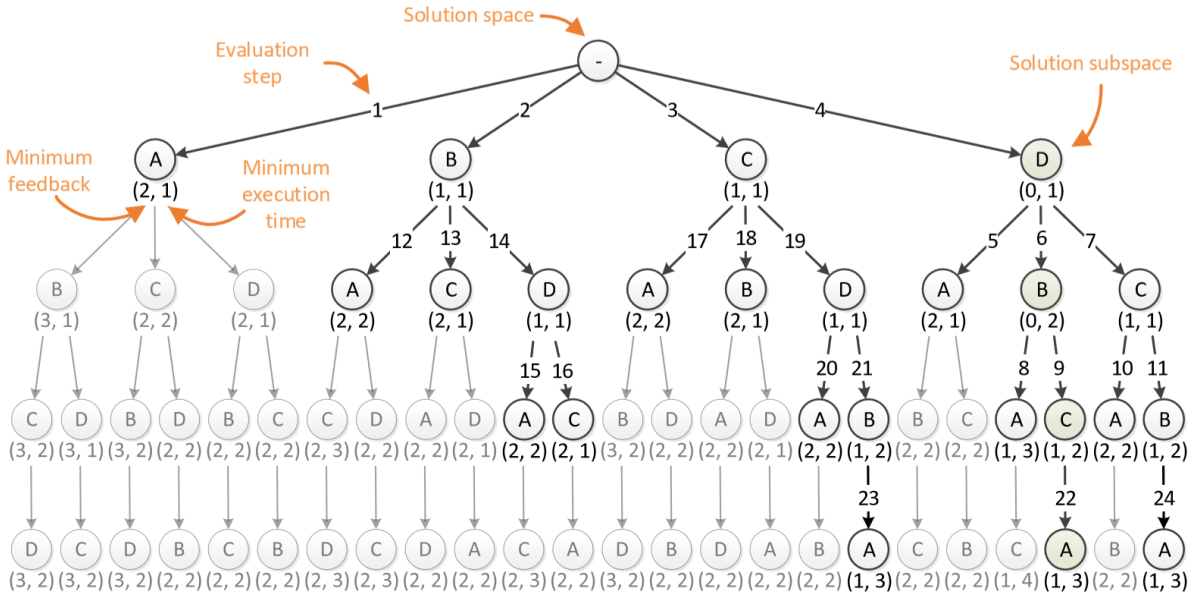


Figure 4.2: Overview of the steps taken by the branch-and-bound algorithm when minimizing the number of feedback connections, while also considering the execution time of the sequence. The entire solution space is visualized in grey, while the steps taken by the branch-and-bound algorithm are shown in black.

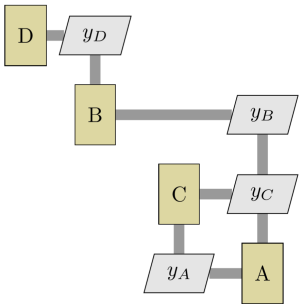


Figure 4.3: Final solution obtained by the branch-and-bound algorithm

4.1.2. Iterative Improvement Methods

Besides the branch-and-bound algorithm, three algorithms are implemented which are variations on the iterative improvement methods. As explained in Section 3.2, iterative improvements methods try to improve a given solution x by searching the neighborhood for a better solution y. The neighborhood of x ($N(x)$) is defined as all the solutions that can be obtained by one permutation of x. The algorithm terminates when no improvement can be found. [55]

Three different variations of the iterative improvement methods have been implemented: the single-swap, two-swap and hybrid-swap. These algorithms are variations on the node-swapping algorithms explained in Section 3.2.1. In Section 3.2.1, the algorithms were applied to the Travelings Salesman Problem. Here, they are adapted such that they can be applied to the sequencing problem. The three algorithms differ in the permutations of x that are allowed. Each of them will be shortly discussed below.

Single-swap

In the single-swap algorithm, a permutation consists of moving one node to a different place in the execution order. The algorithm systematically tries to find an improved position for each node. It starts at the back to find a better position for the last node. If no better position can be found, it continues to find an improved position for the second last node. Each time a better execution order is found, the algorithm starts again by finding a new position for the last node in the sequence.

An example is shown in Figure 4.4. In the first step, the last node (D) is moved to the first position. The new order is accepted as the number of feedback connections reduces from three to two. In the second step, the last node (C) is again moved to the first position as the previous order was accepted. Moving C to the

first position does not lead to an improved sequence as both the number of feedback connections (2) and the execution time (2) remain the same. Therefore, this solution is discarded in step three and C is moved to the second position in step 4. The algorithm continues, but cannot find a better position for node C. Therefore, node B is moved. An improved sequence is found when node B is moved to the second position as the number of feedback connections reduces from two to one. Note that the execution time increases from two to four seconds. The sequence is still accepted as the main goal is to minimize the number of feedback connections. The algorithm continues and the next sequence that is accepted is moving node C to the third position. This reduces the execution time from four to three seconds, while not increase the number of feedback connections. Again, the algorithms continues to find an improvement order. However, as no improved sequence can be found, the algorithm terminates and solution [D, B, C, A] is returned. Note that this is the same solution as the solution found by the branch-and-bound algorithm. However, in contrary to the branch-and-bound algorithm, the result obtained from the single-swap algorithm is not guaranteed to be a global optimum.

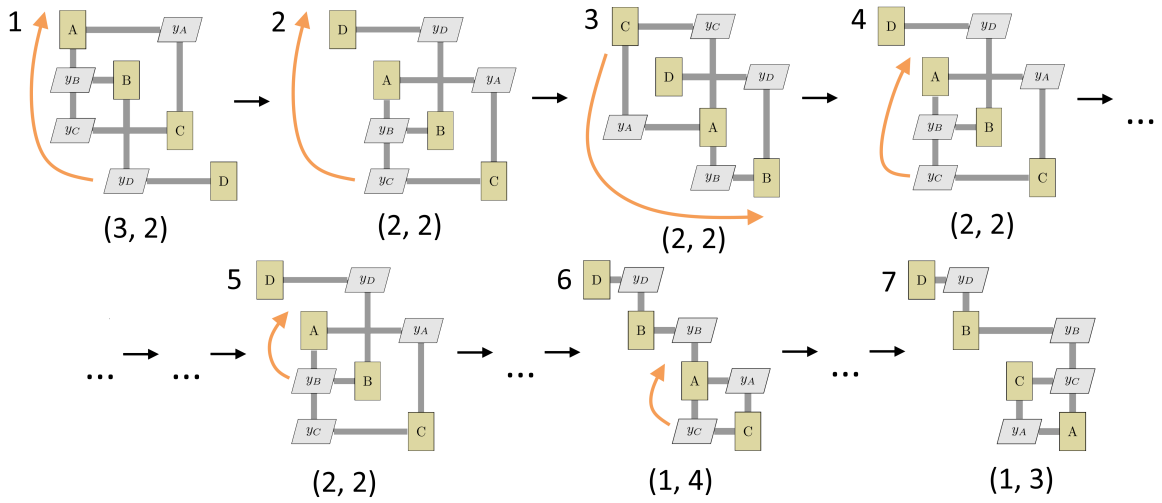


Figure 4.4: Overview of the steps taken by the single-swap algorithm when minimizing the number of feedback connections, while also considering the execution time of the sequence

Two-swap

In the two-swap algorithm, $N(x)$ consists of all the solutions that can be obtained by switching the positions of two nodes. The algorithms starts by finding a new position for the last node by switching it with the first node. If that sequence is not an improvement, the last node is swapped with the second node. If no improved position for the last node can be found, the algorithm tries to find a better position for the second last node, again by switching it with the first node. Each time an improvement is found, the algorithm starts from the beginning by swapping the first and the last node.

An example of the two-swap algorithm is shown in Figure 4.5. The algorithms starts with switching the first node (A) and the last node (B). The new sequence is an improvement as the number of feedback connections reduces from three to one. Therefore, the new order is accepted and the algorithm starts again from the beginning by switching the first and the last node. This leads to a worse sequence, so the solution is discarded in step three and the first and third node are swapped in step four. The algorithm continues to swap the nodes, however, as no improved solution can be found, the algorithm terminates and solution [D, B, C, A] is returned.

Hybrid-swap

Per permutation, more nodes are swapped in the two-swap than in the single-swap algorithm. Therefore, it is expected that the sequence of the two-swap will improve faster per accepted permutation than for the single-swap. However, the neighborhood of the single-swap algorithm consists of a larger number of permutations than the two-swap algorithm. So, it is also expected that the single-swap will be more accurate while the two-swap algorithm will be faster. Therefore, the hybrid-swap algorithm was implemented.

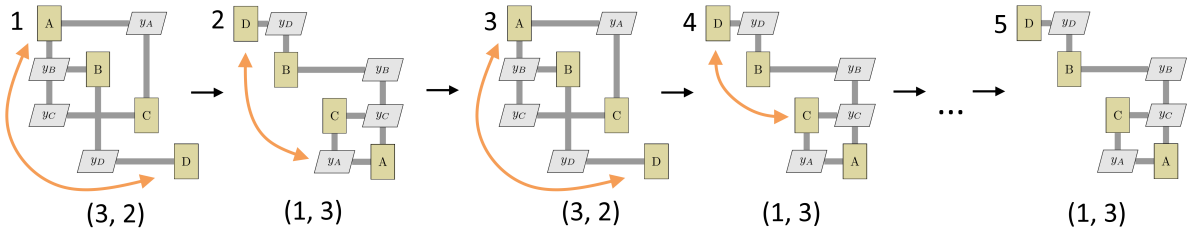


Figure 4.5: Overview of the steps taken by the two-swap algorithm when minimizing the number of feedback connections, while also considering the execution time of the sequence

The hybrid-swap algorithm will first find a local optimum using the two-swap algorithm. This solution is then taken as a start point for the single-swap algorithm. The single-swap algorithm has a larger neighborhood, so it can further improve the solution. An example of a local optimum which can be improved using the single-swap algorithm, but not with the two-swap algorithm is shown in Figure 4.6.

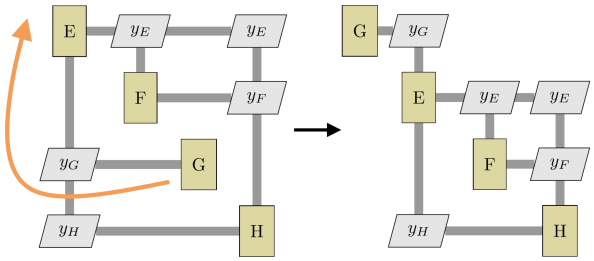


Figure 4.6: Example of a local optimum in the two-swap algorithm which is resolved by the single-swap algorithm

4.1.3. Full sequencing of MDAO systems

In section 2.2, the division of the disciplines into three categories was explained, step 2B in Figure 2.6. The three categories were the pre-coupling, coupled and post-coupling disciplines as visualized in Figure 2.7. As the pre- and post-couping disciplines have no circular dependencies, the number of feedback connections for these nodes is zero. Therefore, it is faster to sequence these nodes using a topologically sort instead of using the algorithms from the previous section.

A topologically sorted sequence is a sequence for which every directed edge $\{u, v\} \in E$, u comes before v [46]. The topological sorting is performed using the topological sort function of the Python package NetworkX[1]. The algorithm starts with finding all source nodes. A source node is a node which has only outgoing edges and no incoming edges. As no circular couplings are present, at least one source node must be present. The source nodes are placed at the beginning of the sequence and removed from the graph. By removing the nodes, new source nodes are created. These new source nodes are again added to the sequence and removed from the graph. This is repeated until all the nodes are sequenced. A more detailed explanation of the algorithm can be found in [46]. As it is assumed that each node will be executed as soon as all its data is available, the execution time of this sequence cannot be further improved.

4.1.4. Algorithm used for the Validation of Large Problems: Genetic Algorithm

The four algorithms described in Section 4.1.1 and 4.1.2 will be verified and validated in the next chapter using a brute-force algorithm. However, the number of possibilities in the sequencing problem is $n!$, in which n is the number of disciplines. Therefore, the execution time of the brute-force algorithm increases significantly when the number of disciplines is increased and can only be used to verify and validate the sequencing of small MDAO problems.

To verify and validate the larger MDAO problems, a Genetic Algorithm (GA) has been implemented. As explained in Section 3.3.3, the GA contains methods to escape local optima, while the iterative improvement methods do not. The results from the GA will be compared to the results obtained with the iterative improve-

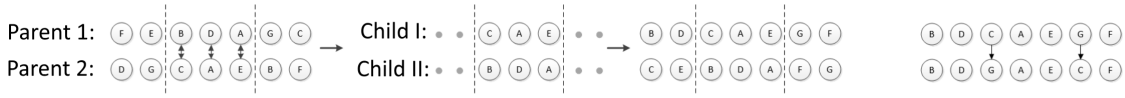
[1] `https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.dag.topological_sort.html`, accessed: January 12th 2019

ment methods. Of course, the GA is not guaranteed to find the global optimum, however it will give a good indication to see how the iterative improvement methods perform.

The GA is implemented using the Python package DEAP [18]. DEAP contains several predefined algorithms for performing GAs. The user only needs to specify the basic settings, like the layout of the individuals, cross-over method and mutation method. A full explanation of the DEAP package can be found in Fortin et al. [18]. As a cross-over and mutation are not straightforward in a combinatorial optimization, the methods used for both the cross-over and mutation are shortly explained below.

Each individual in the GA represents an execution order. A cross-over is performed using an ordered crossover[2] as shown in Figure 4.7a. In this case, two parent individuals are selected and randomly cut at two places as shown in step 1. The nodes between the cuts are swapped to create new children in step 2. Each node can only occur once in each individual. Therefore, the remaining nodes are wrapped around the fixed part according to the node order from the parents starting from the second cut. For example, child I receives nodes C, A and E from parent 2. Starting from the second cut, child I receives the remaining nodes from parent 1. In this case G, F, B and D. These nodes are wrapped around the fixed part of child I.

An example of a mutation is shown in Figure 4.7b. In this case, a mutation is created by swapping several random nodes.



(a) Cross-over (b) Mutation

Figure 4.7: Cross-over and mutation used in the Genetic Algorithm

4.2. Decomposition

One decomposition algorithm has been developed to automatically decompose the disciplines in KADMOS: MDK (Metis-based Decomposition of KADMOS graphs). As explained in Section 2.4, decomposition means that the disciplines are divided over several partitions, which can be executed in parallel. The connections between the different partitions are called cut edges (see equation 3.4 in Section 3.2.2). In relation to the coordination strategies from Section 2.3, these cut edges need to be removed and connected to a converger or optimizer, such that the partitions can be executed in parallel. The converger or optimizer is then responsible for converging these couplings and obtaining a consistent solution. Therefore, the cut edges (together with the feedback connections in the partitions) form the coordination complexity of a given partitioning.

4.2.1. Partition Quality

The first step in the decomposition process is to determine the quality of a given partitioning. As explained in Section 2.4, increasing the number of partitions will reduce the execution time of one iteration, but will increase the number of iterations as the coordination complexity is increased. So, the quality of a given partitioning is determined both by the execution time as well as the number of cut edges and feedback connections. Therefore, the objective function, stated in equation 4.1, is a weighted average between the coordination complexity and the execution time of the decomposition.

$$f = RCB \cdot \frac{n_{ce} + n_f}{n_c} + (RCB - 1)\,\frac{t_{decomp}}{\sum_{i=1}^{n_d} t_i} \tag{4.1}$$

In this equation, n_{ce} is the number of cut edges between the partitions. The number of feedback connections within the partitions is indicated with n_f. n_c is the total number of couplings between the disciplines in the graph. t_{decomp} indicates the execution time of the decomposition when the partitions are executed in parallel and $\sum_{i=1}^{n_d} t_i$ is the sum of the execution time of each discipline in the graph. Lastly, RCB stands for the Runtime-Coupling-Balance. The RCB is a factor between 0 and 1 that determines the weights in the weighted average between the coordination complexity and the execution time of the decomposition.

[2] `https://deap.readthedocs.io/en/master/api/tools.html#deap.tools.cxOrdered`, accessed: January 20th 2019

4.2.2. Metis

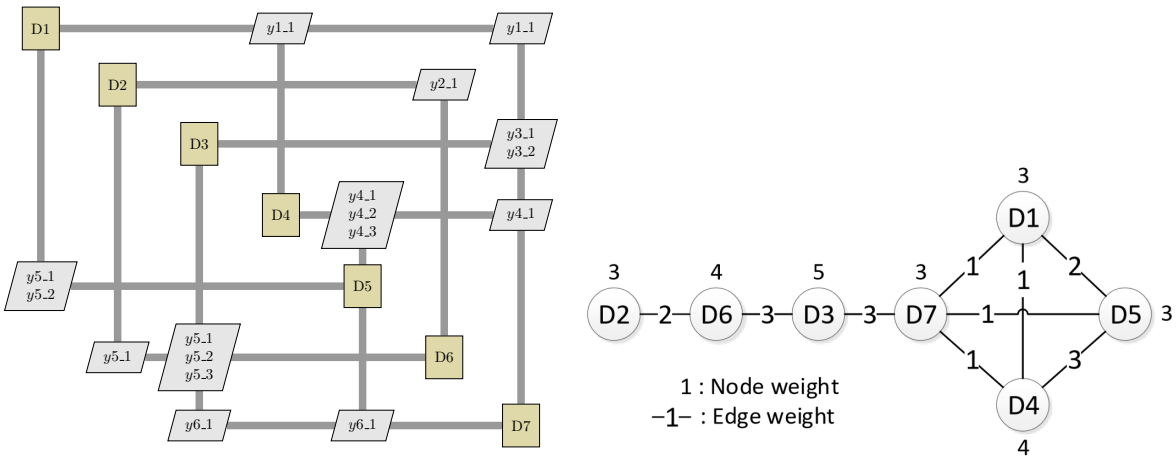
Developing a fast and accurate decomposition algorithm can be challenging and is time-consuming. Therefore, the MDK algorithm is based around the Metis package [28]. Metis is a ready-to-use software package used for graph partitioning which is known to produce high-quality partitions in a short time frame [28].

Metis uses a multilevel graph partitioning to obtain a decomposition of a given graph. Multilevel graph partitioning algorithms repeatedly reduce the size of a graph by merging several nodes. The smallest graph is partitioned and the results are projected back to the bigger graphs. A detailed explanation of the multilevel graph partitioning algorithms was given in Section 3.4.3.

In Metis, the coarsening of the graph is performed using a heavy-edge matching. As explained in Section 3.4.3, heavy-edge matching tries to include the highest edge weights in the maximal matching. Using this method, the cut edge weight after partitioning the smallest graph will be low, as only the low edge weights are present in this graph.

The initial partitioning of the smallest graph is performed using a KL-algorithm (see Section 3.2.2). Note that the KL-algorithm decomposes a graph into two partitions only. If more partitions are needed, Metis uses a recursive bisection. This means that the graph is initially decomposed in two partitions. The resulting partitions are then again decomposed in smaller partitions until the required number of partitions is obtained. A more detailed explanation of the recursive bisection used in Metis can be found in Karypis and Kumar [29]. Finally, Metis uses a BKL algorithm (see Section 3.2.2) to refine the graphs in the uncoarsening phase.

Note that the graphs that Metis uses during the decomposition are slightly different than the graphs used in KADMOS. An example is shown in Figure 4.8. Figure 4.8a shows a data graph as used in KADMOS (visualized using an XDSM diagram), while Figure 4.8b shows the corresponding Metis graph.



(a) KADMOS data graph, visualized using an XDSM diagram

(b) Metis graph

Figure 4.8: Example of the differences between a KADMOS data graph and Metis graph

The main difference is that the Metis graphs are undirected and have no variable nodes. This means that both the feedforward and feedback connections between two nodes are represented by only one edge. The number of variables that are passed between the two nodes is indicated using an edge weight. Furthermore, one or more node weights can be assigned to the nodes in the Metis graphs. In this case, each node gets one node weight which represents the execution time of the node.

4.2.3. Balance Factor

The default goal in Metis is to minimize the number of cut edges, while also balancing the total node weight of the different partitions. However, the objective function that was defined in Section 4.2.1 stated that the objective is a weighted average between the execution time and number of disconnected couplings. So, an unbalanced solution might be a better solution if this reduces the number of cut edges significantly. Therefore, a balance factor was calculated and passed on to Metis. The balance factor indicates how much Metis is allowed to deviate from the target node weight for each partition. Using the balance factor, Metis will still try to minimize the number of cut edges but it will return more unbalanced partitions if that favors the number of cut edges. In Metis, the balance factor is defined as [28]:

$$u = 1 + \frac{x}{1000} = max_i \left(\frac{w[j,i]}{t[j,i]} \right) \tag{4.2}$$

In which, u is the balance factor and x is the value that must be given to Metis. $w[j,i]$ is the node weight of a partition, while $t[j,i]$ is the target weight of a partition. In this case, the target weight for each partition is defined as the total node weight divided by the number of partitions. Note that x is the value that must be given to Metis. Therefore, equation 4.2 is rewritten into:

$$x = \left[max_i \left(\frac{w[j,i]}{t[j,i]} \right) - 1 \right] \cdot 1000 \tag{4.3}$$

The maximum node weight of a partition $w[j,i]$ is calculated based on the RCB. If the RCB is zero, the objective is a minimization of the execution time. This is generally achieved by perfectly balancing the node weights. Therefore, the maximum node weight of a partition is, in this case, set to be equal to the target weight. If the RCB is one, the objective is a minimization of the cut edges and number of feedback connections, so the best solution can be strongly unbalanced. Therefore, the maximum weight of a partition is set to highest partition weight that can occur in the partitioning. When the RCB is set to a value between zero and one, the maximum node weight of a partition is calculated using a linear interpolation:

$$w(RCB) = (w_{RCB=1} - w_{RCB=0}) \cdot RCB + w_{RCB=0} \tag{4.4}$$

4.2.4. Limitiations of Metis

The main benefit of using Metis is that it is fast and proven to give good quality partitions [28]. However, there are three limitations when it is applied to the decomposition of the FPGs in KADMOS:

L1: *Metis can only decompose undirected graphs*
As mentioned in Section 4.2.2, the graphs used by Metis are undirected and the feedforward and feedback connections between two nodes are represent by only one edge. Therefore, Metis cannot distinguish between feedforward and feedback connections between two nodes.

L2: *Metis cannot take the execution order within a partition into account*
Even if the first issue could be resolved, Metis has no sequencing option for the nodes within a partition. Therefore, the number of feedback connections are not included when the quality of the decomposition is assessed by Metis.

L3: *Metis cannot take the parallel execution of disciplines within a partition into account*
As explained in the previous section, Metis balances the execution time of the partitions. However, if two disciplines in one partition have no data dependency, they can potentially be executed in parallel depending on the execution order within the partition (see for example Figure 4.3 in Section 4.1.1). Therefore, the actual execution time of a partition can be smaller than predicted by Metis.

The Metis package cannot be altered directly. Therefore, the MDK algorithm was developed around Metis to overcome some of these limitations.

4.2.5. Full Decomposition Algorithm: MDK

An overview of the complete decomposition algorithm is shown in Figure 4.9. The algorithm will be explained based on a small example shown in Table 4.1. The example consists of 7 disciplines. The disciplines differ in their execution time. Nodes A, B, E and G have an execution time of three seconds. Nodes D and F have an execution time of four seconds and node C five seconds. The goal is to decompose the problem into two partitions.

The input of the decomposition algorithm is the RCB value and the MDAO architecture that will be applied to the problem. Based on the MDAO architecture, different nodes are selected for the decomposition in step 1. Only those nodes are selected which are present in the smallest iteration loop. For example, if the MDF architecture is chosen, the nodes present in the converger loop will be decomposed. The nodes which are present in the converger loop are the coupled functions (see Figure 2.7). In case the IDF architecture is chosen, the nodes in the optimizer loop will be partitioned, which are the pre-coupling, coupled and post-coupling nodes. The example in Table 4.1 consists of coupled nodes only. Therefore, all nodes will be used for the decomposition, independent of the chosen architecture.

In the second step, the KADMOS data graph is translated to a Metis graph. As explained in Section 4.2.2, a KADMOS graph is a digraph with both function as well as variable nodes, while a Metis graph is undirected and only contains function nodes. So, as can be seen in the second row of Table 4.1, the variable nodes are removed and replaced by edge weights. Furthermore, node weights are added which represent the execution time of the disciplines. Also, the feedforward and feedback edges between function node pairs are merged.

The balance factor from Section 4.2.3 is calculated using equations 4.3 and 4.4 in step three. Note that the value of the balance factor depends on the value of the RCB, which has been set as input. In this example, the RCB is set to a value of 0.5. The total node weight in the Metis graph is 25. Therefore the target weight for each partition is 12.5. Furthermore, the most unbalanced case would be if a node with node weight three is placed in one partition and the other nodes in the second partition. Therefore, the maximum node weight of a partition is 22. The calculation is shown in third row of Table 4.1. The value of x that is given to Metis is in this case 380.

The next step is the partitioning of the graph using Metis. As can be seen in the fourth row of Table 4.1, Metis creates two partitions, with nodes B, F and C in partition 1 and nodes G, A, E, D in partition 2. This leads to a cut edge weight of three and a total node weight of 12 for partition 1 and 13 for partition 2.

As this is the first iteration, the algorithm will continue with step 6 as shown in Figure 4.9. The second limitation of Metis was that it cannot take the execution order within a partition into account (see Section 4.2.4). Therefore, the sequencing algorithms from Section 4.1 are used to determine the execution order of the nodes in step 6. Sequencing is performed on the KADMOS graph and the result for the example is shown in the sixth row of Table 4.1.

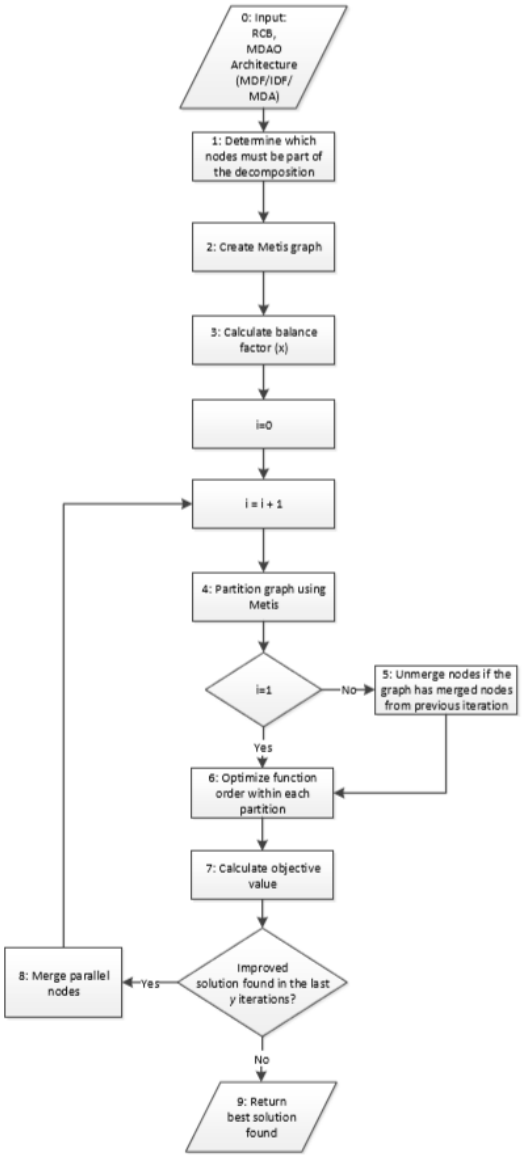


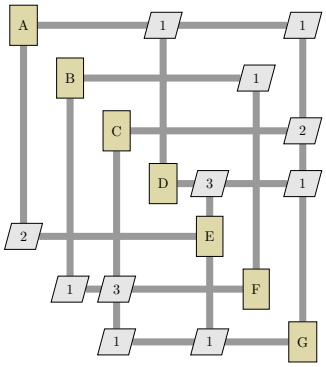
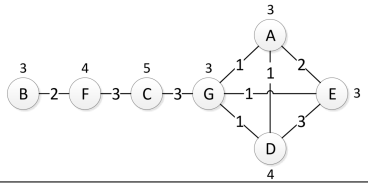
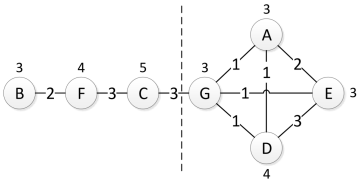
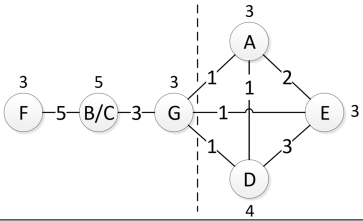
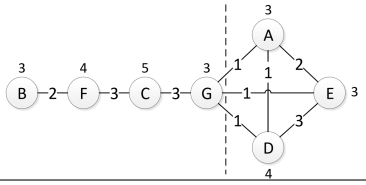
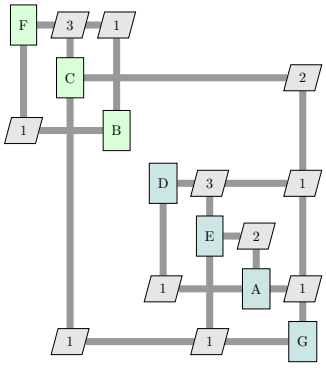
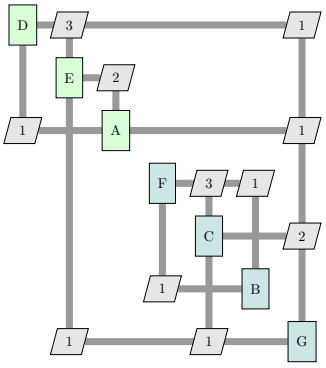
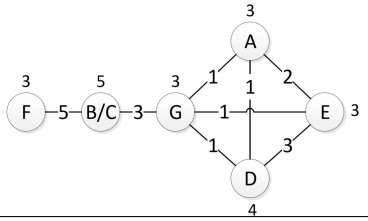
Figure 4.9: Flowchart of the MDK algorithm

After the sequencing of the nodes, the decomposition is complete and the objective value as defined in Section 4.2.1 can be calculated using equation 4.1 as shown in the seventh row of Table 4.1. In this case, the sum of the execution time of the nodes and total number of couplings is 25 and 17, respectively. The partition has 3 cut edges, 3 feedback variables and an execution time of 12 seconds. Therefore, the objective value is 0.42. This value is checked against the stopping criteria. The stopping criteria is satisfied when the maximum number of iterations without an improvement is reached. As this is the first iteration, the stopping criteria is not yet met, and the algorithm continues with step 8 as shown in Figure 4.9.

As explained in Section 4.2.4, the third limitation of Metis is that it cannot take the parallel execution of the disciplines within a partition into account. Therefore, the execution time of a partition can be lower than predicted by Metis. This is the case in the example shown in Table 4.1. In the first partition, nodes B and C have no data dependencies. Therefore, they can be executed in parallel. In order for Metis to make a better execution time estimation, these two nodes are merged in step 8. Node C has an execution time of five seconds and node B an execution time of three seconds. Therefore, the combined node B/C will be given an execution time of five seconds.

In the second iteration, the merged graph is again partitioned using Metis. As can be seen in the second column of Table 4.1, the partitioning has changed compared to the previous partitioning due to the merged node. Node G has moved from the second partition to the first partition. The resulting partitioning has still a cut edge weight of three, but the execution time of partition 1 is now 11 seconds and 10 seconds for partition

Table 4.1: Example of the MDK algorithm

Step	Iteration 1, i=1	Iteration 2, i=2
1		
2		
3	$x = 0.5 \cdot \left[0 + \left(\frac{22}{12.5} - 1\right) \cdot 1000\right] = 380$	$x = 0.5 \cdot \left[0 + \left(\frac{18}{10.5} - 1\right) \cdot 1000\right] = 357$
4		
5	Skipped if i=1	
6		
7	$f = 0.5 \cdot \frac{6}{17} + 0.5 \cdot \frac{12}{25} = 0.42$	$f = 0.5 \cdot \frac{6}{17} + 0.5 \cdot \frac{11}{25} = 0.40$
8		

2. Therefore, the total execution time of the decomposition has reduced from 12 to 11 seconds.

Before the graph is sequenced using the sequencing algorithms from the previous section, the merged nodes are unmerged to their original nodes. This is done to increase the degree of freedom to obtain better results for the sequencing process. The objective for the decomposition is again calculated and the stopping criteria is checked. If the stopping criteria is satisfied, the algorithm terminates. If the stopping criteria is not met, a new iteration loop is started.

In this chapter, the theoretical explanation of the sequencing and decomposition algorithms has been presented. In the next chapter, the algorithms are applied to a scalable mathematical problem to verify and validate them and to test their performance.

5

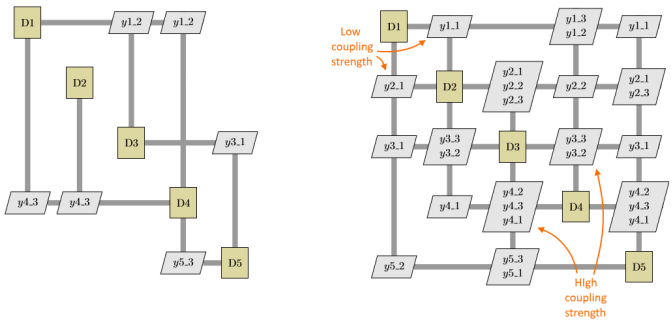
Verification & Validation

After the sequencing and MDK algorithm have been implemented, they must be verified and validated. Furthermore, it is important to know how they perform on different types of MDAO systems. Therefore their solution quality and performance is tested in Sections 5.1 and 5.2 for the sequencing and MDK algorithms, respectively.

To easily generate a lot of different MDAO systems, the Variable Complexity Problem (VCP) from Zhang et al. [57] was implemented. The VCP is a scalable mathematical problem that allows for the quick generation of different types of MDAO problems. In the mathematical problem, the following variables can be set:

- The number of disciplines
- The number of output variables per discipline
- The number of local and global design variables
- The number of local and global constraints
- The coupling density
- The execution time of each discipline

Note that the coupling density differs from the coupling strength. The coupling density is the percentage of connections that are made between the output variables and the disciplines with respect to all possible connections. The coupling strength is the number of variables that are passed between two disciplines. The difference is shown in Figure 5.1. An MDAO problem with a low coupling density is shown in Figure 5.1a. The problem has five disciplines. Each discipline has three output variables, which can be passed to the other four disciplines. Therefore the total number of possible connections is $5 \cdot 3 \cdot 4 = 60$. Only six couplings are present, so the coupling density for this problem is $6/60 = 0.1$. Figure 5.1b shows a problem with a higher coupling density. In this case 27 couplings are present, so the coupling density is $27/60 = 0.45$. This figure also shows the difference between a higher and lower coupling strength. Disciplines D1 and D2 have a lower coupling strength of 2, while disciplines D3 and D4 have a higher coupling strength of 5.



(a) Low coupling density (0.1)

(b) High coupling density (0.45)

Figure 5.1: Difference between coupling density and coupling strength. Each discipline has three output variables

Note that the VCP test cases are randomly generated based on the input variables. So, in order to make conclusions about the accuracy and the performance of the algorithms, each test has to be repeated multiple times to obtain a good average and to rule out coincidences. Therefore, in all the figures presented in this chapter, each test was performed using 200 randomly generated test cases. The average result of the 200 test cases and their 95% confidence interval is plotted in the results. The confidence interval is calculated using the following equation [12]:

$$(\bar{x}_n - \frac{2z_{\alpha/2}s_n}{\sqrt{n}},\ \bar{x}_n + \frac{2z_{\alpha/2}s_n}{\sqrt{n}}) \tag{5.1}$$

In this equation, $\bar{x}_n$ is the average value, s_n is the standard deviation, n the number of test cases (200) and $z_{\alpha/2}$ the value for the confidence level. In this case, $z_{\alpha/2}$ was set to 1.96 to obtain a confidence interval of 95%.

In this chapter, the different MDAO systems are sequenced and decomposed, but not yet solved. The execution of the VCP will be performed in Chapter 6. Therefore, the detailed explanation of the mathematical equations will be explained in that chapter, as they are not relevant for the verification and validation of the sequencing and decomposition algorithms.

5.1. Sequencing

The sequencing algorithms are verified, validated and tested by applying them to different types of MDAO systems. As explained in Section 2.4, the objective is to minimize the number of feedback connections while also taking the execution time of the sequence into account. The verification and validation of the sequencing algorithms is explained in Section 5.1.1, while the performance is discussed in Section 5.1.2.

5.1.1. Verification & Validation

The quality of the generated solutions of the sequencing algorithms is assessed using a brute-force algorithm. However, the brute-force algorithm is only suitable for small MDAO systems, due to its bad performance scalability. Therefore, as explained in Section 4.1.4, a GA has been implemented to validate the sequencing solutions for bigger test cases.

Figures 5.2 and 5.3 show the results for the cases with a relatively small amount of disciplines. The test cases had a coupling density (ρ_c) of 0.08, 5 coupling variables per discipline (n_{cv}) and no clusters. Figure 5.2a shows the average number of feedback couplings, while figure 5.3a visualizes the average execution time of the obtained sequences. Furthermore, Figure 5.2b shows the average difference in feedback couplings, while Figure 5.3b depicts the average difference in execution time with respect to the exact solution.

The average difference in feedback couplings and execution time are calculated using equations 5.2 and 5.3, respectively.

$$\Delta n_f = \frac{\sum_{i=1}^{n_{tc}} \left(n_{f_{approx}} - n_{f_{exact}} \right)}{n_{tc}} \tag{5.2}$$

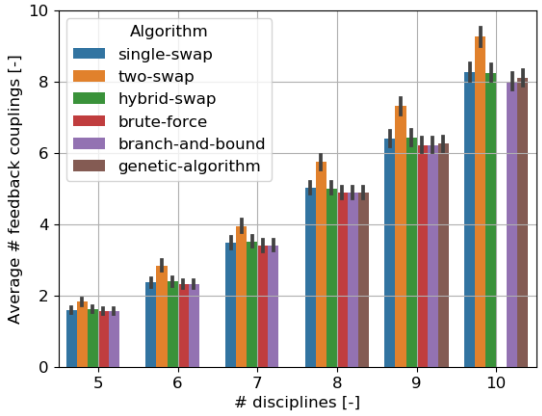
$$\Delta t = \frac{\sum_{i=1}^{n_{tc}} \left(t_{approx} - t_{exact} \right)}{n_{tc}} \tag{5.3}$$

In these equations, Δn_f is the average difference in feedback couplings with respect to the exact solution. $n_{f_{approx}}$ is the number of feedback couplings when the solution is obtained using one of the swap algorithms, the branch-and-bound algorithm or the GA. $n_{f_{exact}}$ is the number of feedback couplings when the solution is obtained using the brute-force algorithm. Furthermore n_{tc} is the number of test cases. As explained above, each test has been performed using 200 different test cases. Δt is the average difference in execution time of the sequences with respect to the exact solutions. t_{approx} is the execution time when the solution is obtained using the swap algorithms, branch-and-bound algorithm or the GA. Lastly, t_{exact} is the execution time when the solution is obtained using the brute-force algorithm.

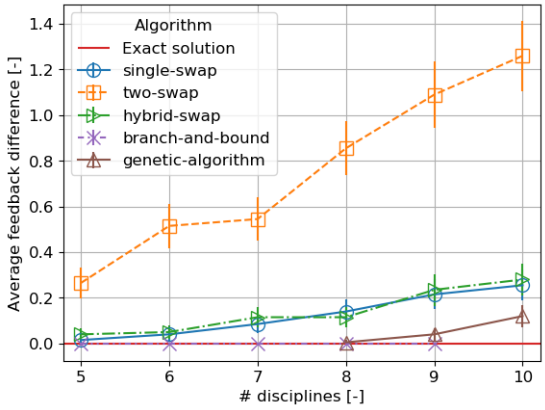
Figures 5.2 and 5.3 show that the branch-and-bound algorithm always find the same solution as the brute-force algorithm. This is expected, as the branch-and-bound algorithm is an exact algorithm which is guaranteed to find the global optimum.

When comparing the three swap algorithms, it can be concluded that the two-swap algorithm is the least accurate, while the hybrid-swap and single-swap have a similar accuracy. As explained in Section 4.1.2, the single-swap has a larger neighborhood $N(x)$ than the two-swap algorithm. Therefore, the single-swap can explore more options than the two-swap algorithm. This results in a higher accuracy for the single-swap algorithm when compared with the two-swap algorithm. Finally, it can be concluded that the hybrid-swap has a similar accuracy as the single-swap algorithm. This indicates that the local optimum in which the two-swap algorithm gets stuck, can indeed be resolved by refining the solution using the single-swap algorithm.

Finally, it can be concluded that, in these cases, the GA performs better than the swap algorithms. As explained in Section 3.3, the GA can escape the local optima in which the swap algorithms get stuck. Therefore, the solution accuracy of the GA is higher than for the swap algorithms.

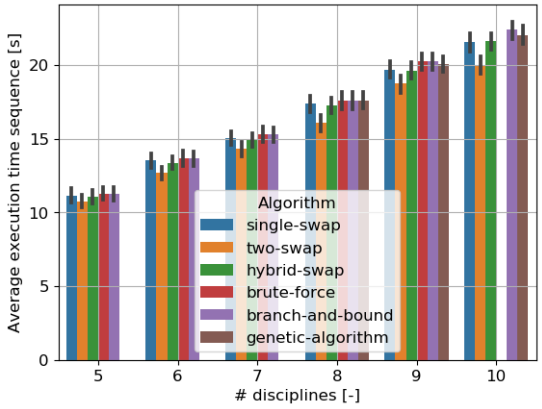


(a) Average number of feedback couplings

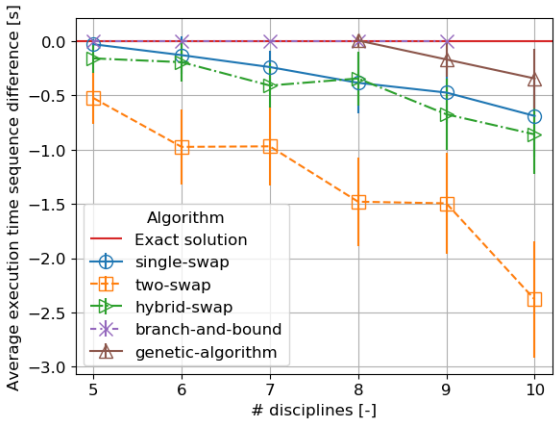


(b) Average difference in feedback couplings with respect to the exact solution

Figure 5.2: Solution accuracy of the sequencing algorithms for a small amount of disciplines when comparing the number of feedback couplings. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters



(a) Average execution time



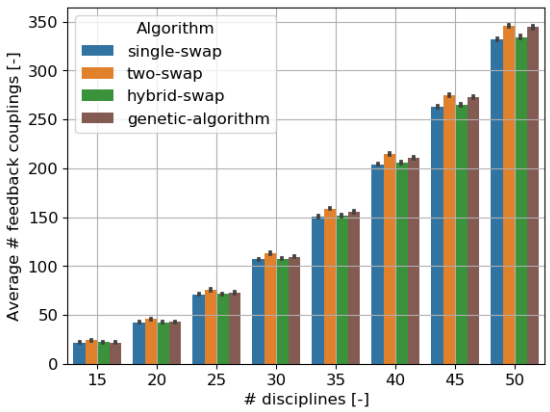
(b) Average difference in execution time with respect to the exact solution

Figure 5.3: Solution accuracy of the sequencing algorithms for a small amount of disciplines when comparing the execution time of the obtained sequences. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters

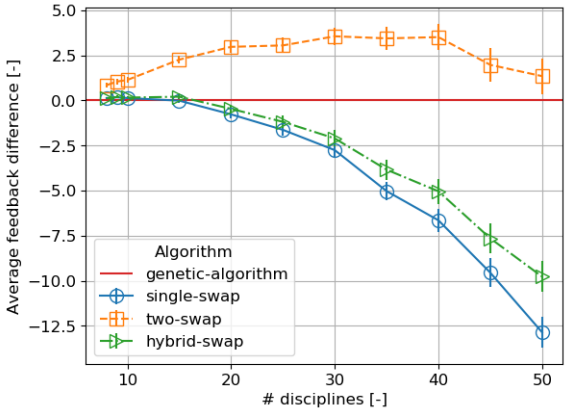
Figures 5.4 and 5.5 show the results when the number of disciplines is increased. The branch-and-bound and brute-force algorithm are not shown. Due to their bad performance scalability, they are only suitable for small MDAO systems. Figures 5.4a and 5.5a show the average number of feedback couplings and average execution time of the obtained sequences, respectively. Figures 5.4b and 5.5b show the average difference in number of feedback couplings and execution time. These values are again calculated using equations 5.2 and 5.3, except in this case the difference is not calculated with respect to the exact algorithm, but with respect to the GA.

When comparing the different swap algorithms, it can be concluded that in these cases, the two-swap algorithm has the lowest solution accuracy, while the single-swap algorithm has the highest solution accuracy. This difference is again due to the differences in the size of the neighborhood. Furthermore, it can be concluded that the hybrid-swap performs only slightly worse than the single-swap algorithm. Finally, it can be concluded that the GA performs better than the two-swap, but worse than the single and hybrid-swap.

Besides the influence of the number of disciplines, also the coupling density and number of coupling variables per discipline were varied. The corresponding figure are shown in Appendix A. For these figures, similar conclusions can be made. The branch-and-bound algorithm always finds the global optimum, while the two-swap is the least accurate of the swap algorithms and the single-swap the most accurate.

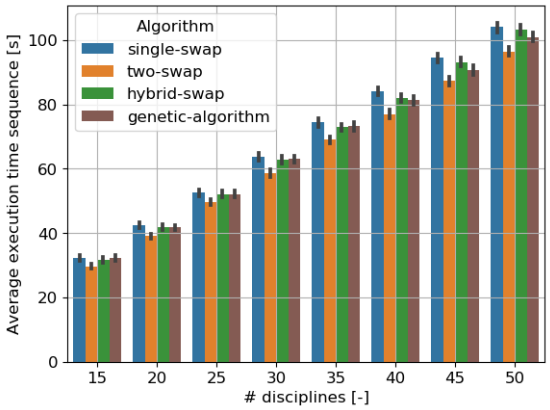


(a) Average number of feedback couplings

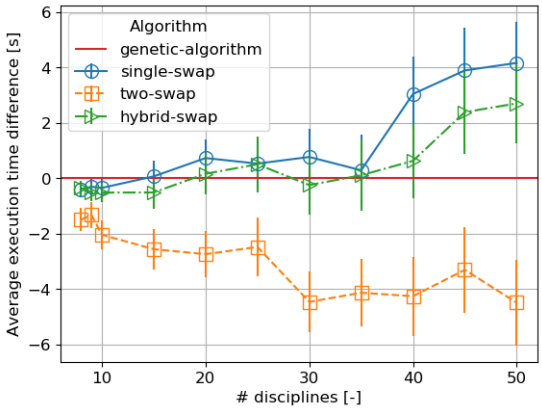


(b) Average difference in feedback couplings with respect to the GA

Figure 5.4: Solution accuracy of the sequencing algorithms for a large amount of disciplines when comparing the number of feedback couplings. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters



(a) Average execution time



(b) Average difference in execution time with respect to the GA

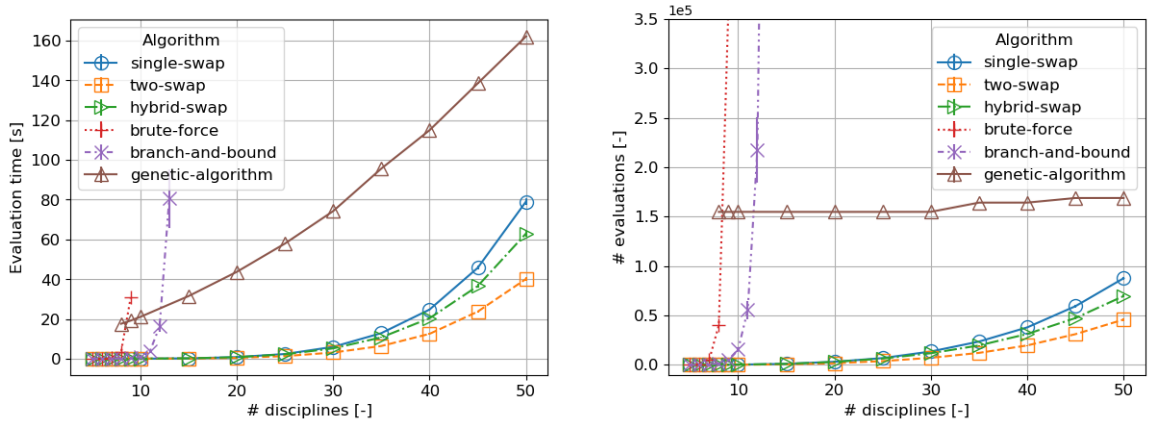
Figure 5.5: Solution accuracy of the sequencing algorithms for a large amount of disciplines when comparing the execution time of the obtained sequences. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters

5.1.2. Performance

Besides the solution accuracy, also the performance of the different algorithms was measured, this is visualized in Figure 5.6. Figure 5.6a shows the evaluation time of the algorithms versus the number of disciplines. However, the evaluation time depends heavily on the implementation details of the algorithm and the computational resource that is used. Therefore, Figure 5.6b shows the number of function evaluations for each algorithm.

These two figures clearly show the bad performance scalability of the exact algorithms. The number of function evaluations and evaluation time increase fast when the number of disciplines is increased. Furthermore, the figures show that the branch-and-bound algorithm is significantly faster than the brute-force algorithm. The brute-force algorithm can only solve test cases up to nine disciplines in a reasonable time, while the branch-and-bound algorithm can solve test cases up to thirteen disciplines.

The iterative improvement methods are significantly faster than the exact methods. As was explained in the previous section, the two-swap algorithm had the worst accuracy. However, Figure 5.6 shows that this algorithm is the fastest algorithm. Furthermore, the single-swap algorithm is the slowest algorithm from the swap algorithms. The two-swap algorithm is faster, because more nodes are swapped per accepted permutation. Therefore, the sequence will improve faster per accepted permutation than for the single-swap algorithm. The hybrid-swap algorithm is faster than the single-swap algorithm. The hybrid-swap algorithm



(a) Evaluation time

(b) Number of function evaluations

Figure 5.6: Performance of the sequencing algorithms for a large amount of disciplines. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters

benefits from the speed of the two-swap algorithm to quickly find a local optimum. This solution is then refined using the slower single-swap algorithm.

From the performance plots, it can be concluded that the branch-and-bound algorithm and swap algorithms differ in accuracy and speed. Therefore, a default algorithm can be used for different sizes of MDAO systems in KADMOS. The branch-and-bound algorithm is most suitable for the small MDAO systems due to its high accuracy, while the swap-algorithms are more suitable for the bigger MDAO systems due to their performance. More specifically, the following rules have been implemented which KADMOS uses to determine the most suitable algorithm for the given MDAO system:

- Branch-and-bound: <12 disciplines
- Single-swap: 12-35 disciplines
- Hybrid-swap: 35-45 disciplines
- Two-swap: >45 disciplines

Of course, the user can always choose a different algorithm if that is preferred.

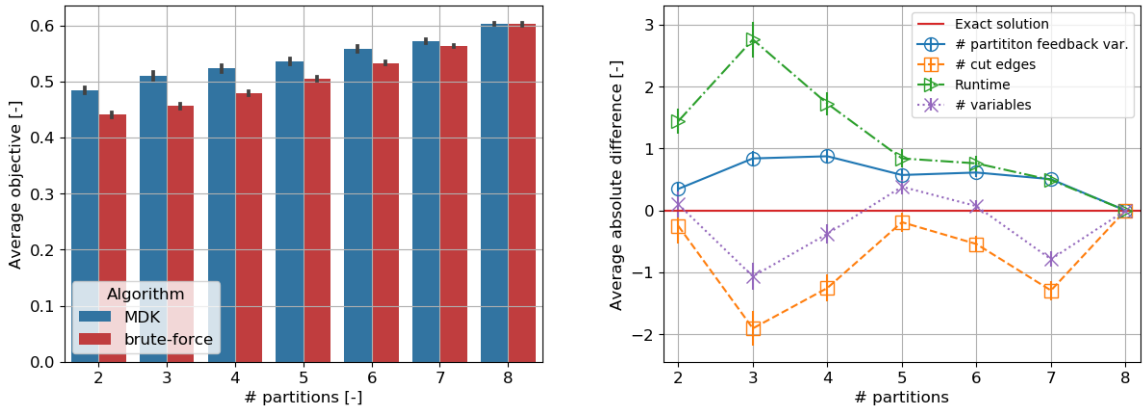
5.2. Decomposition

Just as for the sequencing algorithms, the MDK algorithm is verified, validated and tested by applying it to different types of MDAO systems. As defined in equation 4.1 in Section 4.2.1, the objective of the MDK algorithm is a weighted average between the coordination complexity and the execution time of one iteration. The relative importance between the coordination complexity and execution time is determined by the RCB. The results in this section were obtained using an RCB of 0.5. This means that the coordination complexity and execution time are equally important. The MDK algorithm is verified and validated in Section 5.2.1, while its performance is discussed in Section 5.2.2.

5.2.1. Verification & Validation

The MDK algorithm is verified and validated using a brute-force algorithm. Figure 5.7 shows the influence of the number of partitions on the solution accuracy of the MDK algorithm for test cases with eight disciplines and Figure 5.8 shows the influence of the number of disciplines when the test case is decomposed in two partitions. Figure 5.7a and Figure 5.8a show the average objective value, while Figure 5.7b and Figure 5.8b show the average difference in execution time, the number of feedback couplings within the partitions, the number of cut edges between the partitions and the total number of couplings that need to be converged with respect to the exact solution. These values are again calculated using equations 5.2 and 5.3. Note that the total number of couplings that need to be converged is the sum of the cut edges and the feedback couplings. Besides the number of disciplines and the number of partitions, the coupling density and the number of coupling variables per disciplines were also varied. The results for these cases can be found in Appendix A.

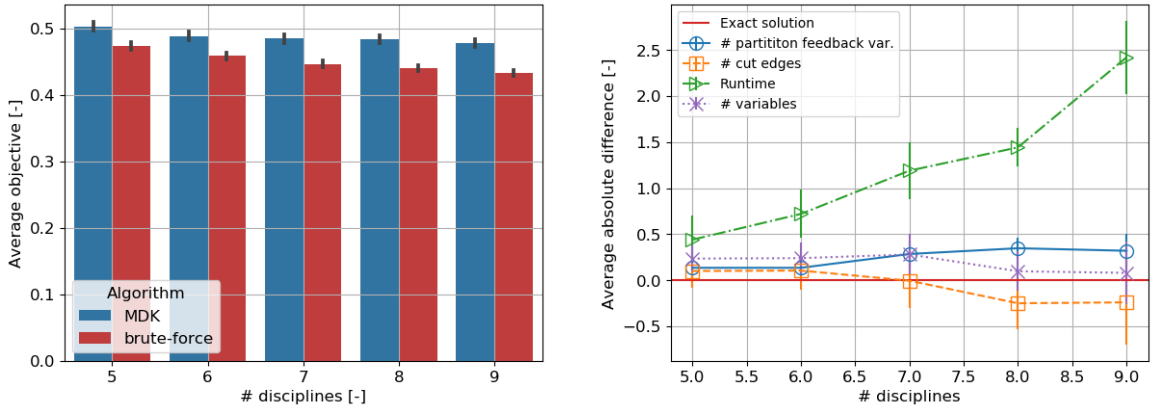
Figure 5.7 shows that the MDK algorithm obtains good partitions. The quality of the decomposition increases as more partitions are generated. The reason for this is that when the number of requested partitions



(a) Average objective value

(b) Absolute differences with respect to the exact solution

Figure 5.7: Solution accuracy of the MDK algorithm when varying the number of partitions. $n_d = 8$, $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters



(a) Average objective value

(b) Absolute differences with respect to the exact solution

Figure 5.8: Solution accuracy of the MDK algorithm when varying the number of disciplines. $n_p = 2$, $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters

is increased, the number of possible solutions decreases. For example, when eight partitions are requested for an MDAO problem with eight disciplines, only one partition is possible.

One exception is the decomposition in two partitions. The MDK algorithm obtains better solutions for two partitions instead of three partitions. The reason for this is that Metis uses a KL-algorithm to partition the graphs. The KL-algorithm was originally designed for the decomposition in two partitions [30]. When more partitions are requested, Metis uses a recursive bisection. The recursive bisection is known to give less optimal results than the simpler single bisection [28].

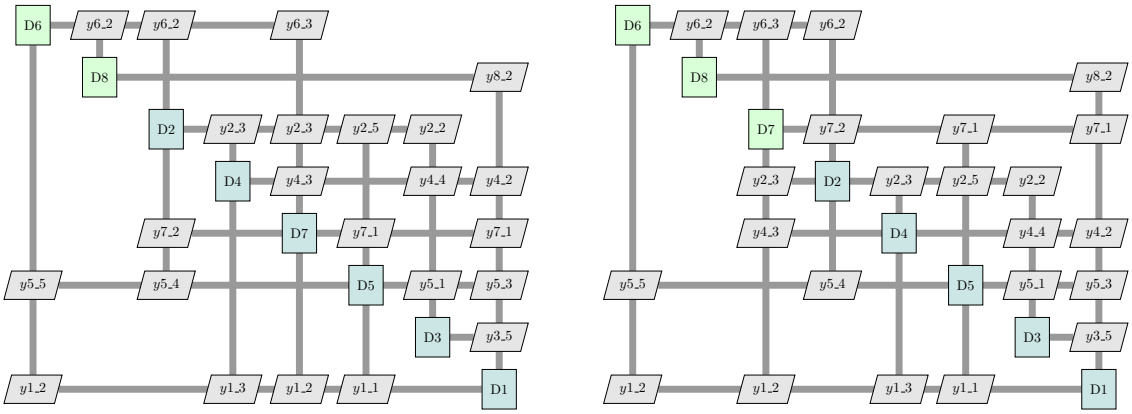
The same conclusions can be drawn from Figure 5.8. This figure shows that the solution accuracy decreases when the number of disciplines increases. This is again due to the increased number of possible solutions.

Both figures show that the solutions obtained by the MDK algorithm have a lower number of cut edges than the solutions obtained by the brute-force algorithm. This difference is due to the second limitation of Metis as explained in Section 4.2.4, which stated that Metis cannot take the execution order within a partition into account. Therefore, the number of feedback variables are not considered when Metis determines the quality of a partitioning. It is possible that the best solution has a higher number of cut edges but less feedback couplings and thus a lower number of total couplings that need to be converged. This solution will never be found by Metis as Metis minimizes the number of cut edges only.

Furthermore, the two figures also show that the solutions obtained by the MDK algorithm have a higher execution time than the solutions obtained by the brute-force algorithm. This difference can be linked to the

third limitation of Metis, which states that Metis cannot take the parallel execution of disciplines within a partition into account. This issue is partly resolved by merging parallel nodes in the MDK algorithm. The MDK algorithm merges parallel nodes after Metis has made its first decomposition. Therefore, it is still possible that a better decomposition can be obtained in which different nodes are executed in parallel.

An example explaining these two limitations of the MDK algorithm is shown in Figure 5.9. Figure 5.9a shows the decomposition obtained by the MDK algorithm, while Figure 5.9b shows the decomposition obtained by the brute-force algorithm. Disciplines D3 and D4 have an execution time of two seconds, D1, D2 and D5 three seconds, and D6, D7 and D8 five seconds. The solution obtained by the MDK algorithm has five cut edges, five feedback couplings and an execution time of eighteen seconds. Therefore, the objective value is $0.5 \cdot \frac{10}{23} + 0.5 \cdot \frac{18}{28} = 0.54$. The solution obtained by the brute-force method has ten cut edges, three feedback couplings and an execution time of eleven seconds. Therefore, the objective value is $0.5 \cdot \frac{13}{23} + 0.5 \cdot \frac{11}{28} = 0.48$. This example clearly shows that the best decomposition has a higher number of cut edges than the solution obtained by the MDK algorithm. Furthermore, the MDK algorithm will never merge nodes D7 and D8, because they are part of different partitions. Also, nodes D4 and D5 are not merged due to node D7, which is placed in between nodes D4 and D5. Therefore, the MDK algorithm cannot obtain the optimal solution from Figure 5.9b.



(a) MDK solution

(b) Brute-force solution

Figure 5.9: Differences in decompositions obtained by the brute-force and MDK algorithm

5.2.2. Performance

Besides the solution accuracy, the evaluation time for both the MDK algorithm as well as the brute-force method was measured. Figure 5.10 shows the evaluation time for the two algorithms when the number of partitions ranges from 2 to 5 disciplines. This figure clearly shows the long evaluation time for the brute-force method. On average, the evaluation of the decomposition of nine disciplines in two or three partitions took more than four minutes to complete. On the other hand, the MDK algorithm finished in less than a second.

Furthermore, this figure clearly shows that the evaluation time decreases when the number of partitions decreases. This is due to decrease in the number of possible solutions. An exception to this is the decomposition in two partitions using the MDK algorithm. A possible explanation could be that the KL-algorithm used by Metis was originally designed for the decomposition into two parts.

In this chapter, the verification and validation of the different sequencing algorithms and the MDK algorithm have been discussed. The results showed that high-quality partitions and execution orders are obtained in a short time frame. The next chapter will discuss the influence of the obtained decompositions on the convergence time of different types of MDAO systems.

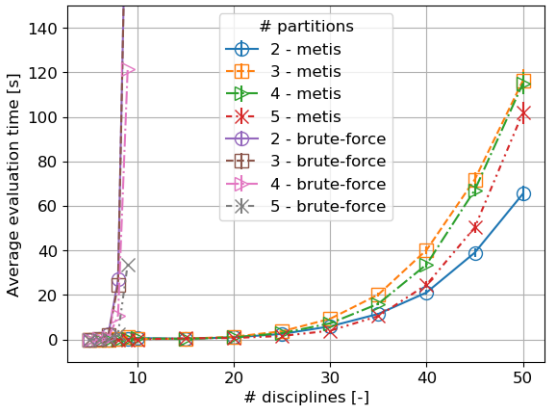


Figure 5.10: Performance plot of the decomposition algorithm. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters

III

Test Cases

6

Test Case 1: Variable Complexity Problem

Chapter 5 investigated how fast and accurate the sequencing and decomposition algorithms are. However, the question arises what the benefits of decomposing the MDAO systems are and whether the convergence time can actually be reduced. In the previous chapter, the Variable Complexity Problem was used to test the sequencing and decomposition algorithms on different types of MDAO systems. This corresponds to step 2C and 2D in Figure 2.6. In this chapter, hundreds of VCP problems are generated and solved to test the full process from Figure 2.6, including the new automated execution process formulation. Using a randomly generated scalable mathematical problem creates the opportunity to quickly generate a large amount of different MDAO systems. It allows to easily change the different properties of the MDAO system and thus creating specific types of problems. Furthermore, as the different modules contain only simple mathematical equations, the execution time to solve a complete optimization problem is relatively low.

The mathematical equations used in the VCP are explained in Section 6.1. The characteristics of the VCP, like convergence and convexity, are also explored in this Section. As explained in Chapter 4, several software packages are available to solve the final workflow. Therefore, a software comparison between OpenMDAO and RCE is performed in Section 6.2 to determine which software package is most suitable to solve the mathematical equations. During the execution of the MDAO systems, each of the modules of the VCP will be given a random execution time. It is not feasible, nor necessary, to run the optimization using this runtime. Therefore, an estimation of the total execution time will be made based on the number of iterations. This estimation is validated in Section 6.3. The architecture benchmarking is performed in Section 6.4. During the benchmarking, the partitions are executed simultaneously. However, the number of possible parallel executions depends on the number of available CPUs. Therefore, the influence of the available computational resources on the total execution time is discussed in Section 6.5.

6.1. Variable Complexity Problem

The optimization problem of the VCP [57] is defined as follows:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}_0, \mathbf{x}, \mathbf{y}) \\ \text{with respect to} \quad & \mathbf{x}_0, \mathbf{x} \\ \text{subject to} \quad & \mathbf{c}(\mathbf{x}_0, \mathbf{x}, \mathbf{y}) \leq 0 \end{aligned} \tag{6.1}$$

In which f is the objective value that is minimized and c the constraint values which must be less than or equal to zero. The global and local design variables are indicated with $\mathbf{x}_0$ and $\mathbf{x}$, respectively. Lastly, $\mathbf{y}$ denotes the coupling variables which are calculated using:

$$\mathbf{B}_{ii}\mathbf{y}_i = -\mathbf{C}_i\mathbf{x}_0 - \mathbf{D}_i\mathbf{x}_i - \sum_{j=1, j\neq i}^{N} \mathbf{B}_{ij}\mathbf{y}_j \tag{6.2}$$

In this equation, matrices **B**, **C**, and **D** represent the influence of the different types of variables on the value of the coupling variables. The **B** matrix indicates the influence each coupling variable has on the other coupling variables. The **C** and **D** matrices determine the influence of the global and local design variables on

the coupling variables, respectively. The three matrices are randomly generated using integer values between -5 and 5. The global constraint values are calculated using:

$$c_{0n_{c_0}} = \mathbf{x}_0^T \mathbf{x}_0 + \sum_{i=1}^{N} \mathbf{x}_i^T \mathbf{x}_i + \mathbf{H}_{n_{c_0}}^T \mathbf{x}_0 + \sum_{i=1}^{N} \mathbf{I}_{n_{c_0} i}^T \mathbf{x}_i + \sum_{i=1}^{N} \mathbf{J}_{n_{c_0} i}^T \mathbf{y}_i - s_{n_{c_0}} \leq 0 \tag{6.3}$$

The matrices **H**, **I** and **J** determine again the influence of the different variables on the constraint values. **H** indicates the influence of the global design variables, **I** the influence of the local design variables and **J** the influence of the coupling variables on the constraint values. These three matrices are also generated using integer values between -5 and 5. Lastly, s is a random positive scaler between 1 and 5, which is added to the constraint.

Finally, the objective value is calculated using:

$$f = \left(\sum_{i=1}^{n_{x_0}} \mathbf{x}_{0i} + \sum_{i=1}^{N} \sum_{j=1}^{n_{x_i}} \mathbf{x}_{ij} + \sum_{i=1}^{N} \sum_{j=1}^{n_{y_i}} \mathbf{y}_{ij} \right)^3 \tag{6.4}$$

Note that the objective function is a cubic function with a saddlepoint in the origin. Therefore, the objective function is not convex. However, as it is monotonically increasing/decreasing, there will be one minimum point bounded by either the constraints or the bounds on the design variables. Therefore, it is expected that the optimizations performed in the next sections will either return this minimum point or the saddlepoint. The lower bound of the design variables was set to -5, while the upper bound was set to 5.

The definition of the VCP as described above and in Chapter 5 is the definition of the VCP as described in Zhang et al. [57]. However, two adaptations were made to the mathematical problem to make it more suitable for the benchmarking performed in this research.

The first adaptation was made to the **B** matrix in equation 6.2 to meet the convergence criteria. In the paper of Zhang et al. [57], the MDA of the MDF architectures were solved using root finding algorithms, namely the Broyden, Excitingmixing and Krylov algorithms. However, during the benchmarking in this research only fixed-point iterators were used to solve the MDAs. Using fixed-point iterators puts an extra requirement on the mathematical problem to ensure convergence. As explained in Olver and Shakiban [39], convergence with fixed-point iterators is guaranteed when the **B** matrix is strictly diagonally dominant. Diagonally dominant means that each value on the diagonal is equal to or higher than the sum of its corresponding row. Strictly diagonally dominant means that at least on value on the diagonal is higher than the sum of its corresponding row. So, in order to meet this convergence requirement, the diagonal of the **B** matrix is calculated by summing the values of the rows. A value of one is added to each diagonal entry to make the matrix strictly diagonally dominant.

The second adaptation was the option to include clusters in the VCP. A cluster is a group of nodes which have very high coupling strengths with respect to each other and low or zero coupling strengths with respect to the nodes outside the cluster. An example of an MDAO system with two perfect clusters is shown in Figure 6.1.

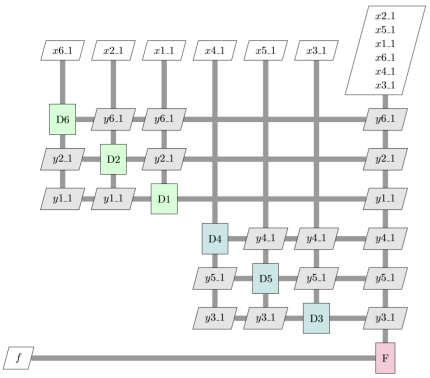


Figure 6.1: Example of an MDAO system with two perfect clusters

The total number of couplings in the VCP are determined by the coupling density. Whether these couplings are placed in the clusters or between the clusters is defined by the cluster strength. The cluster strength determines the bias towards adding couplings in the clusters. A value of one means that all the couplings are placed within the clusters. If the coupling density is high and the clusters are completely full, the remaining couplings are placed in between the clusters. A cluster strength of zero means that the couplings are uniformly distributed between the disciplines, with no bias at all towards the clusters. In other words, when the coupling strength is zero, no clusters are present in the problem. When the coupling strength is increased, the chances that a coupling is placed in the clusters is increased as well.

Now that it is possible to create clusters in the VCP, the influence of the decomposition on the total execution time can be investigated when clusters are present in the MDAO system.

6.2. Software Comparison

In order to investigate the influence of the decomposition on the total execution time of MDAO systems, it is necessary that the software package which solves the MDAO systems supports the execution of the different partitions as well (step 4 in Figure 2.6). The decomposition as defined in KADMOS is saved in a CMDOWS file. The support to interpret this CMDOWS file and translate it into an executable workflow including the different partitions has been implemented in both RCE as well as OpenMDAO.

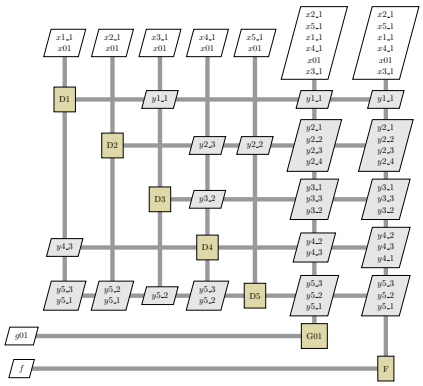


Figure 6.2: VCP problem that was used for the software comparison between RCE and OpenMDAO

To determine which software package is more suitable to solve the mathematical equations, a comparison has been made between the two. The most suitable software package is the package which has the lowest execution time to solve the MDAO systems. Furthermore, in order to create the opportunity to perform large benchmarking studies, it must be easy to create an automatic link between KADMOS and the software package. The comparison was made by solving a small VCP problem, which is shown in Figure 6.2. The problem has five disciplines, a coupling density of 0.23 and one global constraint. The disciplines do not get an execution time, such that only the overhead of the software is compared.

Table 6.1 shows the comparison for the convergence of the mathematical problem using three different architectures: a Gauss-Seidel convergence, a Jacobi convergence and a convergence with two partitions. The convergence tolerance is set to 10^{-10} to increase the number of iterations and to emphasize the time differences. The start point is -0.5 for both the local as well as the global design variables. The different architectures are all executed using one CPU.

Table 6.1: Performance comparison between RCE and OpenMDAO for three different MDA architectures

	RCE			**OpenMDAO**		
Architecture	Execution time [s]	# function evaluations	Objective value	Execution time [s]	# function evaluations	Objective value
MDA-GS	11	11	0.99878	0.39	11	0.99878
MDA-2P	11	12	0.99878	0.46	11	0.99878
MDA-J	21	22	0.99878	0.40	22	0.99878

The results show that RCE has clearly a bigger overhead than OpenMDAO. RCE performs roughly one iteration per second, while OpenMDAO finishes the entire convergence in less than half a second. Both software packages obtain the same design point in the same amount of iterations, except for the partitioned architecture for which RCE performs one iteration more than OpenMDAO.

Table 6.2 shows the comparison for the optimization of the VCP problem using four different architectures. Three architectures used the MDF architecture, while one used the IDF architecture. The converger loop in the MDF architectures is solved using a Gauss-Seidel, Jacobi or partitioned convergence with two partitions. The objective tolerance was set to 10^{-6} and the optimizations are again executed using one CPU. Note that the optimization algorithms are different, due to the differences in optimization packages that are available within the two software packages. RCE uses the DAKOTA package, while OpenMDAO uses the SciPy package. The optimization algorithm used by RCE is the COBYLA algorithm, while OpenMDAO used the SLSQP algorithm.

The results show that the execution time in RCE is significantly larger than the execution time in OpenMDAO. RCE needed 6.5 to 41 minutes to solve the different optimization problems, while OpenMDAO was always finished in less than three seconds. Furthermore, the results show that both software packages found the same objective function for all four architectures. The number of iterations is similar for the two software packages, except that the number of iterations performed by OpenMDAO using the IDF architecture is significantly lower than the number of iterations performed by RCE.

Overall, it can be concluded that OpenMDAO is much faster than RCE. Therefore, it is chosen to use OpenMDAO during the research. Another benefit of OpenMDAO is that KADMOS, OpenMDAO and OpenLEGO (used to translate the CMDOWS files into an OpenMDAO model) are all written in Python. This makes

Table 6.2: Performance comparison between RCE and OpenMDAO for four different MDO architectures

	RCE			OpenMDAO		
Architecture	Execution time [s]	# iterations	Objective value	Execution time [s]	# iterations	Objective value
MDF-GS	1379	176	0.50084	1.80	127	0.50084
MDF-2P	1051	162	0.50084	2.53	184	0.50084
MDF-J	2460	156	0.50084	2.60	117	0.50084
IDF	389	393	0.50084	2.62	125	0.50084

it easy to extend the process in Figure 2.6 with an automatic link between KADMOS and OpenMDAO.

6.3. Validation of the Runtime Estimation

During the benchmarking of the different architectures, each discipline was given a random execution time. These execution times range from 1 to 1000 seconds and are implemented using a sleep function in Python. However, running all the optimization problems in real time is both not feasible nor necessary. When the sleep function is used, the optimization would be sleeping most time, while it can be predicted beforehand how long the runtime of each module will be. Therefore, an estimation of the total execution time is made based on the number of iterations and function evaluations.

This estimation has as disadvantage that the overhead of OpenMDAO is not taken into account. The previous section already showed that the overhead of OpenMDAO is limited when one CPU is used. To validate whether this is also a reasonable assumption when multiple CPUs are used, a small test case with four disciplines was executed in real time using the sleep function. The results are shown in Table 6.3.

Table 6.3: Difference between the estimated and measured execution time for two different architectures

Architecture	**# CPUs [-]**	**Estimated time [s]**	**Measured time [s]**	**Difference [%]**
MDF-GS	1	832	844	1.4
IDF	4	115	121	5.0

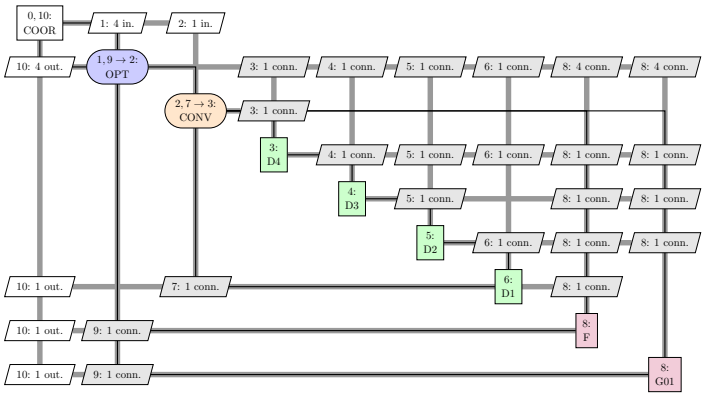
The results show that the overhead of OpenMDAO is limited. The difference is twelve seconds for the MDF with Gauss-Seidel convergence and eight seconds for the IDF optimization. Therefore, it can be concluded that the estimated time is close to the measured time and that the difference are small enough to use the estimated time for the benchmarking performed in the next section.

6.4. Architecture Benchmarking

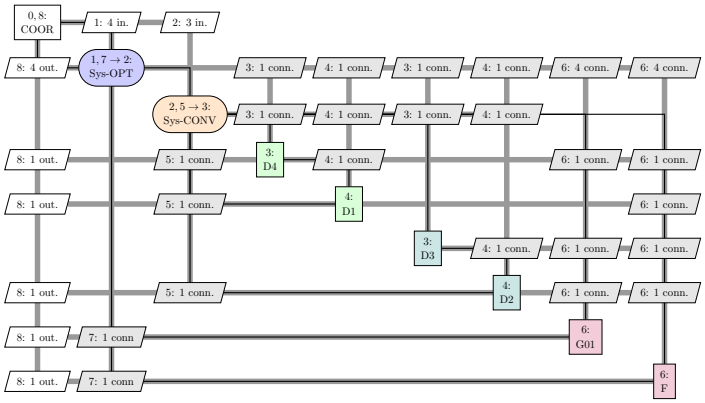
Before this thesis, the disciplines in the MDAO system could only be executed either in sequence or in parallel, as was explained in Section 2.2. Due to the sequencing and decomposition algorithms it is now possible to create more refined execution processes, as will be shown in this section. In order to examine the influence of the architecture and execution process of the MDAO systems on the total execution time, several MDAO architectures are imposed on the VCP to compare their performance. The architectures can be divided into the following five categories. An example of each category is shown in Figures 6.3 and 6.4.

1. *Multidisciplinary Feasible (MDF)*: The optimizer takes care of finding the optimal value of the design variables, while a system converger is responsible for converging the coupling variables and obtaining a consistent solution.
   (a) *Jacobi (J)*: All disciplines in the converger loop are executed in parallel.
   (b) *Gauss-Seidel (GS)*: All disciplines in the converger loop are executed in sequence.
   (c) *Partitioned (2P)*: The disciplines are divided into two partitions. Each partition is executed in parallel, while the disciplines within one partition are run in sequence.
2. *Individual Discipline Feasible (IDF)*: The optimizer is responsible for finding the optimal value of the design variables as well as converging the coupling variables.
   (a) *Classic IDF*: All disciplines in the optimizer loop are executed in parallel (except for the constraints and objective functions)

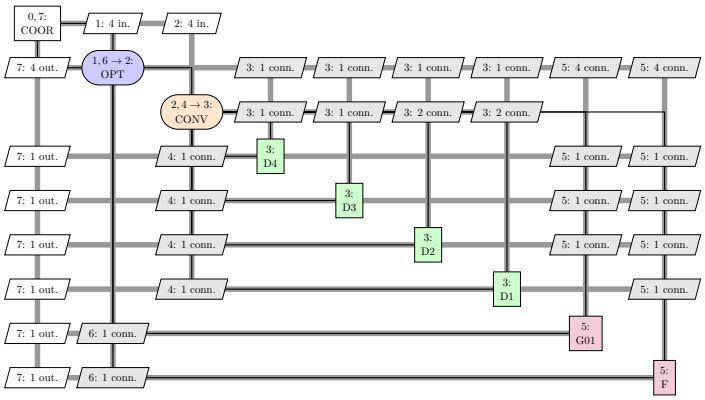
(b) *Partitioned (2P)*: The disciplines in the optimizer loop are divided into two partitions. Each partition is executed in parallel, while the disciplines within one partition are run in sequence.



(a) Gauss-Seidel



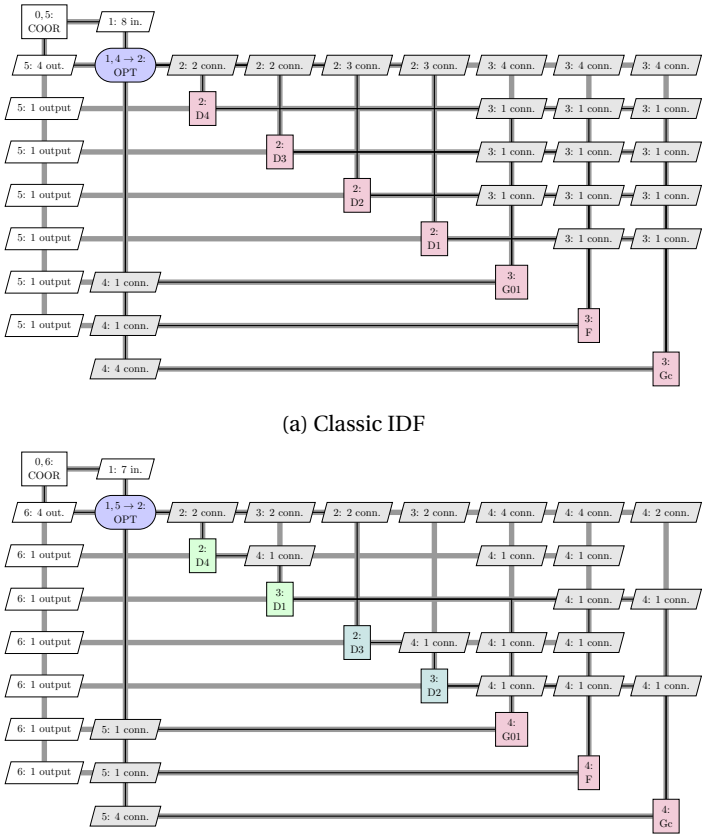
(b) 2 partitions



(c) Jacobi

Figure 6.3: Examples of the different MDF architectures used during the benchmarking

During the benchmarking, each mathematical problem is randomly generated. Therefore, 30 randomly generated problems were formulated and solved for each test case. The average execution time of these 30 problems is shown in the results. Furthermore, a baseline test case was formulated with which the other test cases are compared. The baseline test case consists of 10 disciplines, 5 coupling variables per discipline, a coupling density of 0.08 and no clusters. Lastly, as explained in Section 4.1, it is assumed that there is no limit on the number of available CPUs and thus no limit on the number of parallel executions.



(a) Classic IDF

(b) 2 partitions

Figure 6.4: Examples of the different IDF architectures used during the benchmarking

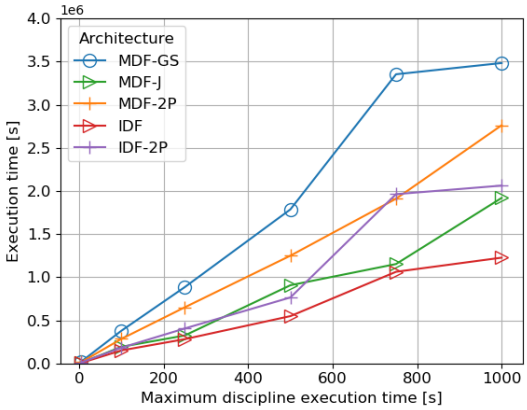
6.4.1. Influence of the Decomposition

The results for the baseline test case are shown in Figure 6.5a. The x-axis shows the maximum execution time that can be given to a discipline. For example, a value of 250 means that each discipline is given a random execution time between 1 an 250 seconds.

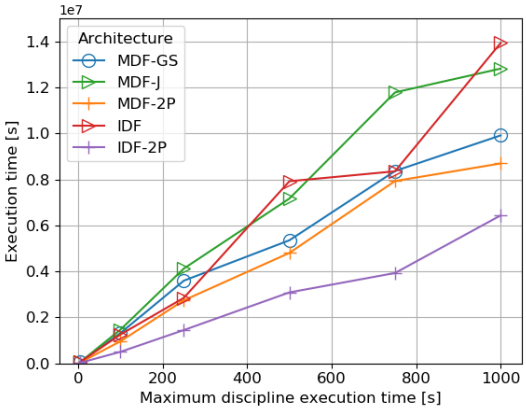
The results show that in this case the IDF is the fastest architecture, while the MDF-GS is the slowest architecture. When comparing the MDF architectures only, it can be seen that the decomposition into two partitions already performs better than the Gauss-Seidel convergence, while the Jacobi is the fastest of the three. This can be linked to the discussion from Section 2.4 on the trade-off between the execution time of one iteration and the coordination complexity. Increasing the number of parallel executions will decrease the execution time of one iteration, but will increase the number of iterations as the coordination complexity is increased. The results in this figure clearly shows that, in this case, the reduction in execution time of one iteration has a bigger impact on the total execution time than the increase in number of iterations. This can also be concluded from the comparison of the two IDF architectures, as executing all the disciplines in parallel has led to a lower total execution time than dividing the disciplines into two parallel partitions.

However, it is important to notice that this is not always the case. Figure 6.5b shows again the baseline case, but this time one discipline is given a significantly longer execution time than the other disciplines. For example, if the maximum discipline execution time (shown on the x-axis) is 250, each discipline is again given a random execution time between 1 and 250 seconds. Except for one discipline, which is given an execution time of $10 \cdot 250 = 2500$ seconds. This means that this discipline has a longer execution time than the sum of all other disciplines, because the baseline test case consists of ten disciplines in total. This case is relevant as this situation can for example happen when a CFD analysis is added in an aircraft design analysis.

In this case, the MDF-J and IDF architectures are not the fastest but the slowest architectures. Instead, both the partitioned MDF and IDF architectures are the fastest architectures. Due to the long execution time of one discipline, the execution time per iteration is the same for the IDF, MDF-J and partitioned architectures. However, as the coordination complexities of the IDF and MDF-J architectures are higher than for the



(a) Baseline case



(b) Baseline case while one discipline gets a significantly longer execution time than the other disciplines

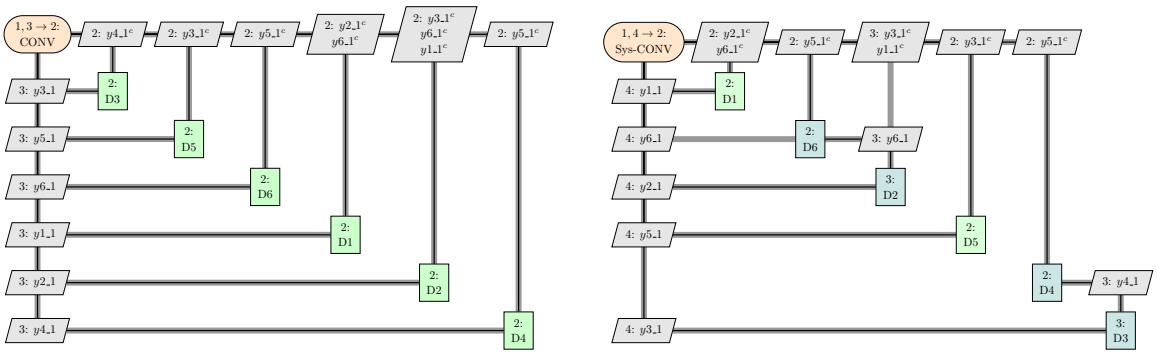
Figure 6.5: Architecture benchmarking using the baseline test case while varying the execution time of the disciplines

partitioned architectures, the number of iterations is higher and thus the total execution time is also higher.

The two cases shown in Figure 6.5 clearly show that there is no 'one-size-fits-all' solution exists when solving MDAO problems. It also shows the importance of a good execution process formulation. A proper decomposition and parallel execution can significantly reduce the total execution time.

6.4.2. Improving the Execution Process

The previous case showed that the shortest execution time of one iteration is equal to the longest execution time of one discipline. So, in order to minimize the coordination complexity and thus the total execution time, it can be beneficial to run other disciplines in sequence as long as it does not increase the execution time of one iteration. An example is shown in Figure 6.6. Figure 6.6a shows the problem formulation when all disciplines are executed in parallel. Disciplines 1 and 5 have an execution time of five seconds, while the other disciplines have an execution time of two seconds. In this case, disciplines 6 and 2, and 4 and 3 can be executed in sequence, as shown in Figure 6.6b. The execution time of one iteration is still five seconds. However, the coordination complexity has reduced with two couplings.



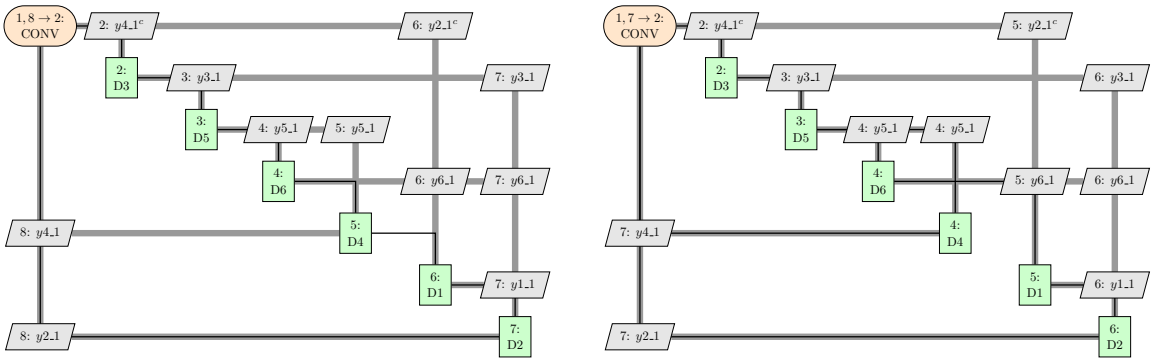
(a) Original execution process

(b) Improved execution process

Figure 6.6: Improving the execution process by running some disciplines in sequence. Disciplines 1 and 5 have an execution time of five seconds. Disciplines 2, 3, 4 and 6 have an execution time of two seconds.

Another possibility to improve the execution process is to improve of the execution time of one iteration while not increasing the coordination complexity by running some disciplines in parallel. An example is shown in Figure 6.7. Figure 6.7a shows an example in which all the disciplines are executed in sequence. In this case, disciplines 6 and 4 have no data dependencies. Therefore they can be executed in parallel. Furthermore, there is no need for disciplines 1 and 2 to wait until discipline 4 is finished as they do not receive input from discipline 4. So the process can be optimized as shown in Figure 6.7b. In this case, the execution time

of one iteration has been reduced with two seconds, while the coordination complexity has not increased.

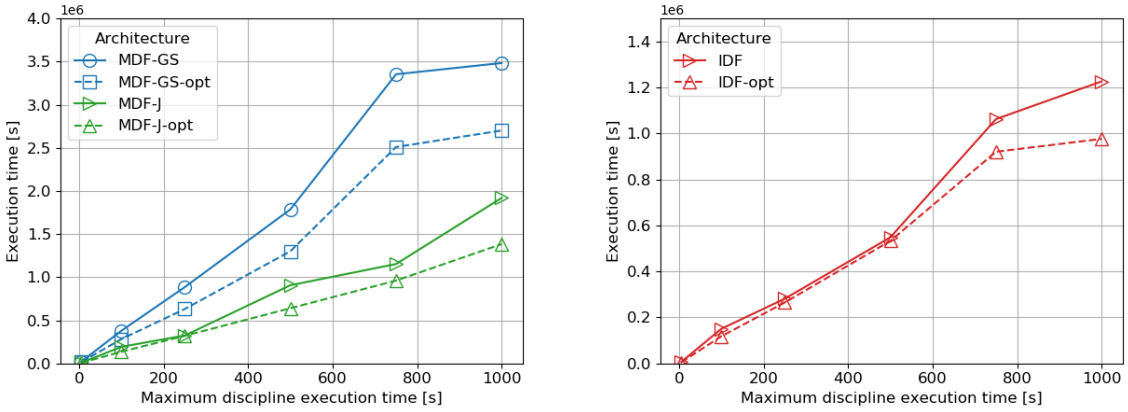


(a) Original execution process

(b) Improved execution process

Figure 6.7: Improving the execution process by running some disciplines in parallel. Disciplines 1 and 5 have an execution time of five seconds. Disciplines 2, 3, 4 and 6 have an execution time of two seconds.

Both process optimizations have been applied to the MDF-GS, MDF-J and IDF architectures for the baseline test case. The results are shown in Figure 6.8. Figure 6.8a shows the influence of the process optimization for the MDF architectures, while Figure 6.8b shows the influence of the process optimization for the IDF architectures. These two figures clearly show that the improvement process reduces the total execution time needed to obtain the solution.



(a) MDF architectures

(b) IDF architectures

Figure 6.8: Comparison of the total execution time between the original and improved execution process using the baseline test case

Due to the sequencing and decomposition algorithms in KADMOS, these improved processes are now easily applied to for any MDAO system.

6.4.3. Addition of Subconvergers

Using the decomposition algorithm, partitions are easily made. This also creates the opportunity to add subconvergers to the different partitions. An example is shown in Figure 6.9. A subconverger converges the feedback couplings within the partition only. If couplings exists between the different partitions, these couplings are converged by either a system-level optimizer (IDF) or a system-level converger (MDF).

Figure 6.10 shows the results when subconvergers are added to the partitioned architectures. The test case is again the baseline test case. The results clearly show that adding subconvergers is not beneficial in this case as the total execution time increases significantly. The reason for the increased execution time is the extra iterations that need to be performed by the subconvergers. This increases the execution time of one iteration performed by the system converger or optimizer. However, as the partitions are already converged, less iterations need to be performed. The results show that, in this case, the decrease in number of itera-

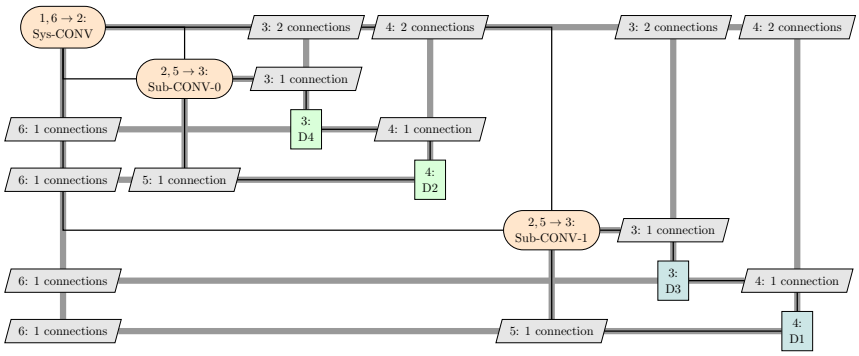


Figure 6.9: Example of a convergence with both a system convergers as well as two subconvergers

tions does not outweigh the increase in execution time of one iteration. Therefore, it is not beneficial to add subconvergers to this test case.

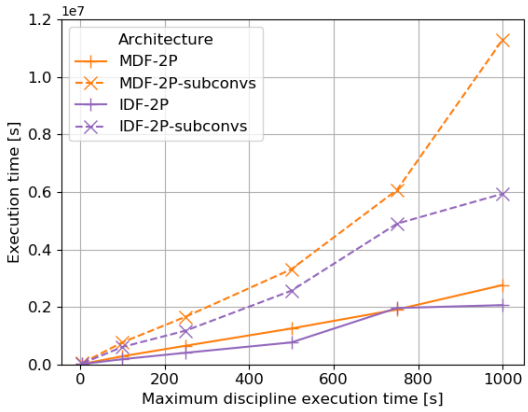


Figure 6.10: Comparison of the total execution time for decomposed MDAO systems with and without subconvergers using the baseline test case

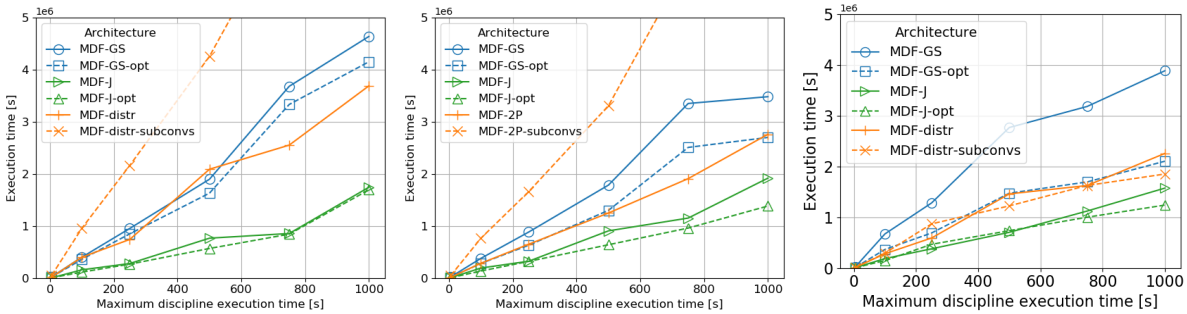
6.4.4. Influence of the Coupling Density and the Presence of Clusters

The previous sections showed the influence of the execution process on the performance of the different architectures. However, as mentioned in Section 6.4.1, the best architecture and execution process also depends on the MDAO problem that is solved. Therefore, the influence of changing the MDAO problem on the performance of the architectures is shown in this Section.

The results are shown in Figure 6.11 and 6.12 for the MDF and IDF architectures, respectively. Figures 6.11b and 6.12b show again the baseline test case. The coupling density is increased from 0.08 to 0.16 in Figure 6.11a and 6.12a, and two perfect clusters are added in Figures 6.11c and 6.12c.

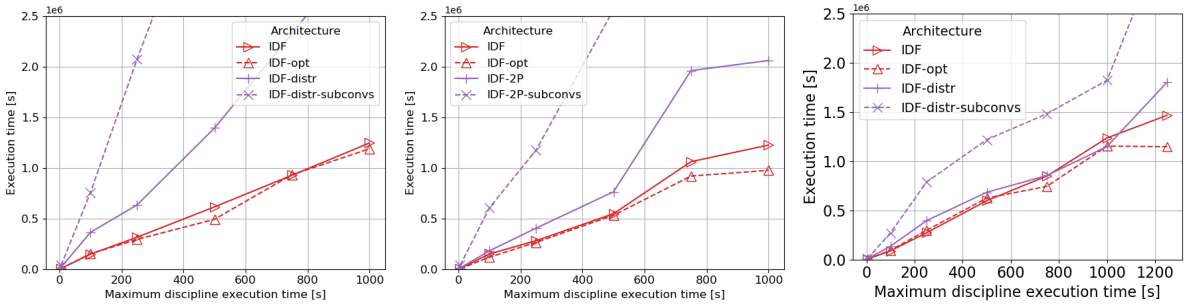
The results show that, in these cases, the MDF-J convergence is always the fastest and the MDF-GS is always the slowest within the MDF architectures. When the coupling density is increased, the execution time for the Gauss-Seidel and partitioned architectures increase as well, while the execution time for the MDF-J and IDF remain more or less the same. Furthermore, the benefits of the improved execution process, as described in Section 6.4.2, reduces for the MDF-GS, because less disciplines can be executed in parallel.

When perfect clusters are added to the problem, the benefits of the improved execution process increases for the MDF-GS architecture and the execution time becomes similar to the partitioned architecture. The reason for this is the fact that two perfect clusters have no data dependencies, therefore, they will automatically be executed in parallel when the improved process is applied to the MDF-GS architecture. Furthermore, it can be seen that this is the only case in which the execution time does not significantly increase when subconvergers are added to the partitioned MDF-architecture. The reason for this is that the different partitions have no data dependencies and therefore, the system converger can be removed.



(a) No clusters, high coupling density (0.16) (b) No clusters, low coupling density (0.08) (c) Two clusters, low coupling density (0.08)

Figure 6.11: Architecture benchmarking using MDF.



(a) No clusters, high coupling density (0.16) (b) No clusters, low coupling density (0.08) (c) Two clusters, low coupling density (0.08)

Figure 6.12: Architecture benchmarking using IDF.

6.5. Influence of the Available Resources

In the previous benchmarking results, it was assumed that enough processors were available to run the disciplines in parallel when necessary. However, this is not always the case. For example, when an MDO problem is run on a single PC instead of a cluster, the amount of available CPUs is limited.

Figure 6.13 shows the influence of the number of available CPUs on the evaluation time for a small test case of four disciplines. This figure clearly shows that the best architecture depends on the number of available CPUs. For example, the Gauss-Seidel architecture is the best MDF-architecture when only one CPU is available. However, as the number of available CPUs is increased, the Gauss-Seidel architecture cannot benefit from this. The Jacobi and distributed convergence can benefit and therefore, their evaluation time is reduced. Eventually, they perform better than the Gauss-Seidel architecture.

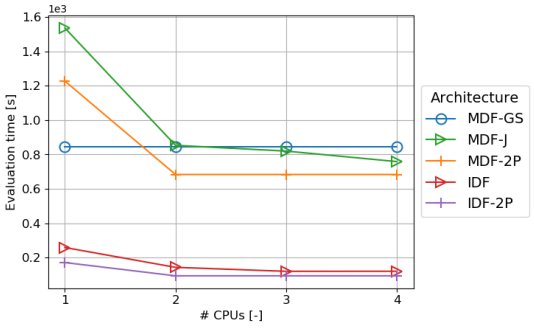


Figure 6.13: Influence of the number of available CPUs on the execution time using different architectures

This figure clearly shows that the best solution strategy depends not only on the MDAO system, but also on the number of available CPUs. Therefore, it is recommended to include the maximum number of available

CPUs as an option in the sequencing and decomposition algorithms. This will create the opportunity to adapt the solution strategy to the specific computational environment.

6.6. Limitations of the Variable Complexity Problem

During the benchmarking studies, the VCP problem has proven to be very useful in solving large amounts of MDAO systems in a short time frame. In Section 6.1, it was explained how the **B** matrix was adapted to meet the convergence criteria of the fixed-point iterators. However, this adaption to the VCP has also created some limitations. For example, when the number of coupling variables per disciplines or number of disciplines were increased significantly it was expected that the MDAO system would be harder to solve and thus the number of iterations would increase. However, this effect was not clearly visible in the results. The reason for this is that when the coupling variables per discipline or the number of disciplines was increased, the values in the diagonal of the **B** matrix had to be increased as well to satisfy the convergence criteria. However, when these values are increased, the influence of each variable on the outcome of the discipline is reduced, in other words the sensitivity of each coupling decreases. Therefore, the number of iterations did not increase significantly even though the complexity of the MDAO system was increased.

Overall, this chapter has shown the successful implementation of the decomposition in the full AGILE process as defined in Figure 2.6. The results show that the decomposition can reduce the convergence time of MDAO systems a lot. Especially the more refined process formulation for the Gauss-Seidel, Jacobi and IDF implementation proved to be very beneficial. Furthermore, this chapter has shown how the opportunity has been created to easily perform large benchmarking studies using a large variety of MDAO systems and MDAO architectures. The benchmarking has shown that there is no 'one-size-fits-all' solution towards solving MDAO systems. The best MDAO architecture and execution process depends heavily on the MDAO system and the number of available resources.

7

Test Case 2: Initiator

The previous chapter showed that a good decomposition and sequencing is important to reduce the convergence time of MDAO systems. This was shown on a scalable mathematical test case. However, a mathematical test case differs from a more realistic design case. In order to show the benefits of the MDAO development process from Figure 2.6 including the automated execution process, a novel implementation of the Initiator toolbox was developed. The Initiator [15] is a software tool for the conceptual design and analysis of aircraft. In this chapter, the full MDAO development process will be applied to the analysis and optimization of an Airbus A320-200.

7.1. Implementation into the AGILE MDAO development process

As explained in Chapter 2, the MDAO development process consists of five steps. The first step in the development process is the creation of the repository. Several developments and changes had to be made to the Initiator in order to implement the software tool in the development process:

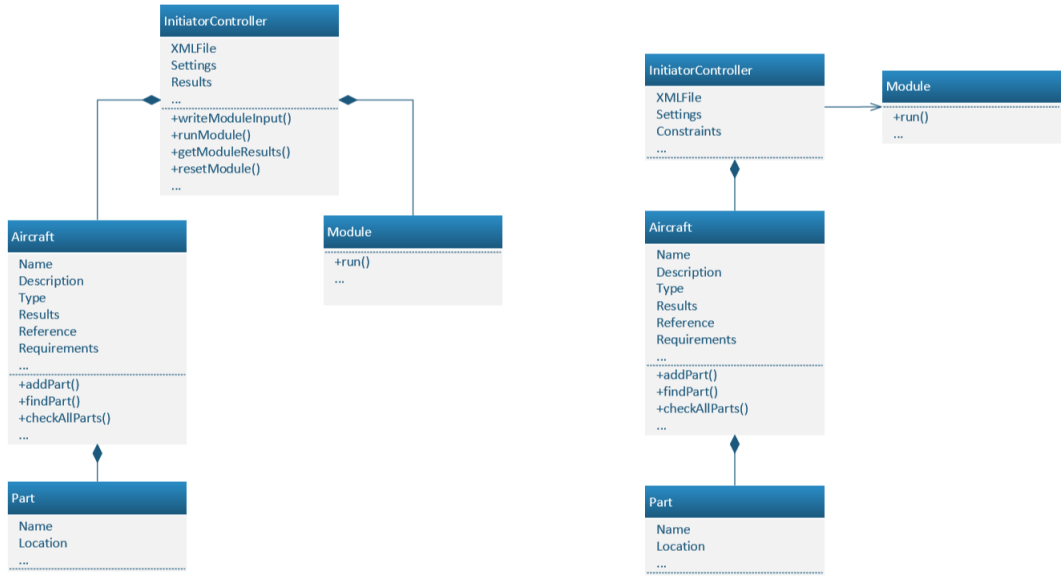
Creation of stand-alone modules

The first step was the creation of stand-alone modules. Figure 7.1 shows the UML-diagrams of the original and new implementation of the Initiator. In the original implementation (Figure 7.1a), the execution process of the different modules is controlled by the InitiatorController class. Each module in the Initiator is a subclass of the InitiatorController class. The Controller writes the module input, runs the modules, and collects their output. It determines when a certain module has to be executed and checks whether the solution has been converged. The resulting aircraft is stored in the Aircraft class, which is also a subclass of the InitiatorController class. The aircraft object contains all properties and requirements for the aircraft. The aircraft consists of several parts. Each of them are stored in a separate instance of the Part class, which is a subclass of the Aircraft class.

Figure 7.1b shows the new implementation of the Initiator in which stand-alone modules have been created. The main difference is that the modules are no longer a subclass of the InitiatorController class. The Module class is now an independent class, which takes a Controller object as input. It uses the data in the Controller object for the calculations and writes the results back to the object. Hence, the creation of the stand-alone modules has changed the function of the InitiatorController class. It is now only used to store data. As the execution process is formulatd by KADMOS and the different modules are executed by OpenMDAO, it has completely lost its coordination function. The Aircraft class and Part class did not change.

Creation of the general data schema

The second step involved the creation of a central data schema. As explained in Section 2.1.2, the MDAO development process is based on a service oriented architecture for efficient collaboration. Therefore, each module must use the same data schema such that the different modules can be easily combined into an executable workflow. Within the AGILE Paradigm, CPACS is used as the standard data schema for the different disciplinary analyses [8]. However, the Initiator has not been developed using CPACS. Hence, it would require a lot of changes to the make the Initiator modules CPACS compatible. Therefore, it was chosen to develop a new central data schema such that the changes to the modules are minimized.



(a) Original implementation (b) New implementation

Figure 7.1: Difference between the original and new implementation of the Initiator toolbox

The central data schema must be interpretable by both KADMOS, OpenLEGO as well as the Initiator modules. The data schema has a similar setup as the Controller object. This makes it easy to translate the data schema into a Controller object and back. The layout of the data schema is shown in Figure 7.2. The dataschema starts with an initiator element. This element is comparable to the InitiatorController class. They both contain the different settings for the aircraft and Initiator modules. Furthermore, when an optimization is performed, the values of the constraints are also stored in the InitiatorController class.

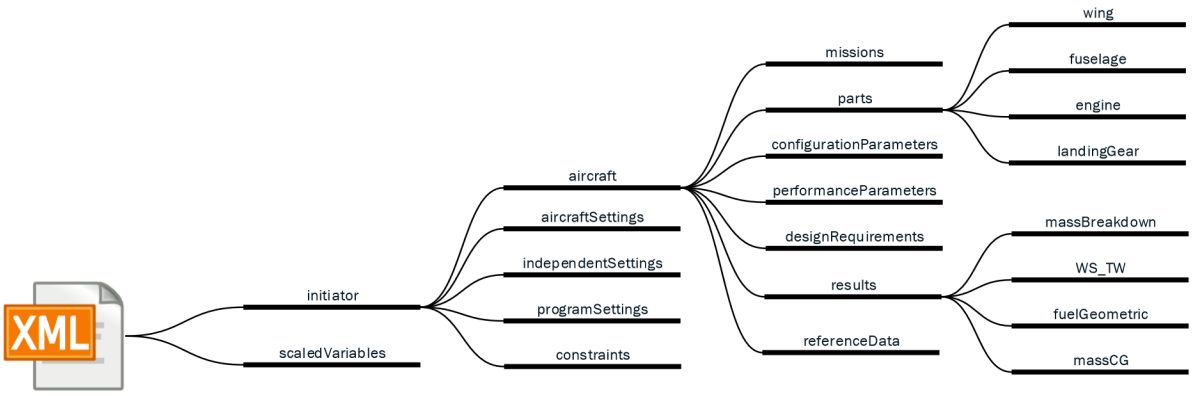


Figure 7.2: Data schema Initiator

As can be seen in the data schema, the initiator element has an aircraft element. This element corresponds to the Aircraft class. It contains the missions, design requirements, performance and configuration parameters, reference data, results. The results correspond to the output by the different modules. For example, the mass breakdown gives an overview of the different masses for the different missions, while the fuelGeometric contains all information on the fuel tank, fuel mass and fuel volume. Also the masses and CG values for the different components are stored in the results element, as well as the wing loading and thrust loading values. The aircraft element also contains a parts element, which correspond to the Part class of the Initiator. The parts elements contain the geometry and properties of the different wings, fuselages, engines and landing gear. Finally, the element scaledVariables contains the values that are used to scale the objective function during the optimization of the aircraft. An example of an Initiator data file using the new schema can be found in Appendix B.

Creation of the input and output files

The next step concerns the creation of the input and output files. For each module, a list is made with the variables that are needed to execute the module. The original XML reader present in the Initiator is slightly adapted such that it can read the new data schema and can translate it into a InitiatorController object. The data dependencies between the different modules are removed, such that the modules only use the input from the InitiatorController object. Furthermore, the XML writer from the Initiator is also slightly adapted such that it can translate the object back into the new data schema. Finally, for each module a list is created which states which output variables are given by each module.

Creation of a Python wrapper

The last step is the creation of a Python wrapper. The Initiator modules are all written in Matlab, but OpenLEGO and OpenMDAO are written in Python. Therefore, a Python wrapper is needed, such that the Initiator modules can directly be executed by OpenMDAO. This was achieved by formulating an execute function, which creates a Matlab engine in which the modules can be run. By providing each module with its own Matlab engine, the different modules can be executed in parallel.

Module Selection

Several modules from the Initiator were included in the repository as listed below. The modules were chosen such that an initial design of the A320-200 aircraft could be made.

- *Database*: collects reference aircraft and engines based on the given input. It stores the reference data in a separate file, which will be used by the other disciplines to make a first estimation of certain parameters.
- *Empirical OEM*: makes a first estimation of the Operational Empty Mass (OEM), based on the reference aircraft from the Database.
- *Class 1 Weight Estimation*: provides an initial estimation of the main masses, based on the OEM.
- *Fuselage Configurator*: configures the entire fuselage layout. It calculates both the outer geometry as well as the cabin layout.
- *Wing Thrust Loading*: determines an initial design point, based on several constraint on the wingloading and thrustloading.
- *Geometry Estimation*: calculates the geometry of the wing, engine and empennage.
- *Class 2 Weight Estimation*: calculates the masses and CG values of the different components.

These modules are loaded into KADMOS using a CMDOWS file. This corresponds to step 1 in Figure 2.6. KADMOS establishes the connections between the different modules and creates the RCG as shown in Figure 7.3. The order of the modules in the RCG was determined using the branch-and-bound algorithm from Chapter 4.

This figure shows how the transparency of the Initiator has already increased by only formulating the repository and RCG. In the original implementation of the Initiator, one could only find the relations between the different modules by digging through the code. Due to the creation of the stand-alone modules and the usage of a central data schema, the input and output connections are now immediately clear.

7.2. Validation

Once the modules are collected and loaded into KADMOS, several problem formulations were made and compared to the original implementation of the Initiator to validate the results (step 2, 3 and 4 in Figure 2.6). The test case that was used, was the conceptual design of an Airbus A320-200. Three different architectures were used: the Gauss-Seidel convergence, the Jacobi convergence, and the partitioned convergence with two partitions. The resulting architectures are shown in Figure 7.4.

The partitioning was performed using an RCB value of 0.5. This means that the execution time and coordination complexity are equally important. Each of the modules was given an execution time. The Database and Empricial OEM got an execution time of four seconds, the Class 1 Weight Estimation three seconds, the Wing Thrust Loading five seconds, both the Fuselage Configurator as well as the Geometry Estimation seven seconds, and finally, the Class 2 Weight Estimation 12 seconds. Note that these values are an estimation. The execution time of the disciplines fluctuated a lot. For example, at the beginning of the convergence, the Class 2 Weight Estimation started with execution times of around 20 seconds. However, when the solution was almost converged, the execution time reduced to around seven seconds or even less.

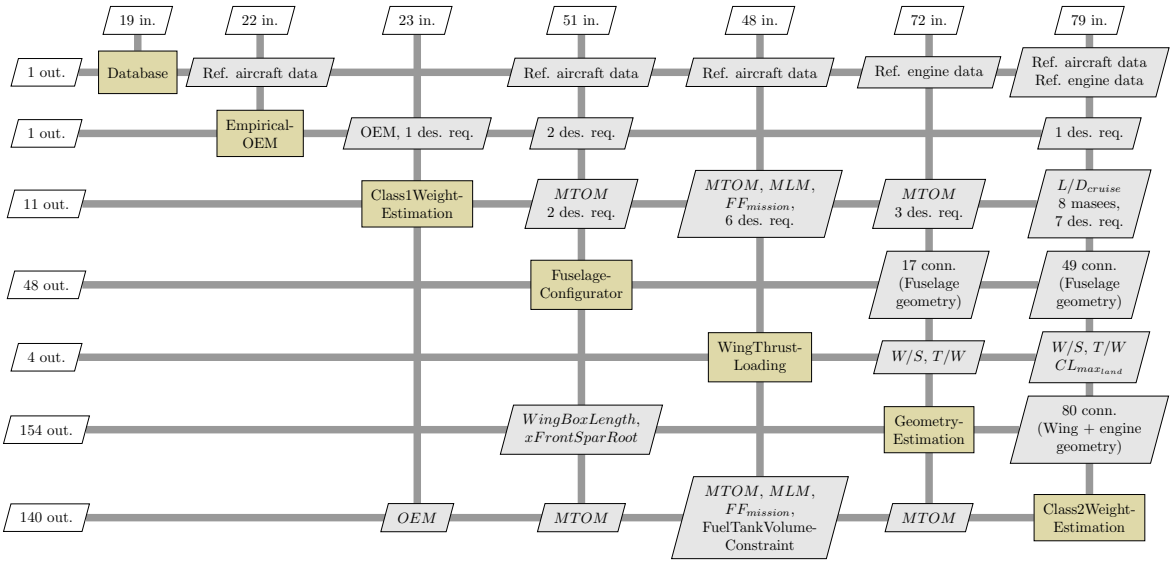


Figure 7.3: RCG of the KADMOSized Initiator

The final partitioning is shown in Figure 7.4b. The first partition consists of the Geometry Estimation and the Class 2 Weight Estimation, with a total execution time of 19 seconds. The second partition consists of the Class 1 Weight Estimation, Fuselage Configurator and Wing Thrust Loading with an execution time of 10 seconds. Note that the Wing Thrust Loading and Fuselage Configurator can be executed in parallel as well, therefore the execution time of this partition is 10 instead of 15 seconds. The first partition has a longer execution time than the second partition. They would have been better balanced if the Geometry Estimation was placed in the second partition. However, as the Geometry Estimation and Class 2 Weight Estimation have a high coupling strength, the number of cut edges would increase too much. Therefore, the unbalanced partitioning is preferred.

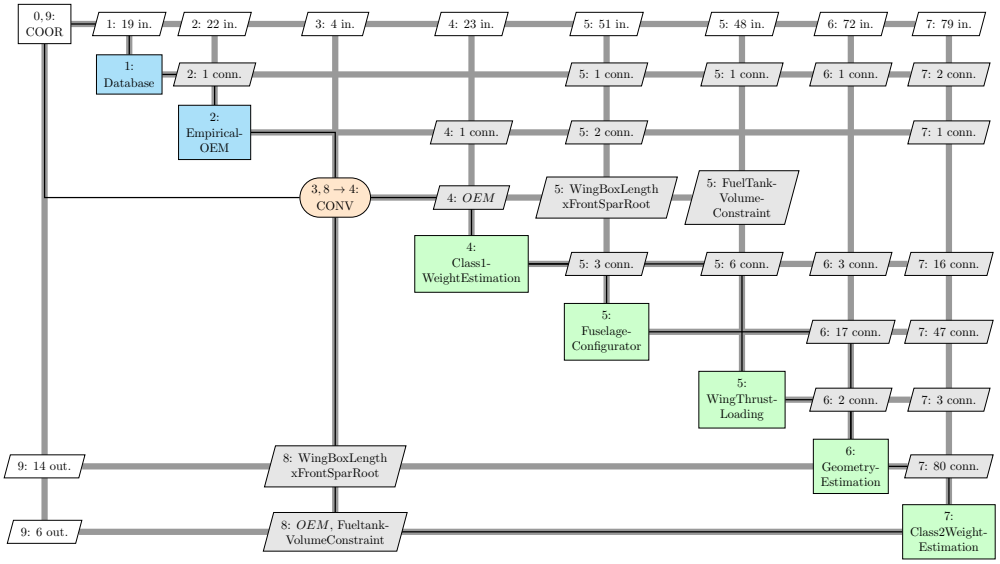
The results are shown in Table 7.1 and the convergence history of both the MTOM and OEM are shown in Figure 7.5. The convergence tolerance was set to 10^{-4}. Note that the original implementation of the Initiator only converges the MTOM, while KADMOS/OpenMDAO converges all variables that are connected to the converger.

Table 7.1: Convergence results of the conceptual design of the A320-200 using four different methods

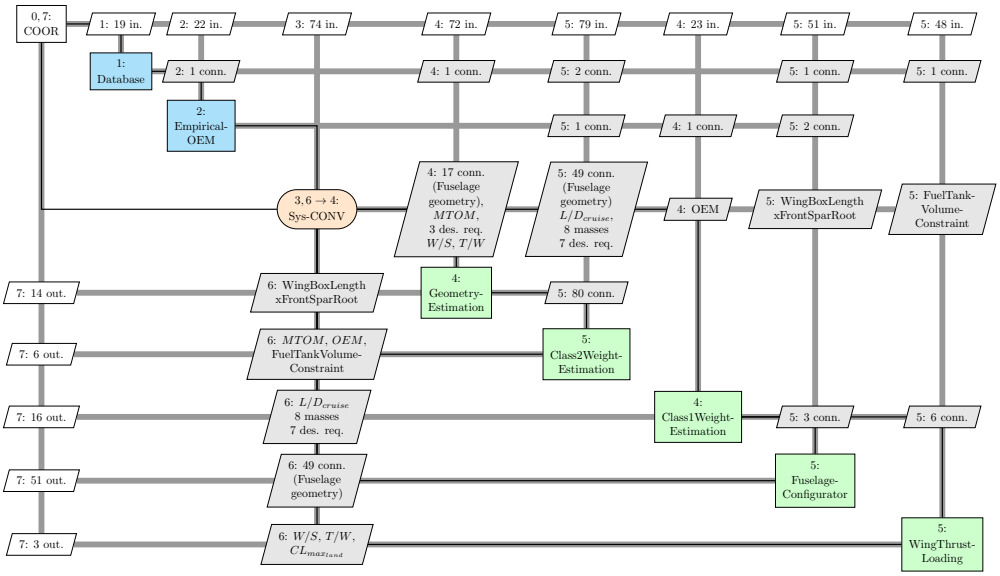
Range: 4000km	**Original Matlab Implementation**	**Gauss-Seidel**	**Two Partitions**	**Jacobi**
# iterations [-]	6	7	9	12
MTOM [t]	81.791	81.789 (-0.0024%)	81.793 (+0.0024%)	81.795 (+0.0049%)
OEM [t]	43.897	43.895 (-0.0046%)	43.898 (+0.0023%)	43.900 (+0.0068%)
Actual execution time [s]		313	620	735
Estimated execution time [s]	105	298 (1 CPU) / 258 (2 CPUs)	214 (2 CPUs)	144 (5 CPUs)

It can be concluded that the different architectures are all converging to the same point and that the differences in MTOM and OEM are minimal. The number of iterations increases when more disciplines are executed in parallel. However, due to the parallel execution of the disciplines, the execution time decreases. This supports the earlier discussion in Section 2.4 on the interaction between the decomposition and coordination process. Increasing the number of partitions will decrease the execution time of one iteration, but increases the number of iterations as the coordination complexity is increased.

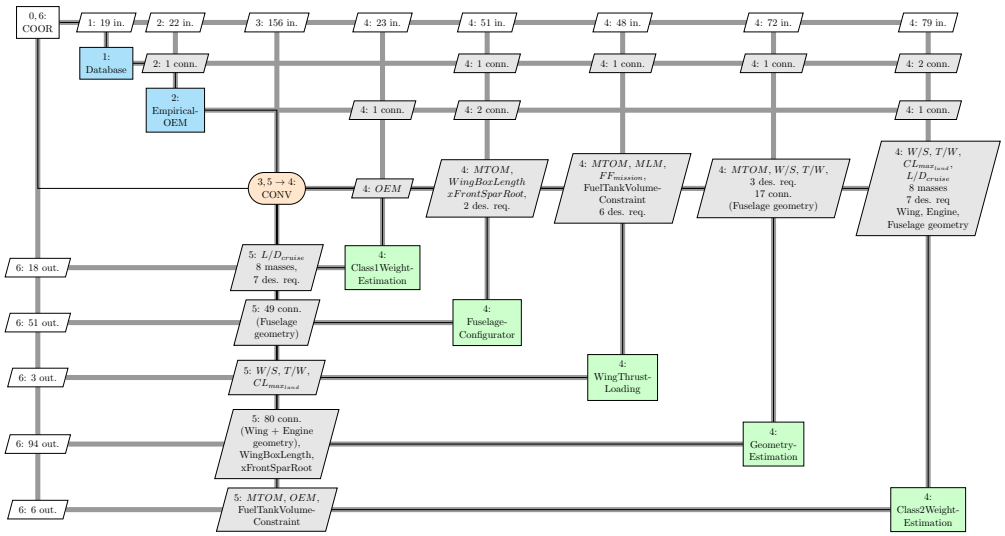
Note that the execution time in these results are an estimation. The different architectures were all executed using 1 CPU. Based on the measured runtime, the total execution time was estimated when more CPUs would have been used. The reason for this is that due to time limitations, there was no time to install the software needed to combine the Initiator modules with the parallel execution in OpenMDAO.



(a) Gauss-Seidel convergence

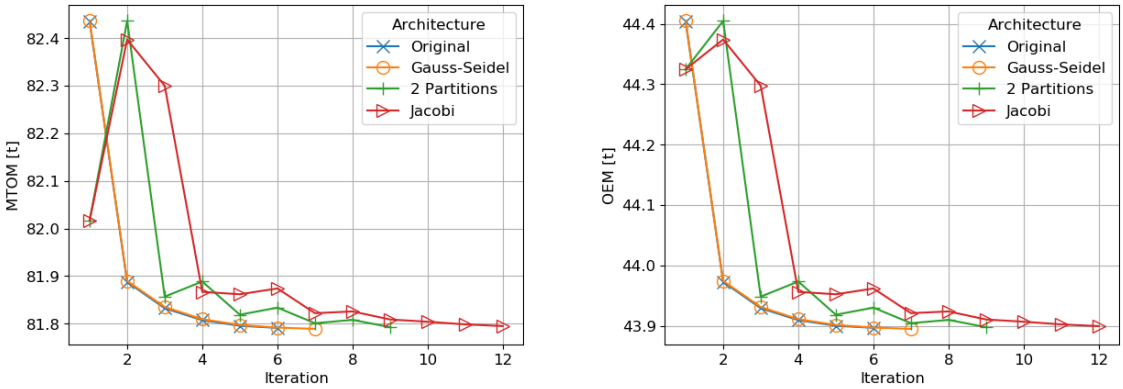


(b) Convergence with two partitions



(c) Jacobi convergence

Figure 7.4: Different convergence strategies used to validate the Initiator and the decomposition



(a) MTOM (b) OEM

Figure 7.5: Convergence history of the MTOM and OEM for different architectures

Finally, it can be concluded that the execution time of the new implementation is significantly longer than the original implementation. This is due to two reasons. First of all, in the new implementation, a separate matlab engine was started for each module. Starting a matlab engine takes around 10-15 seconds. So, starting seven engines takes 70-100 seconds. This could easily be resolved by using Python for the different modules instead of Matlab. The second reason is the reading and writing of the input and output files. Each time a module is called, the input file needs to be translated into an InitiatorController object and once the module is finished, the object needs to be translated back into an output file. This increases the execution time of the modules significantly. Furthermore, all the results from a module are now written to the output file. The performance could be improved by only writing the necessary data to the output file.

However, even though the execution time increases, the new implementation of the Initiator also has several benefits. One of them is the simplicity with which you can run an optimization and the easiness with which you can add and remove modules from the design problem.

7.3. Optimization

The implementation of the Initiator into KADMOS creates several opportunities for analyses, that were difficult to achieve using the original implementation. For example, a hotstart, DOE or optimization are now easy to perform. Furthermore, the agility of the Initiator has improved a lot, as it is now very easy to remove or add new modules to the design case. To demonstrate this, several optimization studies were performed on the A320-200 design.

Three design variables were chosen for the optimization, namely the wing loading, thrust loading and aspect ratio. In order for the aspect ratio to have a proper influence on the design, an aerodynamics module is needed. The aerodynamics module that was implemented calculates the drag polar using equation 7.1.

$$C_D = C_{D_0} + \frac{C_L^2}{\pi A e} \tag{7.1}$$

The induced drag, C_{D_i}, is calculated using the Athena Vortex Lattice (AVL) software[1], while the values for the parasite and wave drag, C_{D_0}, are held constant to a total value of 200 drag counts.

Thanks to KADMOS, the implementation of a new module is now very simple. The AVL module must use the same data schema for its inputs and outputs as the rest of the modules. However, no connections have to made with the other modules, as KADMOS automatically takes care of this. AVL has been implemented using a Python wrapper[2]. This clearly shows that different modules, using different programming languages, can now easily be combined as long as they use the same data schema.

Besides AVL, also a constraint and two objective modules were added to the repository. The constraint module will replace the Wing Thrust Loading module during the optimization. The Wing Thrust Loading assumed the design point by choosing a point on the wing and thrust loading diagram. In the constraint

[1] `http://web.mit.edu/drela/Public/web/avl/`, accessed 14th January 2019
[2] `https://github.com/renoelmendorp/AVLWrapper`, accessed 14th January 2019

module, the current design point will be checked against the constraints in the wing and thrust loading diagram. Therefore, the best design point is no longer an assumption, but the result of the optimization. The objective is either the minimization of the MTOM or the fuel mass. The new repository with both the Initiator modules as well as the newly added modules is shown in Figure 7.6.

The resulting XDSM for the optimization for minimum MTOM is shown in Figure 7.7. As mentioned before, the Wing Thrust Loading module has been removed and the AVL, constraint and objective modules are added. In the first optimization case, the objective is the minimization of the MTOM, while varying the range from 3000 to 5000km.

The results are shown in Figures 7.8 and 7.9. Figure 7.8 shows the history of the design variables for a range of 3000, 4000 and 5000km, respectively. Figure 7.9a visualizes the optimized wing geometries and Figure 7.9b shows the constraint history for a range of 4000km. Finally, Figure 7.10 shows the final wing thrust loading diagram for a range of 4000km. The initial and final values for the objective values, constraints and design variables are shown in Table 7.2. The history of the constraints and wing thrust loading diagram for a range of 3000 and 5000km, can be found in Appendix C.

The results show that the wing loading and thrust loading are increasing with increasing range, while the aspect ratio is decreasing. Due to decrease in aspect ratio, the L/D_{max} decreases as well. Furthermore, the MTOM and OEM increases with increasing range as more fuel mass is needed. The wing geometry shows an an increase in surface area. This is due to the fact that the aspect ratio is decreasing, while the wing span keeps constant. The wing span is constant, because the maximum wing span constraint is active in the final design point. As can be seen in Figures 7.9b and 7.10 and Table 7.2, three constraints are active in the final design point, namely the maximum wing span constraint, landing distance and take-off distance. Each of these constraint put a bound on one of the design variables. In order to minimize the MTOM, the wing thrust loading must be maximal. Therefore, the landing distance constraint is the active constraint for the wing loading. Furthermore, the thrust loading must be minimal. The thrust loading is constraint by the take-off distance. Finally, the aspect ratio must be maximal as this will maximize the L/D_{max} and thus minimizes the MTOM. As can be seen in Figure 7.10, the aspect ratio is bounded by the maximum wing span constraint. Note that the MTOM in the final design point is higher than in the initial design point. The reason for this is that the wing span constraint was violated at the initial point, but satisfied at the final design point.

Because the wing span constraint was active in the previous optimization case, an optimization was performed without this constraint. Changing the optimization is now very simple in KADMOS. The problem role of the wing span constraint must only be changed from constraint to quantity of interest. The results are shown in Appendix C. The results show an significant increase in span, while the MTOM reduces with 730kg.

With the new implementation of the Initiator, it is not only very easy to add and remove constraints, but also to change the objective function. Another optimization was performed by minimizing the fuel mass instead of the MTOM. In this case, the optimization was reformulated in KADMOS by removing the Objective-MTOM module and inserting the Objective-FM module. The results can again be found in Appendix C. The results show a reduction in fuel mass of 350kg, while the MTOM increases with 100kg.

This chapter has shown a successful implementation of the Initiator toolbox into the MDAO development process. A full automation of the process has been achieved, starting with a repository of stand-alone modules, through the MDAO problem formulation until the creation of the executable workflow. Even though the computational time increased significantly with respect to the original toolbox, the new implementation has shown a lot of benefits. First of all, the new implementation has increased the transparency of the Initiator. Thanks to KADMOS, the input and output connections between the different modules can now clearly be visualized. Furthermore, the automated execution process formulation has made it possible to run the Initiator on multiple CPUs. Next to that, the new implementation of the Initiator has shown an improved flexibility. New modules can now easily be combined with the Initiator even when they are written in different programming languages as long as they use the same central data schema. Thanks to the automated execution process the new MDAO system is easily reformulated after the addition of the new modules. Finally, the full MDAO development process has made it possible to run different types of analysis which were not possible with the original Initiator. Hotstarts, DOEs and optimizations are now easily performed.

Table 7.2: Initial and final values for the minimization of the MTOM for a range of 3000, 4000 and 5000km

	Range: 3000 km		*Range: 4000 km*		*Range: 5000 km*			
	Start Value	**End Value**	**Start Value**	**End Value**	**Start Value**	**End Value**	**Lower Bound**	**Upper Bound**
Objective								
MTOM [-] (scaled)	0.89705	0.89725	1.00565	1.01261	1.09106	1.10961	-	-
MTOM [kg] (abs.)	73369	73385	82252	82821	89237	90754	-	-
Design Variables								
A [-]	9.50	9.11	9.50	8.43	9.50	8.02	6	13
W/S [N/m^2]	5100	5072	5100	5289	5100	5516	2000	7000
T/W [-]	0.290	0.293	0.290	0.312	0.290	0.329	0	0.6
Quantities of Interest								
OEM [kg]	39860	39726	44084	43828	46335	46159	-	-
L/D_{max}	17.79	17.53	17.79	17.00	17.78	16.66	-	-
Constraints								
Balked Landing Climb AEO	-0.5953	-0.5846	-0.65308	-0.7009	-0.7131	-0.8229	-	0
Balked Landing Climb OEI	-0.0997	-0.09336	-0.13956	-0.1756	-0.1809	-0.2613	-	0
Cruise Speed	-0.3079	-0.3043	-0.2974	-0.3667	-0.2863	-0.4209	-	0
En Route Climb	-0.5328	-0.5119	-0.5328	-0.5411	-0.5327	-0.5741	-	0
Fuel Tank Volume	-0.5230	-0.5277	-0.4539	-0.4297	-0.3870	-0.3262	-	0
Initial Climb	-0.1729	-0.1529	-0.1729	-0.1675	-0.1729	-0.1876	-	0
Landing Distance	0.0074	**3.140e-05**	-0.0278	**-0.0001**	-0.0618	**6.0835e-05**	-	0
Max Cruise Lift Coefficient	-0.2725	-0.2765	-0.2725	-0.2456	-0.2725	-0.2131	-	0
Max Wing Span	0.0340	**-0.0013**	0.13832	**-8.454e-05**	0.2058	**-0.0001**	-	0
Second Segment Climb	-0.0460	-0.0311	-0.0460	-0.0498	-0.04602	-0.0714	-	0
Take-Off Distance	-0.0076	**7.425e-07**	-0.0076	**5.062e-05**	-0.0076	**6.1021e-08**	-	0
Time To Climb	-0.6524	-0.6604	-0.6522	-0.7598	-0.6519	-0.8427	-	0
Transition Climb	-0.1171	-0.0961	-0.1171	-0.1066	-0.1171	-0.1235	-	0

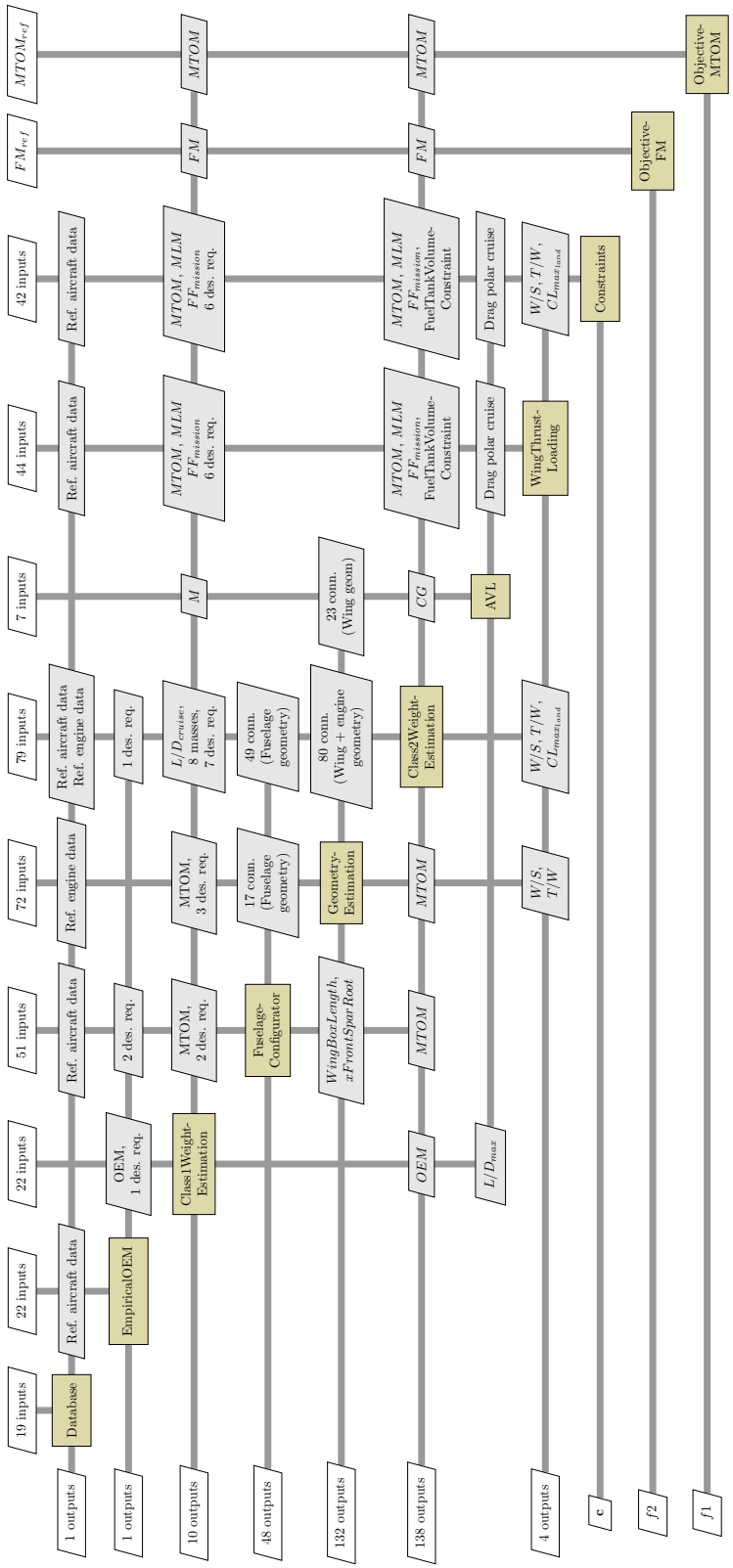


Figure 7.6: Overview of the repository, including the Initiator modules, AVL, the objective and the constraint modules

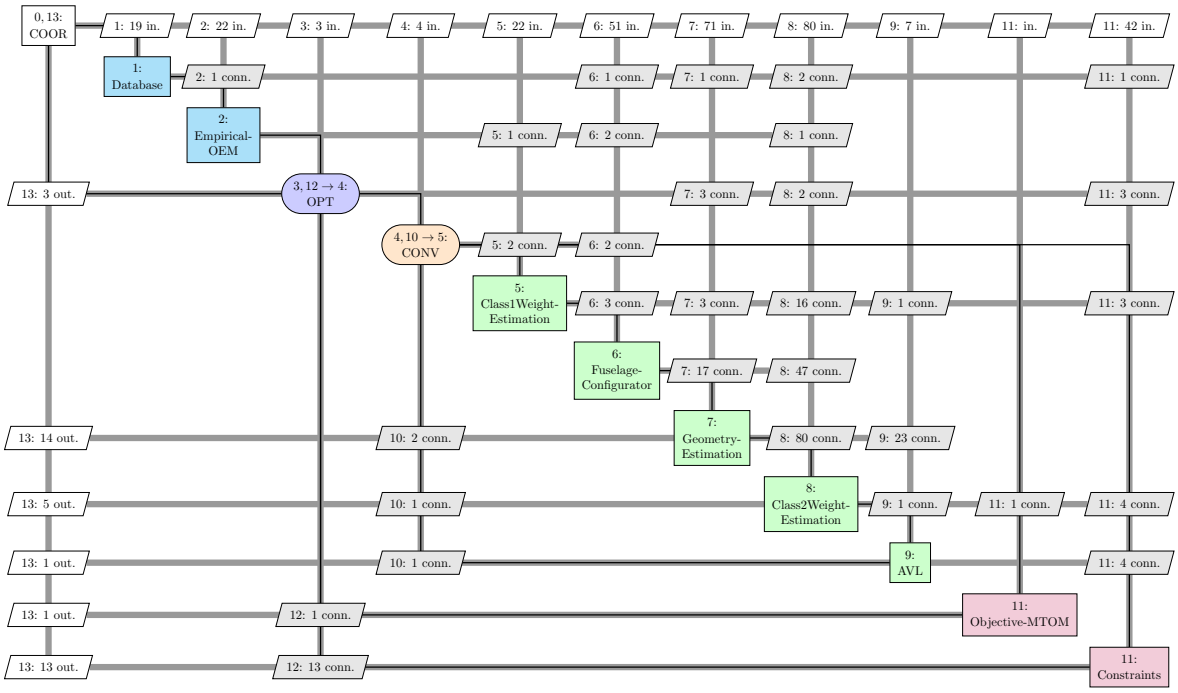
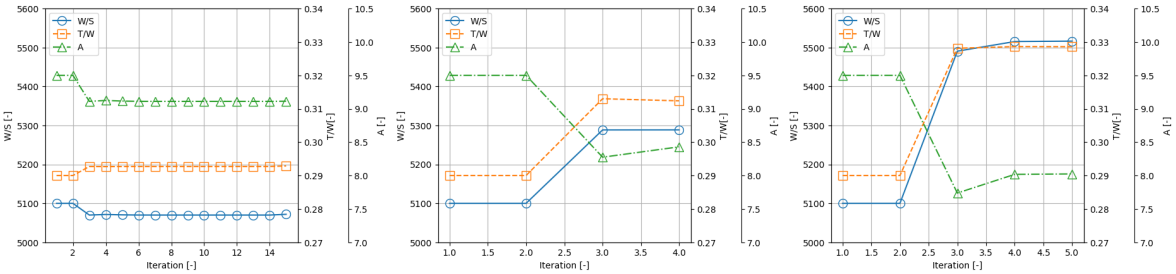
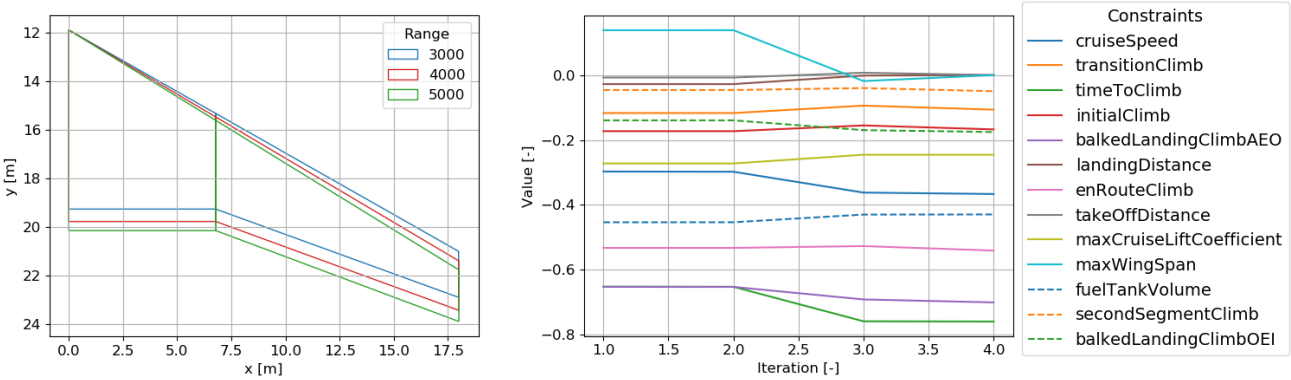


Figure 7.7: XDSM of the optimization for minimum MTOM using the Initiator



(a) 3000 km (b) 4000 km (c) 5000 km

Figure 7.8: History of the design variables for the minimization of the MTOM



(a) Wing geometry (b) History of the constraints for a range of 4000 km

Figure 7.9: Wing geometry and history of the constraints for the minimization of the MTOM

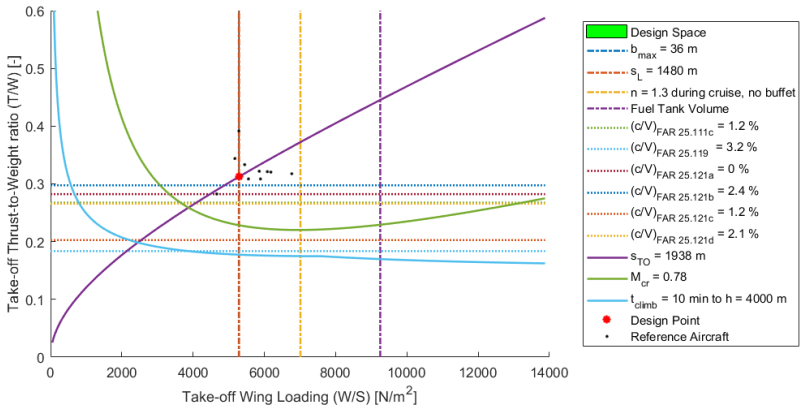
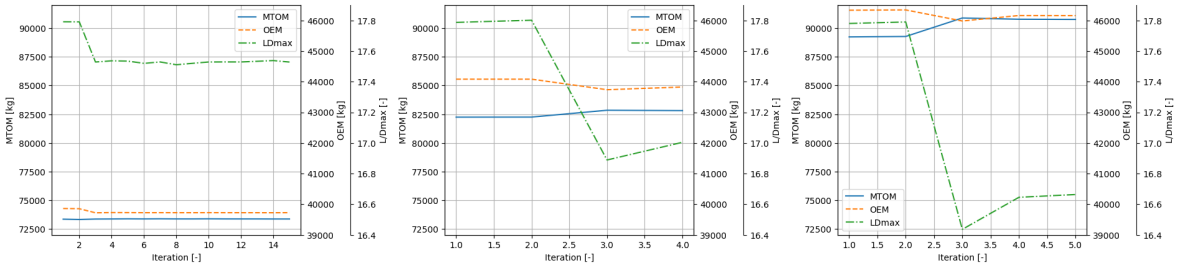


Figure 7.10: Wing thrust loading diagram for a range of 4000km when minimizing the MTOM



(a) 3000 km (b) 4000 km (c) 5000 km

Figure 7.11: History of the quantities of interest for the minimization of the MTOM

8

Conclusions & Recommendations

A proper sequencing and decomposition is a key prerequisite for obtaining MDAO systems that can be executed efficiently. This thesis showed how the setup of the execution process of large coupled MDAO systems can be fully automated to reduce the formulation time and to optimize workflow convergence. Based on the results, several conclusions can be drawn:

- *Successful development of four sequencing algorithms for the automatic determination of the execution order of the disciplinary analysis*
  Before this thesis, sequencing could only be performed manually. By implementing four different sequencing algorithms, which all differ in speed and accuracy, high-quality execution orders are automatically found for both small and large MDAO systems in a short time. The algorithms do not only minimize the number of feedback connections, but also take the execution time of the sequence into account to reduce the total convergence time of MDAO systems even further.

- *Successful implementation of the new Metis-based Decomposition algorithm for KADMOS graphs: MDK*
  Besides the support of decomposition in KADMOS, also a new decomposition algorithm, based on the Metis software package, has been implemented to automatically determine the best division of the different disciplinary analyses over multiple partitions. The algorithm returns high-quality partitions with excellent performance and scalability.

- *Support of decomposition over the full AGILE process*
  Decomposition reduces the convergence time of MDAO systems by executing multiple partitions simultaneously. Initially, KADMOS did not support the formulation of partitions, as all disciplinary analyses could only be executed either in parallel or in sequence. Therefore, the option has been created to formulate partitions in the monolithic architectures of KADMOS. This option has been extended through the full AGILE process. Both RCE and OpenMDAO can now interpret the partitions and execute them accordingly.

- *Creation of the opportunity to perform large benchmarking studies using a large variety of MDAO systems and MDAO architectures*
  The automation of the execution process formulation and the implementation of scalable mathematical problem has created the opportunity to perform large benchmarking studies. Thousands of MDAO systems are easily generated and executed. The benchmarking performed in this thesis showed that there is no 'one-size-fits-all' architecture for solving MDAO systems. The best architecture depends both on the type of MDAO system and the available computational resources.

- *Improved process formulation resulting in a significant convergence time reduction for the Gauss-Seidel, Jacobi and IDF implementations*
  Due to the automated execution process formulation, the convergence time of the Gauss-Seidel, Jacobi and IDF convergence are easily improved. The convergence time of the Gauss-Seidel convergence is improved by executing the disciplines with no data dependencies in parallel. This reduces the execution time of one iteration, while not increasing the number of iterations. The convergence time of

the Jacobi and IDF architecture is improved by executing some disciplines in sequence, while not increasing the execution time of one iteration. This reduces the number of iterations and thus the total execution time. These new processes can now easily be applied thanks to the sequencing and decomposition algorithms.

- *Improved flexibility of the Initiator*
  The implementation of the Initiator toolbox into KADMOS has resulted in an improved flexibility of the toolbox. Different analysis and optimization cases can easily be performed due to the fully automated process in KADMOS. Furthermore, new modules are easily added or removed, as including new modules merely requires one to comply the module to the central data schema. In addition, these modules can be written in different languages as long as they use the same data schema. Thanks to the sequencing algorithms, the new tools are automatically added at the correct location in the execution order.

- *Improved transparency of the Initiator*
  The new implementation of the Initiator also resulted in an improved transparency of the Initiator. Thanks to KADMOS, the input and output connections between the modules are clearly visualized. This increases the understanding of the design process and helps in the debugging process when new modules are added or existing modules are updated.

The automated execution process has showed its benefits towards optimizing the workflow and reducing the setup and convergence time of MDAO systems. However, the results can still be improved further, and based on the work performed in this thesis, the following main recommendations were identified:

- *Addition of the sensitivities in the sequencing and decomposition algorithms*
  The results showed that the sensitivities between the input and output of the different disciplinary analyses have a big influence on the convergence time of MDAO systems. Therefore, it is recommended to include sensitivities into the sequencing and decomposition algorithms. For example, strong couplings (connections with high sensitivity values) could then be placed within the same partition, while the weak couplings could be used as connections between partitions and/or as feedback in the execution sequence. When the sensitivities are known, a better trade-off between the coordination complexity and execution time of the sequence can be made.

- *Include the maximum number of available CPUs as an option in the sequencing and decomposition algorithms*
  The results showed that the best architecture depends on the number of available CPUs. It is therefore recommended to add the maximum number of available CPUs as an option in the sequencing and decomposition algorithms. This will create the opportunity to adapt the solution strategy to the specific computational environment.

- *More research into which architecture is best for which MDAO system*
  As mentioned above, the results showed that there is no 'one-size-fits-all' solution towards solving MDAO systems. The best architecture depends, amongst others, on the characteristics of the MDAO system. Therefore, it is recommended to perform more research into which solution strategy is best for which type of MDAO system. Using the best solution strategy in combination with a proper execution process will further reduce the time necessary to perform MDAO.

- *Improve the execution time of the Initiator*
  The new implementation of the Initiator showed a lot of benefits in terms of improved flexibility and transparency. However, its main disadvantage is the significant increase in convergence time with respect to the original implementation. Several measures can be taken to decrease the convergence time of the new implementation. For example, translating the modules from Matlab to Python will eliminate the need for Matlab engines. Furthermore, the output writer of the Initiator can be improved. At the moment, all generated data is written to the output file. Writing only the required data and variables of interest will reduce the execution tiem fo the individual modules significantly.

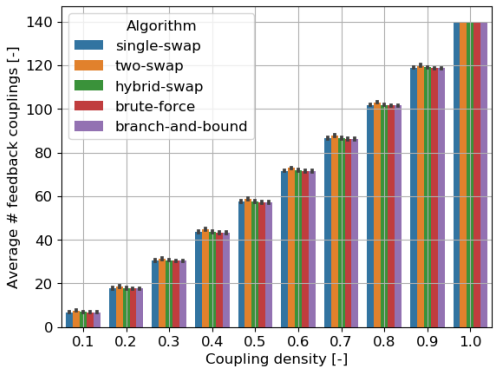
- *Improve Initiator data schema*
  The final recommendation concerns the implemented data schema for the Initiator prototype. It will be easier to add new modules when a more standard data schema is used. For example, if a standard data schema like CPACS is used, the Initiator could easily be connected to external design and analysis tools that also make use of this data schema.

This thesis has made a significant step towards the full automation of the execution process formulation in MDAO systems. The obtained solutions can be improved in future work by optimizing the execution process for the given computational environment and the inclusion of sensitivities. However, the current results already showed a significant decrease in the execution time of MDAO systems. Furthermore, thanks to the automation, the formulation of MDAO systems is easier and the setup time has been reduced. Therefore, this thesis has made a direct contribution in reducing some of the barriers that still exist for using MDAO today.

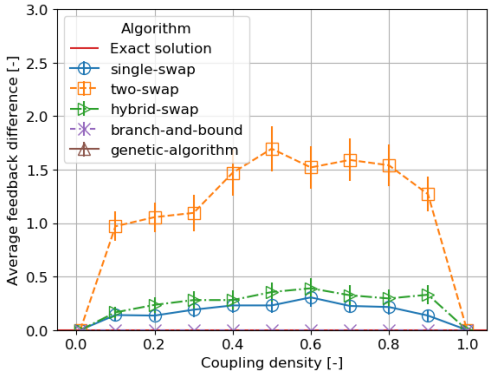
A

Verification & Validation Plots

A.1. Sequencing plots

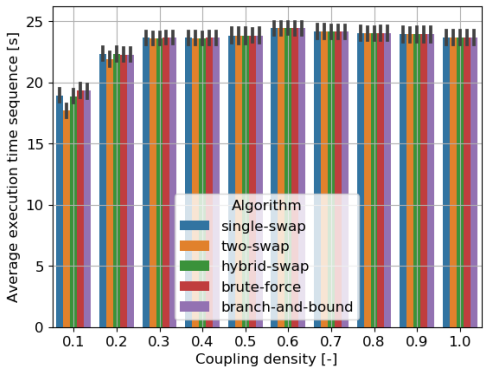


(a) Average number of feedback couplings

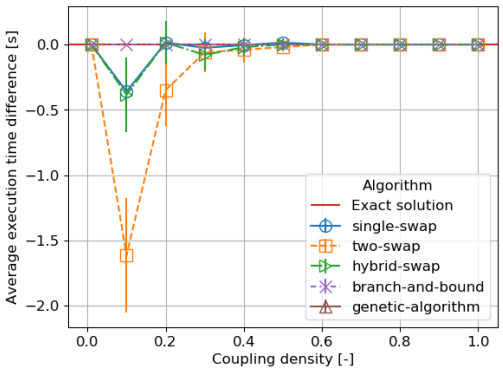


(b) Average difference in feedback couplings with respect to the exact solution

Figure A.1: Solution accuracy of the sequencing algorithms for different coupling densities when comparing the number of feedback couplings. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters

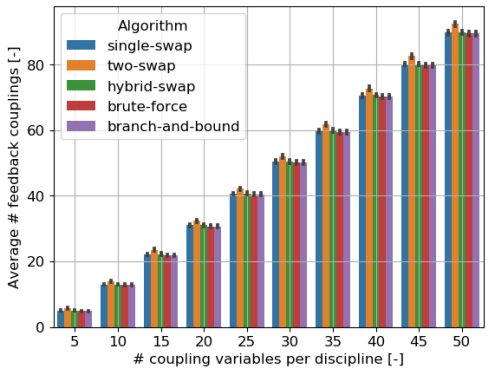


(a) Average execution time

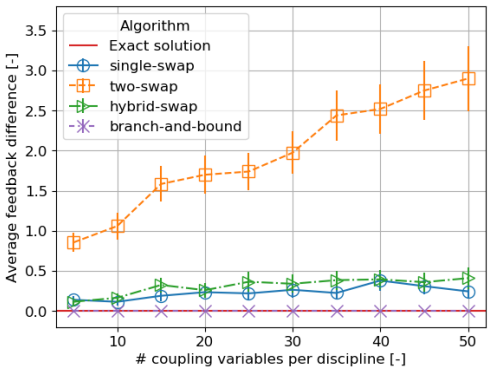


(b) Average difference in execution time with respect to the exact solution

Figure A.2: Solution accuracy of the sequencing algorithms for different coupling densities when comparing the execution time of the obtained sequences. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters

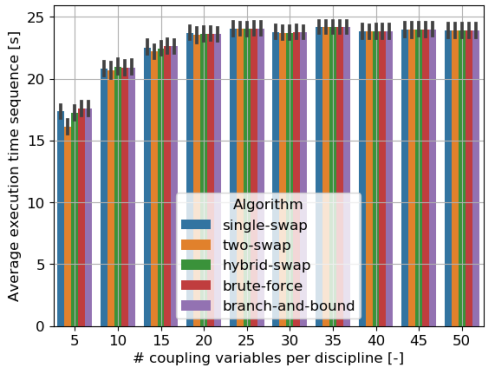


(a) Average number of feedback couplings

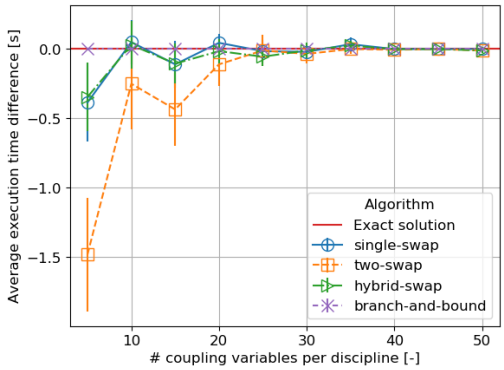


(b) Average difference in feedback couplings with respect to the exact solution

Figure A.3: Solution accuracy of the sequencing algorithms for different numbers of coupling variables per discipline when comparing the number of feedback couplings. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters



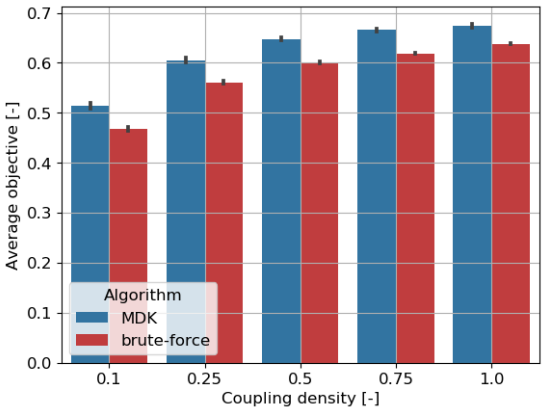
(a) Average execution time



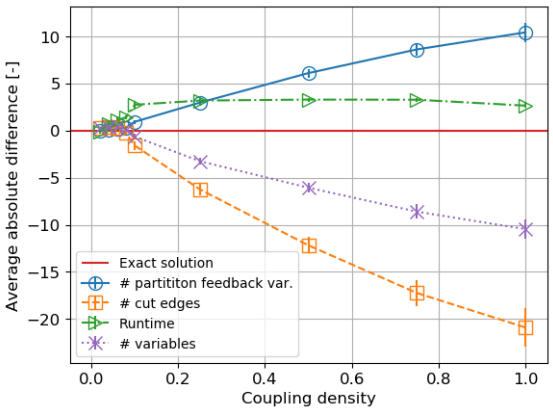
(b) Average difference in execution time with respect to the exact solution

Figure A.4: Solution accuracy of the sequencing algorithms for different numbers of coupling variables per discipline when comparing the execution time of the obtained sequences. $\rho_c = 0.08$, $n_{cv} = 5$, $n_{tc} = 200$, no clusters

A.2. Decomposition plots

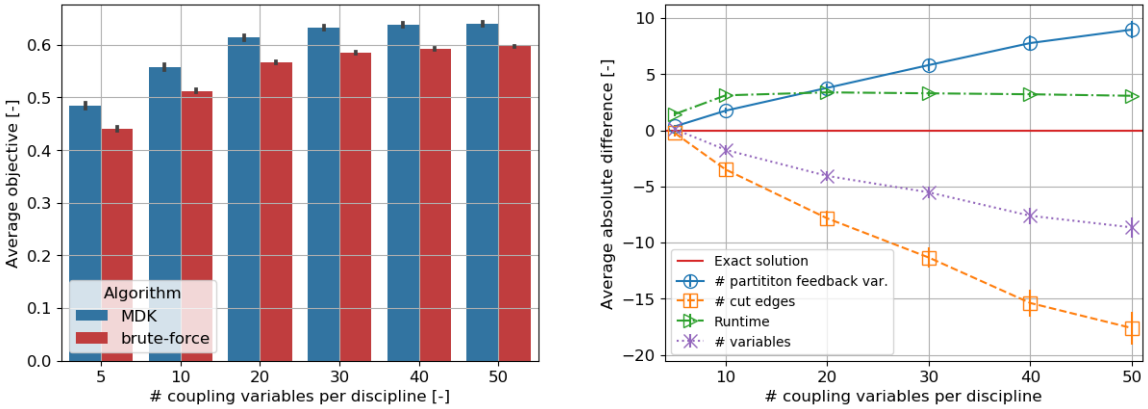


(a) Average objective value



(b) Absolute differences with respect to the exact solution

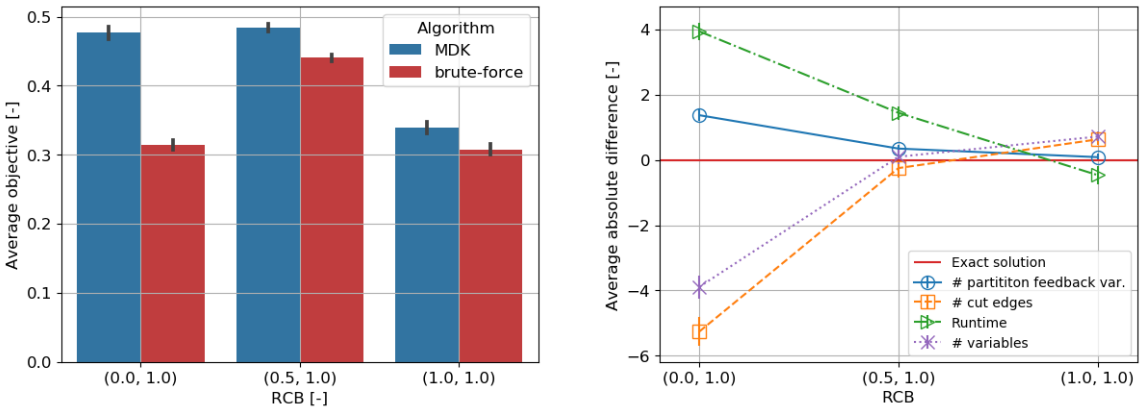
Figure A.5: Solution accuracy of the MDK algorithm when varying the coupling density. $n_d = 8$, $n_p = 2$, $n_{cv} = 5$, $n_{tc} = 200$, RCB = 0.5, no clusters



(a) Average objective value

(b) Absolute differences with respect to the exact solution

Figure A.6: Solution accuracy of the MDK algorithm when varying the number of coupling variables per discipline. $n_d = 8$, $n_p = 2$, $\rho_c = 0.08$, $n_{tc} = 200$, RCB = 0.5, no clusters



(a) Average objective value

(b) Absolute differences with respect to the exact solution

Figure A.7: Solution accuracy of the MDK algorithm when varying the RCB. $n_d = 8$, $n_p = 2$, $\rho_c = 0.08$, $n_{tc} = 200$, $n_{cv} = 5$, no clusters

B

Initiator Data File

```
<?xml version='1.0' encoding='UTF-8'?>
<cpacs>
  <toolspecific>
    <TUDinitiator>
      <initiator>
        <aircraft uID="A320-200">
          <parts>
            <Wing uID="MainWing">
              <Type>MainWing</Type>
              <Sections>
                <Airfoil>
                  <AirfoilName>b737a</AirfoilName>
                  <Chord>7.8183432339057886</Chord>
                  <Position>
                    <x>11.9000636337804444</x>
                    <y>0.0000000000000000</y>
                    <z>-0.5957003281282690</z>
                  </Position>
                </Airfoil>
                <Airfoil>
                  <AirfoilName>b737b</AirfoilName>
                  <Chord>4.0817555449718910</Chord>
                  <Position>
                    <x>15.6366513227143429</x>
                    <y>7.3990874721923641</y>
                    <z>0.1819751023003570</z>
                  </Position>
                </Airfoil>
                <Airfoil>
                  <AirfoilName>b737d</AirfoilName>
                  <Chord>1.9813859757195100</Chord>
                  <Position>
                    <x>21.8114368139764601</x>
                    <y>19.6262267166906206</y>
                    <z>1.4670992220802870</z>
                  </Position>
                </Airfoil>
              </Sections>
              <Kink>0.3770000000000000</Kink>
              <SectionPositions mapType="vector">0.0000000000000000;0.3770000000000000;1.0000000000
                  0000</SectionPositions>
              <SparPositions mapType="vector">0.1000000000000000;0.6500000000000000</SparPositions>
              <FuelTank>
                <HasTank>1.0000000000000000</HasTank>
                <SpanPosition mapType="vector">0.0000000000000000;0.8000000000000000</SpanPosition>
                <Volume>41.2962815385265003</Volume>
              </FuelTank>
              <Orientation>
                <phi>0.0000000000000000</phi>
                <psi>0.0000000000000000</psi>
```

```
50                <theta>0.0000000000000000</theta>
51              </Orientation>
52              <Position>
53                <x>11.9000636337804444</x>
54                <y>0.0000000000000000</y>
55                <z>-0.5957003281282690</z>
56              </Position>
57              <Span>39.2524534333812412</Span>
58              <SweepsLE mapType="vector">26.7940749732813650;26.7940749732813650</SweepsLE>
59              <Dihedrals mapType="vector">6.0000000000000000;6.0000000000000000</Dihedrals>
60              <Symmetric>true</Symmetric>
61              <RootChord>7.8183432339057886</RootChord>
62              <Tapers mapType="vector">0.5220742327185830;0.4854249486254220</Tapers>
63              <TcRatios mapType="vector">0.1536505470988710;0.1256873739180040;0.1080099688107340</
                    TcRatios>
64              <Twists mapType="vector">0.0000000000000000;0.0000000000000000;0.0000000000000000</
                    Twists>
65            </Wing>
66            <Wing uID="HorizontalStabiliser">
67              <Type>HorizontalTail</Type>
68              <Sections>
69                <Airfoil>
70                  <AirfoilName>N64A010</AirfoilName>
71                  <Chord>4.0089558836986443</Chord>
72                  <Position>
73                    <x>31.2923081532591532</x>
74                    <y>0.0000000000000000</y>
75                    <z>1.2994030987654330</z>
76                  </Position>
77                </Airfoil>
78                <Airfoil>
79                  <AirfoilName>N64A010</AirfoilName>
80                  <Chord>1.3149375298531549</Chord>
81                  <Position>
82                    <x>35.1359475679787963</x>
83                    <y>6.6548667669397501</y>
84                    <z>1.9988577810996000</z>
85                  </Position>
86                </Airfoil>
87              </Sections>
88              <Tapers>0.3280000000000000</Tapers>
89              <SparPositions mapType="vector">0.1000000000000000;0.6500000000000000</SparPositions>
90              <FuelTank>
91                <HasTank>0.0000000000000000</HasTank>
92              </FuelTank>
93              <Orientation>
94                <phi>0.0000000000000000</phi>
95                <psi>0.0000000000000000</psi>
96                <theta>0.0000000000000000</theta>
97              </Orientation>
98              <Position>
99                <x>31.2923081532591532</x>
100               <y>0.0000000000000000</y>
101               <z>1.2994030987654330</z>
102             </Position>
103             <Span>13.3097335338795002</Span>
104             <SectionPositions mapType="vector">0.0000000000000000;1.0000000000000000</
                    SectionPositions>
105             <SweepsLE>30.0093639700751318</SweepsLE>
106             <Dihedrals>6.0000000000000000</Dihedrals>
107             <Symmetric>true</Symmetric>
108             <RootChord>4.0089558836986443</RootChord>
109             <Twists mapType="vector">0.0000000000000000;0.0000000000000000</Twists>
110             <TcRatios mapType="vector">0.1000000000000000;0.1000000000000000</TcRatios>
111             <Area>35.4298013485752961</Area>
112           </Wing>
113           <Wing uID="VerticalStabiliser">
114             <Type>VerticalTail</Type>
115             <Sections>
116               <Airfoil>
117                 <AirfoilName>N64A010</AirfoilName>
```

```
118                 </Airfoil>
119                 <Airfoil>
120                   <AirfoilName>N64A010</AirfoilName>
121                 </Airfoil>
122               </Sections>
123               <Tapers>0.3290000000000000</Tapers>
124               <FuelTank>
125                 <HasTank>0.0000000000000000</HasTank>
126               </FuelTank>
127               <Orientation>
128                 <phi>90.0000000000000000</phi>
129                 <psi>0.0000000000000000</psi>
130                 <theta>0.0000000000000000</theta>
131               </Orientation>
132               <Position>
133                 <x>29.0988960815052700</x>
134                 <y>0.0000000000000000</y>
135                 <z>1.2994030987654330</z>
136               </Position>
137               <Span>6.4743182586675392</Span>
138               <SectionPositions mapType="vector">0.0000000000000000;1.0000000000000000</
                      SectionPositions>
139               <SweepsLE>28.1337787219454327</SweepsLE>
140               <Dihedrals>0.0000000000000000</Dihedrals>
141               <RootChord>6.0894641259100251</RootChord>
142               <Twists mapType="vector">0.0000000000000000;0.0000000000000000</Twists>
143               <TcRatios mapType="vector">0.1200000000000000;0.1200000000000000</TcRatios>
144               <Symmetric>false</Symmetric>
145               <Area>26.1979980715724210</Area>
146             </Wing>
147             <Fuselage uID="Fuselage">
148               <Type>Conventional</Type>
149               <HeightMargin>0.3000000000000000</HeightMargin>
150               <NoseShapeFactor>0.1500000000000000</NoseShapeFactor>
151               <TailShapeFactor>0.2000000000000000</TailShapeFactor>
152               <AftRatioWidth>0.1400000000000000</AftRatioWidth>
153               <AftRatioHeight>0.1400000000000000</AftRatioHeight>
154               <FloorOffset mapType="vector">0.0000000000000000;0.0000000000000000</FloorOffset>
155               <PaxDivision>1.0000000000000000</PaxDivision>
156               <CabinHeight>1.9299999999999999</CabinHeight>
157               <Cabins>
158                 <Cabin uID="Cabin1">
159                   <ClassDistribution mapType="vector">0.0800000000000000;0.0000000000000000;0.0000000
                          000000000;0.9200000000000000</ClassDistribution>
160                   <Classes>
161                     <EC>
162                       <seatingArr mapType="vector">3.0000000000000000;3.0000000000000000</seatingArr>
163                       <seatingDim mapType="vector">0.4600000000000000;0.0480000000000000;0.81
                              29999999999999;0.8000000000000000;0.3000000000000000</seatingDim>
164                     </EC>
165                     <FC>
166                       <seatingArr mapType="vector">2.0000000000000000;2.0000000000000000</seatingArr>
167                       <seatingDim mapType="vector">0.5700000000000000;0.0780000000000000;0.9140000000
                              000000;0.8000000000000000;0.3000000000000000</seatingDim>
168                     </FC>
169                   </Classes>
170                   <Position>
171                     <x>4.1398070832253282</x>
172                     <y>0.0000000000000000</y>
173                     <z>0.0500000000000000</z>
174                   </Position>
175                   <Orientation>
176                     <phi>0.0000000000000000</phi>
177                     <psi>0.0000000000000000</psi>
178                     <theta>0.0000000000000000</theta>
179                   </Orientation>
180                   <x mapType="vector">0.0000000000000000;1.5053843939001199;25.9678807947770
                          629;27.8496113871522084</x>
181                   <w mapType="vector">3.6258430602194132;3.6539999999999999;3.6539999999999999;3.381
                          6688129564079</w>
182                   <FloorArea>101.4850595368324946</FloorArea>
```

```
183                 <PlotGL>
184                   <GL>
185                     <x mapType="vector">0.0000000000000000;1.9249544855876921;1.9249544855876921;0.
                            0000000000000000</x>
186                     <y mapType="vector">-1.8129215301097070;-1.8270000000000000;1.8270000000000000;
                            1.8129215301097070</y>
187                   </GL>
188                   <GL>
189                     <x mapType="vector">26.0077544855876823;27.8496112006814549;27.849611200681
                            4549;26.0077544855876823</x>
190                     <y mapType="vector">-1.8241146641303849;-1.6908344199715819;1.6908344199715819;
                            1.8241146641303849</y>
191                   </GL>
192                 </PlotGL>
193               </Cabin>
194             </Cabins>
195             <Length>37.6346098475029862</Length>
196             <Diameter>3.9710773257326659</Diameter>
197             <Height>3.9713355208551269</Height>
198             <Position>
199               <x>0.0000000000000000</x>
200               <y>0.0000000000000000</y>
201               <z>0.0000000000000000</z>
202             </Position>
203             <Orientation>
204               <phi>0.0000000000000000</phi>
205               <psi>0.0000000000000000</psi>
206               <theta>0.0000000000000000</theta>
207             </Orientation>
208             <Container>LD3-45</Container>
209             <NoseLength>4.1398070832253300</NoseLength>
210             <TailLength>5.6451914771254490</TailLength>
211             <MidHeight>0.7979222923099480</MidHeight>
212             <FuelTank>
213               <HasTank>0.0000000000000000</HasTank>
214             </FuelTank>
215             <CargoBays uID="BulkBay1">
216               <CargoType uID="LD3-45">
217                 <Type>ULD</Type>
218                 <ULDType>LD3-45</ULDType>
219               </CargoType>
220               <NumContainers>5.0000000000000000</NumContainers>
221             </CargoBays>
222             <CargoBays uID="BulkBay2">
223               <CargoType uID="LD3-45">
224                 <Type>ULD</Type>
225                 <ULDType>LD3-45</ULDType>
226               </CargoType>
227               <NumContainers>8.0000000000000000</NumContainers>
228             </CargoBays>
229           </Fuselage>
230           <Engine uID="Engine1">
231             <Type>TurboFan</Type>
232             <Location>MainWing</Location>
233             <BypassRatio>5.7000000000000002</BypassRatio>
234             <NacelleSection>
235               <Airfoil>
236                 <AirfoilName>PARSEC1</AirfoilName>
237               </Airfoil>
238             </NacelleSection>
239             <Position>
240               <x>12.9415537964743503</x>
241               <y>-6.6729170836748111</y>
242               <z>-1.3675854899731841</z>
243             </Position>
244             <Orientation>
245               <phi>0.0000000000000000</phi>
246               <psi>0.0000000000000000</psi>
247               <theta>0.0000000000000000</theta>
248             </Orientation>
249             <Length>3.6958360612265722</Length>
```

```
250             <NacelleLength>2.2563892962744929</NacelleLength>
251             <NacelleDiameter>1.7332200104892050</NacelleDiameter>
252             <CenterbodyDiameter>0.7841440054357790</CenterbodyDiameter>
253             <FanDiameter>1.3159299607638240</FanDiameter>
254           </Engine>
255           <Engine uID="Engine2">
256             <Type>TurboFan</Type>
257             <Location>MainWing</Location>
258             <BypassRatio>5.7000000000000002</BypassRatio>
259             <NacelleSection>
260               <Airfoil>
261                 <AirfoilName>PARSEC1</AirfoilName>
262               </Airfoil>
263             </NacelleSection>
264             <Position>
265               <x>12.941553796474350 3</x>
266               <y>6.6729170836748111</y>
267               <z>-1.3675854899731841</z>
268             </Position>
269             <Orientation>
270               <phi>0.0000000000000000</phi>
271               <psi>0.0000000000000000</psi>
272               <theta>0.0000000000000000</theta>
273             </Orientation>
274             <Length>3.6958360612265722</Length>
275             <NacelleLength>2.2563892962744929</NacelleLength>
276             <NacelleDiameter>1.7332200104892050</NacelleDiameter>
277             <CenterbodyDiameter>0.7841440054357790</CenterbodyDiameter>
278             <FanDiameter>1.3159299607638240</FanDiameter>
279           </Engine>
280         </parts>
281         <missions>
282           <mission uID="Mission1">
283             <Altitude>11280.0000000000000000</Altitude>
284             <CruiseMach>0.7800000000000000</CruiseMach>
285             <DivRange>370.0000000000000000</DivRange>
286             <LoiterTime>30.0000000000000000</LoiterTime>
287             <Pax>156.0000000000000000</Pax>
288             <PayloadMass>20536.0000000000000000</PayloadMass>
289             <Range>3917.0000000000000000</Range>
290             <CargoMass>8056.0000000000000000</CargoMass>
291             <CruiseSpeed>230.1542971421397965</CruiseSpeed>
292           </mission>
293           <DefaultMissionuID>Mission1</DefaultMissionuID>
294         </missions>
295         <configurationParameters>
296           <CompositeStructures mapType="vector">0.0000000000000000;0.0000000000000000;1.0000000000
                00000</CompositeStructures>
297           <WingLocation>low</WingLocation>
298           <Freight>0.0000000000000000</Freight>
299           <WingAspectRatio>9.5000000000000000</WingAspectRatio>
300           <SupercriticalAirfoil>1.1000000000000001</SupercriticalAirfoil>
301           <TEinboardSweep>0.0000000000000000</TEinboardSweep>
302           <TailType>standard</TailType>
303         </configurationParameters>
304         <performanceParameters>
305           <SFC>0.6000000000000000</SFC>
306           <LDmax>18.000000000000000</LDmax>
307           <CLmaxLanding>2.7999999999999998</CLmaxLanding>
308           <CruiseLD>15.5879999999999992</CruiseLD>
309           <CDmin>0.0000000000000000</CDmin>
310           <CLmin>0.0000000000000000</CLmin>
311           <CLmaxClean>1.6000000000000001</CLmaxClean>
312           <CLmaxTakeOff>2.2000000000000002</CLmaxTakeOff>
313           <DragPolarClean mapType="vector">0.0000000000000000;0.0000000000000000;0.0000000000000000
                ;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.00000
                00000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0000000000000
                000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.00
                00000000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0000000000
                000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0
                .0000000000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0000000
```

```
000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.000000000000000
0;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0000
000000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.000000000000
0000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0000000000000000;0.0
000000000000000;0.0000000000000000;0.0000000000000000</DragPolarClean>
<CDw>0.0020000000000000</CDw>
<CDf>0.0180000000000000</CDf>
</performanceParameters>
<results>
<MassBreakdown>
<OEM>45847.5342473829223309</OEM>
<HarmonicMission>
<MTOM>81789.2111000000004424</MTOM>
<ZFM>66383.5342473829223309</ZFM>
<PLM>20536.0000000000000000</PLM>
<FFTotal>0.2091746296528110</FFTotal>
<DLM>69372.2549167080287589</DLM>
<MLM>71805.9107915131316986</MLM>
<FFMission>0.1735700756473650</FFMission>
<MTOM2>81789.2111000000004424</MTOM2>
<MLM2>71805.9107915131316986</MLM2>
<FFMission2>0.1735700756473650</FFMission2>
</HarmonicMission>
<InputMissions uID="Mission1">
<PLM>20536.0000000000000000</PLM>
<FFTotal>0.2091746296528110</FFTotal>
<FFReserve>0.0438950346093460</FFReserve>
</InputMissions>
<FerryMission>
<Extended>0.0000000000000000</Extended>
<RangeFactor>1.0000000000000000</RangeFactor>
</FerryMission>
</MassBreakdown>
<WS_TW>
<WS>5080.7751831845444030</WS>
<TW>0.2871414604191640</TW>
</WS_TW>
<FuelGeometric>
<Vs mapType="vector">20.6481407692632501;20.6481407692632501</Vs>
<TotalFV>41.2962815385265003</TotalFV>
<TotalCG mapType="vector">16.3761238091188730;0.0000000000000000;-0.0694242269696460</
TotalCG>
<TotalFM>33449.9880462064684252</TotalFM>
</FuelGeometric>
<MassCG>
<TotalCGHarmonic mapType="vector">17.4449534261684960;0.0000000000000000;0.027018721741
5320</TotalCGHarmonic>
</MassCG>
</results>
<designRequirements>
<MaxPaxInitial>156.0000000000000000</MaxPaxInitial>
<CargoMass>8056.0000000000000000</CargoMass>
<Pax>156.0000000000000000</Pax>
<CargoMass2>8056.0000000000000000</CargoMass2>
<Altitude>11280.0000000000000000</Altitude>
<CruiseMach>0.7800000000000000</CruiseMach>
<CruiseSpeed>230.1542971421397965</CruiseSpeed>
<MaxPax>156.0000000000000000</MaxPax>
<Range>3917.0000000000000000</Range>
<DivRange>370.0000000000000000</DivRange>
<LoiterTime>30.0000000000000000</LoiterTime>
<AirworthinessRegulations>FAR-25</AirworthinessRegulations>
<TakeOffDistance>1938.0000000000000000</TakeOffDistance>
<LandingDistance>1480.0000000000000000</LandingDistance>
<AirportClassification>FAA-III</AirportClassification>
<TimeToClimb mapType="vector">10.0000000000000000;4000.0000000000000000</TimeToClimb>
</designRequirements>
</aircraft>
<programSettings>
<uGridScaling>2.0000000000000000</uGridScaling>
<vGridScaling>2.0000000000000000</vGridScaling>
```

```
<PaxBoundary>20.000000000000000</PaxBoundary>
<ShowMessages>1.0000000000000000</ShowMessages>
<UseArtificialDB>0</UseArtificialDB>
<InsideOutMargin>0.0010000000000000</InsideOutMargin>
<AirfoilPoints>121.00000000000000000</AirfoilPoints>
<DCtolerances mapType="vector">0.0010000000000000;0.0010000000000000;0.0010000000000000</
    DCtolerances>
<plotWSmin>50.000000000000000</plotWSmin>
<plotWSstep>25.000000000000000</plotWSstep>
<NumberReferenceAC>10.000000000000000</NumberReferenceAC>
<ShowReferenceACNames>0.0000000000000000</ShowReferenceACNames>
</programSettings>
<independentSettings>
<PaxMass>80.000000000000000</PaxMass>
<LuggageMass>22.000000000000000</LuggageMass>
<CompositeCorrection mapType="vector">0.8500000000000000;0.7500000000000000;0.7500000000000
    000</CompositeCorrection>
<FuelHg>4350.000000000000000</FuelHg>
<FFStartUp>0.9900000000000000</FFStartUp>
<FFTaxi>0.9900000000000000</FFTaxi>
<FuelDensity>810.000000000000000</FuelDensity>
<PylonEstimationAirfoil>N0012</PylonEstimationAirfoil>
<MaxLoadFactor>2.5000000000000000</MaxLoadFactor>
<SafetyFactor>1.5000000000000000</SafetyFactor>
<DiveMachFactor>1.2000000000000000</DiveMachFactor>
</independentSettings>
<referenceData>
<AircraftData>aircraftData.xml</AircraftData>
<EngineData>engineData.xml</EngineData>
<Span>39.2524534333812412</Span>
<Area>162.1847474252382142</Area>
<Chord>4.7776015610451017</Chord>
</referenceData>
<aircraftSettings>
<NoseDroop>0.1500000000000000</NoseDroop>
<UpSweep>0.3000000000000000</UpSweep>
<average_lavatory_area>1.0000000000000000</average_lavatory_area>
<galley_area_per_passenger mapType="vector">0.1100000000000000;0.0900000000000000;0.0600000
    000000000;0.0400000000000000</galley_area_per_passenger>
<lavatory_per_passenger mapType="vector">0.0417000000000000;0.0588000000000000;0.0333000000
    000000;0.0167000000000000</lavatory_per_passenger>
<aisle_widths mapType="vector">0.7650000000000000;0.5700000000000000;0.5300000000000000;0.5
    100000000000000</aisle_widths>
<CargobayMargin>0.1000000000000000</CargobayMargin>
<FreightFraction>0.0000000000000000</FreightFraction>
<FreightPackingEfficiency>0.5000000000000000</FreightPackingEfficiency>
<CargoDensity>423.1999999999999886</CargoDensity>
<ContainerCutoff>0.6500000000000000</ContainerCutoff>
<BulkMargin>0.4000000000000000</BulkMargin>
<WingBoxLength>0.0500000000000000</WingBoxLength>
<xFrontSparRoot>0.3162000000000000</xFrontSparRoot>
<FuselageFractions mapType="vector">0.1100000000000000;0.7400000000000000;0.150000000000000
    0</FuselageFractions>
<DesignMethod>inside-out</DesignMethod>
<HeadRoom>1.6499999999999999</HeadRoom>
<DesignToDivergenceMachIncrease>0.0150000000000000</DesignToDivergenceMachIncrease>
<MainWingXPosition>0.3162000000000000</MainWingXPosition>
<LowWingPosition>-0.3000000000000000</LowWingPosition>
<MinDistanceSpars>0.0500000000000000</MinDistanceSpars>
<MinAngleSpars>8.0000000000000000</MinAngleSpars>
<HorizontalTailVolumeCoefficient>1.0000000000000000</HorizontalTailVolumeCoefficient>
<HorizontalTailAspectRatio>5.0000000000000000</HorizontalTailAspectRatio>
<VerticalTailVolumeCoefficient>0.0900000000000000</VerticalTailVolumeCoefficient>
<VerticalTailAspectRatio>1.6000000000000001</VerticalTailAspectRatio>
<VTTrailingEdgeOffset>0.0650000000000000</VTTrailingEdgeOffset>
<HTTrailingEdgeOffset>0.0620000000000000</HTTrailingEdgeOffset>
<TurboFanNacelleThicknessFraction>0.1200000000000000</TurboFanNacelleThicknessFraction>
<SingleEngineSpanLocation>0.3400000000000000</SingleEngineSpanLocation>
<EngineWingXOffset>0.6300000000000000</EngineWingXOffset>
<EngineWingZOffset>0.8500000000000000</EngineWingZOffset>
<TurboPropSection>N0012</TurboPropSection>
```

```
443             <UseAuxiliarySparForFuelTank>1.0000000000000000</UseAuxiliarySparForFuelTank>
444             <UsableFuelVolume>1.0000000000000000</UsableFuelVolume>
445             <C2WeightMethod>Torenbeek</C2WeightMethod>
446             <UseClass25>0.0000000000000000</UseClass25>
447             <EnableFWE>0.0000000000000000</EnableFWE>
448             <ElevatorChordCoeff>0.2000000000000000</ElevatorChordCoeff>
449             <NumberOfCrew>7.0000000000000000</NumberOfCrew>
450             <FWEMaximumCabinDifferentialPressure>8.5999999999999996</
                    FWEMaximumCabinDifferentialPressure>
451             <MinNoseGearLoad>0.0600000000000000</MinNoseGearLoad>
452             <MaxNoseGearLoad>0.1500000000000000</MaxNoseGearLoad>
453             <MainGearStowage mapType="vector">0.3500000000000000;0.5000000000000000</MainGearStowage>
454             <NoseGearStowage mapType="vector">0.0500000000000000;0.2000000000000000</NoseGearStowage>
455             <ScrapeAngleLOF>12.0000000000000000</ScrapeAngleLOF>
456             <MainGearStowageWidth>1.0000000000000000</MainGearStowageWidth>
457             <FuelTankVolumeConstraint>8549.9504159228235949</FuelTankVolumeConstraint>
458             <MaxSpanConstraintActive>0.0000000000000000</MaxSpanConstraintActive>
459             <SpanEfficiency>0.8500000000000000</SpanEfficiency>
460             <SpanEfficiencyIncrement_LandingFlaps>0.1000000000000000</
                    SpanEfficiencyIncrement_LandingFlaps>
461             <SpanEfficiencyIncrement_TakeOffFlaps>0.0500000000000000</
                    SpanEfficiencyIncrement_TakeOffFlaps>
462             <CD0increment_LandingGear>0.0200000000000000</CD0increment_LandingGear>
463             <CD0increment_LandingFlaps>0.0650000000000000</CD0increment_LandingFlaps>
464             <CD0increment_TakeOffFlaps>0.0150000000000000</CD0increment_TakeOffFlaps>
465             <CD0increment_Compressibility>0.0005000000000000</CD0increment_Compressibility>
466             <CruiseReqMTOM>0.8500000000000000</CruiseReqMTOM>
467             <MaxContPowerSetting>0.9500000000000000</MaxContPowerSetting>
468             <MaxCLmaxLanding>4.0000000000000000</MaxCLmaxLanding>
469             <DefaultAbsoluteCeilingTurboFan>13500.0000000000000000</DefaultAbsoluteCeilingTurboFan>
470           </aircraftSettings>
471         </initiator>
472       </TUDinitiator>
473     </toolspecific>
474 </cpacs>
```

C

Initiator Optimization Results

C.1. Min MTOM - Variation in Range

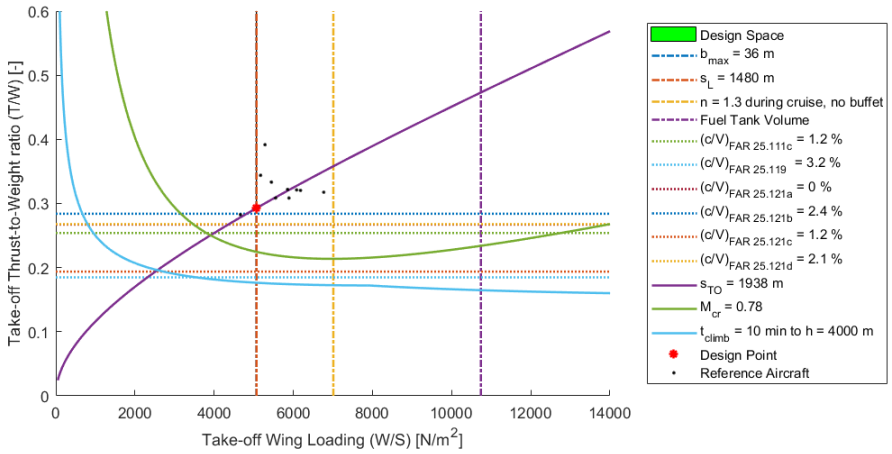


Figure C.1: Wing thrust loading diagram for a range of 3000km

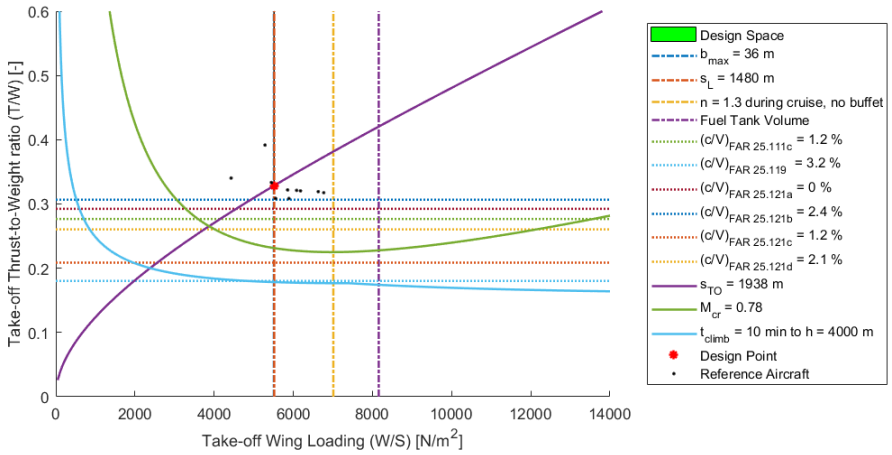
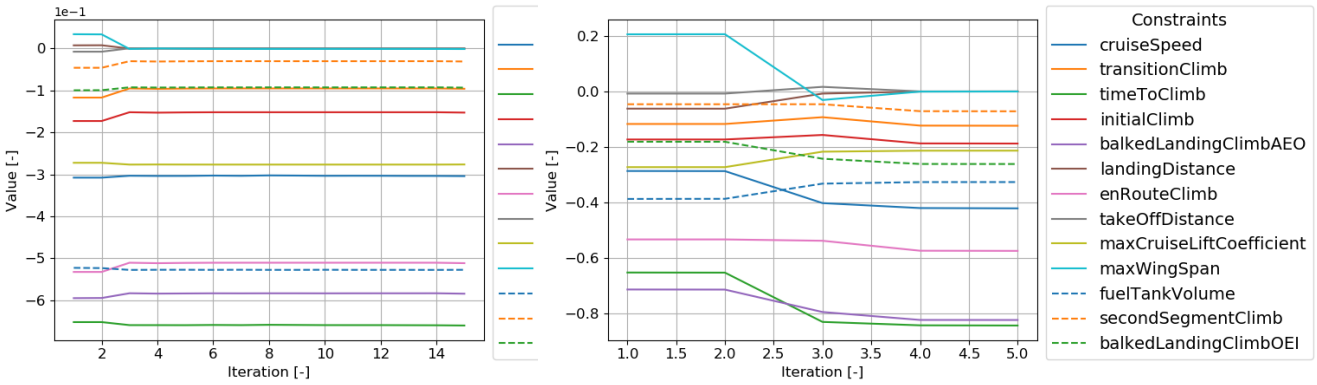


Figure C.2: Wing thrust loading diagram for a range of 5000km



(a) 3000km (b) 5000km

Figure C.3: History of the constraints for a range of 3000km and 5000km

C.2. Min MTOM - No Wingspan Constraint

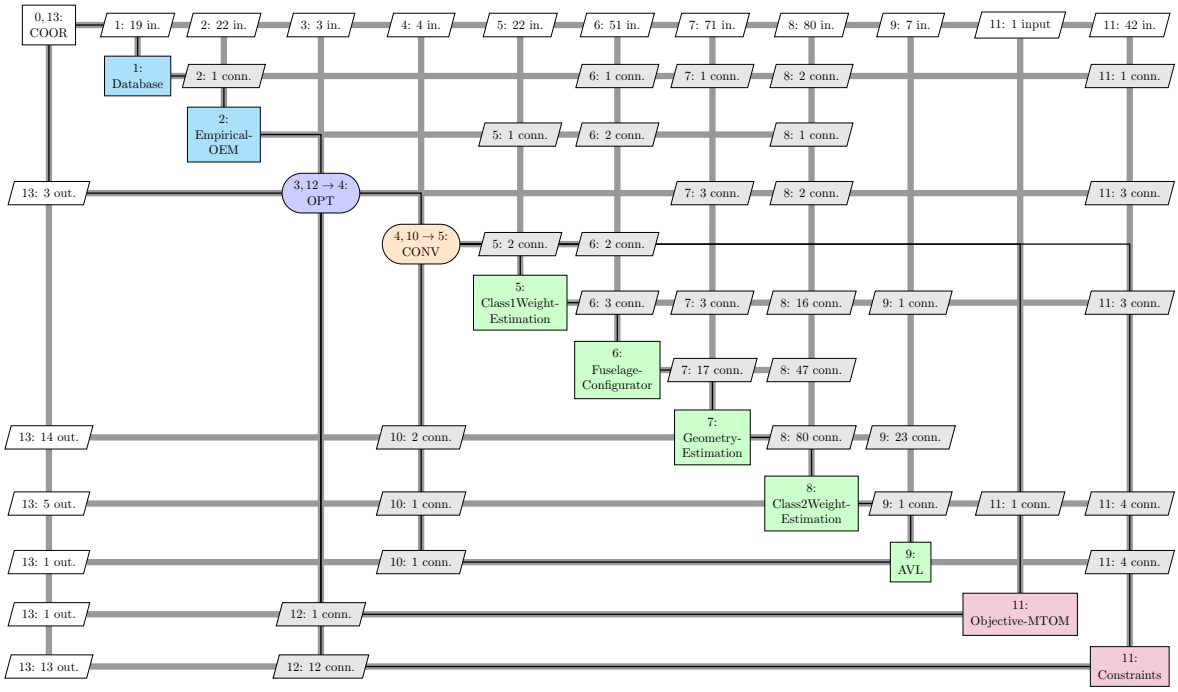
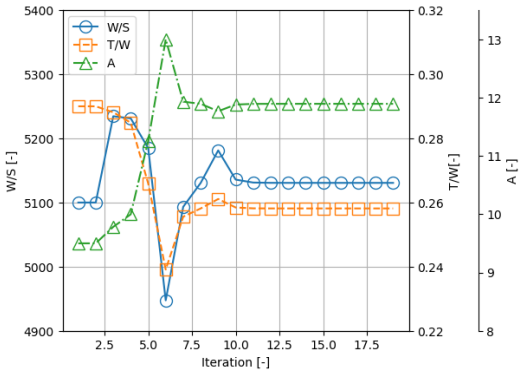
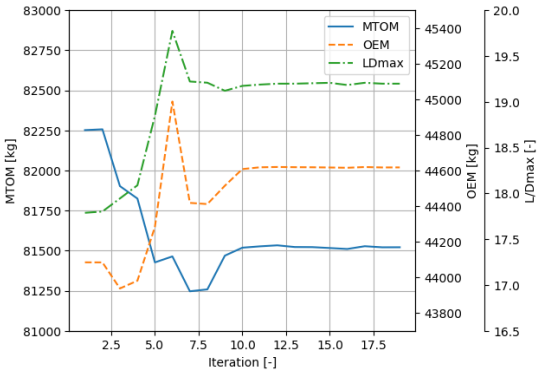


Figure C.4: XDSM for the minimization of the MTOM with no wingspan constraint

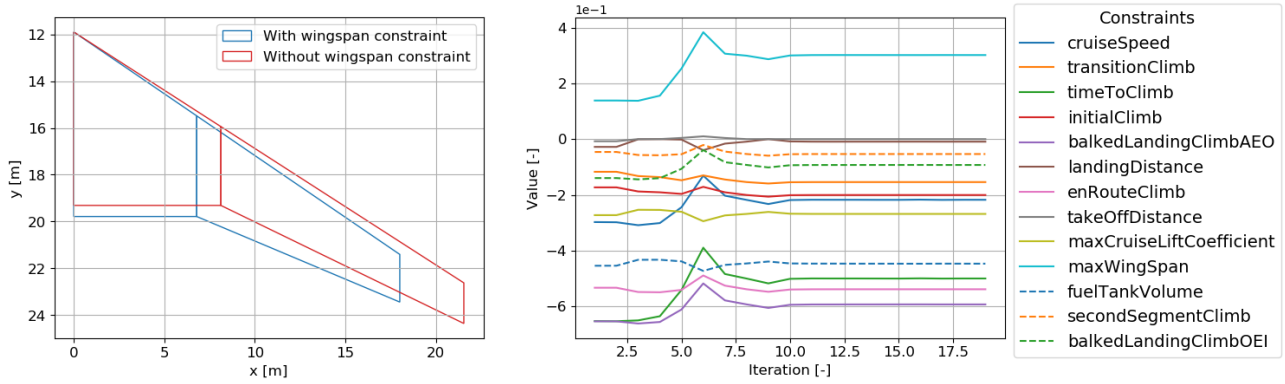


(a) History of the design variables



(b) History of the quantities of interest

Figure C.5: History of the design variables and quantities of interest for the minimization of the MTOM with no wingspan constraint



(a) Wing geometry

(b) History of the constraints

Figure C.6: Wing geometry and history of the constraint variables for the minimization of the MTOM with no wingspan constraint

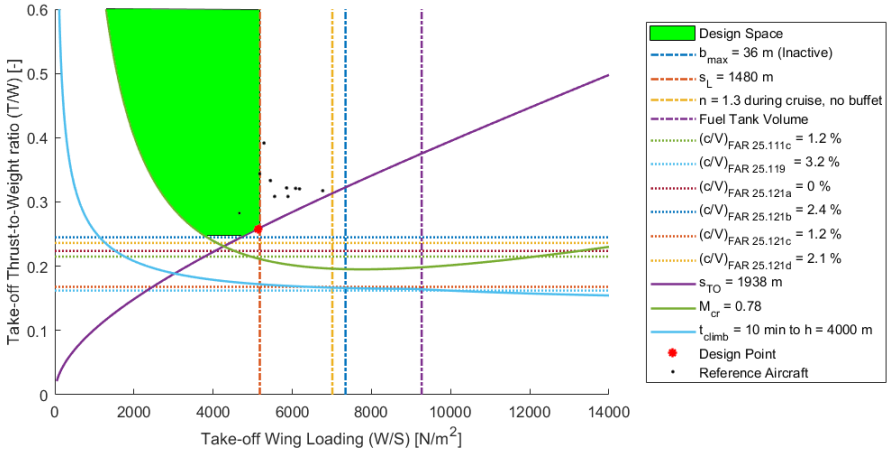


Figure C.7: Wing thrust loading diagram for the minimization of the MTOM with no winspan constraint

Table C.1: Initial and final values for the minimization of the MTOM with no wingspan constraint

	Range: 4000 km			
	Start Value	**End Value**	**Lower Bound**	**Upper Bound**
Objective				
MTOM [-] (scaled)	1.00565	0.99672	-	-
MTOM [kg] (absolute)	82252	81521	-	-
Design Variables				
A [-]	9.50	11.89	6	13
W/S [N/m^2]	5100	5130	2000	7000
T/W [-]	0.29	0.258	0	0.6
Quantities of Interest				
OEM [kg]	44084	44617	-	-
FM [kg]	17631	16368		
L/D_{max}	17.79	19.20	-	-
Constraints				
Balked Landing Climb AEO	-0.65308	-0.59251	-	0
Balked Landing Climb OEI	-0.13956	-0.092618	-	0
Cruise Speed	-0.29737	-0.21746	-	0
En Route Climb	-0.53276	-0.5382	-	0
Fuel Tank Volume	-0.4539	-0.44642	-	0
Initial Climb	-0.17294	-0.2004	-	0
Landing Distance	-0.027793	-0.0091292	-	0
Max Cruise Lift Coefficient	-0.27253	-0.26816	-	0
Second Segment Climb	-0.046017	-0.05345	-	0
Take-Off Distance	-0.0076323	4.8714e-05	-	0
Time To Climb	-0.65222	-0.49953	-	0
Transition Climb	-0.11707	-0.15401	-	0

C.3. Min FM - No WingSpan Constraint

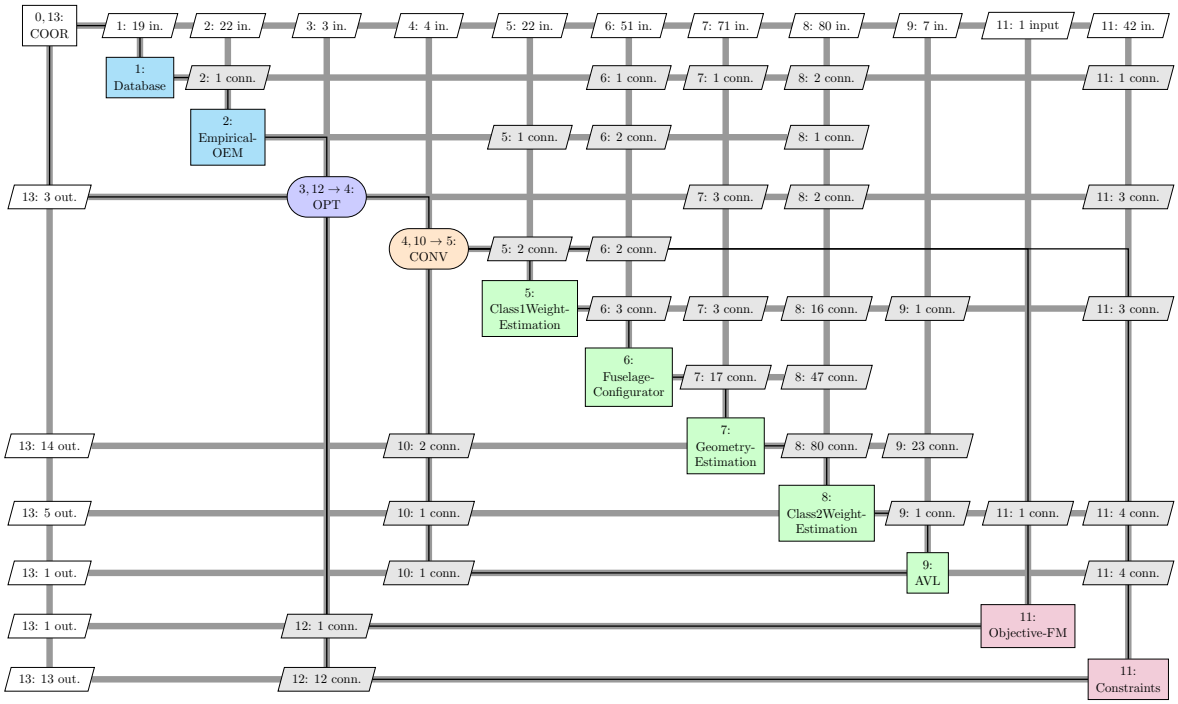
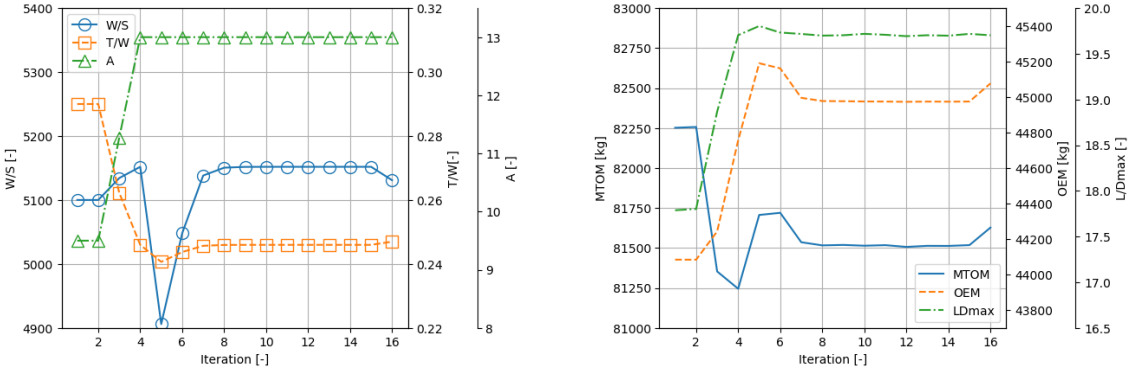


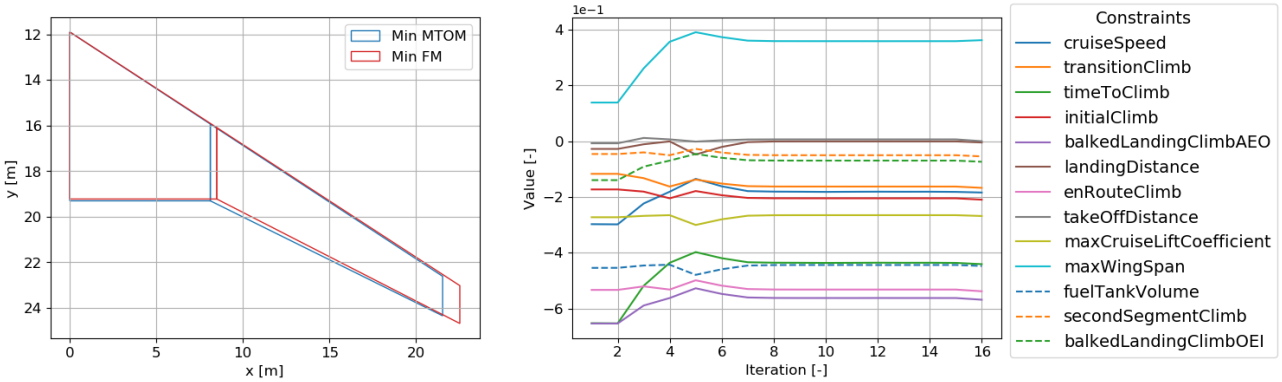
Figure C.8: XDSM for the minimization of the FM with no wingspan constraint



(a) History of the design variables

(b) History of the quantities of interest

Figure C.9: History of the design variables and quantities of interest for the minimization of the FM with no wingspan constraint



(a) Wing geometry

(b) History of the constraints

Figure C.10: Wing geometry and history of the constraint variables for the minimization of the FM with no wingspan constraint

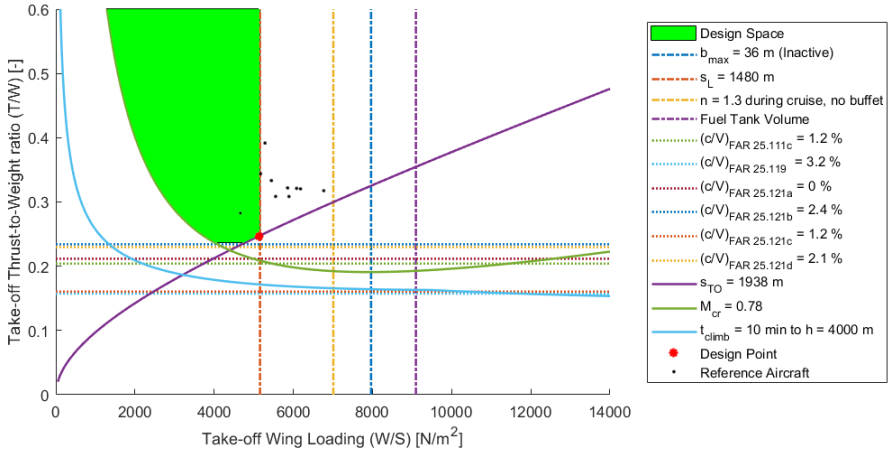


Figure C.11: Wing thrust loading diagram for the minimization of the FM with no winspan constraint

Table C.2: Initial and final values for the minimization of the FM with no wingspan constraint

	Range: 4000 km			
	Start Value	**End Value**	**Lower Bound**	**Upper Bound**
Objective				
MTOM [-] (scaled)	1.01330	0.92042	-	-
MTOM [kg] (absolute)	82252	81627	-	-
Design Variables				
A [-]	9.50	13.0	6	13
W/S [N/m^2]	5100	5131	2000	7000
T/W [-]	0.290	0.25	0	0.6
Quantities of Interest				
OEM [kg]	44084	45076	-	-
FM [kg]	17631	16015		
L/D_{max}	17.79	19.70	-	-
Constraints				
Balked Landing Climb AEO	-0.65308	-0.56795	-	0
Balked Landing Climb OEI	-0.13956	-0.073823	-	0
Cruise Speed	-0.29737	-0.184	-	0
En Route Climb	-0.53276	-0.53772	-	0
Fuel Tank Volume	-0.4539	-0.4471	-	0
Initial Climb	-0.17294	-0.20957	-	0
Landing Distance	-0.027793	-0.0047599	-	0
Max Cruise Lift Coefficient	-0.27253	-0.26813	-	0
Second Segment Climb	-0.046017	-0.05461	-	0
Take-Off Distance	-0.0076323	-0.00010649	-	0
Time To Climb	-0.65222	-0.44057	-	0
Transition Climb	-0.11707	-0.16728	-	0

Bibliography

[1] Allison, J. T., Kokkolaras, M., and Papalambros, P. Y. Optimal Partitioning and Coordination Decisions in Decomposition-Based Design Optimization. *Journal of Mechanical Design*, 131(8): pp. 081008–1 – 0810088, 2009.

[2] Belie, R. Non-technical barriers to multidisciplinary optimization in the aerospace industry. In: *9th AIAA/ISSMO Symposium of Multidisciplinary Analysis and Optimization*, 2002.

[3] Bowcutt, K. G. A perspective on the future of aerospace vehicle design. In: *12th AIAA International Space Planes and Hypersonic Systems and Technologie*, 2003.

[4] Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., and Schulz, C. Recent Advances in Graph Partitioning. In: *Algorithm Engineering: Selected Results and Surveys*, pp. 117–158. Springer International Publishing, 2016.

[5] Cacuci, D. G., Ionescu-Bujor, M., and Navon, I. M. Sensitivity and Uncertainty Analysis, Volume II Applications to Large-Scale Systems. CRC press, Boca Raton, 2005.

[6] Christofides, N. and Eilon, S. Algorithms for Large-scale Travelling Salesman Problems. *Operational Research Quarterly*, 23(4): pp. 511–518, 1972.

[7] Ciampa, P. D. and Nagel, B. Towards the 3rd generation MDO collaborative environment. In: *30th Congress of the International Council of the Aeronautical Sciences*, 2016.

[8] Ciampa, P. D. and Nagel, B. The AGILE Paradigm: the next generation of collaborative MDO. In: *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[9] Ciampa, P. D., Moerland, E., Seider, D., Baalbergen, E., Lombardi, R., and D'Ippolito, R. A Collaborative Architecture supporting AGILE Design of Complex Aeronautics Products. In: *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[10] de Vries, D. Towards the Industrialization of MDAO. Master's thesis Delft University of Technology, 2017.

[11] de Weck, O., Agte, J., Sobieszczanski-Sobieski, J., Arendsen, P., Morris, A., and Spieck, M. State-of-the-art and future trends in multidisciplinary design optimization. In: *48th Aiaa/Asme/Asce/Ahs/Asc Structures, Structural Dynamics, and Materials Conference*, 2007.

[12] Dekking, F. M., Kraaikamp, C., Lopuhaä, H. P., and Meester, L. E. A Modern Introduction to Probability and Statistics: Understanding why and how. Springer Science & Business Media, New York, 2005.

[13] Diestel, R. Graph Theory, volume 5. Springer, Berlin, heidelberg, 2017.

[14] Du, K. L. and Swamy, M. N. S. Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature. Birkhäuser, Switzerland, 2016.

[15] Elmendorp, R. Synthesis of Novel Aircraft Concepts for Future Air Travel. Master's thesis Delft University of Technology, 2014.

[16] Fiduccia, C. M. and Mattheyses, R. M. A Linear-Time Heuristic for Improving Network Partitions. In: *Papers on Twenty-five years of electronic design automation*, pp. pp. 241–247. ACM, 1988.

[17] Flager, F. and Haymaker, J. A comparison of multidisciplinary design, analysis and optimization processes in the building construction and aerospace industries. In: *24th international conference on information technology in construction*, pp. 625–630, 2007.

[18] Fortin, F. A., De Rainville, F. M., Gardner, M. A., Parizeau, M., and Gagné, C. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13: 2171–2175, 2012.

[19] Gendreau, M. and Potvin, J.-Y. Tabu search. In: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pp. 165–186. Springer Science + Business Media, LLC, 2005.

[20] Gray, J., Moore, K., and Naylor, B. OpenMDAO: An open source framework for multidisciplinary analysis and optimization. In: *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, 2010.

[21] Hager, W. W., Phan, D. T., and Zhang, H. An exact algorithm for graph partitioning. *Mathematical Programming*, 137(1-2): pp. 531–556, 2013.

[22] Hajikolaei, K. H., Cheng, G. H., and Wang, G. G. Optimization on Metamodeling-Supported Iterative Decomposition. *Journal of Mechanical Design*, 138(2): pp. 021401–1 – 021401–11, 2016.

[23] Helsgaun, K. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1): pp. 106–130, 2000.

[24] Hendrickson, B. and Leland, R. W. A Multi-Level Algorithm For Partitioning Graphs. *Supercomputing*, 95(28), 1995.

[25] Jia, H., Ding, S., Xu, X., and Nie, R. The latest research progress on spectral clustering. *Neural Computing and Applications*, 24(7-8): pp. 1477–1486, 2014.

[26] Jung, S., Park, G. B., and Choi, D. H. A Decomposition Method for Exploiting Parallel Computing Including the Determination of an Optimal Number of Subsystems. *Journal of Mechanical Design*, 135(4): pp. 041005–1 – 041005–9, 2013.

[27] Kahng, A. B., Lienig, J., Markov, I. L., and Hu, J. VLSI physical design: from graph partitioning to timing closure. Springer, Dordrecht, 2011.

[28] Karypis, G. and Kumar, V. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, 1998.

[29] Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1): pp. 359–392, 1998.

[30] Kernighan, B. W. and Lin, S. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49(2): pp. 291–307, 1970.

[31] Kramer, O. Genetic Algorithm Essentials. Springer, Cham, 2017.

[32] Kusiak, A. and Wang, J. Decomposition of the Design Process. *Journal of Mechanical Design*, 115(4): pp. 687–695, 1993.

[33] Lambe, A. B. and Martins, J. R. Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes. *Structural and Multidisciplinary Optimization*, 46(2): pp. 273–284, 2012.

[34] Lin, S. Computer Solutions of the Traveling Salesman Problem. *The Bell System Technical Journal*, 44 (10): pp. 2245–2269, 1965.

[35] Lu, Z. and Martins, J. R. R. A. Graph Partitioning-Based Coordination Methods for Large-Scale Multidisciplinary Design Optimization Problems. In: *12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSM*, 2012.

[36] Martins, J. R. R. A. and Lambe, A. B. Multidisciplinary Design Optimization: A Survey of Architectures. *AIAA Journal*, 51(9): pp. 2049–2075, 2013.

[37] McCulley, C. and Bloebaum, C. L. A genetic tool for optimal design sequencing in complex engineering systems. *Structural Optimization*, 12(2): pp. 186–201, 1996.

[38] Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19: pp. 79–102, 2016.

[39] Olver, P. J. and Shakiban, C. Applied linear algebra, volume 1. Springer, 2006.

[40] Park, H. W., Kim, M. S., and Choi, D. H. A New Decomposition Method for Parallel Processing Multi-Level Optimization. *KSME international journal*, 16(5): pp. 609–618, 2002.

[41] Potvin, J.-Y. and Rousseau, J.-M. An Exchange Heuristic for Routeing Problems with Time Windows. *Journal of the Operational Research Society*, 46(12): pp. 1433–1446, 1995.

[42] Qiu, Q., Li, B., Feng, P., and Gao, Y. Decomposition Method of Complex Optimization Model Based on Global Sensitivity Analysis. *Chinese Journal of Mechanical Engineering*, 27(4): pp. 722–729, 2014.

[43] Rogers, J. L. DEMAID/GA - An enhanced design manager's aid for intelligent decomposition (genetic algorithms). In: *6th Symposium on Multidisciplinary Analysis and Optimization*, pp. 1497–1504, 1996.

[44] Shahpar, S. Challenges to Overcome for Routine Usage of Automatic Optimisation in the Propulsion Industry. *The Aeronautical Journal*, 115(1172), 2011.

[45] Shaja, A. S. and Sudhakar, K. Optimized sequencing of analysis components in multidisciplinary systems. *Research in Engineering Design*, 21(3): pp. 173–187, 2010.

[46] Skiena, S. S. The Algorithm Design Manual. Springer-Verlag, 2nd edition, London, 2008. ISBN 978-1-84800-069-8.

[47] Sobieszczanski-Sobieski, J. and Haftka, R. T. Multidisciplinary aerospace design optimization: survey of recent developments. *Structural optimization*, 14(1): pp. 1–23, 1997.

[48] Suman, B. and Kumar, P. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10): pp. 1143–1160, 2006.

[49] van Gent, I., Ciampa, P. D., Aigner, B., Jepsen, J., La Rocca, G., and Schut, J. Knowledge architecture supporting collaborative MDO in the AGILE paradigm. In: *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[50] van Gent, I., La Rocca, G., and Hoogreef, M. F. M. CMDOWS: A Proposed New Standard To Store And Exchange MDO Systems. In: *Aerospace Europe 6th CEAS Conference*, 2017.

[51] van Gent, I., La Rocca, G., and Veldhuis, L. L. M. Composing MDAO symphonies: graph-based generation and manipulation of large multidisciplinary systems. In: *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[52] van Gent, I., Aigner, B., Beijer, B., and La Rocca, G. A Critical Look at Design Automation Solutions for Collaborative MDO in the AGILE Paradigm. In: *2018 Multidisciplinary Analysis and Optimization Conference*, 2018.

[53] Vecharynski, E., Saad, Y., and Sosonkina, M. Graph partitioning using matrix values for preconditioning symmetric positive definite systems. *SIAM Journal on Scientific Computing*, 36(1): pp. A63–A87, 2014.

[54] Von Luxburg, U. A tutorial on spectral clustering. *Statistics and Computing*, 17(4): pp. 395–416, 2007.

[55] Voudouris, C. and Tsang, E. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2): pp. 469–499, 1999.

[56] Walshaw, C. and Cross, M. Jostle: parallel multilevel graph-partitioning software–an overview. *Mesh partitioning techniques and domain decomposition techniques*, pp. pp. 27–58, 2007.

[57] Zhang, D., Song, B., Wang, P., and He, Y. Performance Evaluation of MDO Architectures within a Variable Complexity Problem. *Mathematical Problems in Engineering*, 2017.

[58] Zhang, J. Simulated Annealing: In Mathematical Global Optimization Computation, Hybrid with Local or Global Search, and Practical Applications in Crystallography and Molecular Modelling of Prion Amyloid Fibrils. In: *Simulated Annealing: Strategies, Potential uses and Advantages*, pp. 1–33. Nova Science Publishers, Inc., 2014.