Mathematical Formulations in Conceptual Design, Analysis, and Optimization

by

Johannes J. Norheim

B.S., Aeronautics and Astronautics, Massachusetts Institute of Technology (2016)
S.M., Aeronautics and Astronautics, Massachusetts Institute of Technology (2018)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Engineering Systems

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author .....
Department of Aeronautics and Astronautics
August 16, 2022

Certified by .....
Olivier L. de Weck
Apollo Program Professor of Astronautics and Engineering Systems
Thesis Supervisor

Certified by .....
Joaquim R. R. A. Martins
Professor of Aerospace Engineering, University of Michigan

Certified by .....
Bart P. G. Van Parys
Assistant Professor of Operations Research and Statistics

Accepted by .....
Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Mathematical Formulations in Conceptual Design, Analysis, and Optimization

by

Johannes J. Norheim

Submitted to the Department of Aeronautics and Astronautics
on August 16, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Engineering Systems

Abstract

Engineering design problems are often multidisciplinary in nature, giving them a natural decomposition into subproblems. The field of multidisciplinary design, analysis, and optimization (MDAO) has developed methods to integrate these sub-problems and generate standardized mathematical formulations. In the detailed design phase of engineering, multiple technical reasons beyond the nature of the discipline justify the decomposition. However, in the early phase, during conceptual design, when the models involve equations or constraints on elementary functions with highly heterogeneous structures, multiple decompositions are often possible. Furthermore, the decompositions often involve inversions of the elementary functions, leading to a restructuring of the design equations. Applying MDAO methods to specific decomposition choices can lead to mathematical formulations with fewer variables and constraints or equations, which are often desirable properties. This thesis explores how to restructure the conceptual design equations and generate different mathematical formulations based on the same underlying model. For this purpose, we develop a new graph-based mathematical formalism, which generalizes many of the existing MDAO methods through a set of atomic transformations. We find desirable problem structures by solving different combinatorial optimization problems related to the well-known feedback arc set and assignment problems. The formulations of the optimization problems seem to be exponential on average but make it possible to find optimal structures with 30 variables in less than 1 second. Finally, we apply the restructuring method to three conceptual design problems from aerospace and naval engineering domains, with up to 40 sizing relationships. In all but one case, we find feed-forward structures. Although the restructured problems have the desired properties, a new set of unexpected numerical challenges results in some cases.

Thesis Supervisor: Olivier L. de Weck
Title: Apollo Program Professor of Astronautics and Engineering Systems

Acknowledgments

The work behind this thesis would not have been possible without the help, guidance, and support of the people inside and outside MIT. First, I owe a lot to Oli, who has been my advisor during the time of the Ph.D. thesis, but has also been a lot more over the many years we have known each other since the first days of Unified almost 9 years ago. Not once have I felt him losing confidence in my work; he has been a constant source of motivation and rekindling during moments of disillusion. I also thank him for supporting the ideas that eventually developed into this thesis, especially since they were on the unbeaten path for both of us. I also owe him not to have had to worry about funding over the last years, which has been a blessing even at the price of working in parallel on an unrelated topic to this thesis. Next, to the rest of my committee, to Quim for the guiding pieces of advice along the way and for letting me play a (very) small role in your book, and to Bart, for signing on to this project with little exposure to the domain and taking a leap of faith. I wanted to have someone with the operations research point of view on optimization, and you did not hesitate to accept. I also owe a big thank you to my readers, Faez and Maha, for their input, and especially to Maha for the early conversations when I struggled to formulate the early ideas.

Many things have led to the idea in this thesis. Still, one of probably the key decisive elements was the input from Berk and Woody that launched me into the convex engineering optimization bandwagon back in 2016. Had it not been for them, it is hard to know which strange, different path I would have been on, but it's a good chance it would not have been in engineering design and optimization. Some of the first ideas relating to this thesis came during the year I worked for Valispace. I'm deeply grateful to Marco and Louise for taking me on and giving me the freedom of thought, even at the very early stages of the startup. I am also profoundly grateful to those who offered an ear when I struggled to formulate the problem that eventually became this thesis: Cameron, Thomas Coffee, and Andrew Wang.

The thesis topic was never attached to a particular funding source, which meant

I constantly had to keep tabs and make progress on other research projects and teaching. Thankfully I have been lucky to have a very supportive environment for those projects, particularly the Airbus project, where Eric, Alain and Thierry have always been supportive, and flexible when I was in crunch for the thesis work.

Thanks to all the people in 33-409: the earlier generation with Marc, Iñigo, and Markus, and the current generation: George, Skylar, Nils, Anne, Kir, and especially Elwyn and Chloe. Thank you to Maria Khotimsky for all the patience and kindness while I was a listener in your Russian classes until my last semester; your classes provided a refreshing environment where I could take a break from the research. A warm thank you to my flatmate Dani, who always encouraged me whenever I came home to our shared house late at night from MIT. Also, a big thanks to the Airbus gang (unrelated to the company), Scarlett for always going the extra mile to help with stress relief, and Berk for all the research-related conversations we had. Also thank you to my family, especially my mum, and to my host families: Jeff and Elain in the US, and Eric and Christine in France, for their support from the distance.

Finally, and most importantly, I would not have been able to make it through this without Marina's constant and ongoing support. Thank you, thank you for all the times you flew to me, especially when it was hard for me to fly to you, for keeping me mentally sane and cheered up towards the hard moments of crossing the finish line of the thesis. And thank you for all the sacrifices you have made, especially given the constraints that completing this Ph.D. put on our ability to be on the same side of the ocean. Your patience has no bounds.

Contents

1 Introduction **19**

1.1 Two interpretations of discipline 22

1.1.1 Spacecraft design problem 23

1.1.2 Spacecraft optimization problem 26

1.2 Engineering design problems 29

1.2.1 Mathematical formulations 30

1.2.2 Mathematical Structures 31

1.3 Multidisciplinary design models 34

1.4 Research Question 37

1.5 Thesis Objectives 39

1.6 Thesis outline 40

2 Multidisciplinary Formalisms and Methods **43**

2.1 Multidisciplinary problem formalisms 44

2.1.1 Aspects of a multidisciplinary problem 44

2.1.2 Graph concepts 46

2.1.3 Data graphs 49

2.1.4 Process graphs 53

2.1.5 Extended Design Structure Matrix 54

2.1.6 Formal languages 54

2.1.7 Mathematical representations 55

2.2 Optimization Architectures 56

2.2.1 Multidisciplinary Feasible 56
2.2.2 Individual Discipline Feasible 57
2.2.3 Simultaneous Analysis and Design 58
2.3 Partitioning and coordination 59
2.4 Related problems and literature gap 60

3 Graph formulations for multidisciplinary models 63
3.1 Formalization of multidisciplinary models 63
3.1.1 Multidisciplinary model 63
3.1.2 Consistency problem 64
3.1.3 Components revisited 66
3.2 Multidisciplinary formulations 67
3.2.1 Formulation graphs 67
3.2.2 Flat formulations 69
3.2.3 Flat formulation sets 70
3.2.4 Design problem 70
3.3 Reducibility of multidisciplinary sets 71
3.3.1 Reducing two components 71
3.3.2 Reducing a component with an equality constraint 72
3.3.3 Reducibility and elimination notation 72
3.3.4 Elimination order 75
3.3.5 Elimination order and ayclic graphs 75
3.4 Nested formulations 76
3.4.1 Formulation trees 76
3.4.2 Constraints on tree 77
3.4.3 Nested formulation 77
3.4.4 Nested formulation set 78
3.5 The Hierarchical Structure Matrix (HSM) 80
3.5.1 Example 80
3.5.2 Average row density 81

3.5.3 Showing inputs . . . 82
3.5.4 Constructing the HSM . . . 82

4 Equivalent formulation transformations 85

4.1 Equivalent formulations . . . 86
4.2 Equivalence preserving transformations for flat formulations . . . 86
4.2.1 Reduction of intermediary component nodes . . . 87
4.2.2 Inversion of end component nodes . . . 87
4.2.3 Inversion of reduced components . . . 88
4.2.4 Reduction of inverted component-functions . . . 88
4.3 Equivalence preserving transformations for nested formulations . . . 89
4.3.1 Merging components into partitions . . . 89
4.3.2 Merging of one component as inversion . . . 90
4.3.3 Merging of intermediary components . . . 90
4.3.4 Reduction of branches . . . 91
4.4 Reducible formulations from transformations . . . 91
4.4.1 Reducible formulations from inversion . . . 91
4.4.2 Reducible formulations from merging . . . 92
4.4.3 Elimination for affine functions . . . 93
4.5 Multidisciplinary architecture transformations . . . 94
4.5.1 Multidisciplinary feasible transformations . . . 95
4.5.2 Simultaneous analysis and design transformations . . . 95
4.5.3 Individual discipline feasible transformations . . . 95
4.6 Acyclic transformations . . . 95
4.6.1 Reducing all components . . . 96
4.6.2 Reduction based on formulation order . . . 96
4.6.3 Strongly connected components methods . . . 97

5 Optimal execution structures 99

5.1 Minimum component reduction structure . . . 100
5.1.1 Tearing . . . 100

5.1.2 Integer program formulation . . . 101
5.1.3 Examples and validation . . . 102
5.2 Extended tearing . . . 104
5.2.1 Integer program formulation . . . 104
5.2.2 Examples and validation . . . 105
5.3 Reduced subproblems . . . 106
5.3.1 Integer program formulation . . . 106
5.3.2 Examples and validation . . . 107
5.4 Optimizing structure of random bipartite graphs . . . 108
5.4.1 Minimum reduction structure results . . . 109
5.4.2 Extended tearing results . . . 110
5.4.3 Reduced subproblems results . . . 111

6 Synthetic conceptual design 113
6.1 Polynomial functions . . . 113
6.1.1 Structure . . . 114
6.1.2 Polynomial equations . . . 115
6.2 Feasibility design problem . . . 116
6.3 Optimization design problem . . . 116
6.3.1 Ground truth . . . 117
6.3.2 Fine tuning . . . 118
6.3.3 Reformulation . . . 118
6.4 Optimization problem example instance . . . 119
6.4.1 SAND formulation . . . 119
6.4.2 block-IDF and block-MDF of initial formulation . . . 120
6.4.3 Optimal structure re-formulation . . . 121
6.4.4 Discussion . . . 122
6.5 Results from random graphs . . . 123

7 Case Studies: Conceptual Design of Real Systems 125
7.1 Spacecraft conceptual design . . . 126

7.1.1 Sizing relationships and initial formulation structure 127
7.1.2 Design analysis problem 132
7.1.3 Design problem 135
7.1.4 Design optimization problem 137
7.2 High altitude balloon design 139
7.3 Sizing relationships 140
7.3.1 Design feasibility problem 143
7.3.2 Design optimization problem 146
7.4 Marine system 148
7.4.1 Initial multidisciplinary formulation 150
7.4.2 Design analysis problem 158
7.4.3 Design Optimization problem 160

8 Conclusion **163**
8.1 Summary 163
8.2 Contributions 165
8.3 Limitation and future work 167

A miniMDO code snippets **171**
A.1 Syntax for specify sizing relationships 171
A.2 Data structure for storing the formulation 172
A.3 Visualizing the hierarchical structure matrix 173
A.4 Syntax for computing with components 175
A.4.1 Component handles unit conversions 175
A.4.2 Component attributes 175
A.4.3 Evaluate components 175
A.4.4 Evaluate gradients through automatic differentiation 175
A.4.5 Component manual creation 176
A.5 Restructuring introduction 178
A.5.1 Starting model 178
A.5.2 Restructuring interface 178

A.5.3 Visualize HSM 179
A.6 Restructuring algorithms 180
A.6.1 Calculated row density 180
A.6.2 Undirected formulation 180
A.6.3 Directed formulation 181
A.6.4 Polynomial restructuring algorithms 182
A.6.5 Optimization-based algorithms (NP-complete) 185
A.7 Generate workflow 190
A.8 Execute model without running optimization 194
A.9 Optimize model 195

List of Figures

1-1 Illustration of simplified communications satellite design variables. . . 22

1-2 Simplified aerostructural problem 34

1-3 Overview of methods in multidisciplinary design 36

2-1 Phases of a multidisciplinary problem 44

2-2 Aspects of a multidisciplinary problem 45

2-3 Example of a directed graph 47

2-4 Example of a directed bipartite graph 49

2-5 Design structure matrix example. 50

2-6 N^2 Chart example. 51

2-7 Dependency graph representation taken from [1]. 51

2-8 Example of bipartite graph representation. 52

2-9 Example of a process graph. 53

2-10 Example of the XDSM of a multidisciplinary analysis, taken from [2]. 54

3-1 One generalized component-based model 67

3-2 Example of a flat formulation graph 68

3-3 Example G 73

3-4 Modified graph G 74

3-5 Tree T 76

3-6 Example of a nested formulation tree 76

3-7 Example of a nested formulation 77

3-8 Example of hierarchical structure matrix 80

3-9 Illustration of row density. The row density of this figure is 3.5. . . . 81
3-10 Hierarchical structure matrix of example from Fig. 3-7 with additional inputs 82
3-11 Tree of a more complex example with 9 components (where end components are in light gray, and partitions are in dark gray) 83
3-12 Hierarchical structure matrix of a more complex example 83

4-1 Two flat formulation graphs: G_1 on the left and G_2 on the right . . . 86
4-2 Graph of original formulation on the left and transformed on the right 87
4-3 Graph of original formulation on the left and transformed on the right 87
4-4 Degenerate graphs 89
4-5 Degenerate graphs 90
4-6 Degenerate graphs 91
4-7 Nested linear formulations 94
4-10 Example showing result of minimum size strongly connected component problem 97

5-1 Example showing result of minimum feedback problem 103
5-2 Example showing results of extended tearing 105
5-3 Example showing result of minimum size strongly connected component problem 107
5-4 HSM for different row densities for a graph with $m = 10$, $n = 12$. . . 108
5-5 Computational time required for different row densities 109
5-6 Computational time required for different problem sizes 110
5-7 Resulting size of reduced system across multiple values for $\rho, m, n_{\text{coeff}}$. 111
5-8 Computational time required for different problem sizes. 111
5-9 Distribution of size of largest subproblems 112

6-1 Two formulations of the same problem 116
6-2 SAND 120
6-3 Initial formulations 121

6-4 Optimized structure formulations 122
6-5 Distribution of largest strongly connected component 123
6-6 Convergence results, showing the number of components that converged to optimality for a fixed structure (given by a random seed). . 124

7-1 Notional representation of an imaging spacecraft. 126
7-2 Empirical models used for propulsion module of spacecraft model . . 130
7-3 XDSM of modules in spacecraft conceptual design model. The order of the modules is based on the order of given in Section 7.1.1. 133
7-4 HSM of satellite model 133
7-5 HSM of re-structured satellite model 134
7-6 HSM of satellite design model to ensure variables in design constraints are inputs 136
7-7 Artist rendition of high altitude balloon concept 139
7-8 XDSM showing couple between modules in high altitude balloon model 144
7-9 HSM of formulations based on the initial structure 144
7-10 Restructured problem. Now m_t is an input instead of v. 146
7-11 PEARL concept of operations with one AUV docking, recharging and offloading data, and undocking to continue its mission. 148
7-12 Rendering of a PEARL prototype, showing solar panels, satellite communications terminal payload box and flotation devices. Multiple elements like a damping plate and the propulsion subsystem are underwater. 149
7-13 PEARL components sized 150
7-14 XDSM showing variable coupling among PEARL modules. The order of the modules is based on the order the modules are presented in this thesis. 151
7-15 PEARL simplified geometry 151
7-16 HSM of initial multidisciplinary formulation 159
7-17 HSM of restructured problem used for optimization 160

8-1 Big picture of where the results fit within the world of design problems 165

List of Tables

1.1 Results of introductory design example 25
1.2 Results of optimization of introductory design example 28

2.1 Summary of related methods 60

6.1 SAND 120
6.2 Results of synthetic example with two different MDO strategies . . . 121
6.3 block-IDF 122
6.4 block-MDF 122
6.5 Results from optimized structure 122

7.1 Summary of density parameters for mass models 131
7.2 Summary of fixed parameters for the current satellite design example 132
7.3 Analysis inputs (including guesses on the left) and key output values on the right 134
7.4 Analysis inputs (inculding guesses on the left) and key output values on the right 135
7.5 Design results 136
7.6 Satellite optimization results 137
7.7 Spacecraft design optimization runtime with different structures . . . 138
7.8 Fixed parameters for high altitude balloon design example 143
7.9 Balloon design input and key output values for initial structure . . . 145
7.10 Balloon design input and key output values for initial structure . . . 146
7.11 Balloon design optimization, input and results for initial structure . . 147

7.12 Summary of fixed parameters for the current satellite design example. 158
7.13 Analysis results 159
7.14 PEARL design optimization results 161
7.15 Marine design optimization runtime with different structures 162

Chapter 1

Introduction

> *In the case of all things which have several parts and in which the totality is not, as it were, a mere heap, but the whole is something besides the parts ...*
>
> *Aristotle*

Engineering design problems often have a multidisciplinary nature, giving them a natural decomposition or *partition*: sub-problems are created for each discipline. An extensive set of custom methods have been developed based on a *discipline* decomposition for solving multidisciplinary problems: mainly the so-called multidisciplinary (design) analysis (MDA) and multidisciplinary (design) optimization (MDO) problems. From a mathematical perspective, MDA is an instance of a feasibility problem, and MDO, as the name gives away, is an optimization problem.

However, what makes multidisciplinary problems different from any general feasibility or optimization problem? As Allison writes in [1]: "These methods are more broadly applicable than to just systems partitioned by discipline; partitions may be along disciplinary, physical, process boundaries, or some combination thereof." However, this broader interpretation of discipline does not seem very satisfactory, as it still leaves much ambiguity. This ambiguity is a point that has received little attention in the literature: whether a problem is a multidisciplinary problem is currently considered a judgment that can be made based on whether such "natural" decomposition

exists or a "natural" definition of discipline comes to mind. An additional question emerges: what *mathematical property* is it that the multidisciplinary-specific methods take advantage of? If they take advantage of particular properties, this could make the methods applicable beyond the *conventional* definition of disciplines whenever such properties are present.

In this thesis, we claim that multidisciplinary problems exhibit two essential properties: *sparsity* and what we will call *invertibility*, a concept based on ideas borrowed from [3]. These properties are necessary (but not sufficient) conditions for the last property that multidisciplinary methods can exploit: *reducibility*. We will introduce these concepts in more detail later.

The multidisciplinary methods developed can be seen as a "preprocessing step" [1] to either improve the performance of the solution method or even to solve the problem since some of the sub-problems are sometimes black-box problems. The preprocessing step transforms the original formulation of the problem into a different one that is then given to standard solvers for the problem at hand, which is often optimization.

So far, these preprocessing steps have only considered re-formulations based on the *intuitive* definition of a discipline. However, with the mathematical definition of a discipline, there can be many different *equivalent* problems with alternative disciplines, introducing a more extensive set of re-formulations and solution methods that can be applied to a problem. This does, in fact, only happen under a set of constraining conditions, which hold for the models used in one of the critical phases of engineering design: the *conceptual design phase*. The functions used for modeling at this stage are typically lumped, first-order, and have the property that they often involve elementary functions, which have properties that a mathematical perspective on *disciplines* can exploit.

In this thesis, we explore how to generate such formulations and solve a combinatorial optimization problem to find alternative "disciplines", and apply the existing architectures to this formulation.

This thesis has multiple audiences. The main audience consists of systems engineers or design engineers that are involved in the early formulation of design problems and want to produce point designs with starting solely of from the sizing equations while providing as few assisting parameters, like guessing values for design variables, or tuning parameters for a solver. This could happen in the context of being new to a domain of modeling, or to reduce the amount of *built-in* assumptions of a model, potentially allowing for the exploration of unknown designs. These engineers also often want to tweak different known problem parameters, like costs or material properties, and change the assumptions that are known, like maximum cost or total mass, and generate new point designs quickly. A second audience consists of the existing MDO community: this thesis introduces new theoretical concepts and ideas that can be used beyond the area of application of conceptual design, which is the focus in this thesis. This thesis is also written with the more general mathematical optimization community in mind, and aims to be a bridge between the, more historically, industry motivated domain of multidisciplinary design and the more rigorous domain of mathematical optimization. Finally, this thesis introduces new formulations to problems on directed bipartite graphs. Therefore, this thesis might also be of interested to audiences working on graph based problems.

Key topics that would make this thesis easier to read and understand are graph theory, set theory, optimization theory, and research efforts in the domain of multidisciplinary design. Additionally, knowledge in the domain of aerospace and marine design will make the results from the case studies more relevant to the reader.

1.1 Two interpretations of discipline

To motivate the key idea of alternative interpretations of *discipline* we introduce two disciplinary formulations of the sames set of design equations. We then solve two conceptual design problems for each formulation and compare the results. The equations are due to de Weck and Willcox [4], and describe first-order sizing relationships for key sizing variables of a communications satellite shown in Fig. 1-1. The sizing relationships are artificial and have been significantly simplified from real relationships, with dummy parameters for illustration purposes. The simplified model is a set of four equations with seven variables.

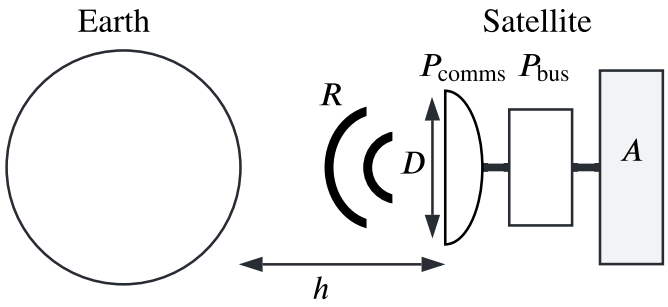


Figure 1-1: Illustration of simplified communications satellite design variables.

The first interpretation of discipline gives rise to the following formulation which we call the *initial* formulation:

$$P_{\text{comms}} = f_1(A, P_{\text{bus}}) \qquad = 483.3A - P_{\text{bus}} \tag{1.1a}$$

$$P_{\text{bus}} = f_2(P_{\text{comms}}) \qquad = 10\sqrt{P_{\text{comms}}} \tag{1.1b}$$

$$R = f_3(P_{\text{comms}}, D, h) \qquad = P_{\text{comms}}\frac{D^2}{h^2} \times 10^{18} \tag{1.1c}$$

$$C = f_4(D, A, P_{\text{bus}}) \qquad = 2,500D^2 + 12,000(A+1) + 100P_{\text{bus}} \tag{1.1d}$$

where P_{comms} is the power in Watts required for the communications system, A is the surface area in square meters of the solar panel, P_{bus} is the power in Watts consumed by the rest of the satellite, h is the altitude in meters of the orbit, D is the diameter

in meters of the satellite communications antenna, R is the data rate in megabits per second of the communication system, and C is the cost in thousands of dollars of the satellite.

The second interpretation of discipline gives rise to the following formulation: we modify Eq. (1.1a) and the Eq. (1.1c) equations to yield the new system

$$A = \tilde{f}_1(P_{\text{bus}}, P_{\text{comms}}) \qquad = (P_{\text{bus}} + P_{\text{comms}})/483.3 \tag{1.2a}$$

$$P_{\text{bus}} = f_2(P_{\text{comms}}) \qquad = 10\sqrt{P_{\text{comms}}} \tag{1.2b}$$

$$P_{\text{comms}} = \tilde{f}_3(R, D, h) \qquad = R\frac{h^2}{D^2} \times 10^{-18} \tag{1.2c}$$

$$C = f_4(D, A, P_{\text{bus}}) \qquad = 2,500D^2 + 12,000(A + 1) + 100P_{\text{bus}}. \tag{1.2d}$$

Note that f_1 and f_3 have been replace by $\tilde{f}_1$ and $\tilde{f}_3$, and the variable on the left side of the equality signs are now respectively A and P_{comms}. Formulations resulting from the two interpretations in Eq. (1.1) and Eq. (1.2) of the idea of discipline are equivalent in this case to the ideas of *exchanging inputs and outputs* given by Coffee [3], and are also related to the ideas of output sets of equations given by [5], and are a special case of the broader mathematical perspective on *discipline* that will be given later.

Based on the system of equations Eq. (1.1) or Eq. (1.2) we are interested in solving two problems: the first one consists in generating design candidates for the communication satellite based on the sizing relationships, and the second, to find the design that optimizes the cost function (also referred to as objective function) $f_4(D, A, P_{\text{bus}})$, which models the cost C of the satellite, under constraints on minimum data rate, since it is a communications satellite, and a constraint that fixes the orbit altitude, to ensure a certain amount of coverage (a property which we do not model here).

1.1.1 Spacecraft design problem

In the first problem we seek to find design candidates for the communications satellite. We refer to this as the *design problem*. These design candidates do not need to be

optimal in any way, just feasible. Mathematically this can then be formulated as a feasibility problem, i.e. finding values for the *design vector* $(P_{\text{comms}}, A, P_{\text{bus}}, h, D, R, C)$, that satisfy the governing equations.

We first solve this problem for the formulation in Eq. (1.1). Turns out if we fix values $A^{(0)}, D^{(0)}, h^{(0)}$ for A, D, h, we only need to solve a smaller system of equations

$$\begin{aligned} 483.3A^{(0)} - P_{\text{bus}} - P_{\text{comms}} &= 0 \\ 10\sqrt{P_{\text{comms}}} - P_{\text{bus}} &= 0 \end{aligned} \tag{1.3}$$

in the variables $P_{\text{comms}}, P_{\text{bus}}$. Once we have found values for $P_{\text{bus}}^{(0)}$ and $P_{\text{comms}}^{(0)}$ that satisfy Eq. (1.3) we can plug these values into f_3 and f_4 to calculate $R^{(0)}$ and $C^{(0)}$. The resulting design vector $(P_{\text{comms}}^{(0)}, A^{(0)}, P_{\text{bus}}^{(0)}, h^{(0)}, D^{(0)}, R^{(0)}, C^{(0)})$ will by construction satisfy Eq. (1.1) and therefore be a valid design candidate. Here we see one of the advantages of the multidisciplinary formulation: we got away with solving a smaller system instead of solving the full 4 by 4 system.

However, we can do better. Let's now focus on the formulation from Eq. (1.2). To find a candidate design we proceed differently: instead of fixing A, as we did previously, we fix R to $R^{(1)}$, and start our calculations with the fixed values $(R^{(1)}, D^{(1)}, h^{(1)})$. We start by calculating P_{comms} from plugging in these fixed values in $\tilde{f}_3$ and recover $P_{\text{comms}}^{(1)}$. Next we calculate $P_{\text{bus}}^{(1)}$ from plugging in the fixed values in f_2. Next we plug in $(P_{\text{bus}}^{(1)}, P_{\text{comms}}^{(1)})$ in $\tilde{f}_1$ and recover $A^{(1)}$. Finally we recover $C^{(1)}$ from plugging in values in f_4 through the same process.

The resulting design vector $(P_{\text{comms}}^{(1)}, A^{(1)}, P_{\text{bus}}^{(1)}, h^{(1)}, D^{(1)}, R^{(1)}, C^{(1)})$ will also by construction satisfy Eq. (1.1) and therefore be a valid design candidate. This means we didn't have to solve any intermediary system, and we were able to solve the entire system by simple *feed forward* of the original values $(R^{(1)}, D^{(1)}, h^{(1)})$.

Table 1.1: Design candidate based on two different methods based on different interpretations of discipline. The first method requires guessing variables.

(a) Initial formulation

Variable	Value	Unit
A	0.5	m^2
D	0.1	m
h	400	km
$P_t^{(\text{guess})}$	1	W
$P_b^{(\text{guess})}$	1	W
P_t	113.3	W
P_b	128.4	W
R	1	MB/s
C	29.4	\$1M

(b) Reparametrized formulation

Variable	Value	Unit
R	1	MB/s
D	0.1	m
h	400	km
P_t	113.1	W
P_b	128.4	W
A	0.5	m^2
C	29.4	\$1M

Table 1.1 compares the results of both methods, where the values held fixed for the second method are based on the design vector from the first method. As expected we get the same results. In the first method we had to solve a two-dimensional system of nonlinear equations(the second equation in Eq. (1.3) has a square root in it). Granted the system is linear in one of the variables, so it could have been further simplified to a nonlinear problem in one variable. To solve the system we applied Newton's method, which required providing initial guess for P_{comms} and P_{bus}. Granted, the guesses could be pretty poor: in this case initial guess with $P_{\text{comms}} = 1\,\text{W}$ and $P_{\text{bus}} = 1W$ converges after 4 iterations.

In the second method, however, we only had to solve two uni-dimensional problems that yield $\tilde{f}_1$ and $\tilde{f}_2$. Additionally these two problems turn out to be linear in the variables that we solve for, i.e. A to recover $\tilde{f}_1$ from f_1, and P_{comms} to recover $\tilde{f}_3$ from f_3, which in this case we solved symbolically for illustration purposes.

1.1.2 Spacecraft optimization problem

In the second design problem, we seek to find among all possible designs candidates, the one that minimizes the cost, while satisfying certain constraints on R, and h. We refer to this quite conventionally as the *optimization problem*. We start first with the general formulation of the optimization, which in the multidisciplinary domain goes by the name *simultaneous analysis and design (SAND)*:

$$\begin{aligned}
\underset{P_{\text{comms}},A,P_{\text{bus}},h,D,R}{\text{minimize}} \quad & f_4(D, A, P_{\text{bus}}) = 2,500D^2 + 12,000(A+1) + 100P_{\text{bus}} && \text{(1.4a)}\\
\text{subject to} \quad & P_{\text{comms}} - 483.3A + P_{\text{bus}} = 0 && \text{(1.4b)}\\
& P_{\text{bus}} - 10\sqrt{P_{\text{comms}}} = 0 && \text{(1.4c)}\\
& R - P_{\text{comms}}\frac{D^2}{h^2} \times 10^{18} = 0 && \text{(1.4d)}\\
& R \geq R^{(2)} && \text{(1.4e)}\\
& h = h^{(2)} && \text{(1.4f)}
\end{aligned}$$

where $R^{(2)}$ and $h^{(2)}$ are fixed values. From monotonicity arguments [6], we can argue that Eq. (1.4e) will be tight constraints at the optimal point

$$x^* = (P^*_{\text{comms}}, A^*, P^*_{\text{bus}}, h^*, D^*, R^*).$$

Furthermore, turns out the optimization problem can be formulated as a convex optimization problem as it can be relaxed to a geometric program [7], which has a global solution. This gives us a benchmark against which to compare other multidisciplinary methods. We now show a *multidisciplinary feasible* formulation instead (these terms

will be introduced in more detail later):

$$
\begin{aligned}
\underset{A,D,h}{\text{minimize}} \quad & 2,500D^2 + 12,000(A+1) + 100P_{\text{bus}} \\
\text{subject to} \quad & R \geq R^{(2)} \\
& h = h^{(2)} \\
\text{while solving} \quad & 483.3A - P_{\text{bus}} - P_{\text{comms}} = 0 \\
& 10\sqrt{P_{\text{comms}}} - P_{\text{bus}} = 0 \\
\text{for} \quad & P_{\text{comms}}, P_{\text{bus}} \\
\text{and} \quad & R = f_3(A, D, h) = P_{\text{comms}} \frac{D^2}{h^2} \times 10^{18}
\end{aligned}
\tag{1.5}
$$

This notation might be novel but is based on the notation from [8], and essentially it means that we have eliminate three equations and three variables, and that at every step in the optimization problem we have to solve the subproblem of *multidisciplinary analysis*, which given a value for A, solve the three equations. This can be solved as a 2 by 2 problem we saw earlier, to recover P_{bus} and P_{comms}, and then forward substitute to recover R. In exchange, the optimization problem has a smaller design space to optimize over: only 3 variables, A, D, and h.

The SAND formulation is the same for the second interpretation of discipline, as expected, however, the *multidisciplinary feasible* formulation is different, as seen in Eq. (1.6).

$$
\begin{aligned}
\underset{R,D,h}{\text{minimize}} \quad & 2,500D^2 + 12,000(A+1) + 100P_{\text{bus}} \\
\text{subject to} \quad & R \geq R^{(2)} \\
& h = h^{(2)} \\
\text{where} \quad & P_{\text{comms}} = \tilde{f}_3(R, D, h) = R \frac{h^2}{D^2} \times 10^{-18} \\
& P_{\text{bus}} = f_2(P_{\text{comms}}) = 10\sqrt{P_{\text{comms}}} \\
& A = \tilde{f}_1(P_{\text{bus}}, P_{\text{comms}}) = (P_{\text{bus}} + P_{\text{comms}})/483.3
\end{aligned}
\tag{1.6}
$$

In this case at every iteration of the multidisciplinary problem a simple feed forward substitution must be carried out to recover A and P_{bus}, and ultimately calculate the cost function $2,500D^2 + 12,000(A+1) + 100P_{\text{bus}}$ as a function of the design variables R, D, h. This means that no intermediary system of equations needs to be solved, and we have reduced the number of constraints in the system. We would expect that for general purpose optimization methods, this reduction in problem size should speed up the solution. Furthermore it also reduces the guess variables to only the design variables R, D and h.

In Table 1.2 we show the optimal design, which all three different methods find, and we also show the initial guesses required for the different methods. This solution also matches the optimal solution of the geometric program formulation of the problem, which we know is a global optimum.

Table 1.2: Optimal design candidate, and initial values used for the optimization problem.

Method 1 guess			Method 2 guess			Optimal value		
Variable	Value	Unit	Variable	Value	Unit	Variable	Value	Unit
A	0.03	m^2	R	0	MB/s	A^*	0.04	m^2
D	0.65	m	D	0	m	D^*	0.67	m
$P_t^{(\text{guess})}$	1	W	h	400	km	h^*	400	km
$P_b^{(\text{guess})}$	1	W				P_t^*	16.9	W
h	400	km				P_b^*	2.9	W
						R^*	1	MB/s
						C^*	15.3	\$1M

Performance of optimization

Formulation	Optimization iteration count	Subproblem iteration count
Alternative 1	12	27
Alternative 2	5	9

It's interesting to note that for the given parameters of this problem, the optimal

solution looks like a Cubesat given the sizing of the power system and the size of the solar panels, however the diameter of the antenna dish with a diameter of 67 cm is significantly larger than what we would expect for a Cubesat, however it could be accomplished through a meshed antenna deployment. The meshed antenna, however, would imply a modification to the model we are using, as it would be in a different category of antennas than that assumed for the current model. This shows the limitations of the simplified sizing model, however, it still makes it possible to make a ballpark estimation of the sizing of some key variables of the spacecraft, and an order of magnitude for the cost of the satellite, which might have been slightly conservative given that we would expect similarly sized spacecraft to have a cost of maximum a few million dollars. These results however, reflect the need for a refinement of the underlying model, which would take place as the mission matures.

The initial guess for the optimization of the second formulation where arbitrarily set. We use an off-the-shelf gradient based, constrained optimization method, sequential least square programming. For the first formulation it turns out that the method is very sensitive to initial values: if the initial value for A is changed from 0.03 to 0.05, a very small step, the method fails, and ends at a suboptimal value of $C = 15.7\,\text{M\$}$. We make one final comparison, in terms of iterations required, using the same optimization solver, based on the initial values shown in Table 1.2.

Because of the system of equations solved as a subproblem at every iteration of the optimization algorithm, the first interpretation has a significantly larger number of subiterations that need to be calculated. As a result, it also takes significantly longer for the first method to converge, in comparison to the second method.

1.2 Engineering design problems

Now that we have seen two instances of design problems and their formulation we go back to the more general picture and see them as instance of more general mathematical problems: feasibility(sometimes called satisfiability) problems and optimization

problems. We first introduce the notation that has become quite universal for these type of problems, and then also introduce, in the light of the examples from Section 1.1 the three key properties mentioned at the beginning of the chapter: *sparsity*, *invertibility*, and *reducibility*.

1.2.1 Mathematical formulations

The two problems introduced in Section 1.1 are instances of more general mathematical problems that have a standard formulation. We use a similar description to [6] and define the *design problem* (Papalambros and Wilde use the term general design problem) as a feasibility problem:

$$\begin{aligned} &\text{find} && x \in \mathbb{R}^n \\ &\text{subject to} && h_i(x) = 0, \quad i = 1, \ldots, m \\ &&& g_i(x) \leq 0, \quad i = 1, \ldots, p. \end{aligned} \tag{1.7}$$

where x is the *design vector* described earlier, also often referred to as the set of *design variables*, h_i are equality constraint functions, and g_i are inequality constraint functions. Often we can add the requirement that $x \geq 0$, which is sometimes written as $x \in \mathbb{R}_+$, as in engineering design, quantities like mass, power, or dimensions of geometries are positive(often even strictly positive) quantities. The design problem from Section 1.1.1 could be described through this formulation: with only equality constraints h_i and no constraints g_i, where for example

$$h_1(P_{\text{comms}}, A, P_{\text{bus}}, h, D, R, C) = 483.3A - P_{\text{bus}} - P_{\text{comms}}.$$

The *design optimization problem* has the same formulation as a standard optimization

problem:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & h_i(x) = 0, \quad i = 1, \ldots, m \\ & g_i(x) \leq 0, \quad i = 1, \ldots, p. \\ & x \in \mathbb{R}^n \end{array} \tag{1.8}$$

where f is called the objective, or cost, function; otherwise the equality constraint and inequality constraint notation is the same as for the feasibility problem. The design optimization problem from Section 1.1.2 has the cost function

$$f(A, D, P_{\text{bus}}) = 2,500D^2 + 12,000(A + 1) + 100P_{\text{bus}}$$

Finally, we present an engineering design problem we did not introduce in Section 1.1.1, a special case of the design problem: the *design analysis problem*

$$\begin{array}{ll} \text{find} & x \in \mathbb{R}^n \\ \text{subject to} & h_i(x, y) = 0, \quad i = 1, \ldots, n \end{array} \tag{1.9}$$

where $y \in \mathbb{R}^m$ is a fixed parameter. As a matter of fact, we could see the way we solved the design problem in Section 1.1.1, as two different analysis problems, where in the first problem, $y = (A, D, h)$ and $x = (P_{\text{comms}}, P_{\text{bus}}, C, R)$ in the second version $y = (R, D, h)$ and $x = (P_{\text{comms}}, P_{\text{bus}}, A, C)$.

1.2.2 Mathematical Structures

Properties of the functions encountered in the general problems described in Section 1.2.1 can, and are often exploited to speed up the solution time of any problem that can be put in such a general formulation. The most common structures encountered in the literature are *linearity*, and *convexity*, which are especially exploited in the context of optimization, where key references exist [9], [10]. In this thesis we exploit three properties, two of which are well defined properties, and which we introduce here. We will also argue later that these are the properties that multidisciplinary

problems exhibit, and exploit in particular ways.

Sparsity

The first property that will be important in this thesis is *sparsity*. Sparsity is often used as a term for matrices, where it refers to the number of non-zero entries of the matrix to be small, or to some extent, linear in the number of rows and columns (as opposed to quadratic, which is what we would expect for *dense* matrices). In this thesis we use the extended notion of sparsity that also applies to functions in the same way as Dennis et al. in [11] and in other similar literature: that the Jacobian matrix, $\nabla f(x)$ of a function f is sparse for all x, which assumes we know the structural sparsity pattern ahead of time, i.e. which entries of $\nabla f(x)$ are always zero. For the problems encountered in this thesis it will often be the case that we have access to the sparsity pattern based on an explicit version of f. In the more general case, for black box functions, one could compute the gradient through for example automatic differentiation, of all the functions at a random number, and assume that a zero close enough to machine precision corresponds to a function being independent of a specific variable. This idea can be robustified by running this test over multiple random points. Essentially what sparsity means is that each function that show up in the general problems depends only on a small set of the design variables. We will introduce a more narrow definition of sparsity in Chapter 3 that is relevant for conceptual design: a specific conceptual design relationships normally only depend on a few variables in the entire design, and the average number of variables that a relationship depends on seems to be within the bounds of three to a bit over four. This type of sparsity is different from the sparsity often encountered in large MDAO problems, where each discipline exhibits high levels of sparsity in the disciplinary models which are often based on discretizations of continuous problems which introduce very different types of sparsity. Furthermore, although the problem sizes in MDAO disciplinary models can be on the order of millions of variables, the sparsity often has homogeneous patterns(for example, a banded matrix pattern) that result from the formulation of the discretization. In conceptual design there are rarely homogeneous sparsity

patterns.

Invertibility

The second property that is important in this thesis is *invertibility* of a function, which is related to the implicit function theorem. A function f is partially invertible for some point x if f is continously differentiable, with nonzero derivative at the point x. In the multivariate version we say that a function $f(x_1, \ldots, x_n)$ is *partially invertible with respect to* x_i if holding all other variables $x_{i \neq j}$ constant means that the resulting function is invertible. When the function is invertible for all values of the variables $x_{i \neq j}$ in the domain of f we call it *globally invertible* with respect to x_i. This makes the concept a bit wordy, so normally we leave it up to the context of the text to determine whether we mean *globally* invertible or just locally invertible (for a specific set of values of $x_{i \neq j}$).

Example 1.2.1. The function $f(x_1, x_2, x_3, x_4, x_5) = x_1 x_2 + x_1 x_3 x_2^2 - x_3 x_4 + \exp(x_5 x_3)$ is (globally) invertible with respect to the variables x_1, x_4 and x_5 if the doman of the function is $x_i \neq 0$.

A sufficient condition with some caveats for a function to be *globally invertible* is that it can be manipulated algebraically so that one variable can be isolated on the left hand-side of the expression. The main caveat is that for this to be valid we might have to impose some constraints on the domain of the variables, for example the function $x_1 x_2 = 1$ is (globally) invertible with respect to x_1: $x_1 = 1/x_2$, but only if we assume that $x_2 \neq 0$.

A function can be *invertible* with respect to multiple of the variables. When a function is *globally invertible* with respect to all variables we will call it *fully invertible*. For example, linear functions with one-dimensional output are fully invertible (without any constraints on the domain).

The idea of *invertible* structure is very similar to the ideas found in the tearing literature [12], so-called acausal modeling language like Modelica [13], and other pub-

lications like [3].

1.3 Multidisciplinary design models

Multidisciplinary models have their origins in aerostructural optimization, an archetype problem in the multidisciplinary design field [14, 15, 8]. In the aerostructural problem there is a dedicated solver for structural equations, and a dedicated solver for fluid dynamics equations [16]. The structural equations depend on a subset of the variables that the fluid dynamics solver solves for, and vice-versa, the fluid dynamics equations depend on a subset of the variables that the structural equations solve for. This gives a natural view on the problem where we think of each solver as a function characterized by its input and output variables. Since the function, in reality, is actually a solver, and since what the output variable refers to matters, we use the name of *component*, based on conventions from the literature [8].

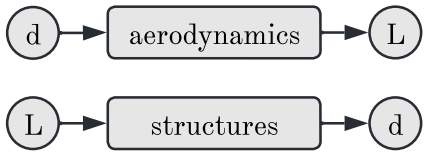


Figure 1-2: Simplified aerostructural problem, where L and d are vector quantities.

In Fig. 1-2 we visualize two simplified coupled components of an aerostructural problem: the *aerodynamics* component depends on one variable, d, the displacement of a wing along its axis during flight, that is computed by the *structures* component, and vice-versa the *structures* component depends on one variable, L, the lift distribution over the wing that is computed by the *aerodynamics* component.

The main design problem that the multidisciplinary design field has focused on is *optimization*, where the objective is often to minimize one of the variables computed as an output of one of the disciplines, and where equality and inequality constraints are often imposed on the output variables as well. Constraints on engineering variables abound, from temperature limits, to strength limits, to restrictions on the domain

of certain variables, likes masses and dimensions having to be positive [1]. This is the multidisciplinary design optimization or MDO problem. A related but simpler problem, is the problem of multidisciplinary design analysis, or MDA, analogous to the analysis problem introduced earlier, but where we have components in the model. When solving these problems, we additionally impose the requirement of *consistency* on the solution. By *consistency* we mean that if we were to evaluate each component at the solution, the output values produced are the same as those used for evaluation. In light of the example in Fig. 1-2 this would mean that a consistent value for $L^{(0)}$ and $d^{(0)}$ evaluating the structures components and the aerodynamics component produces values $d^{(1)} = d^{(0)}$ and $L^{(1)} = L^{(0)}$, respectively. This is one of the reasons $(L^{(0)}, d^{(0)})$ is often denoted a *fixed point*.

The research on multidisciplinary problems have been mainly carried out in two directions. Most generally, how to generate standardized formulations similar to the ones seen in Section 1.2.1 to take advantage of off-the-shelf solvers: this can be seen as how to generate equality constraint and inequality constraint functions, and objective functions. This stage could be thought of as filling a "parsing" or *preprocessing* role. A second research question that has received a fair amount of attention has also been on how to solve subproblems based on each component. Going back to the simplified aerostructural problem, this could be that instead of running an optimization based on the components, we create an optimization problem for each component, and then give a method for solving the original optimization problem by solving the two smaller optimization subproblems. We group these two problems under the category of *coupling*. Although we phrase the general idea of the research as *generating standardized formulations*, in fact this formulation is closer in nature to the idea of describing the computational process, or algorithm, that given a set of components, solves the original problem. In the context of optimization, these are known as *architectures* [17], and most of them are inspired by the process that engineers have found to solve coupled problems in practice, which is often based on industry experience.

Given the fair amount of research that has gone into solving problems that can be

formulated as *multidisciplinary*, i.e. as functions with inputs and outputs, another line of research has been along the line of transforming problems into a "multidisciplinary" form. This involves taking any subset of objective functions, equality constraint functions, inequality constraint functions, or even components, and partitioning them into *artificial* components, based on sparsity and decoupling criteria [18, 19, 20]. This would then make it possible to apply existing multidisciplinary methods to the artificial components. We show a summary of these two lines of research in Fig. 1-3.

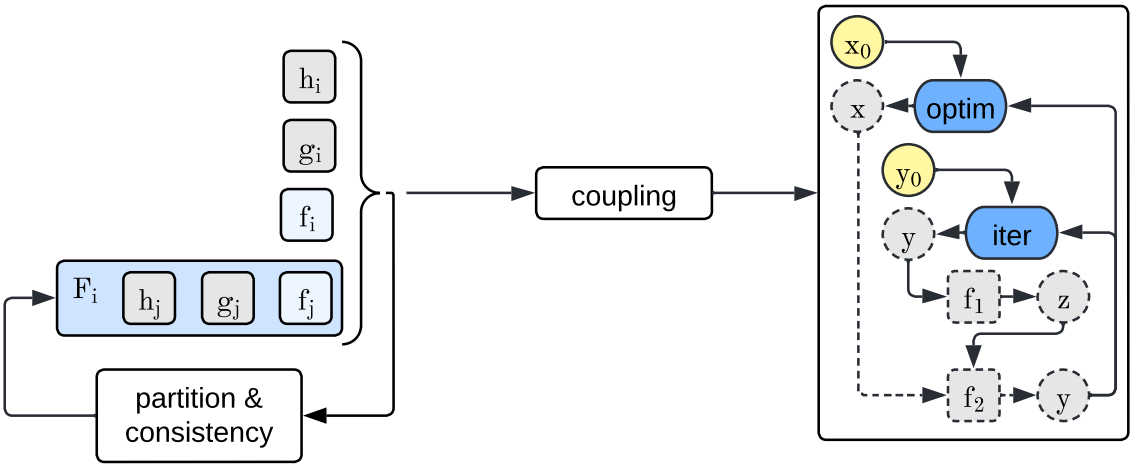


Figure 1-3: Overview of current methods. The research on solution methods based on multidisciplinary components is most often describe in an algorithmic fashion, which is illustrated through a flow chart shown on the right.

For engineering design problems, the equality constraints $h_i(x)$ represented in Fig. 1-3 which would appear in all design problems Eqs. (1.7) to (1.9) are normally given by the so-called *governing equations* from engineering modeling. In the case of optimization or design problems, the inequality constraint g_i are normally based on physical constraints related to the variables modeled. In the case of multidisciplinary problems, when the solver is a black box and where there is no access to the governing equations that it solves for. Going back to the general design problem formulations, one way to formulate components f_i is to see them as equality constraints h_i that have an implicit inverse with respect to the variables that are output by the component (this is the *residual* or *functional* perspective [8]). The partitions F_i can also be seen

as *artificial components* from the perspective of the problem, with the inputs that all the functions h_j, g_j and sub-components f_j depend on, and output a set of variables that the "component" solves for. The sparsity in h_i and g_i can be used to generate partitions F_i thereby reducing the dimensionality of the overall problem, and it is often the case that each component f_i only depends on a few of the outputs of the other components, which is then exploited by the multidisciplinary methods.

1.4 Research Question

The components f_i(black box based) and F_i (partition based) so far described, have historically had their source in the detailed engineering design phase. The computational models that are generated during this phase have medium to high fidelity, and the components have conventionally solved large systems of equations, often domain specific differential equation solvers. This includes finite element methods (FEM) often used for structural or thermal analysis, and computational fluid dynamics (CFD) used for fluid dynamics analysis. More often than not the designer does not have access to the underlying governing equations, but just the numerical solvers. The outputs of the modules are often aggregate values [21], since it is rarely the case that the information from the full solution is required by other disciplinary solvers. For example, in an aerodynamics problem, the entire state vector of the flow field is unlikely to be necessary in the structural analysis of the wing, which only requires the distribution of lift, which is aggregated based on the pressure distribution on the wing. This is one of the motivations behind the development of the multidisciplinary methods in the first place, as it significantly reduces the size of the overall problem to be solved.

More recently, a much larger set of domains have started applying the ideas behind multidisciplinary methods in the early engineering phases of conceptual design [1, 22]. At this stage, the models used are often first-order, lumped, and low-fidelity, with the goal being only to capture key sizing relationships that will have major impact on the design. As a result, components are often much "simpler", often to the point that

a component represents an elementary function, i.e. "one which can be obtained by addition, multiplication, division, and composition from the rational functions, the trigonometric functions, and their inverses, and the functions log and exp" [23]. These functions are also often partially invertible with respect to many of its variables, for example, in the introductory example in Section 1.1, the component

$$R = P_{\text{comms}} \frac{D^2}{h^2} \times 10^{18}$$

could have easily been instead

$$h = D\sqrt{\frac{P_{\text{comms}}}{R}} \times 10^{9}$$

(all variables have to be positive, so there is only one inversion possible). This creates many different ways in which the same sizing relationship can be viewed as a "discipline", in the way we saw in Section 1.1. This in turn affects the properties of the formulation of the problem that results from the application of multidisciplinary problems. This motivates the main research question of this thesis:

> *Which interpretation of "discipline" is better during conceptual design? in other words, which are advantageous ways of rearranging the original sizing relationships?*

Which leads to a further set of questions:

> *What are ways to reduce the size of the problem formulation?*
>
> *Does reducing the problem formulation actually result in any computational gains?*

This can be seen as a special case of the problem of generating partitions F_i base on one equality constrains, i.e. in the previous example starting with

$$h(R, P_{\text{comms}}, D, h) = R - P_{\text{comms}} \frac{D^2}{h^2} \times 10^{18} = 0$$

And taking the implicit inverse with respect to one of the variables. Wagner, Michelena and Papalambros have carried out research in the direction of partitioning h_i into "components" F_i [18, 19, 6], however their goal was to generate a small set of partitions, for example 4 partitions for a problem with more than 100 functions. We would require generating m partitions, where m is the number of functions h_i. Studying this special case requires a more granular perspective.

Another line of research very close in nature, but not in the domain of multidisciplinary design, is the idea of *tearing* of system of equations [24, 25, 13, 26], where the problem just described is seen as finding an *output set*, an assignment of variables with which a function h_i can be inverted, such that no set of equations is inverted with respect to the same variable. However, this research has been focused on the domain of solving systems of equations, where the number of variables and equations are the same. In the context of conceptual design, the number of variables is normally strictly larger than the number of sizing relationships, resulting in design freedom.

A last research focus that is also connected to the current problem has to do with the special case of linear systems. In this case, both structural based methods [27, 11, 28] and reduce-filling methods have been applied to reduce the number of calculations required to solve the problem. The research in this thesis focuses on the non-linear case, in which case these are not applicable. On the flip side, many of the solution methods (fixed point iteration or Newton's method) rely on linear approximations of the nonlinear problem at every iteration point of the algorithm, and often involve solving a linear system of equations. Investigating this connection was beyond the scope of this thesis.

1.5 Thesis Objectives

To answer the research question we need some key machinery that makes it possible to operate on the structure of the conceptual design functions. So far, formalisms developed for describing multidisciplinary problems have been close in nature to describing

the flow of information, and the process structure of a specific solution method. A rich set of formalisms, often based on graphs, have been developed [29, 30, 2, 31, 32], however, these have often been detached from a mathematical perspective on the functions. In [17], and more recently [8] mathematical templates are given for the architectures, however, they leave certain details implicit. The sizing relationships from conceptual design instead motivate the development of a formalism that operates on mathematical functions, as opposed to flow and process graphs. This leads to one of the first objectives of this thesis:

> *Develop a mathematically rigorous formalism for describing and manipulating multidisciplinary problems, and especially problems where the problem is composed of functions that are invertible.*

The next set of objectives use this formalism to help answer the research questions:

1. Describe a set of transformations based that can generate equivalent problems
2. Find transformations that optimally reduce the structure of the problem
3. Benchmark the resulting formulations against baseline formulations

1.6 Thesis outline

We give some further background and literature review in Chapter 2, particularly on both the multidisciplinary methods and how they can reduce or decompose the original problem. We also use this chapter to introduce the graph notation we will use in the thesis - it is mostly standard, but will be helpful for reading Chapter 3 and 4. We describe a new formalism that is mathematically rigorous in Chapter 3. We also introduce a new visualization based on this formalism, the Hierarchical Structure Matrix or HSM. In Chapter 4 we then describe manipulations on the graph structure that conserve the underlying mathematical representation of the problem. We then formulate in Chapter 5 optimization problems on the graph to yield manipulations of the original problem that help reduce the size of the problem in different ways. The

motivation is that the reduced problems will have desirable properties: less input from the engineer(like guessing variables or tuning parameters), or with improved guarantees of convergence of the numerical algorithm used to solve the problem. The optimization problems on the graphs are described as as binary (integer) programs, which we solve exactly, sometimes through the help of heuristics to speed up the process. We apply the different methods to optimize the structure of randomly generated polynomial problems that are constructed in a way to imitate conceptual design equations in Chapter 6. In Chapter 7 we describe the conceptual design model of three different engineering systems: a spacecraft, an aircraft, and a marine system, and we compare the performance of solving different design problems with different strategies, and particularly the strategies that rely on optimizing the structure. The goal of this chapter is to serve as a validation of the methodology on real world examples. We summarize the finding of the thesis and spend more time pointing out limitations, and particularly directions of future research in Chapter 8.

Chapter 2

Multidisciplinary Formalisms and Methods

Multidisciplinary methods aim to take a problem description majorly based on components, and transform it into standard formulations that can be used with general off-the-shelf solution methods, for example, in the case of optimization, a non-linear constrained optimizer like a sequential least squares solver. This preprocessing step can have multiple objectives: reducing the problem by sequencing the computation of the components or introducing auxiliary variables in the problem to allow for parallelization. In this chapter, we review three key elements: the formalisms that have been developed to describe the generation of such standard formulations, a subset of key strategies from the literature, and methods that exist to take a problem already in standard formulation, turn it into a multidisciplinary problem, which can then be further transformed based on the strategies already mentioned. Finally, we go a bit more in-depth onto related problems. We show that the formalisms existing are either too algorithmic and process-oriented in describing the sequence of computations or too high-level and general to formulate the research questions. We also compare and contrast the different research questions investigated so far and conclude that neither address the current research questions.

2.1 Multidisciplinary problem formalisms

We introduce in this section the idea of aspects of a multidisciplinary problem and then discuss which formalisms have been developed that can store the information needed for each aspect. The most common formalisms rely on graphs, so we use this section to introduce conventional concepts from graph theory that will also be useful for the remainder of the thesis. We then discuss representations based on mathematical notation and make the mapping between the mathematical notation and the graph representation. We finally discuss formal languages that have been created and shortly describe their grammar and the mapping from the formal languages to mathematical and graph-based notations.

2.1.1 Aspects of a multidisciplinary problem

We use the classification of [32] to divide a multidisciplinary problem into a *formulation phase* and an *execution phase*. The formulation phase concerns itself with all the information required up to running the problem, and the execution phase concerns itself with running computations according to the formulation. From this perspective, the formulation phase eventually describes the set of instructions that will be run during the execution phase, comparable to a programming language. For multidisciplinary problems we define three aspects: *component description*, *problem description* and *solution strategy*.

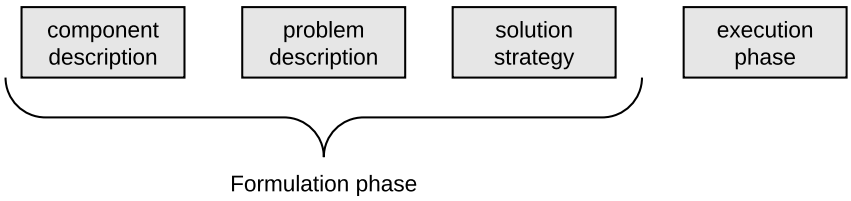
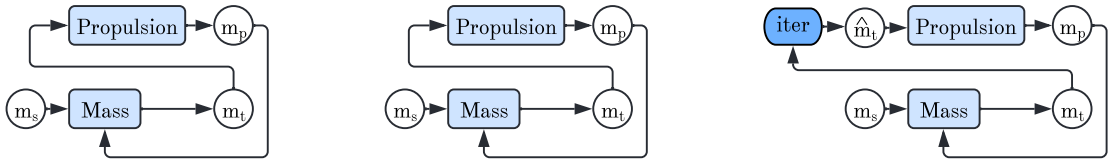


Figure 2-1: Phases of a multidisciplinary problem

Each of the aspects correspond to the *tool repository*, *MDO problem* and *MDO solution strategy* nomenclature described in [33]. Pate et al. describe similar ideas with

different names: the problem description is referred to as the *fundamental problem* and the problem solution strategy is referred to as the *specific problem*. The solution strategy results in the set of subproblems and functions for each subproblem, and how to sequence the execution of each subproblem.



(a) Component description (b) Problem description (c) Solution strategy

Figure 2-2: Aspects of a multidisciplinary problem

The *component description* aspect describes inputs and outputs, that is to say, on which data, either assumed as given or computed by other components, it depends. We use the concept of inputs and outputs to differentiate a component from a mathematical function, where the only relevant information is the range and the domain of the function, and the function is seen as a mapping from the range to the domain. For the component, not only does the range of the function matter, but also the reference to the data properties, i.e., variables, that it depends on. Fig. 2-2a shows two components: propulsion and mass, with each of their input dependencies. The propulsion module can be described in two ways. From a function perspective, it takes a real number in and outputs a real number. From a dependency perspective, it has one input m_t, which is the output of another component, the mass. The mass function has the range $\mathbb{R}_+^2$ and the domain $\mathbb{R}_+$, and inputs m_s and m_p, where m_p is also an output of the propulsion component, and output m_t.

The *problem description* describes the problem and additional meta-data specific to a design problem. For example, for an optimization problem, it assigns the role of objective, equality constraint, and inequality constraint to either variables or components. For a design problem, the role of equality constraints would be specified in the case of an isoperformance problem. When the design problem is analysis, the problem description and the component description graph are the same, as no additional

meta-data is required.

The *solution strategy* describes the dependencies between the components and connectors, and the order in which to execute the components to solve the respective design problem: analysis, design or optimization.

Formalisms represent information describing each of the different aspects in compact forms and might only apply to one or all three of the aspects. The most popular formalisms are based on a graph description.

2.1.2 Graph concepts

We introduce a conventional description of both undirected and directed graphs and bipartite graphs. These are well covered in the literature, so the definitions here are introduced mainly for nomenclature purposes for the remainder of the thesis. The following definitions are grounded in the nomenclature used in Chapter 7 of [9].

Undirected graphs

An *undirected graph* $G = (N, E)$ consists of a set N of nodes(also referred to as vertices) and a set E of edges(also referred to as arcs), where each edge consists of a subset $\{i, j\}$ of N, where i cannot be the same node as j, i.e. self-edges are not allowed. We say that the edge $\{i, j\}$ is *incident* to its *endpoints* i and j. We use the notation $|N|$ to refer to the number of nodes in the graph, and $|E|$ to refer to the number of edges.

The *degree* of a node in an undirected graph is the number of edges incident to that node.

A *walk* from node i_1 to node i_n is defined as an ordered set $W = (i_1, \ldots, i_n)$ subset of N, such that each consecutive set of nodes in the sequence, $\{i_k, i_{k+1}\},\ k = 1, \ldots, n-1$, is an edge.

A *path* is a walk W such that no element in the ordered set W is repeated.

A *simple cycle*, *cycle*, *circuit* or *loop* is also a walk W, where only one element is repeated: the first element and the last element are the same.

Directed graphs

A *directed graph* $G = (N, E)$ consists of a set N of nodes and a set E of (directed) edges, where each edge consists of an ordered pair, or tuple, (i, j) of N. Although some conventions allow for self edges for directed graphs, we use the definition where self-edges are not allowed, i.e. i cannot be the same node as j. We use the same notation for the number of nodes and edges as with undirected graphs.

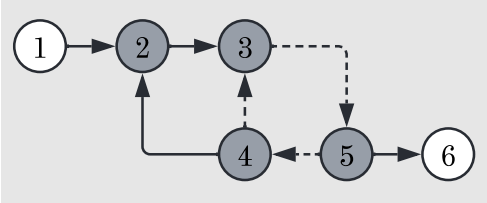


Figure 2-3: A directed graph $G = (N, E)$ with $N = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (2, 3), (3, 5), (5, 4), (5, 6), (4, 2), (4, 3)\}$.

We distinguish between two types of *endpoints*: for an edge (i, j), we say that i is the *start node* and j the *end node*. We also define the edge (i, j) to be *incoming* to node j and *outgoing* from node i. Two edges which share at least one endpoint are called *adjacent*.

A *walk* from node i_1 to node i_n is defined as an ordered set $W = (i_1, \ldots, i_n)$ subset of N, such that each consecutive set of nodes in the sequence, (i_k, i_{k+1}), $k = 1, \ldots, n-1$, is a directed edge. A *path*, and *cycle* are defined in the same way as for undirected graphs.

Example 2.1.1. For the graph shown in Fig. 2-3, the walk of nodes adjacent to dashed edges $W = \{3, 4, 5\}$ is a cycle.

A *strongly connected component* consists of a maximal subset S of N, such that for every node i in S there exists a walk $W = (i, \ldots, j)$ to every other node in the S. By

maximal we mean that S is not a subset of any other set satisfying the condition just described. A strongly connected component consist of at least one cycle.

Example 2.1.2. For the graph shown in Fig. 2-3, the set $S_1 = \{2, 3, 4, 5\}$ is a strongly connected component. The set $S_2 = 2, 3, 4$ is not a strongly connected component, because it is a subset of the set S_1, and is therefore not maximal.

A *topological order* of G is an ordered set $T = (\sigma(1), \ldots, \sigma(n))$, where σ is a bijection from the set $1, \ldots, n = |N|$ to all nodes N, such that if (i, j) is an edge, then $\sigma(i) < \sigma(j)$.

Example 2.1.3. For the graph shown in Fig. 2-3, if we remove the edge $(3, 5)$, $\sigma = \{(5, 1), (6, 2), (4, 3), (1, 4), (2, 5), (3, 5)\}$ satisfies that $\sigma(i) < \sigma(j)$ for all (i, j) in E, and gives the topological order $(5, 6, 4, 1, 2, 3)$. There are often more than one topological order of G, for example $(5, 4, 1, 6, 2, 3)$ is also a topological order of G.

A *weighted directed graph* additionally assigns weights, possibly to the nodes and the edges, through functions $v : N \to \mathbb{R}$ and $w : N \times N \to \mathbb{R}$, such that $w(i, j) = 0$ if (i, j) is not in E.

An *adjancency matrix* of a (weighted) directed graph is a matrix A such that $A_{ij} = w(\sigma(i), \sigma(j))$, where again σ is any arbitrary bijection from the set $1, \ldots, n = |N|$ to all nodes N. This representation uses the conventions that rows show the dependencies of the current node, and columns show nodes that depend on the current node. The reverse convention applies if we instead construct $A_{ji} = w(\sigma(i), \sigma(j))$. If the graph is not weighted we get a binary matrix $w(i, j) = 1$ for all (i, j) in E.

Bipartite graphs

A *bipartition* of an undirected(or directed) graph G is a partition of the nodes into two disjoint sets A and B such that for every edge $\{i, j\}$(or respectively (i, j) for a directed graph) either i is in A and j is in B or viceversa (i is in B and j is in A). A graph $G = (A, B, E)$ is called *bipartite* if a bipartition of N into A and B exists.

A *matching* of a bipartite graph is a subset M of edges, such that at most one edge in M is incident to every node. A maximal matching is a matching such that there exists no edge we can add to the matching that does not have an endpoint with an incident edge in M already. A maximum matching is a matching so that there exist no matching M of larger size. Every maximum matching is maximal, but the converse is not necessarily true. The problem of finding the a maximum matching is also often referred to as the *assignment problem*.

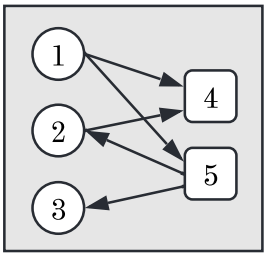


Figure 2-4: A directed bipartite graph $G = (A, B, E)$ with $A = \{1, 2, 3\}$ and $B = \{4, 5\}$ and $E = \{(1, 4), (1, 5), (2, 4), (5, 2), (5, 3)\}$.

Example 2.1.4. One possible matching for the graph in Fig. 2-4 is $M = \{(1, 4)\}$, a maximal, and maximum matching is $M = \{(1, 4), (5, 2)\}$ or $M = \{(2, 4), (5, 3)\}$.

Graphs with additional semantics have been proposed in the literature to describe all three aspects of multidisciplinary problems. Graphs describing the component and problem description are often referred to as *data* or *dataflow* graphs, and the graphs that describe the solution strategy are commonly referred to as *process*, *workflow* or *control flow* graphs [2, 34, 32].

2.1.3 Data graphs

Data graphs are used to describe dependencies between the components in a multidisciplinary problem; some types of data graphs have been extended to also describe the dependencies of non-component functions, like equality functions or objective functions in the case of optimization.

One of the first graph-based descriptions used is Steward's design structure matrix (DSM) [35]. The DSM is based on a directed graph where the nodes represent the different components, and edges represent the data dependency between components. The DSM is most commonly visualized through a binary adjacency matrix of the underlying graph. Although the DSM just captures the presence or absence of dependencies between analyses the idea can be extended to specify which data a component depends on more specifically. Minor modifications of the DSM have been made to adjust for adding weights on the edges and the nodes [20]. Although the DSM can be used to represent dependencies between computational elements, like components, most domains have used the DSM to represent dependencies between processes, tasks, people, and interfaces between physical components; Browning has provided multiple surveys on the area of application in [36] and more recently in [37]. Fig. 2-5 shows an example of a DSM of three components, where the variables that a component depends on are along the vertical axis, and the variables that a component computes, or outputs, are along the horizontal axis. Since the components often depend on variables that are not outputs of any of the components, an *artifical* component is introduced for any inputs x to the entire computation.

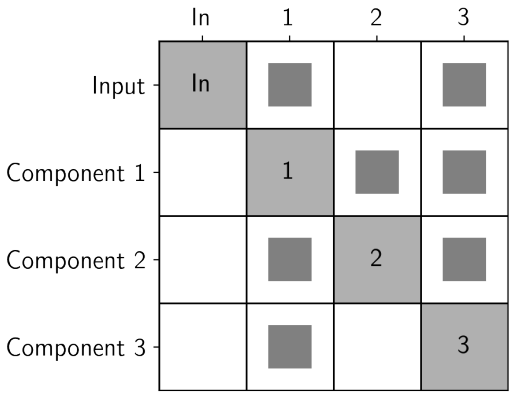


Figure 2-5: Design structure matrix example.

The N-square or N^2 Chart [38] has a similar matrix form to the DSM but might be better described as a table where the i-th row, j-th column entry instead of containing a binary entry, now describe the name of the variables calculated by component i and

required by analysis j. Fig. 2-6 shows an example of an N^2 Chart of three components in Fig. 2-5. Note that in this diagram we now see the exact dependencies between the components in terms of which exact data (for example Component 1 depends on y_{22} computed by Component 2, but no on y_{21}, on which Component 3 depends).

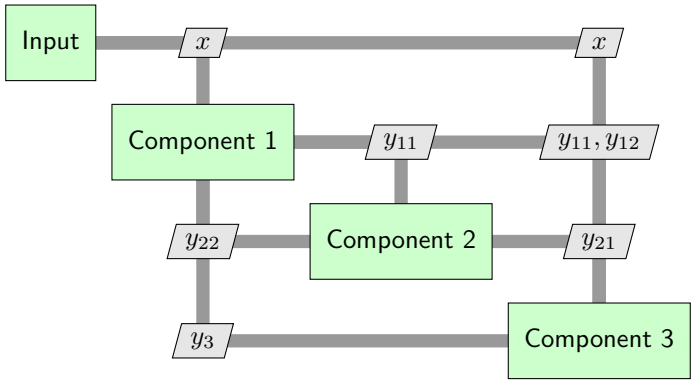


Figure 2-6: N^2 Chart example.

Both the DSM and the N^2 diagram can be used to describe component dependencies, but do not contain additional information for the problem description for any design problem other than analysis.

Allison [1] also uses a graph based on components, and where edges describe the dependencies between components.

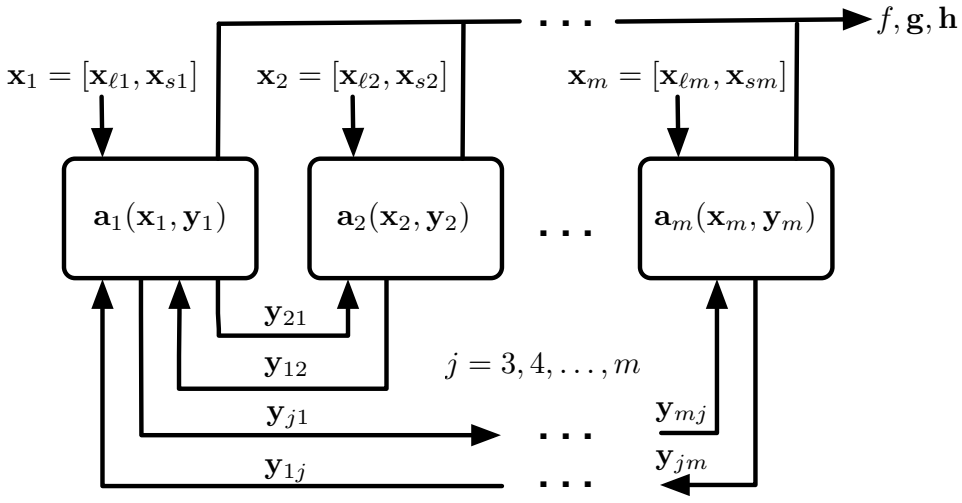


Figure 2-7: Dependency graph representation taken from [1].

An alternative to representing dependencies through edges is to use a second set of nodes for variables, giving rise to a bipartite graph. This solves a major limitation of the graph just based on component nodes: it makes it possible to describe in higher granularity the exact dependencies between components, in a similar way to what can be achieved in the N^2 Chart. Bipartite representations have been used for more recent formalisms, both for the components description and for the problem description aspects. There are both representations relying on undirected and directed versions of the bipartite graph. The undirected version simply describes that a variable appears in a certain component, or even function. The functional dependency table, or FDT, developed by Wagner in [18] captures this information. This essentially represents the structural information of the problem, i.e. the dependency of a component or function on specific variables. The undirected bipartite graph is also used in the tearing literature, and the sparse matrix literature [26, 39, 40].

Directed bipartite graphs are used in numerous recent representations. Alexandrov et al. used a directed bipartite representation composed of *data nodes* and *function nodes* for REMS[29]. The data nodes represent the dependencies, i.e. inputs and outputs of functions, in the same way as described so far. In REMS, function nodes can represent both components or objective functions and constraints. Pate et al. and van Gent et al. use similar concepts for the fundamental problem graph (FDP), which consists of variable nodes and function nodes [31, 32].

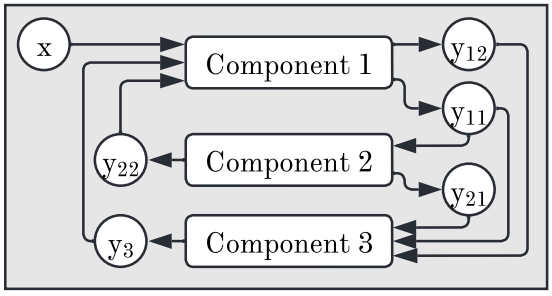


Figure 2-8: Example of bipartite graph representation.

More recent graph representations add additional semantics to the bipartite nodes:

van Gent et al. use a set of different directed graphs: the Repository Connectivity Graph(RCG), the fundamental problem graph(FDP) and the MDAO Data Graph (MDG), where additional meta-data is added to nodes [32]. The graphs have increasingly more granularity in the meta-data associated with the nodes, and specifying further nodes connected to specific strategies in which the tools can be coupled with numerical methods for different purposes like optimization. In the RCG, nodes are split into function, input/output, coupled and problematic. In the FPG additional categories associated with the nodes are design variable, objective and constraint. Finally in the MDG, a new set of component nodes are used to describe computational elements, like iteration, optimization, or residual convergence methods can be added.

2.1.4 Process graphs

Although the data graphs might sometimes be displayed with an order, as in the case of the DSM or N^2 Chart, the order displayed might not always be the order of execution of the components. To address this, the process graph shows the sequence in which the components are to be executed: if there is a directed edge from one component to the next, it means that it executes before the next. When there is a directed edge from one component to two or more, it means that all of these can be executed in parallel. The process graph is therefore a directed tree (or ordered tree, or polytree). The nodes in this graph only includes components, but also normally includes nodes for computational elements, like solvers and optimizers.

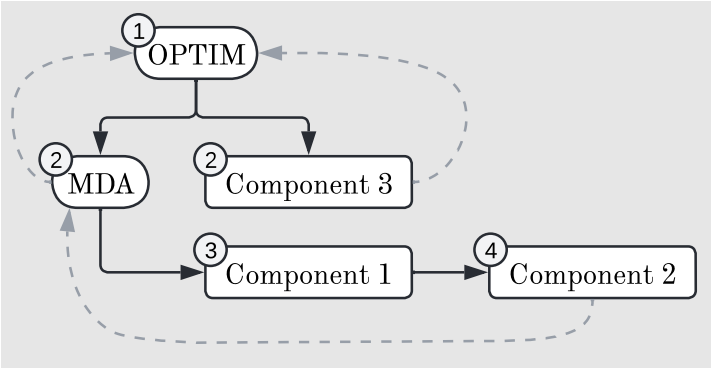


Figure 2-9: Example of a process graph.

Fig. 2-9 shows an example of a process graph, also showing additional meta-data data is often added: a number that explicitely gives the order, and backwards edges when there is an iterative algorithm being used. In this case the OPTIM node stands for an optimization algorithm, and MDA stands for a fixed point iteration algorithm, that at every iteration step executes first Component 1, and then Component 2.

2.1.5 Extended Design Structure Matrix

A more advanced representation mixes both elements from the data graph and the process graph. This representation is referred to as the extended design structure matrix or XDSM as it has multiple elements that resembles the DSM (although in fact it is closer to the N^2 Chart). It contains information on the order of execution of the different functions, and how they can be wrapped in solvers to converge analysis, or feed into the standard form formulations of optimization solvers. An example for a multidisciplinary analysis of three modules is represented in Fig. 2-10, taken from the paper that describes the notation in more detail [2].

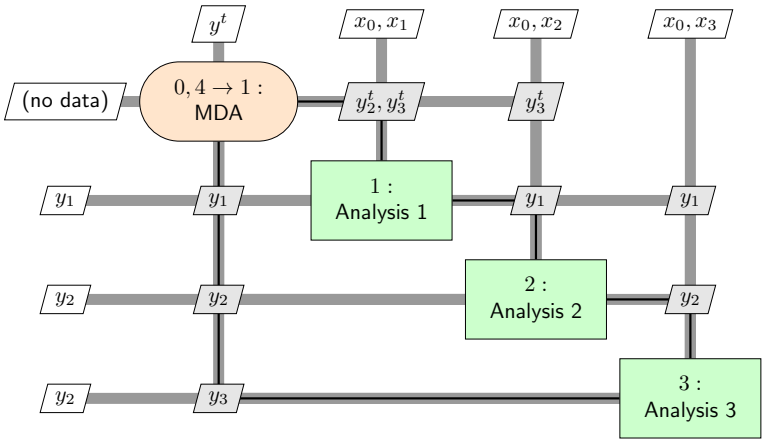


Figure 2-10: Example of the XDSM of a multidisciplinary analysis, taken from [2].

2.1.6 Formal languages

Although most representations for multidisciplinary problems are based on graphs, there have also been more linguistic approaches, that make it possible to describe

both the problem description and solution strategy. One notable one is the *partition and specification*, or Ψ, developed by Tosserams [30]. The language makes it possible to specify a set of hierarchical computations: components, and systems consisting of compositions of either components or other systems. The components consists of a set of variables and functions which can be either response functions, constraint functions, or objective functions. However, Tosserams makes an important note that the specification of the components only indicate *where the variables and functions* originate from, and that they do not necessarily imply that each component corresponds to an actual subproblem to be solved. Therefore component only provide a lower bound on the coordination techniques that can be used, however the information could still be flattened and merged into a single-level coordination technique.

Ψ and the FDP are thereby similar in the way that they both specify inputs and outputs of functions and classify them into different categories relevant for an optimization problem. However they are different since Ψ makes it possible to specify a hierarchical decomposition, which FDP does not.

Other language based representations rely on programming languages, like Python, as is the case with pyMDO(presently discontinued) [41], openMDAO [42], and GEMSEO [43].

2.1.7 Mathematical representations

Mathematical representations have mainly been used for the design optimization problem. It has typically relied on variations of the conventional template from Eq. (1.8):

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, m \\ & g_i(x) \leq 0, \quad i = 1, \ldots, p. \\ & x \in \mathbb{R}^n \end{aligned} \tag{2.1}$$

Where custom notation is used for showing hierarchical solvers, like in the case of the

multidisciplinary design feasible (MDF) architecture [8]:

$$\begin{aligned} &\text{minimize } f\left(x;\hat{u}^*\right) \\ &\text{by varying } x \\ &\text{subject to } g\left(x,\hat{u}^*\right) \leq 0 \\ &\text{while solving } \hat{r}(\hat{u};x) = 0 \\ &\text{for } \hat{u} \end{aligned} \tag{2.2}$$

The custom notation is used to give the queues about specific details that don't fall under the general mathematical notation.

2.2 Optimization Architectures

Given a set of multidisciplinary components, different template solution strategies exist that take advantage of some of the properties of multidisciplinary systems, as a function of the design problem. A significant focus has existed on the optimization problem, or MDO, and the solution strategies have come to be known as *architectures* [17]. In this section we give a brief description of some of the most popular optimization architectures, all classified as *monolithic*: *multidisciplinary design feasible (MDF)*, *individual design feasible (IDF)*, *simultaneous analysis and design (SAND)* and *collaborative optimization (CO)*. We then describe a different strategy, of *optimal partitioning and coordination* [1]. We also highlight the specific multidisciplinary structure that each takes advantage of.

2.2.1 Multidisciplinary Feasible

At every iteration of the optimization problem, the equality constraints corresponding to the multidisciplinary system are solved with the optimization variable held fixed, and then the optimization step is run with the multidisciplinary output variables fixed. The following mathematical description is from [8]:

$$
\begin{aligned}
&\text{minimize } f\left(x; \hat{u}^*\right) \\
&\text{by varying } x \\
&\text{subject to } g\left(x, \hat{u}^*\right) \leq 0 \\
&\text{while solving } \hat{r}(\hat{u}; x) = 0 \\
&\text{for } \hat{u}
\end{aligned}
\tag{2.3}
$$

Another mathematical interpretation of MDF is given through the idea of elimination: we have a function $\phi : \mathbb{R}^k \to \mathbb{R}^n$, which we will call the multidisciplinary analysis of the problem, such that $y = \phi(x)$, where x would correspond to the input variables of the problem, and $y = (x, \hat{u})$ and the optimization problem simplifies to:

$$
\begin{aligned}
&\text{minimize } f\left(\phi(z)\right) \\
&\text{subject to } g\left(\phi(z)\right) \leq 0
\end{aligned}
\tag{2.4}
$$

Note that in this template, the details of calculating the multidisciplinary analysis are left as a design decision. This problem itself could be solved as a fully coupled system that does not take advantage of any of the partition specific solvers, or could rely on solvers that handle specific subpartitions.

2.2.2 Individual Discipline Feasible

This strategy solves over the full design vector $y = (x, \hat{u})$, but uses the dedicated solvers for each partition to solve the set of equations that correspond to a partition first, with the assumption that a partition does not depend on a subset of the variables y_i. Each dedicated solver then calculates y_i, and uses the fixed value in an auxiliary equality constraints.

$$
\begin{aligned}
&\text{minimize } f(x;\hat{u}) \\
&\text{by varying } x, \hat{u}^t \\
&\text{subject to } g(x;\hat{u}) \leq 0 \\
&\qquad\qquad h_i^c = \hat{u}_i^t - \hat{u}_i = 0 \quad i = 1,\ldots,m \\
&\text{while solving } r_i\left(\hat{u}_i; x, \hat{u}_{j\neq i}^t\right) = 0 \quad i = 1,\ldots,m \\
&\qquad\text{for } \hat{u}
\end{aligned}
\tag{2.5}
$$

Another mathematical interpretation of IDF is again given through the idea of elimination: however here we have multiple functions $\phi_i : \mathbb{R}^n \to \mathbb{R}_i^n$, such that $y_i = \phi_i(y)$; we also assume very importantly that ϕ_i is independent from y_i. We then can rewrite the problem as:

$$
\begin{aligned}
&\text{minimize } f(y) \\
&\text{subject to } g(z) \leq 0 \\
&\qquad\qquad \phi_i(y) - y_i = 0
\end{aligned}
\tag{2.6}
$$

Again, for IDF, the underlying assumption is that a dedicated solver for each partition exists, which assumes all variables fixed, and only solves certain variables specific to the partition.

2.2.3 Simultaneous Analysis and Design

This strategy solves over the full design vector y and is simply the general optimization problem, where we do not use the custom solvers for any of the parts of the problem.

$$
\begin{aligned}
&\text{minimize } f(y) \\
&\text{subject to } g(y) \leq 0 \\
&\qquad\qquad h(y) = 0
\end{aligned}
\tag{2.7}
$$

2.3 Partitioning and coordination

In parallel to the research on solution strategies for multidisciplinary problems, a second line of research has been along the line of generating *artificial* components, or subproblems, based on partitions of the original problem. These have mostly been applied in the context of optimization problems.

Wagner used the incidence structure of equality and inequality constraints to partition a problem into decoupled partitions. Each partition is dependent on a set of *local* variables that do not appear in any other partition, and shared variables that appear in two or more of the partitions [18]. A heuristic method was developed. A method based on an integer formulation was developed by Krishnamachari and Papalambros [44, 45]. Michelena and Papalambros built on this idea further to develop a method that would generate k partitions based on k-cut graph algorithm [19]. In both of these methods the partitions do not depend on each other, and only require information from the system level problem, based on the shared variables.

In [1], Allison focused instead on partitioning on components, as opposed to equality and inequality constraints. He developed a method to find partitions that allow for coupling between them; this depended on having a method for coordinating the different subproblems. The augmented Lagrangian coordination architecture was used [46]. This method uses penalty functions to achieve consistency between the problems solved in the partitions and the master problem. Allison used a genetic algorithm to find combined optimal partitions and sequencing of the components inside each partition.

More recently, Lu and Martins also described a strategy for partitioning components into partitions, and optimizing the sequencing of the components inside each partition. The method allowed for a multi-level partitioning, where a partition could be further decomposed into even smaller partitions. In this work the partitioning was done based on graph partitioning algorithms, with the goal to minimize the *coupling* between the partitions, through a surrogate function that tried to estimate the coupling.

2.4 Related problems and literature gap

So far all the formalisms discussed have either have a high focus on describing the *algorithmic* steps that take a set of components and how to integrate them with different solvers. This is useful from an implementation point of view, but makes it a lot harder to operate on the functions that are embedded in each component. For example there is no way to describe the inverse of a function, other than to couple a component with a solver. This motivates the need for a more mathematical formalism that we can manipulate.

In terms of methods, Table 2.1 gives an overview over some of the methods discussed already, and an additional set of methods that are related in nature to the research question of this thesis.

Table 2.1: Summary of related methods

	focus	application area	technical method
Steward 1965 [47]	reordering	$f(x)$	Heuristic
Grotschel 1984 [39]	reordering	$h(x) = Ax + b$	Linear Programming
Fletcher 1993 [40]	reordering	$h(x) = Ax + b$	Heuristic
Wagner 1993 [18]	partition	$h(x), g(x)$	Heuristic
Dennis [11]	inversion of jacobian	$h(x)$	Dulmage-Mendelsohn
Krishnamachari 1997 [45]	partition	$h(x), g(x)$	Integer Programming
Allison 2008 [1]	partition	$f(x)$	Genetic Algorithm
Coffee [3]	inversion	$h(x)$	Dulmage-Mendelsohn
Lu 2012 [20]	partition	$f(x)$	Graph Partitioning
Baharev 2016 [26]	inversion and reodering	$h(x)$	Heuristic and Integer Programming
Gallard 2018 [34]	reordering	$f(x)$	condensation
Van Gent 2019 [32]	reordering	$f(x)$	condensation

Finding inversions can be seen as a special case of the partitioning discussed so far, when the partition includes one component only. However, instead of turning every partition into a subproblem, we have an explicit form for the inverted function. This is similar to the concept of tearing developed by [13] and embedded in the solver behind the Modelica language [24] - which is slightly different from the tearing of Steward developed in [47]. However, in both cases, the application is solving systems of equations $h(x) = 0$, where the number of equations, or dimensionality of h and the dimensionality of x are the same. In this thesis we are interested in the rearrangements that can be done when the number of variables is larger than the number of equations. This is shortly mentioned by [12], but not further developed.

Chapter 3

Graph formulations for multidisciplinary models

3.1 Formalization of multidisciplinary models

In this chapter we develop a new formalization of multidisciplinary models that make the manipulation and restructuring of the underlying formulation much easier to describe through the use of graphs. Although, as pointed out in Chapter 2, multiple graph-based formalizations already exist in the literature, the graph introduced in this chapter is deeply tied to the general mathematical representations for feasibility and optimization problems discussed in Section 1.1.1. We start by first giving a definition of a multidisciplinary model based on properties of the function in this general representation.

3.1.1 Multidisciplinary model

In this thesis we define a multidisciplinary model as a set of *component-functions* $f_i : \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots, m,\ m \leq n$ and *equation-functions* $h_i(x) : \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots, p,\ p \leq n$. We refer to n as the dimension of the model. Component-functions have the defining property that f_i is independent of x_i: $\partial f_i / \partial x_i = 0$ for all x. Note

that the dimensionality of the range of all f_i and h_i is the same: n.

Example 3.1.1. We take a multidisciplinary model of dimension $n = 3$ consisting of one component-function $f_1(x_1, x_2, x_3) = x_2 x_3$, and one equation-function $h_1(x_1, x_2, x_3) = x_1 x_3 - 1$. We can verify that f_1 is a component function: it is independent of x_1.

3.1.2 Consistency problem

We next formalize the multidisciplinary *consistency problem*: find a vector $x = (x_1, \ldots, x_m, \ldots, x_n)$ element of the *consistency set*

$$\{x \mid f_i(x) = x_i,\ i = 1, \ldots, m,\ h_i(x) = 0,\ i = 1, \ldots, p\ \}.$$

We will also use the notation

$$\begin{array}{lll} \text{find} & x & \\ \text{subject to} & f_i(x) = x_i, & i = 1, \ldots, m \\ & h_i(x) = 0, & i = 1, \ldots, p. \end{array} \tag{3.1}$$

Note that this is a special case of the *design problem* introduced earlier, but with a specific structure. We call any element x of the *consistency set*, or equivalently, solution to Eq. (3.1) a *consistent design*. We refer to both the equations $f_i(x) = x_i$ and $h_i(x) = 0$ as equality constraints. We call x_i the *output* of component-function f_i, since when x is consistent we have that $f_i(x) = x_i$.

Example 3.1.2. The *consistency set* of the multidisciplinary model given in Example 3.1.1 is

$$\{(x_1, x_2, x_3) \mid x_2 x_3 = x_1,\ x_1 x_3 - 1 = 0\}.$$

Which can be simplified to the triplet $(x_1, x_1^2, 1/x_1)$ for $x_1 \neq 0$. A solution to the *consistency problem*, i.e. a *consistent design*, would be $x = (2, 4, 0.5)$. Note that an infinite set of solutions exist. Note that x_1 is called the *output* of f_1.

We also look at a special case:

Example 3.1.3. Consistency of an arbitrary one component problem, where $f : \mathbb{R}^n \to \mathbb{R}$ is given by

$$\begin{aligned} &\text{find} && x \\ &\text{subject to} && f(x) = x_1. \end{aligned}$$

Since by definition f is independent of x_1, finding a consistent x is trivial: we pick values $x_2^{(0)}, \ldots, x_n^{(0)}$ for variables $x_2, \ldots, x_n$ then calculate $x_1^{(0)} = f(0, x_2^{(0)}, \ldots, x_n^{(0)})$ (we could also have picked any other arbitrary number instead of 0, since f is independent of x_1). We then construct a consistent x: $(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})$.

This second example justifies calling x_1 the *output variable* as the value $x_1^{(0)}$ of x_1 is computed with the numerical values of all variables other than x_1 and without the need of any numerical method to solve for x_1. This is analogous to the idea of a component representing a model that can be executed to produce values for its output variables. Now that we have a formal definition we revisit these informal descriptions of component-based and equation-based models to highlight differences and similarities.

3.1.3 Components revisited

With this formal definition, a component-function corresponds almost one-to-one to the idea of components introduced in Chapter 1. However, two differences need to be clarified, one with respect to the dimensionality and number of outputs of a component, and the second with respect to the connection between the function that models the component and the component-function.

The first difference is that the informal description of components allowed for vector functions, where the range is n-dimensional, whereas the formal definition introduce in Section 3.1.1 only allows scalar functions, i.e. with a uni-dimensional range. This will simplify the mathematical notation of the general cases without sacrificing loss of generality, as we can easily transform from the informal description to the more formal one. We illustrate the process through an example. Given a component described originally by a vector function $f : \mathbb{R}^n \to \mathbb{R}^2$ we can instead represent it without loss of generality by two functions $f_1 : \mathbb{R}^n \to \mathbb{R},\ f_2 : \mathbb{R}^n \to \mathbb{R}$, where

$$f_1(x) = (1, 0) \cdot f(x),\ f_2(x) = (0, 1) \cdot f(x).$$

This case also covers components with multiple scalar outputs, or a combination of scalars and vector outputs. Not bending our formal definition to enable a one-to-one mapping to the informal description of components might seem like a shortcoming, but we believe that it will significantly simplify notation. However it does mean that multiple component-functions can map to the same "component" from the engineering perspective.

The second difference is between the function modeling the component and the function-component. The function modeling an arbitrary component f with input variables $x_2, \ldots, x_n$ and output x_1 in our informal terminology, is depicted in Fig. 3-1.

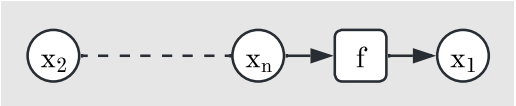


Figure 3-1: One generalized component-based model

Although $f : \mathbb{R}^{n-1} \to \mathbb{R}$ in the informal description(there are $n-1$ variables in the set $\{x_2, \ldots, x_n\}$), in the formal description we would use $\hat{f} : \mathbb{R}^n \to \mathbb{R}$, where $\hat{f}(x_1, \ldots, x_n) = f(x_2, \ldots, x_n)$. We can now verify that $\hat{f}$ is a component function: $\hat{f}$ is independent of x_1 and the range has dimensionality n. Although $\partial f/\partial x_1 = 0$ is also independent of x_1, the dimensionality of the range of f is $n-1$.

3.2 Multidisciplinary formulations

3.2.1 Formulation graphs

We introduce in this section the new concept of a *formulation graph*. For now we will call the formulation graph flat, in contrast with a *nested formulation graph* which will be covered in Section 3.4.1. A flat formulation graph is a *directed bipartite graph* $G = (V, F, E)$ consisting of two sets of nodes: V of n *variable nodes*, or simply variables, and F of m *component nodes*, or simply components, and a set E of directed edges. Note that the term *variable* by itself could have other meanings, but when it is clear from context that we refer to variables nodes of a formulation graph we stick to this simpler term. The same idea goes for the term component. The set $E = E_{in} \cup E_{out}$ can be further subdivided into two mutually exclusive sets: E_{in} of *input edges* and E_{out} of *output edges*. Each edge consists of an ordered pair or tuple of nodes (i, j), for the set E_{in}, $i \in V$ and $j \in F$, and for the set E_{out}, $i \in F$ and $j \in V$. A flat formulation graph needs to satisfy that for each $j \in V$, there is only one $i \in F$ such that (i, j) in E_{out}, and for every for each $i \in F$, there is at most one $j \in V$ such that (i, j) in E_{out}. The first constraint encodes the idea that a variable can only be the output of one component, and the second constraint that each component node

can have at most one output variable.

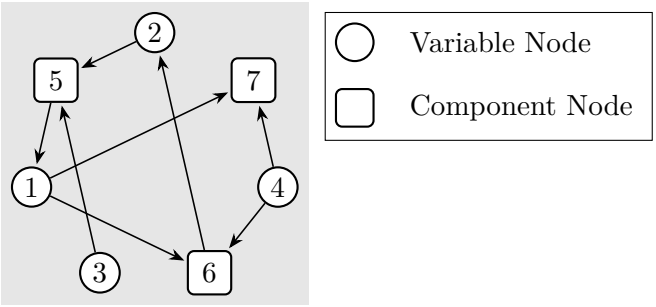


Figure 3-2: A *flat formulation graph* $G = (V, F, E_{in} \cup E_{out})$ with $V = \{1, 2, 3, 4\}, F = \{5, 6, 7\}$, $E_{in} = \{(2, 5), (3, 5), (1, 6), (4, 6), (1, 7), (4, 7)\}$ and $E_{out} = \{(5, 1), (6, 2)\}$.

For bookkeeping purposes we further subdivide $V = V_0 \cup V_1 \cup V_2$ into three mutually exclusive subsets V_0, V_1, V_2: respectively *input*, *intermediary* and *output* variable nodes. V_0 consists of the set of variables nodes $i \in V$ for which there is no edge $(j, i) \in E_{out}$ for all $j \in F$. V_2 consists of the set of variables $i \in V$ for which there is no edge $(i, j) \in E_{in}$, and finally $V_1 = V \setminus V_0 \setminus V_2$. Similarly, $F = F_0 \cup F_1$ where F_0, F_1 are respectively *intermediary* and *end* component nodes. F_1 consists of nodes $i \in F$ for which there is no edge $(i, j) \in E_{out}$ for all $i \in V$, and $F_0 = F \setminus F_1$ (the $\setminus$ is the *set difference* operator).

Example 3.2.1. For the graph shown in Fig. 3-2, $V_0 = \{3, 4\}, V_1 = \{1, 2\}, V_2 = \varnothing$, and $F_0 = \{5, 6\}, F_1 = \{7\}$.

For any component node $j \in F$ we define the set of *input variables* $\mathrm{I}(j)$ as the set of variable nodes $i \in V$ such that $(i, j) \in E_{in}$. Similarly the *output variable* or simply *output* $\mathrm{O}(j)$ of a component node $j \in F$ is the variable node $i \in V$ such that $(j, i) \in E_{out}$.

Example 3.2.2. For the graph shown in Fig. 3-2, $\mathrm{I}(5) = \{2, 3\}$ and $\mathrm{O}(5) = \{1\}$.

3.2.2 Flat formulations

A *flat formulation* will consist of a *flat formulation graph* $G = (V, F, E)$ with additional information consisting of a bijective mapping $\varphi : F \to \{1, \ldots, m\}$ which maps each component node to an index, a bijective mapping $\upsilon : V \to \{1, \ldots, n\}$ which maps each variable node to an index, and functions

$$f_j : \mathbb{R}^n \to \mathbb{R}, \quad j = \varphi(i), \quad n = |V|, \quad i \in F$$

Each function f_j for $j \in F$ is dependent on x_i, where $i = \upsilon(k),\ k \in \mathrm{I}(j)$, i.e. for each input variable node. For all $i \in F_0,\ f_i$ is a component-function, i.e. it satisfies the conditions of Section 3.1.1: *f_i is independent of, x_i* where $i = \upsilon(k)$ and $k = \mathrm{O}(j)$.

Example 3.2.3. A flat formulation is given by G from Fig. 3-2, φ, υ and f_1, f_2, f_3. We set $\varphi(5) = 1,\ \varphi(6) = 2,\ \varphi(7) = 3$ and set $\upsilon(1) = 1, \upsilon(2) = 2, \upsilon(3) = 3, \upsilon(4) = 4$ and define arbitrarily three functions that depend on these variables, $f_i : \mathbb{R}^4 \to \mathbb{R} \quad \forall i = \{1, 2, 3\}$:

$$f_1(x) = x_2 + x_3$$
$$f_2(x) = a_1 x_1 + a_2 x_4$$
$$f_3(x) = a_3 x_1 - a_4 x_4$$

where a_1, a_2, a_3, a_4 are fixed parameters. f_1 and f_2 are component functions, whereas f_3 is an equation function. We verify that f_1 is dependent on the variables whose indices are associated with the input nodes of component 5: {2, 3}. Similarly we can confirm that $\{1, 4\}$, the indices of the variables in f_2 and f_3 are the indices associated with the inputs to components 6 and 7. Finally we confirm that f_1 is independent of x_1 and f_2 is independent of x_2.

3.2.3 Flat formulation sets

Flat formulations can describe different sets depending on roles that we can assign each component. Without any roles it describes the consistency set

$$S = \left\{ x \in \mathbb{R}^n \mid \begin{array}{lll} f_j(x) = x_k, & k = v_{out}(i),\ j = \varphi(i), & \forall i \in F_0 \\ f_j(x) = 0, & j = \varphi(i), & \forall i \in F_1 \end{array} \right\}.$$

Where $v_{out}(i) = v(\mathrm{O}(i))$, i.e. the index of the output node of component $i \in F_0$. We continue the previous example:

Example 3.2.4. The consistency set of the formulation given in Example 3.2.3 is

$$S = \left\{ x \in \mathbb{R}^4 \mid \begin{array}{l} f_1(x) = x_1 \\ f_2(x) = x_2 \\ f_3(x) = 0 \end{array} \right\} = \left\{ x \in \mathbb{R}^4 \mid \begin{array}{l} x_2 + x_3 = x_1 \\ a_1 x_1 + a_2 x_4 = x_2 \\ a_3 x_1 - a_4 x_4 = 0 \end{array} \right\}$$

3.2.4 Design problem

If we add the role of equality(F_h) and inequality(F_g) constraints such that $F_1 = F_h \cup F_g$ it describes the design set:

$$S = \left\{ x \in \mathbb{R}^n \mid \begin{array}{lll} f_j(x) = x_j, & k = v_{out}(i),\ j = \varphi(i), & i \in F_0 \\ f_j(x) = 0, & j = \varphi(i), & i \in F_h \\ f_j(x) \leq 0, & j = \varphi(i), & i \in F_g \end{array} \right\}.$$

The set of functions f_i associated with end components $F_1 = F_h \cup F_g$ can be further subdivided into inequality constraint functions for each component node $i \in F_h$ and equality constraint functions for each component node $i \in F_g$. This is similar to the idea of assigning *problem roles* given in [32].

3.3 Reducibility of multidisciplinary sets

One of the key properties of multidisciplinary formulations is *reducibility*, which relates to concepts of *elimination* and *feed forward substitution*. To introduce reducibility, we go back to the example in Example 3.1.3, where we had only one component-function, and we found a simple method to find design points that required evaluating the function once. We now want to generalize this idea. Reducibility extends an idea of *equivalence* discussed by Boyd and Vandenberghe in [10] where it is applied to optimization problems: "two problems [are] *equivalent* if from a solution of one, a solution to the other is readily found, and vice versa. (It is possible, but complicated, to give a formal definition of equivalence.)". In the same spirit we say a set $S_1 \in \mathbb{R}^n$ based on functions f_i is equivalent to a set $S_2 \in \mathbb{R}^m$ if for every element in S_2 we can easily re-construct an element in S_1. By saying *easily* we are not very precise, but we mean that there is a function $\phi : S_2 \to S_1$, where the function ϕ has comparable computational cost to the functions f_i. We say that S_1 is reducible when $m < n$.

Before discussing reducibility further we give an example of equivalent sets.

$$S_1 = \{x \mid x_1 + x_2 = 1, x_1 - x_2 = 0\},\ S_2 = \{x \mid 2x = 1\}.$$

The sets S_1 and S_2 are not equal, as S_1 is a two-dimensional vector space, whereas S_2 is a one dimensional vector space. However, if we take a point from S_2 (as it turns out S_2 has only one element in it), $x^{(0)} = 0.5$, we can easily construct a solution $x^{(1)} = (0.5, 0.5)$ that is a member of S_1. We therefore say that S_1 and S_2 are equivalent.

Having a formulation graph gives us a way to generalize the idea from the one component example for any multidisciplinary formulation. We first revisit another example, to make the idea of reducibility clearer.

3.3.1 Reducing two components

We start with a formulation with 3 variables $\{1, 2, 3\}$ and 2 components $\{4, 5\}$, $\varphi(4) = 1$, $\varphi(5) = 2$, and two functions $f_1(x_3) = x_3^2$ and $f_2(x_1, x_3) = x_1 - x_3$. This formulation

describes the set

$$S = \{x \in \mathbb{R}^3 \mid x_3^2 = x_1,\ x_1 - x_3 = x_2\}.$$

We create the function $\phi : \mathbb{R} \to \mathbb{R}^3$, that satisfies $x = \phi(z)$ where

$$\phi(z) = (f_1(z), f_2(f_1(z), z), z).$$

Based on this function we can now take the simpler set:

$$S_r = \mathbb{R}$$

And pick an element from S_r, for example $2 \in \mathbb{R}$, and construct an element of S through $\phi(2) = (4, 3, 1)$.

3.3.2 Reducing a component with an equality constraint

For this example we remove the variable 3, and make the component 5 and end component, and change f_2 to be an inequality constraint function, so that the set we describe is:

$$S = \{x \in \mathbb{R}^2 \mid x_2^2 = x_1,\ x_1 - x_2 = 0\}.$$

Here we create the function $\phi(z) = (f_1(z), z)$ and use the reduced uni-dimensional set

$$S_r = \{z \in \mathbb{R} \mid f_2(f_1(z), z) = z^2 - z = 0\}.$$

In this case, there are only two real elements in S_r, namely $\{0, 1\}$, and we can reconstruct the elements of set S through $\{\phi(0), \phi(1)\} = \{(0, 0), (1, 1)\}$.

3.3.3 Reducibility and elimination notation

We next generalize the procedure we have carried out in the previous two examples to find a reduced set S_r and a reconstruction function ϕ. Particularly, the reductions were all carried out through *elimination* and subsequent *forward substitution*, since

we can eliminate the equation $x_i = f_i(x)$ and substitute it in the functions where x_i is used.

To make these eliminations explicit and to avoid the verbosity and nesting of the function composition like $f_2(f_1(z))$ in the reduced set we introduce a new operator, the left arrow or $\leftarrow$. For the previous example instead of

$$S_r = \{z \in \mathbb{R} \mid f_2(f_1(z), z) = z^2 - z = 0\}$$

we can use a cleaner notation, which is also closer to the original notation used for S, which we call *elimination notation*,

$$S_r = \left\{ z \in \mathbb{R} \mid \begin{array}{l} w_1 \leftarrow z^2 \\ w_1 + z = 0 \end{array} \right\}.$$

We introduce another example with the formulation given by the graph G in Fig. 3-3 with $\varphi(4) = 1,\ \varphi(5) = 2\ \upsilon(1) = 1,\ \upsilon(2) = 2, \upsilon(3) = 3$ and $f_1(x) = x_3 + a,\ f_2(x) = x_1 + a\ f_3(x) = x_2 - a$, and where a is a fixed constant, and f_3 and inequality constraint function.

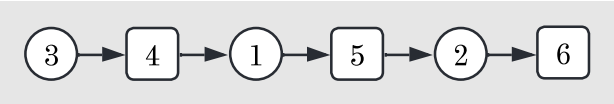


Figure 3-3: Example G

The formulation set associated is given by

$$S = \left\{ x \in \mathbb{R}^3 \mid \begin{array}{l} x_3 + a = x_1 \\ x_1 + a = x_2 \\ x_2 - a \leq 0 \end{array} \right\}.$$

And the reduced set, based on elimination is

$$S_r = \left\{ z \in \mathbb{R} \mid \begin{array}{c} w_1 \leftarrow z + a \\ w_2 \leftarrow w_1 + a \\ w_2 - a \leq 0 \end{array} \right\}$$

This examples allows us to introduce some more vocabulary: we call w_1 and w_2 *elimination variables*, and the set $F_L = \{4, 5\}$ is the *component elimination set* of S.

Which could have been written in the conventional way:

$$S_r = \left\{ z \in \mathbb{R} : \ f_2 \circ f_1(z) = z + 2a \leq 0 \right\}$$

This example illustrates that left arrow notation used to denote elimination (and feed forward substitution) is not always the most compact, however it has two advantages. First, it implicitly gives ϕ. And second, it makes it easier to write up explicit elimination expressions in the same way we would write equalities and inequalities.

This notation might make the reduced set seem more complicated than it actually is. For example, with a slightly altered example with the following graph instead:

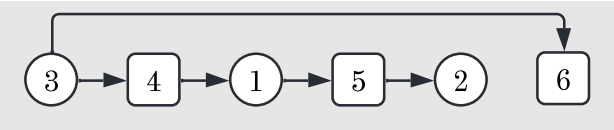


Figure 3-4: Modified graph G

Where now $f_3(x) = x_3$. The elimination notation is given below, with the version without the elimination notation to the right:

$$S_r = \left\{ z \in \mathbb{R} : \begin{array}{c} w_1 \leftarrow z + a \\ w_2 \leftarrow w_1 + a \\ x - a \leq 0 \end{array} \right\} = \{ z \in \mathbb{R} : x \leq a \}$$

The elimination notation makes it possible to easily recover $\phi(z) = (z + a, z + 2a)$.

3.3.4 Elimination order

The order of the elimination notation matters for the left arrow operator, as opposed to the equality and inequality operators (where placing an equality constraint earlier or later in the set definition has no importance). The order that the eliminations appear in encodes the order of the elimination. This matters as $f_2(f_1(x_1)) = (x_2 \leftarrow f_1(x_1), x_3 \leftarrow f_2(x_2))$ is not the same as $f_1(f_2(x_1)) = (x_2 \leftarrow f_2(x_1), x_3 \leftarrow f_1(x_2))$.

To encode this order we use the idea from [39] of a *linear ordering* or *permutation* of component nodes $i \in F_L$. The linear ordering is a bijective mapping $\sigma : F_L \subseteq F_1 \rightarrow \{1, 2, \ldots, |F_L|\}$; the order of elimination is given by this mapping, i.e. if for $i \in F_L$, $\sigma(i) = 1$, then the function $f_j,\ j = \varphi(i)$ is eliminated first, and so forth. We define $\bar{\sigma} = \sigma^{-1}$, i.e. the reverse mapping. We say that a component $i \in F_L$ is upstream from a different component $j \neq i \in F_L$ if $\sigma(i) < \sigma(j)$, and downstream if $\sigma(i) > \sigma(j)$. In the previous example $\sigma(4) = 1,\ \sigma(5) = 2$, i.e. component node 4 is upstream from 5 and vice-versa 5 is downstream from 4. We next describe the general conditions under which an elimination set can exist.

3.3.5 Elimination order and ayclic graphs

If the directed bipartite graph $G = (V, F, E)$ is acyclic, any set $F_L \subset F_1$ of intermediary components constitutes a valid choice of elimination set. A valid permutation σ can be considered a topological sorting of the component nodes in F_L satisfying $\sigma(i) < \sigma(j) \quad \forall k \in V$ s.t. $(i, k) \in E_{out} \wedge (k, j) \in E_{in}$. Assuming still that G is acyclic we maximize the size of the elimination set by picking $F_L = F_1$.

Given a valid permutation σ we get a general way to generate the reduced set of any formulation set:

$$S_r = \left\{ \begin{array}{lll} & w_i \leftarrow f_i(z,w), \quad i = \bar{\sigma}(j), & j \in \{1,\ldots,|F_L|\} \\ z \in \mathbb{R}^l : & f_i(z,w) = 0 & i \in F_h \\ & f_i(z,w) \leq 0 & i \in F_g \end{array} \right\}$$

where $l = N - |F_L|$, is the dimension of the reduced set, and $j = 1$ corresponds to the first elimination, for which we recover the index of the function through $\bar{\sigma}$.

3.4 Nested formulations

In the previous section we introduced flat formulation graphs, and conditions on the graph to recover reduced sets, which we would hope could translate into advantages for the solution method. In this section we introduce new notation that makes it possible to decompose a problem into multiple set, and define sets for each of these decompositions, and then later subproblems for them.

3.4.1 Formulation trees

A formulation tree T is a tree consisting of M *partitions*, or branches, B, component nodes F, input variables nodes V_0, and a set of of component leaf edges H_0, which in this case consists of a tuple (i,j), $i \in F$, $j \in B$, partition edges H_1 where the tuple $(i,j) \in B \times B$ and H_2 variable leaf edges where $(i,j) \in V_0 \times B$.

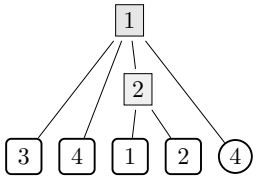


Figure 3-5: Tree T

Figure 3-6: A *nested formulation graph* $T = (V_0, F, B, H_0 \cup H_1 \cup H_2)$ with $V_0 = \{4\}, F = \{1,2,3,4\}$, $B = \{1,2\}$, and $H_0 = \{(3,1),(4,1),(1,2),(2,2)\}$, $H_1 = \{(2,1)\}$, $H_2 = \{(4,1)\}$

We say that for an edge (i,j) of the tree T, the node i is a child of the node j, and

conversely that the node j is a parent of the node j. We use the notation $H_0(i), H_1(i)$ and $H_2(i)$ to refer to the component nodes, variable nodes and partitions, respectively children of partition i.

3.4.2 Constraints on tree

The formulation tree has a similar constraint to the simple formulation graph, that for every end component child of a partition, there needs to be at least one variable node child of the same partition. This is required for the formulation set in Section 3.4.3 not to be over-constrained. Additionally, there needs to be a path from every input variable in the partition to every end component.

3.4.3 Nested formulation

A nested formulation consists of a formulation graph G and a formulation tree T. Together they allow us to describe formulations of multidisciplinary problems with *partitions* or *decompositions*. We can visualize both structures as a nested graph, shown in Fig. 3-7. In this visualization, a subgraph contains the child components and child nodes of a partition, and the output nodes corresponding to each child component.

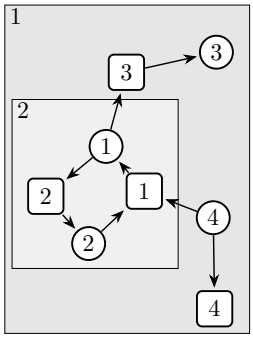


Figure 3-7: A *nested formulation graph* $G = (V \cup F, E_{in} \cup E_{out})$ with $V = \{1, 2, 3, 4\}, F = \{1, 2, 3, 4\}$, $E_{in} = \{(2, 1), (4, 1), (1, 2), (1, 3)\}$ and $E_{out} = \{(1, 1), (2, 2), (3, 3)\}$ and T as given earlier.

3.4.4 Nested formulation set

Nested formulation graphs encode multiple formulation sets S_i for every partition $i \in B$. Within every set, all variables correspond to variable nodes outside the partition are assumed fixed.

Nested formulation graphs encode multiple formulation sets S_i for $i = 1, \ldots, M$:

$$S_i(x^{(0)}) = \left\{ x \in \mathbb{R}^n : \begin{array}{ll} x_k = x_k^{(0)}, & k \in (V_0 \cup V_1) \setminus H_1(i) \\ x_k = \mathcal{P}_j(S_j(x), k), & k \in \textbf{outdesc}(j),\ j \in H_2(i) \\ f_j(x) = x_k, & k = \upsilon_{out}(l),\ j = \varphi(l),\ l \in H_0(i) \cap F_0 \\ f_j(x) = 0, & j \in H_0(i) \cap F_h \\ f_j(x) \leq 0, & j \in H_0(i) \cap F_g \end{array} \right\}$$

where $x^{(0)} \in \mathbb{R}^n$ is assumed to be a fixed variable. The first line in the description of the set is saying that for the set S_i, we fix the value of all input variables that are not outputs of either child components of the partition i, or child variables of the partition i. The second line gives a recursive definition through the operator **outdesc**, which is saying that we also fix all variables that are descendants of the partition, or are outputs of components descendant of the partition. We fix the value to the k-th value of the output of $\mathcal{P}_j : \mathbb{R}^n \to \mathbb{R}^n$. Here $\mathcal{P}$ stands for a problem associated with the set S_j. This problem could be a feasibility, or an optimization problem for example.

We go back to the example from Fig. 3-7. It describes two sets, S_2 and S_1, where

$$S_2(x^{(0)}) = \left\{ x \in \mathbb{R}^4 : \begin{array}{l} x_3 = x_3^{(0)} \\ x_4 = x_4^{(0)} \\ x_1 = f(x_1, x_2, x_4) \\ x_2 = f(x_1, x_2) \end{array} \right\} = \left\{ x \in \mathbb{R}^4 : \begin{array}{l} w_3 \leftarrow x_3^{(0)} \\ w_4 \leftarrow x_4^{(0)} \\ x_1 = f(x_1, x_2, w_4) \\ x_2 = f(x_1, x_2) \end{array} \right\}$$

and

$$S_1 = \left\{ x \in \mathbb{R}^4 : \begin{array}{l} x_1 = \mathcal{P}_2(S_2(x), 1) \\ x_2 = \mathcal{P}_2(S_2(x), 2) \\ x_3 = f(x_1) \\ f(x_4) \leq 0 \end{array} \right\} = \left\{ x \in \mathbb{R}^4 : \begin{array}{l} w_1 \leftarrow \mathcal{P}_2(S_2(x), 1) \\ w_2 \leftarrow \mathcal{P}_2(S_2(x), 2) \\ x_3 = f(w_1) \\ f(x_4) \leq 0 \end{array} \right\}$$

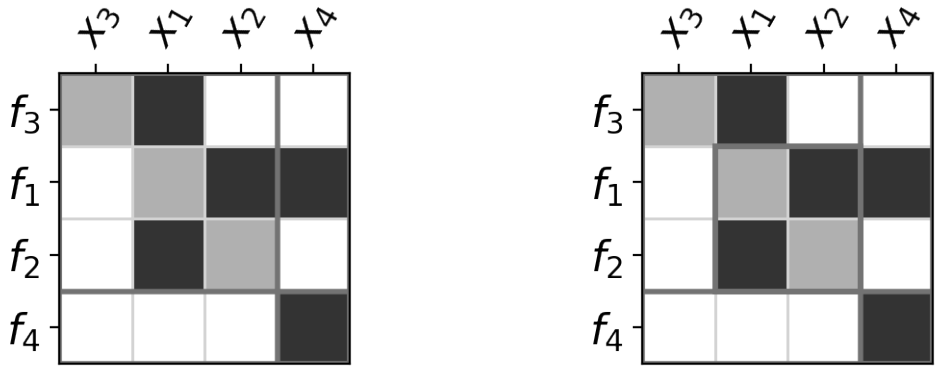
where $\mathcal{P}_2$ is the problem of finding an element of S_2, and $\mathcal{P}_2(S_2(x), 1)$ means taking the first entry of the vector element of S_2, while $\mathcal{P}_2(S_2(x), 2)$ involves taking the second element.

3.5 The Hierarchical Structure Matrix (HSM)

For visualization purposes of the nested formulation graph, this thesis introduces the *hierarchical structure matrix*, which is an incidence matrix where rows represent the components and columns variables. It is an extension of the *structural matrix* developed by Steward [47]. It also shares some of the characteristics of the DSM. The HSM is a visualization support, and if anything is better compared to a table than to a matrix in the mathematical sense. It is a compact way of visualizing both the formulation graph and the formulation tree, and the functions φ(and υ) which makes it perfect for displaying both the nested and flat formulations.

3.5.1 Example

We start with the HSM of the example formulation from Fig. 3-7, and visualize it in Fig. 3-8:



(a) Hierarchical structure matrix of the flat version of Fig. 3-7

(b) Hierarchical structure matrix of example from Fig. 3-7

Figure 3-8: Example of hierarchical structure matrix

For both examples in Fig. 3-8b we have $\varphi = (3,1),(2,1),(2,1),(4,1)$ and $\upsilon = (3,3),(1,1),(2,2),(4,4)$. The axis of the HSM are the component nodes for the rows, and the variable nodes for the columns. The HSM visualizes the split between intermediary components and end components in the formulation graph G. This can be seen in Fig. 3-8a where f_3, f_1 and f_2 are intermediary components, and a gray

thick line marks the separation between these components and the end component f_4. Since intermediary components have outputs, they are highlighted in gray in the matrix, and the order of the columns for the intermediary components are aligned with the order of the rows. The order of the rows and columns for the end components also follows φ but is arbitrary as it does not matter: variables that are being solved for at a particular level are in the columns.

The diagram makes it possible to visualize feedback, which for the HSM is in the upper right corner. This is against many conventions where feedback is in the lower left corner.

A nested graph is shown in Fig. 3-8b: here there is a solver which is a child of the root solver. It has two intermediary components. Each solver is considered an intermediary component from the perspective of the solver one level up.

3.5.2 Average row density

As mentioned in the introduction, sparsity is a key property required for the application of the methods in this thesis. We introduce a key metric that can easily be seen from the HSM: the average row density ρ, which is the average number of non-zero elements across rows in the matrix. Fig. 3-9 illustrated this process.

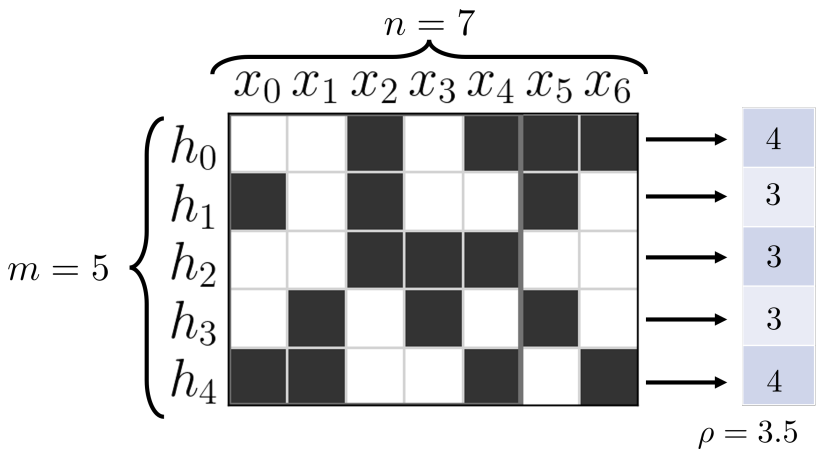


Figure 3-9: Illustration of row density. The row density of this figure is 3.5.

3.5.3 Showing inputs

As an extension to the structural matrix we make a clear separation between input variable nodes on the visualization by placing them all to the right of the table:

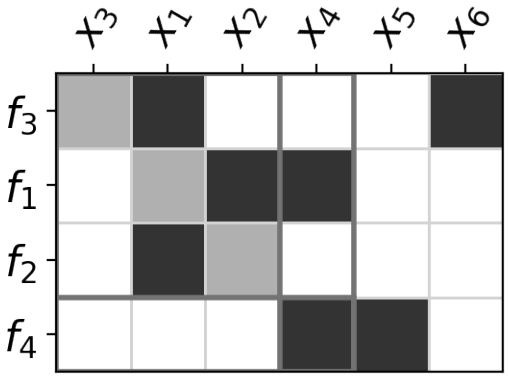


Figure 3-10: Hierarchical structure matrix of example from Fig. 3-7 with additional inputs

3.5.4 Constructing the HSM

Flat HSMs can be easily constructed from ordering the rows according to φ. However, nested ordering need some additional work: now the values associated with the ordering φ just determines the local ordering at a specific solver level. We have developed a simple algorithm that taken an ordering defined by φ, a graph and a tree will generate an HSM. We show an example where we start with the following solver tree:

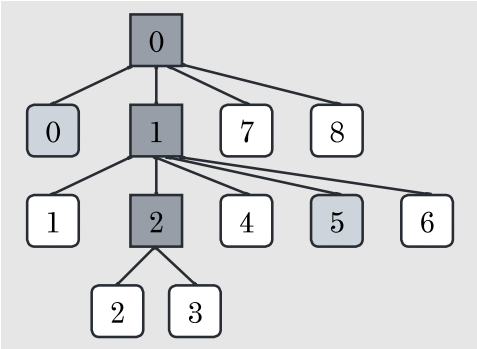


Figure 3-11: Tree of a more complex example with 9 components (where end components are in light gray, and partitions are in dark gray)

Which generates the HSM shown in Fig. 3-12.

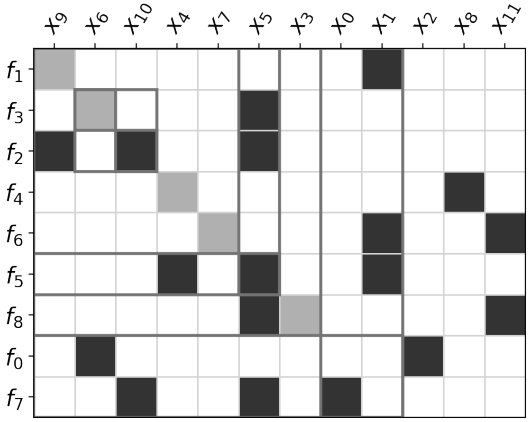


Figure 3-12: Hierarchical structure matrix of a more complex example

The original order given by $\varphi = \{(0,0), \ldots, (i,i), \ldots (8,8)\}$ is clearly not the order of the rows, which has the order $\{1,3,2,4,6,5,8,0,7\}$. This is because at any level the ordering algorithm makes sure that certain conditions are satisfied, namely that intermediary components go before end components (regardless of their order number), and are only ordered within each subgroup.

Chapter 4

Equivalent formulation transformations

Flat and nested formulation are not unique descriptions of formulation sets. This chapter introduces the idea of *equivalent formulations*: formulations that have different graphs and functions, but that describe *equal* sets. We then describe a set of transformations that yield equivalent transformations. We therefore call them *isomorphic transformations* as they conserve the formulation set that they describe. This is not to be confused with the well adopted concept of *graph isomorphisms*. We then show how for optimization problems over the formulation set, these transformations can be applied to generate different MDO architectures, particularly all monolithic architectures. We also introduce the concept of *equivalent sets*, which will be key to show how certain formulations can yield equivalent sets of lower dimensionality. Finally, we show how we can apply the transformations to generate these lower dimensional equivalent sets, which echo the ideas of Section 1.1.

4.1 Equivalent formulations

To illustrate the concept of equivalent formulations we start with an example with two flat formulation graphs G_1 and G_2, given in Fig. 4-1. Both have two variables nodes $\{1, 2\}$ and one component node $\{3\}$.

Figure 4-1: Two flat formulation graphs: G_1 on the left and G_2 on the right

The first formulation has a component-function $f_1(x) = x_2$ and the second formulation has an equation-function $h_1(x) = x_2 - x_1$. For both formulations we have $\varphi(3) = 1$, and for the first formulation $\upsilon(1) = 1$. Both formulations map onto consistency sets S_1 and S_2 respectively, where

$$S_1 = \left\{x \in \mathbb{R}^2 : f_1(x) = x_2 = x_1\right\}$$

and

$$S_2 = \left\{x \in \mathbb{R}^2 : h_1(x) = x_2 - x_1 = 0\right\}.$$

We can verify by inspection that $S_1 = S_2$, i.e. the two sets are equal : $x \in S_1$ if and only if $x \in S_2$.

4.2 Equivalence preserving transformations for flat formulations

Next we describe a set of transformations of both the graph and the functions that preserve equivalence of the formulation set they describe.

4.2.1 Reduction of intermediary component nodes

The first transformation we introduce is *reduction* of a component, and associated function. It reverses the directionality of the edge of the formulation graph that is outgoing from an intermediary component, to become incoming to that component, and transforms the function f_i of component j, where $i = \varphi(j)$ from a component-function to an equation-function, also often referred to as residual form: $\tilde{f}_i(x) = f_i(x) - x_k$, where $k = v_{out}(i)$. To keep track of *reductions* and potentially undo them through the second transformation, we introduce an additional edge set R_{in}, where the set of all input edges is now $E_{in} \cup R_{in}$. The operation on a single edge is illustrated in Fig. 4-2, highlighting the switch in direction of the edge.

Figure 4-2: Graph of original formulation on the left and transformed on the right

In the original graph G, the edge set looks like $E_{in} = \{(2,3)\}$, $E_{out} = \{(3,1)\}$, $R_{in} = \varnothing$ and transforms to $E_{in} = \{(2,3)\}$, $E_{out} = \varnothing$, $R_{in} = \{(1,3)\}$. In the original formulation $f_1(x_2) = x_2^2$, and in the transformed formulation $\tilde{f}_1(x_2, x_1) = x_2^2 - x_1$.

4.2.2 Inversion of end component nodes

The second transformation we introduce is *inversion* of an end component with respect to a variable. It does the opposite of *reduction*: it takes the incoming edge (i, j), where i is a variable node and j an end node of the formulation graph, and transforms it into an outgoing edge.

Figure 4-3: Graph of original formulation on the left and transformed on the right

It then transforms the function f_l, where $l = \varphi(j)$ to it's *partial inverse* with respect to x_k where $k = \upsilon(i)$. We define the *partial inverse* $\hat{f}_l : \mathbb{R}^n \to \mathbb{R}$ of f_l with respect to x_k as $\hat{f}_k = g(x_1, \ldots, x_{i \neq k}, \ldots, x_n)$ where $g : \mathbb{R}^{n-1} \to \mathbb{R}$ is the implicit function with respect to x_1 in the neighborhood of some point $x^{(0)}$. If we now take the example from earlier, and start with $E_{in} = \{(2,3),(1,3)\}, E_{out} = \varnothing, R_{in} = \varnothing$, and

$$f_1 = x_2^2/x_1 + x_2 - 1.$$

After *inversion* of the component 3 with respect to variable 1, we end up with $E_{in} = \{(2,3)\}, E_{out} = \{(3,1)\}, R_{in} = \varnothing$. We have $1 = \varphi(3)$, so we have to transform f_1 to it's implicit inverse with respect to x_1, since $\upsilon(1) = 1$. In this case we can find an explicit expression for $\hat{f}_1$ by manipulating the expression algebraically to get

$$\hat{f}_1 = (1 - x_2)/x_2^2.$$

4.2.3 Inversion of reduced components

By keeping track of reduced components through R_{in}, the inversion of a reduced component i with respect to a variable node j, when $(i,j) \in R_{in}$, just returns the original formulation. However, reducing with respect to a different variable does require computing the *implicit inverse* of the reduced function with respect to this variable. With the running example, if we start with the original graph $G = \{(2,3),(3,1)\}$, and reduce the component 3, we get as previously noted the graph $G = \{(2,3),(1,3)\}$, and we transform a function f_1 to its reduced version $\tilde{f}_1$. If we now invert the component of the resulting graph and $\tilde{f}_1$, we recover f_1.

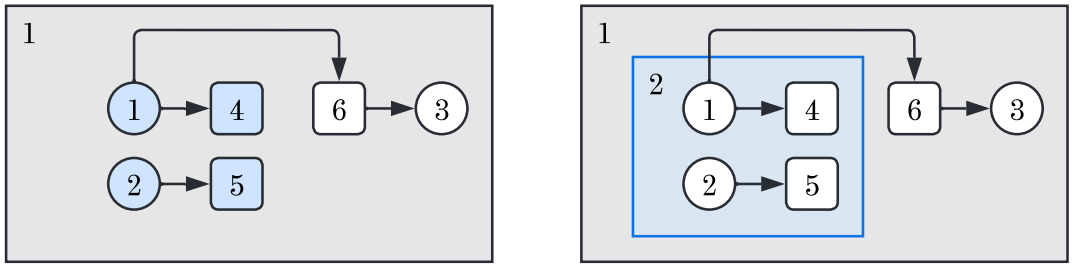
4.2.4 Reduction of inverted component-functions

By keeping track of inverted components through R_{out}, the inversion of a reduced component i with respect to a variable node j, when $(i,j) \in R_{out}$, just returns the original formulation.

4.3 Equivalence preserving transformations for nested formulations

4.3.1 Merging components into partitions

This transformation takes a formulation and merges multiple component nodes and variables nodes by putting them under a new partition in the formulation tree. For a valid merge to happen the rules from the nested tree mentioned in Section 3.4.2 need to hold: the number of variables nodes under the new branch needs to be greater than or equal to number of end components, and every variable needs to have a path to a different component. Fig. 4-4 shows an example of the the merge of two components and two nodes under a branch in the solver tree.



(a) G_1 (b) G_2, with differences to G_1

Figure 4-4: Graph associated with two formulation graphs resulting in the same formulation set

The merge is valid, as there are two variable nodes, $\{1, 2\}$ and two end components $\{4, 5\}$, and every variable has an independent path to a different component: variable node 1 has a path to component 4, and variable node 2 has a path to component 5. With $\varphi = \{(4, 1), (5, 2), (6, 3)\}$, and $\upsilon = \{(3, 3)\}$, the merge transforms the formulation set from:

$$S = \left\{ x \in \mathbb{R}^3 : \begin{array}{r} f_3(x) = x_3 \\ f_1(x) = 0 \\ f_2(x) = 0 \end{array} \right\}$$

to the two sets:

$$S_2 = \left\{ x \in \mathbb{R}^2 : \begin{array}{c} f_1(x) = 0 \\ f_2(x) = 0 \end{array} \right\}, S_1 = \left\{ x \in \mathbb{R}^3 : \begin{array}{c} \mathcal{P}_1(S_2) = x_1 \\ \mathcal{P}_2(S_2) = x_2 \\ f_3(x) = x_3 \end{array} \right\}$$

4.3.2 Merging of one component as inversion

We now take a look at the special case of merging one end component and one variable. We take the earlier example from Fig. 4-3, with edges $\{(1,3),(2,3)\}$. If we merge the component 3 and the variable 1, we end up with the nested graph shown in Fig. 4-5b:

(a) G_1 (b) G_2, with differences to G_1

Figure 4-5: Graph associated with two formulation graphs resulting in the same formulation set

$$S = \left\{ x \in \mathbb{R}^2 : f_1(x) = 0 \right\}$$

becomes

$$S_2(z) = \left\{ x \in \mathbb{R} : f_1(x, z_2) = 0 \right\}, S_1(z) = \left\{ x \in \mathbb{R}^2 : \mathcal{P}_1(S_2(x)) = x_1 \right\}$$

where $\mathcal{P}_1(S_2(x)) = \hat{f}_1(x)$

4.3.3 Merging of intermediary components

This is often how disciplines are defined. In Section 4.1 we will see how the elimination sets that result end up being equivalent.

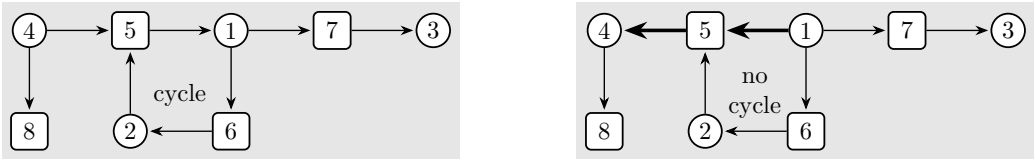
4.3.4 Reduction of branches

Reducing branches is the inverse of the merge operation, in the sense that it *flattens* the formulation by removing nesting, and exposes functions that were seen from a black box perspective at the lower level to the level above.

4.4 Reducible formulations from transformations

4.4.1 Reducible formulations from inversion

Due to the non-unique representation of a formulation we can end up with two different graphs with different functions describing the same formulation set, where, due to the acyclic condition from Section 3.3.3 one formulation is reducible, while the other is not. We illustrate this idea with an example, given by G_1 and G_2 described in Fig. 4-6.



(a) G_1 (b) G_2, with differences to G_1

Figure 4-6: Graph associated with two formulation graphs of equivalent formulation sets. G_2 can be seen as an inversion transformation applied to G_1: component 5 has been inverted with respect to variable 4, which is now the new output.

We set $\varphi_1(i) = i - n$ and $\upsilon_1(i) = i$ for the first formulation, and $\varphi_2 = \varphi_1$ except for $\varphi_2(5) = 4$ and $\upsilon_1 = \upsilon_2$ for the second formulation. The formulation sets described by G_1 and G_2 are given by S_1 and S_2, respectively.

$$S_1 = \left\{ x \in \mathbb{R}^4 : \begin{array}{l} x_2 \exp(x_4) = x_1 \\ x_1^2 + 1 = x_2 \\ \log(x_1) = x_3 \\ x_4 - 1 = 0 \end{array} \right\} = S_2 = \left\{ x \in \mathbb{R}^4 : \begin{array}{l} \log(x_1/x_2) = x_4 \\ x_1^2 + 1 = x_2 \\ \log(x_1) = x_3 \\ x_4 - 1 = 0 \end{array} \right\}$$

Although $f_1(x) = x_2 \exp(x_4)$ in the first formulation is different from $\hat{f}_1(x) = \log(x_1/x_2) = x_4$ in the second formulation, by inspection it is easy to validate that $S_1 = S_2$. However, since G_1 has a cycle, as noted in Section 3.3.5 we cannot use our mechanical ordering method to construct a reduced set based on this formulation, whereas since G_2 is *acyclic*, S_2 does have a uni-dimensional reduced set S_{r2}, as opposed to the 4-dimensional sets S_2 and S_1:

$$S_{r2} = \left\{ z \in \mathbb{R} : \begin{array}{l} w_1 \leftarrow z^2 + 1 \\ w_2 \leftarrow \log(z/w_1) \\ w_3 \leftarrow \log(z) \\ w_3 - 1 = 0 \end{array} \right\} = \left\{ z \in \mathbb{R} : \log\big(z/(z^2+1)\big) - 1 = 0 \right\}$$

4.4.2 Reducible formulations from merging

If we merge the components 5 and 6 from G_1 we get the nested formulation sets S_2 and S_1, which both have reducible formulations:

$$S_2(x^{(0)}) = \left\{ x \in \mathbb{R}^4 : \begin{array}{l} x_3 = x_3^{(0)} \\ x_4 = x_4^{(0)} \\ x_2 \exp(x_4) = x_1 \\ x_1^2 + 1 = x_2 \end{array} \right\} \Rightarrow S_{r2}(x^{(0)}) = \left\{ z \in \mathbb{R}^2 : \begin{array}{l} w_4 \leftarrow x_4^{(0)} \\ z_2 \exp(w_4) = z_1 \\ z_1^2 + 1 = z_2 \end{array} \right\}$$

and

$$S_1 = \left\{ x \in \mathbb{R}^4 : \begin{array}{l} \mathcal{P}_2(S_2(x), 1) = x_1 \\ \mathcal{P}_2(S_2(x), 2) = x_2 \\ \log(x_1) = x_3 \\ x_4 - 1 = 0 \end{array} \right\} \Rightarrow S_{r1} = \left\{ z \in \mathbb{R} : \begin{array}{l} w_1 \leftarrow \mathcal{P}_2(S_2(z), 1) \\ w_2 \leftarrow \mathcal{P}_2(S_2(z), 2) \\ w_3 \leftarrow \log(w_1) \\ z - 1 = 0 \end{array} \right\}$$

Merging also reduced the size of S_1 to uni-dimensional, however, as a hidden element of this reduction, we also need to solve a sub-problem at every iteration.

4.4.3 Elimination for affine functions

We look at the case when f_i are all affine functions, and we assume that the inequality constraint set F_g is empty. If we have only one branch node, this means that the set S_1 will be a linear subspace; if we did include linear inequalities for F_g we would be describing a polyhedron instead.

Small example case

Let S_1 first be the *flat* formulation set given by $V = \{1, 2, 3\}$, and only end component nodes $F_1 = \{4, 5\}$, $E_{in} = \{(1, 4), (3, 4), (1, 5), (2, 5), (3, 5)\}$, $E_{out} = \varnothing$, $\varphi(4) = 1, \varphi(5) = 2$ and finally the affine maps

$$\begin{matrix} f_1(x) = a_1 x + b_1 \\ f_2(x) = a_2 x + b_2 \end{matrix}, \quad A = \begin{bmatrix} 0.2 & 0 & -0.1 \\ 0.4 & 0.2 & 0.5 \end{bmatrix}, \quad b = (1, 1)$$

Then we have:

$$S_1 = \{x \in \mathbb{R}^3 : f_1(x) = 0, f_2(x) = 0\} = \{x \in \mathbb{R}^3 : Ax + b = 0\}$$

In this case S_1 is an affine subspace of $\mathbb{R}^3$; one example point from this set that can easily be verified is $x_1^{(0)} = (5, -5, 0) \in S_1$. We apply two different inversions: for the first set we invert component 4 with respect to variable 1 and component 5 with respect to variable 2. For the second component 5 with respect to variable 3 instead.

These transformations and the result are visualized in Fig. 4-7.

(a) Acyclic nested graph

(b) Cyclic nested graph

Figure 4-7: With two different nestings

Since the first set of inversion results in an acyclic graph, we can further simplify it as:

$$S_r = \left\{ z : \begin{array}{c} w_1 \leftarrow \hat{f}_1(z) \\ w_2 \leftarrow \hat{f}_2(z, w_1) \end{array} \right\}$$

where $\hat{f}_1(x_3) = (1 - 0.1x_3)/0.2$ and $\hat{f}_2(x_3, x_1) = (1 - 0.5x_3 - 0.4x_1)/0.2$ The second set of inversions, however leads to a cyclic component, which means we cannot generate a reduced set. Plugging in $z = 0$ we get as expected, $\phi(0) = (5, -5, 0)$.

4.5 Multidisciplinary architecture transformations

Next we show a different perspective on multidisciplinary architectures: they can be seen as a set of transformations based on a multidisciplinary, in the most general form, nested formulations. Although in the literature they have been applied to optimization problems, we here show how they can be applied to any design problem that can be represented through a nested formulation.

4.5.1 Multidisciplinary feasible transformations

We merge all level-1 components and branches of the tree at this level under a new branch, which will solve a feasibility problem: this is *multidisciplinary analysis problem.*

We can reduce the size of the multidisciplinary analysis problem as follows: We first construct the level-1 formulation graph. We then compute the strongly connected components of the level-1 formulation graph. Finally we merge the component nodes and variable nodes that are part of the strongly connected components. Then we get a level-1 formulation graph that has an elimination order, and possibly a smaller system that has to be solved. This is equivalent to the second step of the Dulmage-Mendhelson decomposition method used in tearing. Similar ideas have been used by [43] and [20].

4.5.2 Simultaneous analysis and design transformations

We reduce all level 1 branches and components.

4.5.3 Individual discipline feasible transformations

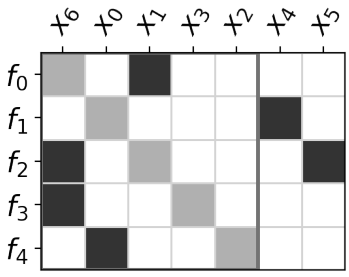
This construction requires auxiliary variables In the case of no branches at level-1 this strategy becomes a reduction of the component.

4.6 Acyclic transformations

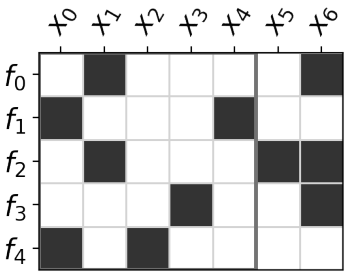
As seen in Section 4.4.1, we can carry out operations on a cyclical graph to turn it into an acyclic graph, and as a result get reduced formulations. We next discuss a set of transformations that guarantee generating an acyclic graph, while having the potential, but not guarantee, to result in a reduced formulation.

4.6.1 Reducing all components

This is the simplest strategy, as it guarantees that the problem will be acyclic, since no component has an output variable any longer. However, in essence we haven't gained much, as the formulation set that we get is the same as if we didn't apply the transformation, i.e. this transformation does not result in a reduced set.



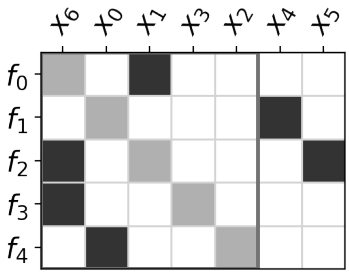
(a) Initial graph with feedback



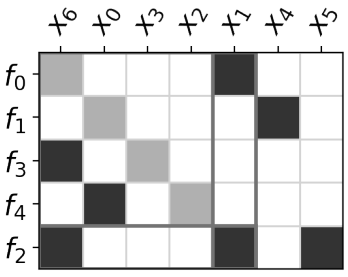
(b) Reduce all

4.6.2 Reduction based on formulation order

Given an (arbitrary) ordering of the components, we reduce any components that have an output variable that feeds back to a component earlier in the order. This guarantees that the resulting graph will be acyclic, but we might end up having to reduce a large number of components depending on how arbitrary the order is; in the worse case, we might have to reduce all components.



(a) Initial graph with feedback

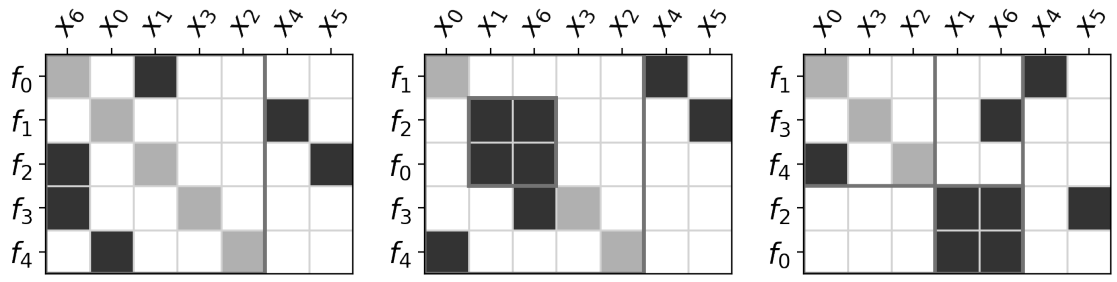


(b) Reduce based on feedback components

4.6.3 Strongly connected components methods

The first method finds clusters of components in the flat graph that when contracted all into one node makes the resulting graph acyclic. Based on the blocks, further transformations can be made on these blocks themselves to ensure the entire graph is acyclic. In this subsection we show the transformation where all components are reduced, and where alternatively, these are merged into a partition.

The method itself consists in finding all strongly connected components of the formulation graph. These strongly connected components correspond to the blocks mentioned earlier. We call the formulation where we reduce the strongly connected components the "block-IDF" formulation, and the formulation where we merge the reduced components into a partition the "block-MDF" formulation. We next take a random flat graph, and show these two transformations through the HSM:



(a) Initial graph with feedback

(b) block-MDF

(c) block-IDF

Figure 4-10: Example showing result of minimum size strongly connected component problem

Chapter 5

Optimal execution structures

In Chapter 4 we discussed the challenge of cyclical formulations, and different transformation strategies, both hierarchical and non-hierarchical, to turn them into executable formulations. In this chapter we describe three different methods to find optimal transformation strategies from a structural perspective. The first method finds a transformation of a flat formulation with the minimal number of reductions on the formulation graph, as every reduction becomes a guessing variable when solving. This problem corresponds to the well-known minimum feedback arc set problem. This method works when components cannot be inverted. The second method allows for inversion and reduction, and tries to minimize the amount of reductions needed. This becomes a combined assignment and minimum feedback arc set problem. We describe two versions of this method: one that does not allow for inversion of components with respect to input variable nodes, and one that does. The first and the second method result in flat formulations. The last method aims at reducing the size of the largest strongly connected component of the formulation graph through the process of inversions only, and merges the elements of the strongly connected component inside a partition branch. For a graph with the same number of components and variable nodes (and structurally non-singular), inversions do not lead to any improvement: therefore we only explore when inversions can be done with respect to input variables. We explore two versions: one where inside the partition we reduce

all components of the strongly connected components resulting, and one where we apply the first method to the partition to reduce the number of reductions needed.

5.1 Minimum component reduction structure

Formulation graphs can be made acyclic by reducing component nodes. On the one extreme we could reduce all component nodes, resulting in a formulation graph with only end components. This in turn would mean that the user would have to provide initial values for each variable node in R_{in} for iterative methods. However, in the case of sparse formulations, a much smaller number of components would ideally require reducing. An almost identical problem to the one just described is called *tearing*, and is relevant for analysis and design problems that can be formulated as solving systems of equations [12]. It can also be shown to be equivalent to the the well-known *minimium feedback arc set* problem, which consists in finding the set of edges that can be removed, or in our case, inverted, such that the resulting graph is acyclic [12]. Therefore we first describe some additional background, before we describe this problem in the context of formulation graphs.

5.1.1 Tearing

As described very concisely in a recent review by Baharev [12]:

> Tearing is the representation of a sparse system of nonlinear equations
>
> $$f(x) = 0, \text{ where } f : \mathbb{R}^n \to \mathbb{R}^m$$
>
> in a permuted form where most of the variables can be computed sequentially once a small auxiliary system has been solved.

This problem can eventually be posed as a minimum feedback arc set, or equivalently a minimum vertex feedback set [48], which is generally NP-hard, but for which good heuristics have been developed. The variable associated with the node on one side of

a *torn* or removed edge, becomes an input to the system and the function associated with a node on the other side gets an associated residual that the system is aimed at converging towards.

In [12], Baharev does a survey on the literature on the concept of tearing to reduce the size of system of equations by eliminating variables and turning some of the equations into an ordered set of explicit computations. Tearing has found multiple applications, especially in the chemical engineering industry, but is also one of the key ideas behind so called *acausal* modeling languages, like Modelica, a simulation language widely used in the automotive industry and which originated from Elmqvists thesis [25]. Many of the key ideas behind tearing were originally developed by Steward, first in [5] and later in [49].

5.1.2 Integer program formulation

We adapt the minimum set cover formulation of the minimum vertex feedback problem as it is given in [48]:

$$\min_{y} \quad \sum_{j=1}^{m} y_j \tag{5.1a}$$

$$\text{subject to} \quad \sum_{j=1}^{m} a_{ij} y_j \geq 1 \quad \text{for each } i = 1, 2, \ldots, \ell \tag{5.1b}$$

$$y_j \text{ is binary} \tag{5.1c}$$

Where m represents the number of component nodes, y_j is 1 if we are *reducing* component j, and $y_j = 0$ if we leave the component node intact. This is analogous to the minimum feedback vertex set problem, where feedback vertices can only be selected among the set of component nodes. Each constraint in Eq. (5.1b) corresponds to a strongly connected component i of a subgraph of the bipartite formulation graph. For each such strongly connected component, a_{ij} is a 1-0 vector of dimension m where

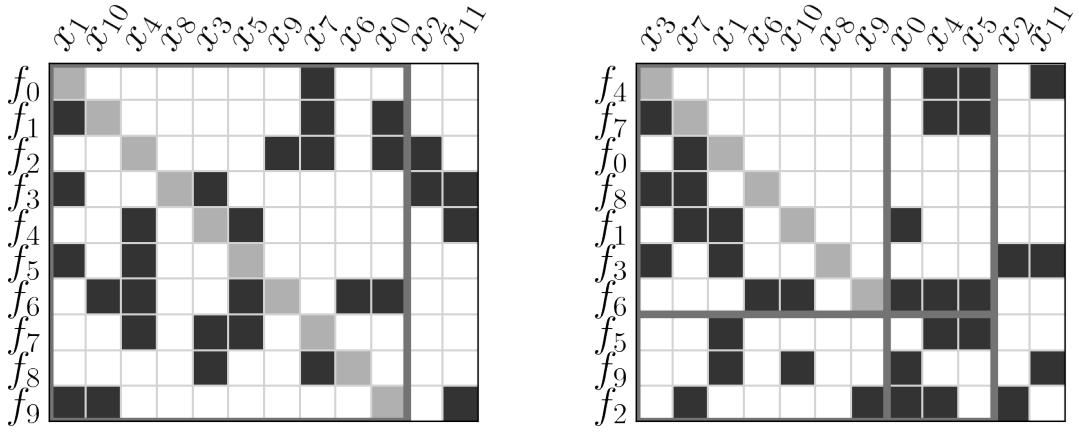
each entry corresponds to a component in the graph. $a_{ij} = 1$ if component j is part of the strongly connected component i, and 0 otherwise. Then Eq. (5.1b) says that at least one of the component nodes need to be reduced. These constraints looks the same as the one given by [50], with one major difference: in the original formulation each inequality results from enumerating cycles in the original graph. In this formulation, each inequality constraint comes from strongly connected components of ℓ subgraphs of the formulation graphs. In the original formulation, the argument is made that enumerating cycles of a graph can be done in $\mathcal{O}(n+e)(l+1)$ where n is the number of nodes and e the number of edges and l the number of cycles [51]. But since the number of simple cycles in a directed graph can become exponential (i.e. 2^n) in the number of nodes in the graph, they use a cutting plane method. We apply a similar strategy, as enumerating strongly connected components of all subgraphs is also an exponential problem. The cutting plane method, which is efficiently implement through so-called *lazy constraints*. A simplified description of this solution method looks as follows. We first enumerate the strongly connected components in the input graph formulation, and generate a set of initial constraint with $a_1, a_2, \ldots, a_{n_0}$ corresponding to the n_0 strongly connected components in the input graph. We then solve the optimization problem, and get $y^{(0)}$. We use the value of $y^{(0)}$ to reduce the graph, which means that for every component j where $y_j = 0$, we flip the direction of the output edge. We then compute the strongly connected components of this new graph, and add n_1 constraints $a_{n_0+1}, \ldots, a_{n_0+n_1}$. We repeat the process until the reduced graph that results has no strongly connected components.

The use of strongly connected components instead of cycles is motivated by the fact that we use an extension of this formulation in Section 5.3.

5.1.3 Examples and validation

To test the binary integer programming formulation, we generate random directed bipartite graphs, and control the row density and the size of the graphs. As an example: we show in Fig. 5-1b the hierarchical structure matrix resulting from finding the

minimum feedback arc set for a graph with 10 component nodes and 10 variable nodes shown on the left in Fig. 5-1a. In the upper left box of the HSM in Fig. 5-1b, we have topologically sorted the resulting directed graph, and the components in the bottom row: $\{f_5, f_9, f_2\}$ correspond to the reduced components, and the matching variables, $\{x_0, x_4, x_5\}$ that where outputs of these components are in the right columns.



(a) Starting problem

(b) One optimal solution

Figure 5-1: Example showing result of minimum feedback problem

We validated the implementation of the algorithm by running a brute force method: for a given k, we iterate through all k-combinations of the set of variables nodes V, and we remove for each node, the outgoing edge (there is only one, by construction of the formulation graph), and test whether the resulting graph is acyclic. We start with $k = 1$, and proceed until we find a k for which the test passes. Since this is a brute force method it only scales to reasonable computational times on a personal computer with up to dimensionality of n and m between 10 and 20, and k being less than 10. We then compare the result of the binary integer program, and the brute force methods for a large set of random graphs. The results matched.

5.2 Extended tearing

We next investigate how to find optimal structures resulting in the smallest amount of end components, which we correlate with reducing the size of the problem. We extend the set of operations allowed to inversions as well: we allow for a component to be inverted with respect to all of the incident variable nodes. When the number of components equals the number of variables, it is normally referred to as *tearing* in the literature. Although the formulation given below already exists in the literature, we extend the applicability of the formulation to problems where the number of variables nodes is larger than the number of component nodes. We therefore call it *extended tearing*.

5.2.1 Integer program formulation

This problem can be formulated as a combined assignment(or matching) and minimum feedback arc set problem. We again use a binary integer formulation.

$$\max_{x} \quad \sum_{(i,j)\in E} x_{ij} \tag{5.2a}$$

$$\text{subject to} \quad \sum_{(i,j)\in E} x_{ij} \leq 1 \qquad \text{for each } i = 1, 2, \ldots, n \tag{5.2b}$$

$$\sum_{(i,j)\in E} x_{ij} \leq 1 \qquad \text{for each } j = 1, 2, \ldots, m \tag{5.2c}$$

$$\sum_{j=1}^{m} a_{kij} x_{ij} \leq \frac{|S_i|}{2} - 1 \qquad \text{for each } k = 1, 2, \ldots, \ell \tag{5.2d}$$

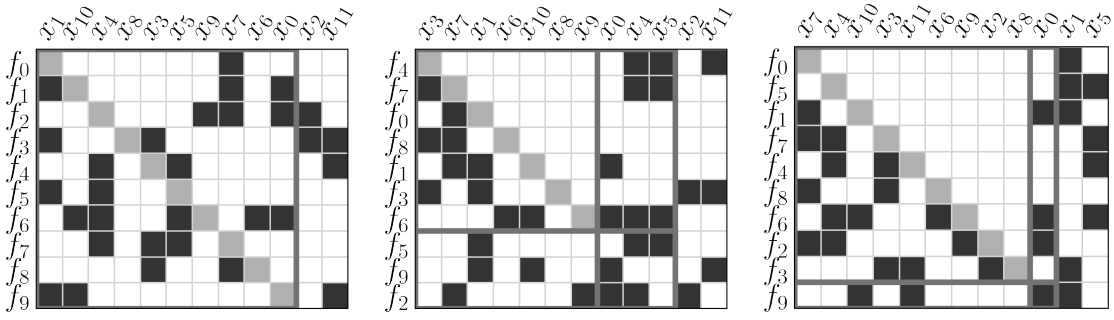
$$x_{ij} \text{ is binary} \tag{5.2e}$$

Where E is the edge set, and we have a binary variable x_{ij} for each edge. x_{ij} is 1 if the variable i is the output of component j, and $x_{ij} = 0$ if the variable i is an input to component j. The goal is to maximize the number of components with an output variable, as this reduces the amount of end components; this justifies the formula in

Eq. (5.2a). Eq. (5.2b) says that we can pick at most one edge out of the edges going out of a variable(it could not be assigned at all), and Eq. (5.2c) we can pick at most one of the multiple edges incoming to a component node. This is the assignment problem. In this formulation, Eq. (5.2d) says that for each cycle S_i in the undirected graph given by E, at most $|S_i|/2 - 1$ of the components participating in that cycle can have an output variable. This is the same as saying that at least one of the components in the cycle needs to be an end component. In this formulation there is a vector a_k of size $|E|$, where each entry corresponds to an edge in the graph and indicates whether the edge is member of cycle i of the undirected graph. $a_{kij} = 1$ if x_{ij} participates in the simple cycle k of the undirected graph, and 0 otherwise. The logic of the formulation is very similar to the logic in Section 5.1.2, and is further detailed in [12].

5.2.2 Examples and validation

We again test the implementation of the algorithm against random matrices. As an example we pick the same random matrix used in Section 5.1.3, with average rho density $\rho = 4$. We compare the size of the problem generated through the minimum feedback formulation from Section 5.1.2 and the formulation given in Section 5.2.1 in Fig. 5-2.



(a) Starting problem (b) Minimum feedback (c) Optimal assignment

Figure 5-2: Example showing results of extended tearing

The size of the final system goes from 3 in Fig. 5-2b (the number of variables in the

banded column) to 1 in Fig. 5-2c, and the input variables changed from $\{x_2, x_{11}\}$ to $\{x_1, x_5\}$.

5.3 Reduced subproblems

The last structural problem we solve is a new formulation that has not been reported in the literature. The goal is to find a set of full inversions of components such that the size of the largest strongly connected component that results from the assignment is as small as possible. If we then apply a merge transformation on the formulation based on the components in the strongly connected component, we get a set of smaller subproblems that could be solved in sequence.

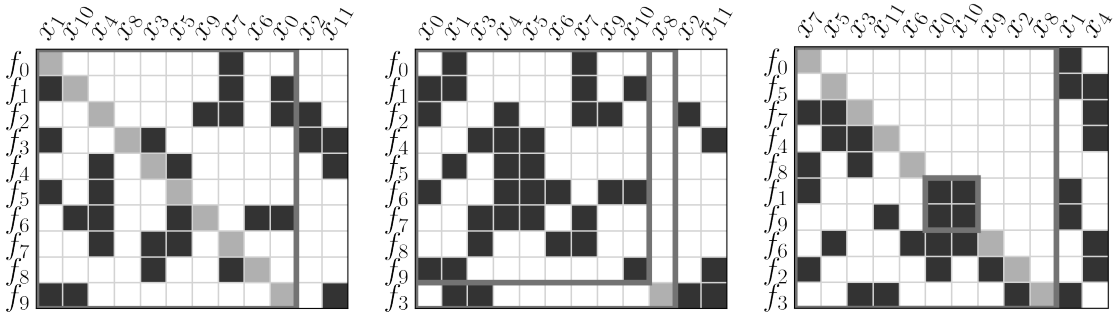
5.3.1 Integer program formulation

$$
\begin{aligned}
\min_{x} \quad & s_{max} \\
\text{subject to} \quad & \sum_{(i,j)\in E} x_{ij} = 1 && \text{for each } i = 1, 2, \ldots, n \\
& \sum_{(i,j)\in E} x_{ij} \leq 1 && \text{for each } j = 1, 2, \ldots, m \\
& \sum_{j=1}^{m} a_{kij} x_{ij} \geq 1 - \frac{s_{max} - 1}{|S_i|} && \text{for each } k = 1, 2, \ldots, \ell \\
& y_j \text{ is binary}
\end{aligned}
\tag{5.3}
$$

Where $|S_i|$ is the length of the simple cycle S_i, and s_{max} is the size of the largest cycle in the model (which matches the size of the larges strongly connected component). In this formulation we force all the components to be assigned to an output set, this is what forces the strongly connected components to appear. The last inequality works, because it ensures that $s_{max} > S_i$, if the cycle S_i appears in the problem.

5.3.2 Examples and validation

We again test the implementation of the algorithm against random matrices. As an example we pick the same random matrix used in Section 5.1.3, with average rho density $\rho = 4$. We compare the size of the problem generated from finding the strongly connected components in the initial formulation with the optimal structure in Fig. 5-3.



(a) Initial formulation (b) Initial SCC (c) Smallest SCC

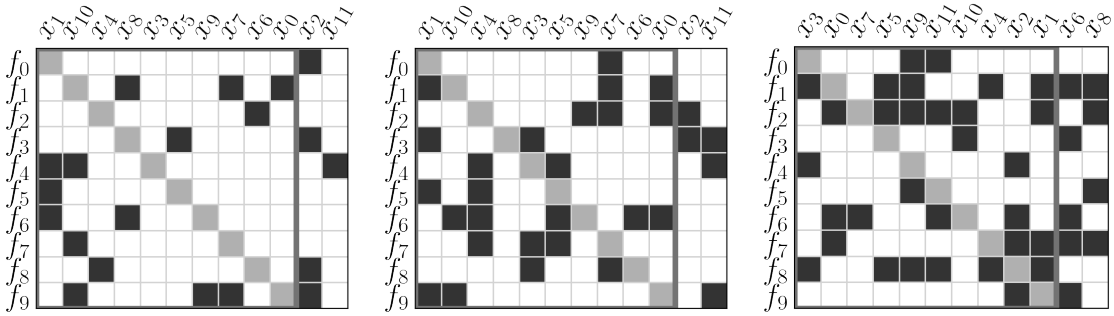
Figure 5-3: Example showing result of minimum size strongly connected component problem

We can see that by allowing for inversions of all components, we can reduce the size of the largest strongly connected component from 9 (in Fig. 5-3b) to 2 (in Fig. 5-3c). In this case the input variables changed from $\{x_2, x_{11}\}$ to $\{x_1, x_4\}$.

To validate the implementation we generate all n choose $n-m$ possible combinations of input variables and carry out the Dulmage-Mendelsohn decomposition on each of the combinations, and we find the size of the largest strongly connected component. For small sizes ($m = 10, n = 15$), this can be done within a few seconds. The results matched as-well. One interesting lesson that came from running the validation code is that there is normally a large number of solutions that can achieve the smallest strongly connected component from 5 to 10 percent of all the combinations of inputs.

5.4 Optimizing structure of random bipartite graphs

To measure the performance of the structural optimization, we test it on random bipartite graphs of different sizes, where we assume an output assignment for each component node is given in the initial graph, meaning that all component nodes are intermediary components. This initial output assignment is also randomly generated. One key controlling parameter that we set is the degree of sparsity, where we use the average row density metric ρ previously introduced. Given a desired ρ_0, we can produce random bipartite graph in such a way that (the details are given in Chapter 5) that the mean row density of a large number of matrices is ρ_0. In Fig. 5-4 we show examples of randomly generated matrices with different values of ρ: 3, 4, and 5. We show both the desired ρ, and the ρ of the randomly generated matrix resulting.



(a) $\rho = 3$ (b) $\rho = 4$ (c) $\rho = 5$

Figure 5-4: HSM for different row densities for a graph with $m = 10$, $n = 12$.

For measuring performance we generate problems of different row densities ρ, size m, and different factors of degrees of freedom n_{coeff}, for example $n_{\text{coeff}} = (n - m)/m$, where n is the number of variables. We use n_{coeff} to provide a normalization factor with respect to the problem size. For the same desired combination of $\rho, m, n_{\text{coeff}}$ we generate 10 random matrices, and record the actual ρ generated. We then test each of the three algorithms and measure the run time and the objective function. We use the next three sections to show the results for $m = (10, 20, 30, 40, 50)$ for $n_{\text{coeff}} = 0.5$ and $m = (10, 20, 30)$ for $n_{\text{coeff}} = (0.1, 0.2)$. We split the observations in two buckets $2.5 \leq \rho \leq 3.5$, $3.5 \leq \rho \leq 4.5$. We also limited the run time to 100 seconds out of

pragmatic reasons, and if a computation took longer than 100 sec we put it as timed out.

5.4.1 Minimum reduction structure results

We test the minimum feedback arc set implementation on the set of random problems. Since this algorithm does not exploit the degrees of freedom, we expect to get the same result running the algorithm across different n_{coeff}. The run time performance for different component size is shown in Fig. 5-5. In this subsection we explore the number of reductions that are needed as a function of row density and size of the problem. Since for this problem the number of input variables does not really matter, we set it to zero, and focus on problems with only intermediary components, without loss of generality. In Fig. 5-5 we show the distribution of computational performance of the current algorithm for different sizes of the graph. The line tracks the average across different values for m, and the shaded region shows the spread in the computational time.

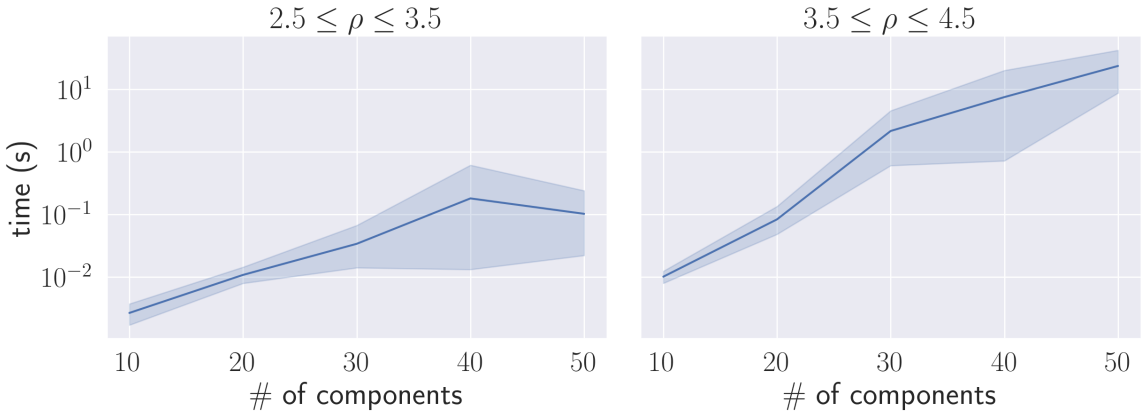


Figure 5-5: Computational time required for different row densities

Note that the y-axis is logarithmic. A couple of interesting observations can be made: for high n_{coeff} the computational cost seems to be sub-linear on the plot, meaning subexponential. However, for slightly larger average row-densities it seems like the algorithm enters a different regime that look linear on the logarithmic plot, which translates to exponential. Problems with up to 50 components can be solved in tens

of seconds on average, and we would expect that problems with 60 components would be solved in hundreds of seconds and so on.

5.4.2 Extended tearing results

We test the extended tearing across the three levels of n_{coeff}. The results are shown Fig. 5-6 where each color correspond to a different n_{coeff}. Since the run time of the optimization started timing out at 100 seconds for $m \geq 30$ and for most cases of $n_{\text{coeff}} \leq 0.2$, there are almost no results shown for these cases in the right of the figure.

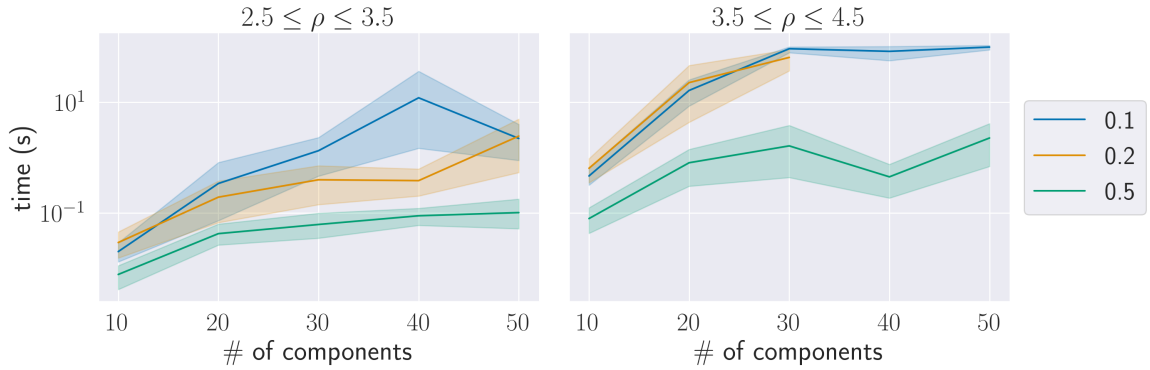


Figure 5-6: Computational time required for different problem sizes

Similar to the minimum reduction structure optimization, for low row densities the run-time is subexponential, as can be seen in the left figure for $2.5 \leq \rho \leq 3.5$. This behavior seems to be conserved for higher ρ, as long as n_{coeff} is high. As soon as n_{coeff} gets smaller, the run time of the algorithm seems to be exponential.

To show that extended tearing can add significant value to reducing the size of the system, we compare the size of the reduced system (the width of bordered column in the hierarchical structure matrix) as we go from only allowing for reduction of existing components, to allowing for inversions with respect to all variables except inputs, to allowing for inversions with respect to all variables including inputs. We compare the results across all $2.5 \leq \rho \leq 4.5$ all n_{coeff} and $10 \leq m \leq 50$.

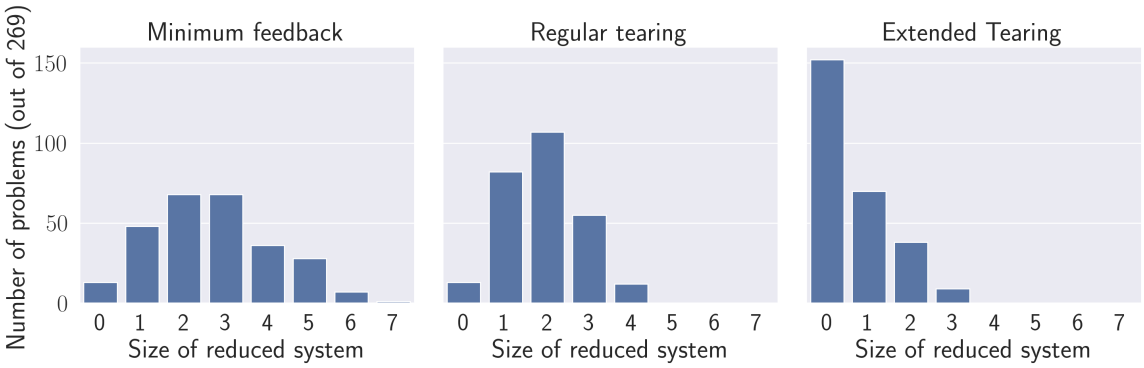


Figure 5-7: Resulting size of reduced system across multiple values for $\rho, m, n_{\text{coeff}}$.

As Fig. 5-7 shows, the number of problems with smaller size increases significantly when we allow for extended tearing. More than half of the problems (152 out of 269) where reduce to size zero, meaning they can be solved as a feed forward problem. Although the extended tearing does not reduce the size a lot with respect to regular tearing in the worst case (the maximum size recorded for regular tearing was four, versus three on extended tearing), in the best case it increases the number of zero sized systems from 13 to 152. This might actually be the main added value of extended tearing: it can find feed-forward systems that are unlikely to exist based on the original formulation.

5.4.3 Reduced subproblems results

We show the run time of the reduce subproblem structure implementation across

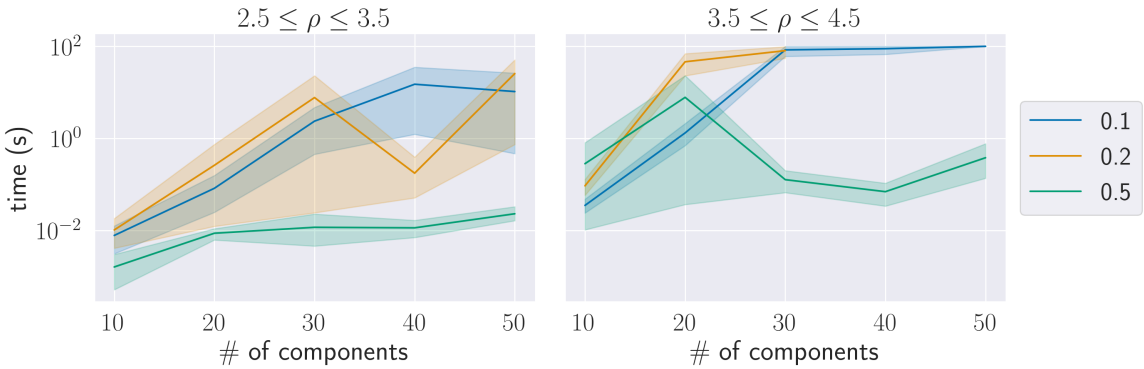


Figure 5-8: Computational time required for different problem sizes.

different problem sizes in Fig. 5-8. The spread in computation time for $n_{\text{coeff}} = 0.2$ is quite large across different sizes of m for $2.5 \leq \rho \leq 3.5$. We observe also similar subexponential behavior of the algorithm for $2.5 \leq \rho \leq 3.5$, whereas for larger densities and smaller n_{coeff}, the algorithm seems to have exponential behavior. For $n_{\text{coeff}} \leq 0.2$ and $m \geq 30$ almost all optimization runs timed out at 100 seconds.

To benchmark the value of finding reduced subproblems we compare the size of the largest strongly connected component based on the initial random output assignment and the optimized structure. The results across different values for n_{coeff} and for $2.5 \leq \rho \leq 4.5$ and $m = (10, 20)$ is shown in Fig. 5-9.

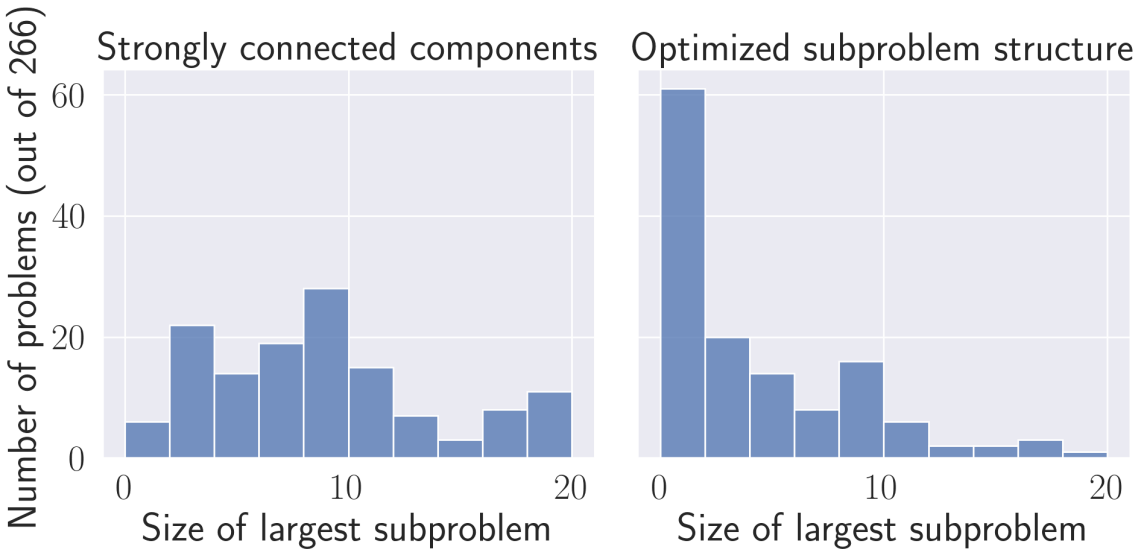


Figure 5-9: Distribution of size of largest subproblems

Fig. 5-9 shows the effect of finding the minimum size strongly connected component: on the left, the plot shows the largest strongly connected component for the original assignment, and on the right for the optimized assignment. We see that the largest strongly connected component skews to the left, and we can find a large number of structures that are feed forward. Note that the data used for the plot in Fig. 5-9 is different from the one in Fig. 5-7, as for both cases we have eliminated the cases that timed out (in which case we cannot say to have found the minimally sized subproblem structure).

Chapter 6

Synthetic conceptual design

To test the methods developed in this thesis we need the ability to quickly generate problems of different sizes to test the methods on a more extensive set of candidate test problems. Test problems have been generated in the literature: Tedford and Martins describe a non-convex quadratic optimization problem in [52], Zhang et al. describe a method to generate multidisciplinary problems with linear structure [53], Dennis describe a polynomial test problem in [11]. In this chapter, we describe a new set of test problems that also rely on polynomial functions. The goal is to imitate the form of conceptual design problems while guaranteeing that the functions can be partially inverted with respect to its variables. Finally, we investigate two applications of the methods developed in Chapter 5: generating reformulations of the design problem and reformulations of an optimization problem, where the objective function is constructed in a special way.

6.1 Polynomial functions

We create a synthetic problem where each component in the graph is associated with a scalar polynomial function. The choice to focus on polynomials is made based on three reasons. First, we can readily generate partially invertible polynomials with a minor constraint on the domain of the variables: they have to be non-zero, which

is often the case for design engineering problems. Second, a large set of conceptual design models can be described or reformulated with polynomial equations. Finally, we can use custom solvers for polynomial problems to benchmark and verify the results from both problems. Polynomial systems can be solved relatively robustly with off-the-shelf homotopy methods [54] that give access to a ground truth against which we can compare our methods. Furthermore, for the optimization problem we can use the SAND formulation (with only equality constraints, and no elimination functions), to derive the Karush–Kuhn–Tucker (KKT) conditions of the optimization problem, which is also a polynomial problem, now in the original variables with the addition of variables for the Lagrange multipliers. We can also use the homotopy solver for this problem. Lastly, although these problems possess signomial structure, including polynomials, we could apply off-the-shelf signomial optimization solvers. We note, however, that signomial solvers have a more challenging time solving for equality constraints and are less of a mature technology than the homotopy methods used.

6.1.1 Structure

To generate a synthetic model, we first generate the graph's structure through the same methods applied in Chapter 5. To recall, we first generate a random bipartite graph, with one set of nodes corresponding to the variables and the other set corresponding to the set of variables. The randomness comes from which edges connect which nodes of the bipartite graph. Different models for randomness can be used, but one that will be considered in the future is the Erdős–Rényi model for random graphs. Next, to ensure that we have an output for each component and thereby do not have any end components in this generated graph, we add an edge to the random bipartite graph if it does not exist. Finally, we add a connection from any isolated variables to any random components to ensure that all variables are used.

6.1.2 Polynomial equations

We start with a very general template for synthetic conceptual design models, given by a set of m polynomial functions $h_i(x), x \in \mathbb{R}^n$ (where $n > m$):

$$h_i(x) = \sum_j a_j \prod_{k \in P_k(i)} x_k + a_0 \tag{6.1}$$

Where $P_k(i)$ stands for the elements in a random partition of the input and output variables $\mathrm{I}(i) \cup \mathrm{O}(i)$ of component i in the graph. For example if $\mathrm{I}(i) \cup \mathrm{O}(i) = (1, 2, 3, 4, 5, 6)$ a random partition would be $(1, 3, 4), (2, 6), (5)$. a_j are coefficients for each partition drawn from a random uniform distribution, which for the purposes of this problem are picked such that $-1 \leq a_j \leq 1$.

Example 6.1.1. We take the set of variables $x_1, \ldots, x_6$. We first generate the random partition $(1, 3, 4)$, $(2, 6)$, (5), random coefficients $a = (0.5, 0.1, -0.9)$ and from it generate the random polynomial:

$$h(x) = 0.5x_1x_3x_4 + 0.1x_2x_6 - 0.9x_5$$

Polynomials generated in such a fashion have the property that we can easily invert the component node with respect to one of the variables, and have an explicit form of the transformed function $\tilde{h}_i$. For example, the component associated with h from Example 6.1.1 can be inverted with respect to the variable node 2, turning the end component into an intermediary component, where the implicit function $\tilde{h}$ can be given explicitly when assuming $x_6 \neq 0$:

$$\tilde{h}(x) = 5\frac{x_1x_3x_4}{x_6} - 9\frac{x_5}{x_6}$$

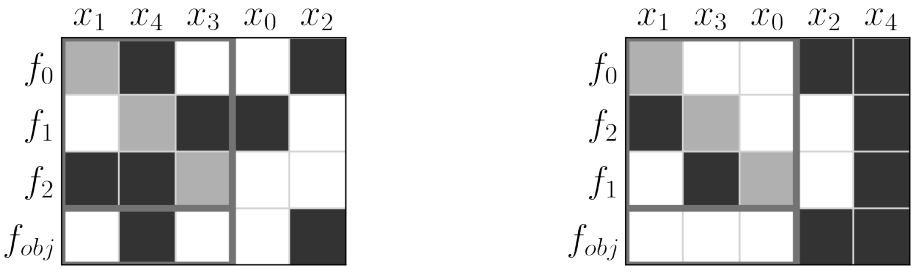
Since by construction we want h_i to be invertible with respect to all variables we need to include the assumption on $x_i \neq 0$; this in practice is not too constraining, we just need to take it into account for the formulation of the different problems we are solving.

6.2 Feasibility design problem

The first design problem we explore for a synthetic problem is the question of finding design candidates through a potential re-parametrization of the problem. We compare four existing strategies: a regular MDF strategy, a compact MDF strategy, a regular IDF strategy, and a compact IDF strategy, with three strategies based on the optimal structures: one to generate the smallest IDF formulation (tearing), with and without allowing for changes in inputs and one to generate the smallest compact MDF formulation, allowing for changes in inputs. We compare the performance based on the number of reductions required, as a reduction will be associated with a guess variable and the computational performance for different-sized problems. Finally, we include the computation time required for finding the optimal structures.

6.3 Optimization design problem

For the optimization problem, we construct an artificial objective function that we expect would make solving the re-formulated optimization problems easier to solve than the original formulation. To do this, we chose the objective function to depend on the input variables of the structurally optimized graph. We denote the set of variables selected in this way X_f. This should make the optimization problem reasonably trivial, and reduces the problem into a feasibility problem, where the goals becomes achieving consistency.



(a) Optimized formulation (b) Initial formulation

Figure 6-1: Two formulations of the same problem

In Fig. 6-1 we show an optimized formulation, and on the right the original formulation, and we can see in the last line that in the first problem the objective function depends only on the variables that were picked as inputs.

Based on X_f we generate a simple objective function:

$$f(x_i) = \sum_{i \in X_f} (x_i - p_i)^2$$

For simplicity we set $p_i = 1 \quad i \in X_f$, but this could also be tuned as long as $p_i \neq 0$, as we are trying to avoid cases where $x_i^* = 0$.

The constraint functions and elimination orders will depend on the formulation chosen, which will be described in the next section.

6.3.1 Ground truth

We constrained the synthetic optimization problem to equality constraints by design: it allows us to solve the equations resulting from the KKT conditions as a larger system of $n+m$ polynomial equations in $n+m$ variables, where m variables correspond to the lagrange multipliers μ_i.

$$\sum_{i=1}^{m} \mu_i \frac{\mathrm{d}h_i}{\mathrm{d}x} + \frac{\mathrm{d}f}{\mathrm{d}x} = 0 \tag{6.2}$$

$$h_i(x) = 0 \quad i = 1, \ldots, m \tag{6.3}$$

We use the `HomotopyContinuation.jl` Julia library to solve the resulting system [55]. We first verify that there is one unique real solution - as homotopy methods solve over the field of complex numbers. This method gives us all points, if there are finitely many of them, that satisfy the KKT conditions; these are local optima that we could expect the more general optimization methods that we will see in next section to find. To find the global optima we would just pick the local optimum with

the lowest cost, but since more general optimization methods aim only at finding a local optimum we have a list of points we can compare against.

6.3.2 Fine tuning

Since we have an off-the-shelf method for finding the ground truth given a randomly generate problem we can fine tune both the variables that we include in the objective functions, by for example including more than $n - m$ variables, or adjusting the parameters p_i to generate different problems, ideally with a smaller number of local optima, as this makes it easier to compare the numerical methods we will apply during optimization, since there are fewer points the methods can possibly converge towards.

6.3.3 Reformulation

Based on the output set by the original matching M we generate different sets of MDF and IDF/AAO formulations, which determine the baseline performance.

The MDF formulation will consist of an implicit component at the very top that solves the optimization problem where the design variables correspond to the variables that are inputs in the graph. It will consist of an elimination order containing a mix of explicit and implicit components, where the implicit components correspond to strongly connected explicit components that have been merged, and where different elimination orders are used. The top level component does not contain any constraint functions.

The IDF formulation will consist of an implicit component at the very top with an elimination order containing some of the explicit components from the original model and with equality constraints for the residual transformation of the explicit components not in the elimination order.

We then apply our method which looks for matchings with few strongly connected components, and compare the different formulations that can be generated based on the explicit components that can be generated from the different outputset.

6.4 Optimization problem example instance

Before showing results from running a large set of random problems, we first show an example of one specific problem. We generate an initial synthetic random problem with $m = 10$ equations and $n = 13$ variables, shown in Eq. (6.4):

$$\begin{aligned}
\min \quad & (x_1 - 1)^2 + (x_3 - 1)^2 + (x_4 - 1)^2 \\
\text{subject to} \quad & 0.6x_{12}x_6 + 0.8x_3x_7 + 0.2x_4 + 0.5 = 0 \\
& 0.5x_1x_{13}x_9 - 0.9x_{10}x_{12} - 0.6x_{14} + 0.7 = 0 \\
& 0.3x_{11}x_{12} + 0.2x_2x_3x_5 - 0.8x_4x_8 - 0.2 = 0 \\
& 0.4x_1x_{14}x_7 + 0.3x_{11}x_8 + 0.4 = 0 \\
& 0.1x_{14}x_4 + 0.4x_6x_7 - 0.4 = 0 \\
& -0.3x_{14}x_6x_7 + 0.1x_3x_5 - 0.4 = 0 \\
& 0.8x_{13}x_9 - 0.2x_{14}x_3 - 0.5 = 0 \\
& 0.2x_2 - 0.8 = 0 \\
& 0.8x_0x_5 - 0.3x_{12}x_2 - 0.7x_9 - 0.1 = 0 \\
& -0.5x_1x_{11}x_8 - 0.9 = 0
\end{aligned} \tag{6.4}$$

Next we compare the different formulations.

6.4.1 SAND formulation

We start with the formulation that is independent of choice of outputs for a component, where all functions are equality constraints.

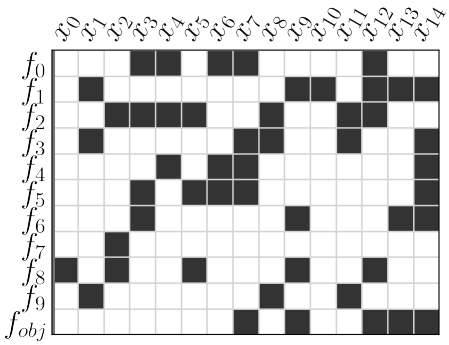


Figure 6-2: SAND

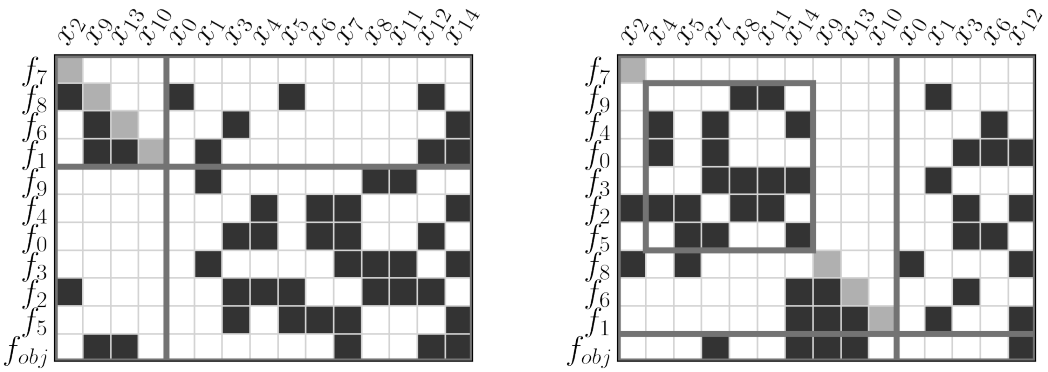
And the number of iterations as a function of how far away we start from the optimal solution point:

factor	Iterations	Optimal value
1.001	2	1.35×10^{-7}
1.01	5	5.33×10^{-6}
1.1	12	2.96×10^{-4}
10	37	2.4×10^{-2}

Table 6.1: SAND

6.4.2 block-IDF and block-MDF of initial formulation

For the block-IDF formulation, we reduce all the components in the groups of strongly connected components, which then become equality constraints in the optimization problem, whereas for the block-MDF we have a smaller optimization problem to solve, but where we need to solve a feasibility problem as part of every iteration of the optimization algorithm. As a result we get smaller problems to solve for (fewer variables that we optimize over). We expect the worse performance here, as we need to solve a significantly large system of equations for every iteration of the optimization. In Fig. 6-3 we show the HSM of both formulations.



(a) Block-IDF (b) Block-MDF

Figure 6-3: Initial formulations

We use a sequential least squared method given the large number of constraints, and set x_0 of the method at different factors from the known optimal point x^* based of the ground truth found through the homotopy method.

Table 6.2: Number of iterations required for sequential solving. In brackets number of iterations required to solve internal system of equations.

(a) block-IDF

factor	Iterations	Optimal value
1.001	3	1.22×10^{-7}
1.01	5	5.46×10^{-6}
1.1	15	2.97×10^{-4}
10	45	0.0235407

(b) block-MDF

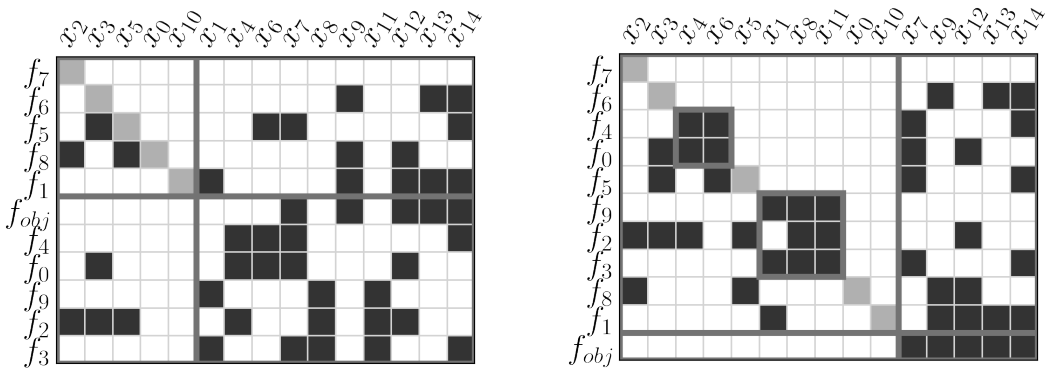
factor	Iterations	Optimal value
1.001	4 (32)	2.06×10^{-7}
1.01	6 (41)	6.53×10^{-6}
1.1	20 (299)	6.63×10^{-7}
10	4 (358)	12.25

We observe that as the initial guess goes further away from the true optimum (which was found from finding a solution to the polynomial equations satisfying $x_i = 1$ for all x_i in the objective function), the number of iterations increases significantly.

6.4.3 Optimal structure re-formulation

We optimize the structure to find the smaller strongly connected component re-formulation. The new input variables now become x_{12}, x_{17}, x_{19}. In this case, the

size of the largest strongly connected component is one, and we therefore do not need to merge any of the component, as is shown in the HSM in Fig. 6-4. This results in slightly smaller problems both for the block-IDF and block-MDF reductions.



(a) Block-IDF (b) Block-MDF

Figure 6-4: Optimized structure formulations

We again use sequential least squares method given the large number of constraints, and set x_0 of the method at different factors from the known optimal point x^* based of the ground truth found through the homotopy method.

factor	Iterations	Optimal value
1.001	3	2.37×10^{-7}
1.01	6	4.67×10^{-6}
1.1	14	2.86×10^{-4}
10	27	0.0235

Table 6.3: block-IDF

factor	Iterations	Optimal value
1.001	2 (9)	ϵ
1.01	2 (12)	ϵ
1.1	2 (14)	ϵ
10	2 (27)	ϵ

Table 6.4: block-MDF

Table 6.5: Results from optimized structure

6.4.4 Discussion

The SAND formulation seems to converge in fewer iterations when very close to the starting problem, but the reduced block-IDF of the optimal structure seesm to converge faster for larger factors, however, for factor 10 it does no longer get close

enough to the optimal solution. The block-MDF of the optimal structure always converges to within machine precision of the optimal value (0), as this should happen by construction. The iterations are instead spend converging the much smaller 4x4 system that can be seen in the HSM in Fig. 6-4b.

6.5 Results from random graphs

We generate 10 random structures, and 10 random sets of polynomial coefficients per structure, for a problem of size $m = 10, n = 15$ and average row density $\rho = 4$. We then compare the computational time, and also whether the problem converged or not.

Largest strongly connected components

Fig. 6-5 shows the distribution of largest strongly connected components across the different structures (with different random seeds).

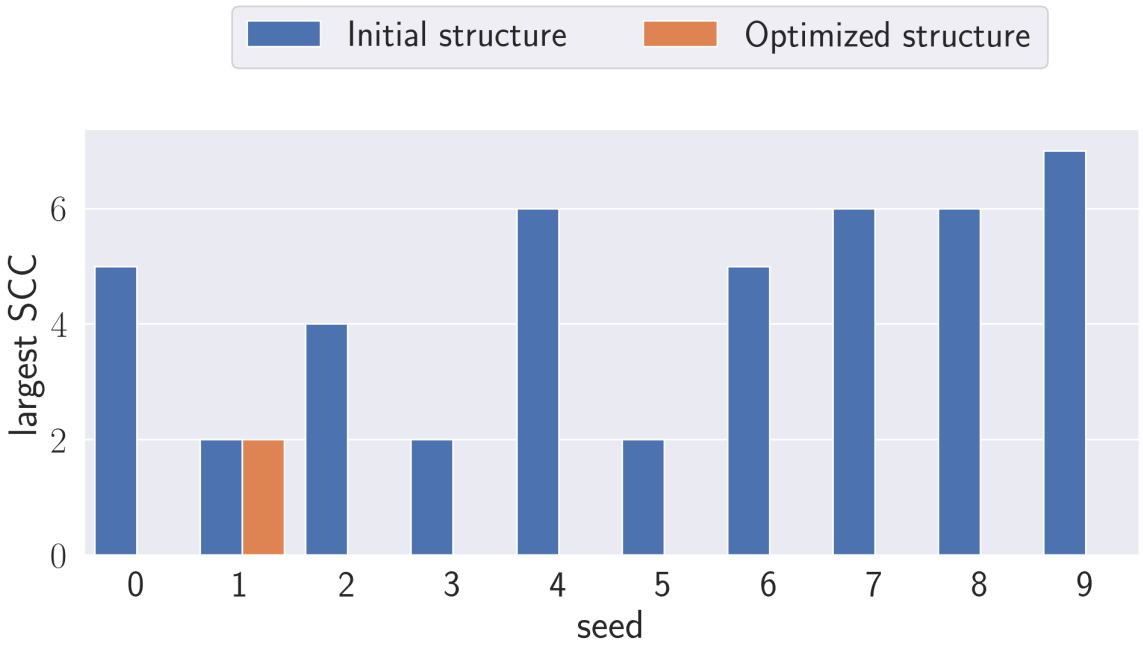


Figure 6-5: Distribution of largest strongly connected component

Only one out of the ten structures cannot be reduced to a feed-forward structure

(without any strongly connected components).

Convergence of optimization

Fig. 6-6 shows the number of problems that converged for the different formulations.

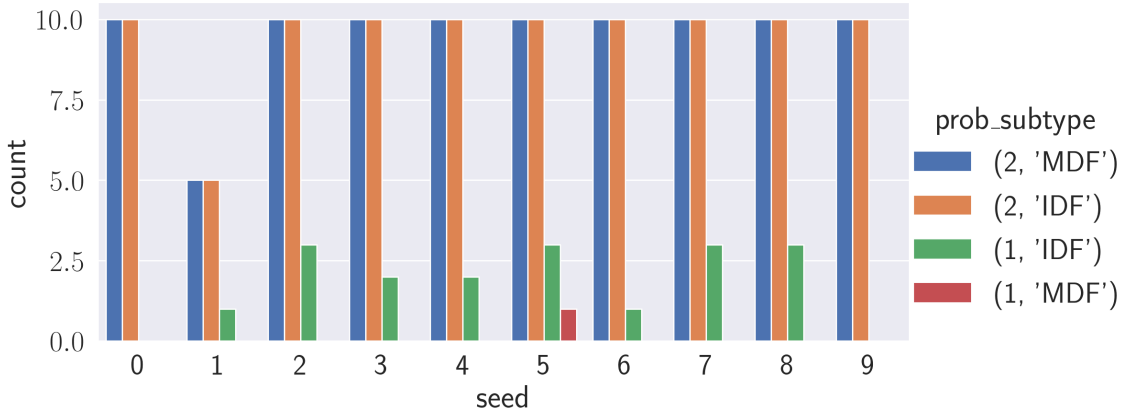


Figure 6-6: Convergence results, showing the number of components that converged to optimality for a fixed structure (given by a random seed).

As can be seen, all of the optimized structures converged when the resulting structure is feed-forward. This is as expected, since the objective function only depends on the input variables, and at this point the problem becomes equivalent to an unconstrained problem. When the optimal structure is not feed-forward, as was the case for the problem with random seed of 1, we see that the number of problems that converged out of all the random polynomial problems with the same structure is larger. For the problems with initial random structure, it seems like the optimizer struggles to achieve convergence, especially as the size of the largest strongly connected component goes up.

Chapter 7

Case Studies: Conceptual Design of Real Systems

In this chapter, we apply the methods developed in Chapter 5 to three real conceptual design problems: a spacecraft design problem, an aerial vehicle design problem, and a marine system design problem. This chapter aims to show the breadth of application of the method and that it applies across different application areas. For each case study, we compare a naive initial formulation in each section to formulations based on a restructuring generated by optimizing the graph structure. Mainly we focus first on the design problem, where we seek to restructure the problem to find feasible concepts; we apply this to all three problems. Next, we optimize the mass of each system and add physical constraints that are different for each of the cases. We again optimize the initial structure and the optimized structure. For both problems, we compare the number of guess variables required and the sensitivity of the result to initial variables. Finally, we use an off-the-shelf sequential-least square programming algorithm for constrained optimization.

7.1 Spacecraft conceptual design

This case study is based on a model of an Earth-observation satellite developed and previously reported in [56] and [57]. A notional representation is given in Fig. 7-1: the spacecraft has an imaging payload that images a subsection of its field of view. As it orbits around the Earth it sweeps out a cross section of the Earth's surface, which is stored as data and relayed to a ground communications station.

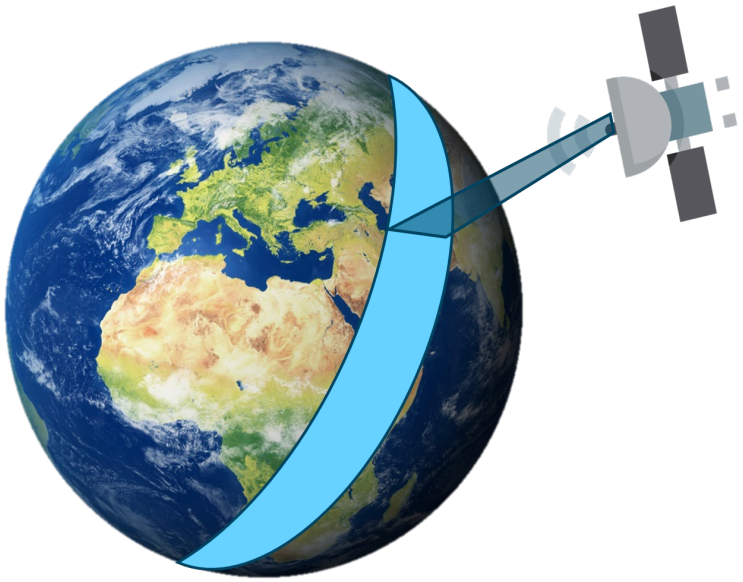


Figure 7-1: Notional representation of an imaging spacecraft.

The model is a simple first-order model that captures the key sizing equations of a spacecraft at the conceptual design level, and is not specific to a particular real spacecraft or ongoing design project; therefore the parameters used in the problem are set to realistic values, but are not representative of an existing system. We first describe the sizing equations, partitioned into their conventional disciplines: *mission design*, *power budget*, *payload design*, *link budget* and *propulsion*.

We use standard sizing relationships to generate an *initial formulation*, based on the variables that appear on the left hand-side of every equation.

7.1.1 Sizing relationships and initial formulation structure

Mission design module

Given a circular orbit altitude h, the semi-major axis of the orbit is defined as

$$a = R_e + h, \tag{7.1}$$

where $R_e = 6738\,\text{km}$ is the radius of the Earth. Then the period T is defined as:

$$T = 2\pi\sqrt{\frac{a^3}{\mu}} \tag{7.2}$$

where $\mu = 3.98 \times 10^{14}\,\text{m}^3/\text{s}^2$. The ground station communication time g is given by the relationship:

$$g = \frac{1}{\pi}\arccos\left(\frac{R_e}{R_e + h}\right) \tag{7.3}$$

The fraction of the orbit that the satellite spends out of eclipse is given by a similar expression and can therefore be recovered from g:

$$d = g + 0.5 \tag{7.4}$$

Power budget module

We model two elements: a solar panel for generating electrical energy, and a battery to store the energy during eclipses. Given a solar panel of efficiency η_s, and area A, the solar panel charging power during sun exposure will be:

$$P_t = dA\eta_s Q \tag{7.5}$$

where $Q = 1367\,\text{W}/\text{m}^2$ is the solar constant, and d is the out-of-eclipse orbit fraction as already defined.

The amount of energy stored in the battery is defined as a function of previously

defined variables:

$$E_b = \frac{1}{d} P_t T \tag{7.6}$$

Observation Payload module

The payload is assumed to be an Earth-imaging camera, and we assume that its resolution is diffraction limited. This gives the following sizing relationship that determines the payload resolution:

$$X_r = 1.22 h \frac{\lambda}{D_p} \tag{7.7}$$

where X_r is the payload resolution, h is the satellite's altitude, λ the imaging wavelength and D_p the aperture diameter of the camera.

Communications module link budget

We will assume that over the duration of one full orbit we need to downlink all the data that has been gathered by the payload. This is a very over-simplified assumption for the purposes of the toy example while it also serves as a worst case scenario. Assuming the payload is always recording a strip underneath it with length $N_{\text{px}} \cdot X_r$ where X_r is the resolution of the payload and N_{px} is the pixel width of the image sensor, the total data D recorded then is just the amount of pixels needed to represent a full sweep of the Earth, i.e. $\frac{2\pi R_e}{X_r}$ times the width N_{px}, times the amount of bits per pixel B, which is set as a parameter:

$$D = \frac{2\pi R_e}{X_r} B N_{\text{px}} \tag{7.8}$$

All this data needs to be downloaded during a ground station pass which lasts $d \cdot T$, so the bit rate required during a download b then becomes:

$$b = \frac{D}{g \cdot T} \tag{7.9}$$

The link budget to estimate the signal to noise ratio E_b/N_o gives the following relationship:

$$E_b/N_o = \frac{P_{Tx} \cdot G_{Rx} \cdot G_{Tx}}{L_{other} \cdot k \cdot T_{sys} \cdot b} \left(\frac{\lambda_c}{4\pi r}\right)^2 \tag{7.10}$$

$$D_{Tx} = \frac{\lambda_c}{\pi}\sqrt{\frac{G_{Tx}}{\eta}} \tag{7.11}$$

$$r = \sqrt{h^2 + 2R_e h} \tag{7.12}$$

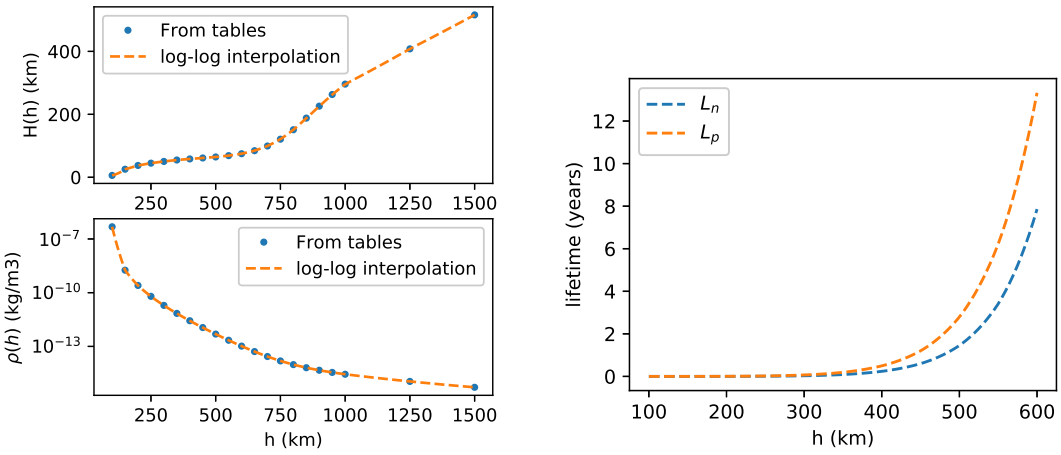
Where r is the distance from the satellite to the furthest most distant point within its field of view, which is taken as the worse case constraint for the design.

Total lifetime and propulsion module

Total lifetime L_t is the sum of two parts, the natural decay time due to drag L_n, and the time we can keep the satellite from decaying by countering the drag with propulsion, L_p. The simplifying assumption is made that drag is countered continuously, whereas in reality this happens in discrete burns.

$$L_n = \frac{H(h) \cdot m_t \cdot T}{2\pi \cdot C_D \cdot A \cdot \rho(h) \cdot a^2} \tag{7.13}$$

Where $H(h)$ and $\rho(h)$ are given by tables at the end of [58], which have been digitized and rendered in Fig. 7-2a.



(a) Atmospheric properties according to NRLMSISE-00 model.

(b) Propulsive model for $m = 2\,\text{kg}, m_P = 0.2\,\text{kg}, C_D = 2.2, A = 0.05\,\text{m}^2, Isp = 70\,\text{s}$.

Figure 7-2: Empirical models used for propulsion module of spacecraft model

For the propulsion lifetime, it is the total momentum available from propulsion, $l = m_P I_{sp} g$ divided by the drag force.

$$F_D = 0.5\rho C_D A v^2 = 0.5\rho C_D A \frac{\mu}{a} \tag{7.14}$$

$$l = m_P I_{sp} g \tag{7.15}$$

$$L_p = \frac{l}{F_D} \tag{7.16}$$

To get an understanding of the order of magnitude of L_p relative to L_n we calculate L_p as a function of orbit altitude for parameters with typical values for a small CubeSat; the relative magnitudes can be seen in Fig. 7-2b. We then recover the value for total lifetime, L_t:

$$L_t = L_n + L_p \tag{7.17}$$

Mass module

Mass models use a simple empirical relationship that was tuned to generate meaningful masses. Mass of the battery and the solar panel follow a linear law:

$$m_i = \rho_i \cdot f_i \tag{7.18}$$

Where m_i is the mass of certain component, ρ_i is the mass density, and f_i is some performance factor, like the solar panel area A for a solar panel. Mass models for circular components: the payload and the diameter follow a 1.5 power law:

$$m_i = \rho_i \cdot f_i^{1.5} \tag{7.19}$$

Table 7.1: Summary of density parameters for mass models

Quantity	Value	Description
ρ_T	$0.2\,\mathrm{kg/m^{1.5}}$	Mass coefficient for antenna
ρ_p	$2\,\mathrm{kg/m^{1.5}}$	Mass coefficient for payload
ρ_A	$10\,\mathrm{kg/m^2}$	Mass surface density for solar panel
ρ_A	$2\,\mathrm{g/kJ}$	Mass density for battery

We use an empirical model of the structural mass:

$$m_s = \eta_s m_t \tag{7.20}$$

Where m_t is the total mass defined below

$$m_t = \textstyle\sum_i m_i \tag{7.21}$$

Parameters

Table 7.2 shows a summary of the values of the parameters used.

Table 7.2: Summary of fixed parameters for the current satellite design example

Quantity	Value	Description
μ	$3.986 \times 10^{14}\,\mathrm{m}^3/\mathrm{s}^2$	Standard gravitational parameter
R_E	$6378\,\mathrm{km}$	Earth(spherical) radius
Q	$1367\,\mathrm{W}/\mathrm{m}^2$	Solar constant
k	$1.381 \times 10^{-23}\,\mathrm{J/K}$	Boltzmann constant
λ_c	$0.13\,\mathrm{m}$	Data link wavelength
G_{Rx}	$39.1\,\mathrm{dB}$	Receiver gain
T_{sys}	$150\,\mathrm{K}$	Noise temperature
η	0.55	Antenna efficiency
λ_v	$500\,\mathrm{nm}$	Optical payload wavelength
N_{px}	2000	Pixel width of image sensor
B	32	bits per pixel
C_D	2.2	Drag coefficient
I_{sp}	70s	Specific impulse of propulsion
m_c	$0.2\,\mathrm{kg}$	Constant mass
η_S	$20\,\%$	Structures subsystem mass percentage

7.1.2 Design analysis problem

In the analysis problem we seek to find a parametrization of the problem that we can use to generate valid designs. We compare the parametrization based on the design equations given in Section 7.1.1, to parametrizations generated to simplify the model.

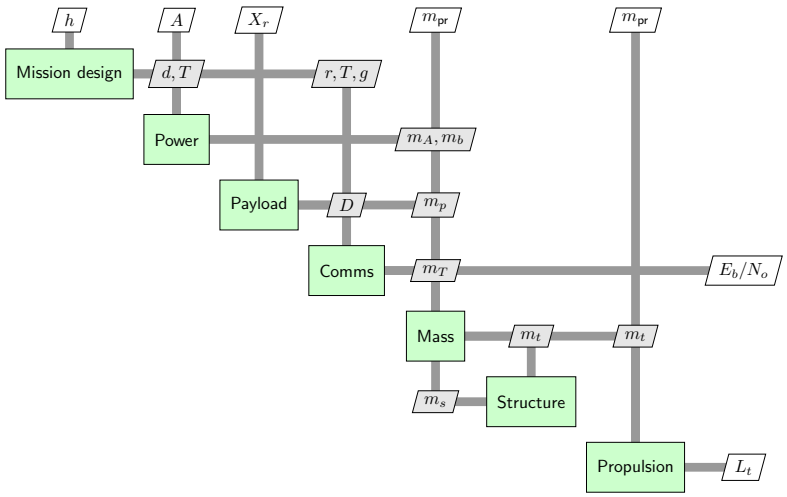


Figure 7-3: XDSM of modules in spacecraft conceptual design model. The order of the modules is based on the order of given in Section 7.1.1.

In Fig. 7-3 we show the XDSM of the modules in the initial formulation. The model is almost feed-forward, with only one feedback loop between the mass and the structure model. This would require either one or two guessing variables depending on the method we used for converging the initial model. There are four inputs to the entire model: m_{pr}, X_{req}, h and A, and one output L_t. A more granular visualization of the full model is given using the HSM in Fig. 7-4.

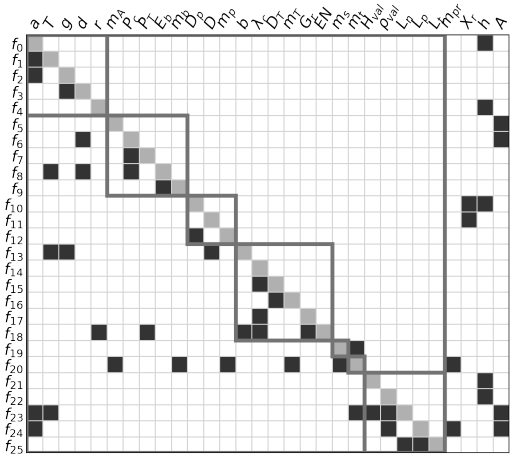


Figure 7-4: HSM of satellite model

Table 7.3 shows the results of one analysis run, that took 9 iterations to converge.

Table 7.3: Analysis inputs (including guesses on the left) and key output values on the right

Quantity	Value	Unit
A	0.05	m^2
h	400	km
m_{pr}	0.5	kg
X_{req}	5	m
$m_t^{(0)}$	0.1	kg
$m_s^{(0)}$	0.1	kg

Quantity	Value	Unit
$E_b/N_{o_{req}}$	9.86	dB
L_t	1.43	years
m_t	1.62	kg
m_s	0.32	kg

Since there is only one loop it makes the task of re-parametrization very simple in this case. We can generate a re-parametrization of the problem by optimizing the assignment. One feasible assignment that results in no cycles is shown in the HSM in Fig. 7-4:

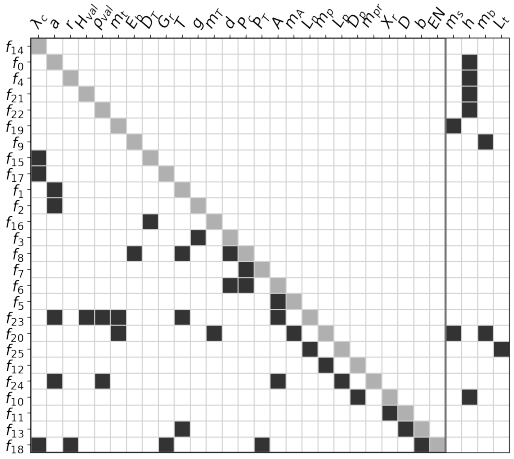


Figure 7-5: HSM of re-structured satellite model

In this case we only need to provide initial values for L_t, m_s, h and m_b, and no guess variables, and the results can be generated without the need for any iteration:

Table 7.4: Analysis inputs (inculding guesses on the left) and key output values on the right

Quantity	Value	Unit
L_t	1.4	years
h	400	km
$m_t^{(0)}$	0.3	kg
$m_s^{(0)}$	0.3	kg

Quantity	Value	Unit
$E_b/N_{o_{req}}$	20.2	dB
m_t	1.5	kg
A	0.066	m^2

One of the things that was noticed during this second formulation is that the model would lead to strange outputs: turns out that for certain combinations of inputs, the total mass m_t became negative, which in turn affects all downstream values, and generated non-sensical values.

7.1.3 Design problem

We now add a set of constraints to the design: we seek to find feasible designs that satisfy constraints on a required E_b/N_o, a specific lifetime, and resolution. We add three end components to the model, that in this problem are interpreted as equality constraints:

$$L_t = L_{min} \quad \Leftrightarrow g_1(L_t, L_{min}) = L_t - L_{min} \tag{7.22}$$

$$X_r = X_{req} \quad \Leftrightarrow h_1(X_R, X_{req}) = X_r - X_{req} \tag{7.23}$$

$$E_b/N_o = E_b/N_{o_{req}} \quad \Leftrightarrow h_2(E_b/N_o, E_b/N_{o_{req}}) = E_b/N_o - E_b/N_{o_{req}} \tag{7.24}$$

We next solve the minimum reduction assignment problem to find a re-arrangement of the problem that could reduce the design problem into an analysis problem.

We solve the restructuring problem and get:

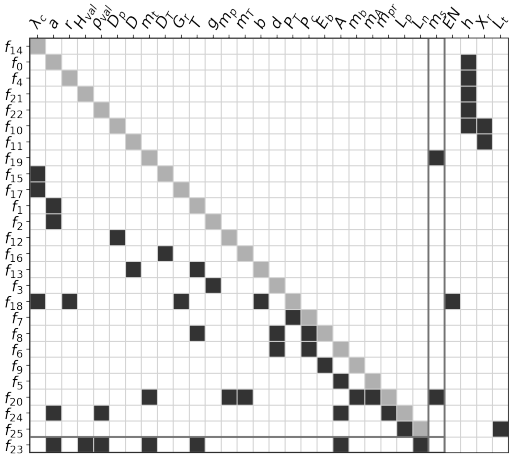


Figure 7-6: HSM of satellite design model to ensure variables in design constraints are inputs

We plug in the values for a set of fixed requirements L_{min}, X_R, and $E_b/N_{o_{req}}$ as given below, and only required 2 iterations to converge.

Table 7.5: Design results

Quantity	Value	Unit
$L_t = L_{\min}$	10	years
h	400	km
$E_b/N_o = E_b/N_{o_{req}}$	11.5	dB
$X_R = X_{\text{req}}$	5	m/px
$m_s^{(0)}$	1	kg

Quantity	Value	Unit
$E_b/N_{o_{req}}$	20.2	dB
m_t	1.5	kg
A	0.066	m^2

7.1.4 Design optimization problem

We create the design problem, which consists in minimizing the mass of the spacecraft, m_t, subject to the constraints seen in the design subsection. Since X_R is an input to the model, we can simply eliminate this constraint by holding X_R fixed; we also hold h fixed, which reduces the problem to a two-dimensional problem over the solar panel size A and the propulsion mass m_{pr}. The two constraints are again shown below

$$L_t \leq L_{min} \qquad \Leftrightarrow g_1(L_t, L_{min}) = L_t - L_{min} \tag{7.25}$$

$$E_b/N_o \leq E_b/N_{o_{req}} \qquad \Leftrightarrow g_2(E_b/N_o, E_b/N_{o_{req}}) = E_b/N_o - E_b/N_{o_{req}} \tag{7.26}$$

We plug in the values for a set of fixed requirements L_{min}, X_R, and $E_b/N_{o_{req}}$ as given below, and get the optimal point shown in Table 7.6

Table 7.6: Satellite optimization results

Quantity	Value	Unit
$L_{\min}$	10	years
h	400	km
$E_b/N_{o_{req}}$	11.5	dB
X_{req}	5	m/px
$A^{(0)}$	1	m^2
$m_{pr}^{(0)}$	1	kg
$m_s^{(0)}$	1	kg

Quantity	Value	Unit
A^*	0.05	m^2
m_{pr}^*	3.84	kg
m_s^*	1.16	kg
m_t^*	5.8	kg

The timing performance of the optimization of the different structures are shown in Table 7.15:

Table 7.7: Spacecraft design optimization runtime with different structures

Structure	Formulation	Function iterations	Runtime (s)	Objective (kg)
Baseline	Block reduced	8	7	5.8
Baseline	Bordered reduced	8	6.8	5.8
Restructured	Bordered reduced	2	0.7	5.8

In this case the optimizer converges quickly on the optimal value for all cases (less than 10 seconds), but the restructured problem solves an order of magnitude faster than the other structures.

7.2 High altitude balloon design

High altitude balloons are normally used for weather and climate surveying applications. Recently, however, there have been proposals of applying these balloons for high altitude tourism, as a much cheaper alternative to space tourism. Current high altitude balloons are designed for light payloads, and for this case study we are interested in exploring the design space for significantly heavier payloads due to the space tourism passengers. The entire vehicle consists of three main subsystems: a passenger module, a high altitude balloon and a paragliding wing. The balloon provides the lift for the passenger module to reach altitude and the paragliding wing enables a much more controlled descent than the balloon itself or a parachute. The passenger module hosts the passengers, and comprises the structure to survive the environment the system encounters over its flight path, life support systems, infrastructure for the passengers and its crew and hosts all the electronics and safety equipment.

Figure 7-7: Artist rendition of high altitude balloon concept

For conceptual design purposes we focus on the sizing of the vehicle at the subsystem level: sizing the radius of the balloon at sea level, i.e. standard pressure and temperature (STP) conditions, and at apogee, i.e. the highest point of flight. We model the lifting gas as a mixture between helium and hydrogen, with α the proportion of helium in the mix, i.e. $\alpha = 1$ would correspond to a balloon based on helium, and $\alpha = 0$ a balloon based purely on hydrogen. We also size the masses of the passenger module. Particularly we are interested in sizing the vehicle as a function of different passenger module weights.

7.3 Sizing relationships

Balloon at apogee

We assume that the balloon has reached its maximum altitude, and that the lifting buoyancy forces L_z therefore match the weight of the balloon at apogee W_z:

$$W_z = g(z)m_t \tag{7.27}$$

$$L_z = W_z \tag{7.28}$$

where $g(z)$ is the gravitational constant at altitude (at higher altitudes this will be slight less than $9.81\,\mathrm{m/s^2}$), m_t is the total mass of the vehicle excluding lifting gas, and m_t is the total mass, including lifting gas, of the balloon. We then recover the volume V_z and mass m_{req_z} of lifting gas required to provide the force L_z at this altitude:

$$V_z = \frac{L_z}{g(z)\rho(z)} \tag{7.29}$$

$$m_{req_z} = V_z \rho_{LG}(z) \tag{7.30}$$

Where $\rho(z)$ is the density of the atmosphere at apogee, and $\rho_{LG}(z)$ the density of the lifting gas at apogee. We model the air density through tables generated based on the standard model, and interpolating based on the altitude z of the vehicle. The density

of the lifting gas can similarly be recovered based on the pressure and temperature conditions of the environment, i.e. the air, at altitude z:

$$\rho_{LG}(z) = \frac{P(z)}{kT(z)}(\alpha m_{He} + (1-\alpha)m_{H2}) \tag{7.31}$$

Where $P(z), T(z)$ are the standard model atmospheric air pressure and temperature at altitude z, m_{He} the molecular weight of helium, and m_{H2} the molecular weight of hydrogen.

Balloon materials

Based on the volume of the balloon at apogee we can estimate the dimensions of the balloon: the radius r_z and the height h_z. We can use this to derive the total surface area of the balloon, assumed elliptical for these calculations.

$$r_z = \frac{3}{4\pi}V_z^{1/3} \tag{7.32}$$

$$h_z = \frac{3}{2}r_z \tag{7.33}$$

$$S = \pi\left(\frac{1}{3}\left(r_z^{2p} + 2r_z^p \cdot h_z^p\right)\right)^{1/p} \tag{7.34}$$

where $p = 8/5 = 1.6$. We can then recover the mass of the balloon material m_b

$$m_b = 3St_m\rho_m \tag{7.35}$$

Where t_m is the thickness of the balloon material, and ρ_m the density of the material.

Mass budget

The total mass m_t, excluding lifting gas, comes from adding up the masses m_v of the vehicle excluding the balloon, m_b the mass of the balloon material, and whichever mass of lifting gas is largest: the mass of lifting gas at apogee or at STP. Further algebra would show that we always have $m_{req_z} < m_{req_0}$, but we assume that this is

not known from a simple modeling perspective.

$$m_t = m_v + m_b + \max(m_{req_z}, m_{req_0}) \tag{7.36}$$

Balloon at standard pressure and temperature

At STP, the three main forces acting on the vehicle are weight, lift and drag. Balloons generate lift from buoyancy, which comes from the volume of air displaced. At STP, if the lift is greater than the weight, the balloon will start raising which gives raise to the drag force. We assume that the balloon has reached steady state rise velocity v, meaning that the lift matches the drag and the weight. We can then recover the drag, and use it to calculate the characteristic area A_0 of the assumed spherical balloon:

$$L_0 = g\rho(0)V_0 \tag{7.37}$$

$$W_0 = gm_t \tag{7.38}$$

$$D = L_0 - W_0 \tag{7.39}$$

$$A_0 = \frac{2D}{C_D\rho(0)v^2} \tag{7.40}$$

Where $g = 9.81\,\mathrm{m/s^2}$ is the gravitational constant at sea level, i.e. for $z = 0$, C_D is the drag coefficient of a sphere, about 0.47 at the Reynolds number imposed at lower speeds of a few m/s and $\rho(0) = 1.225\,\mathrm{kg/m^3}$ is the density of air at STP conditions. The previous calculations are encapsulated into the *aerodynamics* module within the STP module.

We use A_0 to recover the radius of the balloon, and from this we recover both the radius r_0 and volume V_0 of the balloon at STP, from which we calculate the mass of

the lifting gas required at STP: m_{req_0}

$$r_0 = \frac{1}{\pi}\sqrt{A_0} \tag{7.41}$$

$$V_0 = \frac{4}{3}\pi r_0^3 \tag{7.42}$$

$$m_{req_0} = \frac{P(0)}{kT(0)}(\alpha m_{He} + (1-\alpha)m_{H2})V_0 \tag{7.43}$$

These calculations are encapsulated into the *geometry* module with the STP module.

Parameters

Table 7.8 shows a summary of the values of the parameters used for the balloon design case.

Table 7.8: Fixed parameters for high altitude balloon design example

Quantity	Value	Description
k	$1.381 \times 10^{-23}\,\mathrm{J/K}$	Boltzmann constant
R	$287.05\,\mathrm{J/K/kg}$	Individual gas constant for air
ρ_0	$1.225\,\mathrm{kg/m^3}$	Density of air at STP conditions
g	9.81	Gravitational constant at sea level
M_{H2}	$1.66 \times 10^{-27}\,\mathrm{kg}$	Molecular mass of hydrogen
M_{He}	$6.64 \times 10^{-27}\,\mathrm{kg}$	Molecular mass of helium
t_m	$0.0245\,\mathrm{mm}$	Balloon material thickness at apogee
C_D	0.47	Drag coefficient of balloon
m_v	$5045\,\mathrm{kg}$	Vehicle mass (excluding balloon and gas)
α	1	Helium fraction of lifting gas

7.3.1 Design feasibility problem

The coupling among the disciplines based on the initial formulation in Section 7.3 are shown in the block diagram in Fig. 7-8.

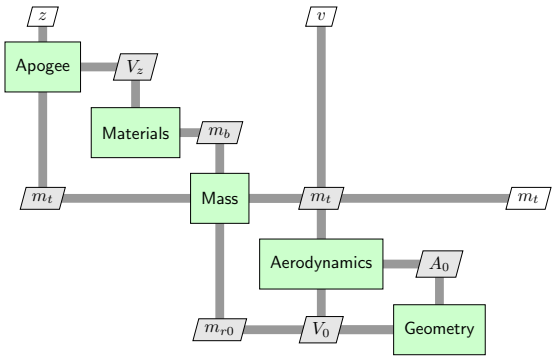
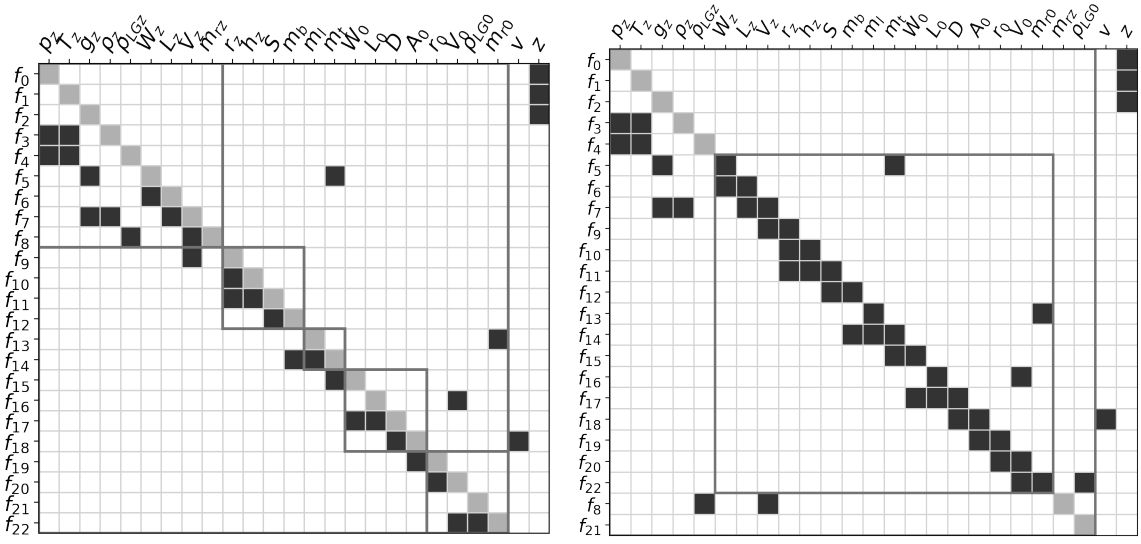


Figure 7-8: XDSM showing couple between modules in high altitude balloon model

The formulation has multiple cycles: there is a cycle between the geometry and the aerodynamic modules, and multiple cycles between the apogee, materials and mass module. The first design problem we seek to solve is finding feasible design candidates. We compare two methods: a block-MDF formulation based on the initial structure (ignoring modules), and a block-MDF methods based on the restructured problem, which we expect to have smaller blocks to solve, after applying the method from Section 5.3. We show the result of the block-MDF of the initial structure in Fig. 7-9.



(a) Initial formulation with modules (b) block-MDF formulation

Figure 7-9: HSM of formulations based on the initial structure

The largest strongly connected component of the initial graph means that the block-MDF structure has a large number of elements (17). In the worst case, we would have to provide 17 guessing variables, however we can reduce this to three feedback variables based on the current arbitrary ordering: m_t, V_0, m_{req_0}. The results of running the analysis based on the block-MDF formulation are shown in Table 7.9.

Table 7.9: Balloon design input and key output values for initial structure

Inputs

Quantity	Value	Unit
v	6	m/s
z	30	km
m_t	**200**	kg
m_{req_0}	1	kg
V_0	**4500**	m^3

Results

Quantity	Value	Unit
m_t	9719.8	kg
m_{req_0}	1677.6	kg
V_0	9393.1	m^3

Note that the initial values of m_t and V_0 are not arbitrary: they had to manually be selected in such a way for the Newton iteration method to converge the design equations of the block. V_0 would also have to be picked large enough in the beginning, this most likely has to do with the design equation $L_0 = g\rho(0)V_0$, which would lead to a negative drag in the case when $L_0 \leq W_0$, which in turn would result in a negative area, at which point the Newton iteration is not able to continue. With the guess in Table 7.9, it took 9 iterations for Newton's algorithm to converge.

In Fig. 7-10, we show the HSM of the restructured formulation, where we have optimize for smallest block, while allowing for inversions to happen with respect to inputs. We also restrict inversions to happen with respect to a specific set of variables: T_z, p_z, g_z, ρ_z, m_{rz}, m_{r0}, h_z, r_z, V_z, L_z, W_z, W_0. As a result we get an acyclic formulation with no solving variables; in the formulation found, the previous input v has been exchanged with the input m_t.

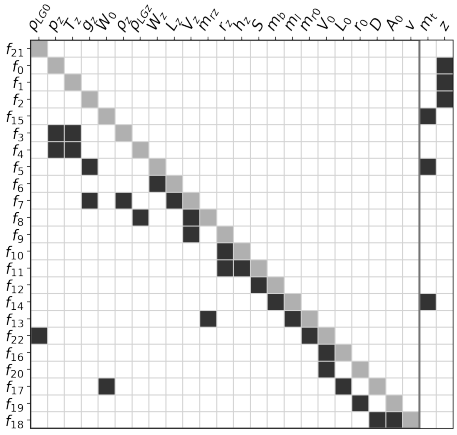


Figure 7-10: Restructured problem. Now m_t is an input instead of v.

We show the results of running the analysis based on the restructured formulation in Table 7.10, which can be solved in one feed-forward sweep.

Table 7.10: Balloon design input and key output values for initial structure

Quantity	Value	Unit
m_t	10000	kg
z	30	km

Quantity	Value	Unit
v	7.5	m/s
m_{req_0}	1900.5	kg
V_0	1.064×10^4	m^3

7.3.2 Design optimization problem

In this section we seek to minimize the mass of the vehicle for a given altitude; we expect this to happen at the lowest bound for v possible. For the initial formulation we therefore require to provide one constraint, $v_{min} \leq v$ and we further reduce the optimization problem to one variable only by fixing $z = z_0$. In this case it turns out that the optimization problem is very trivial, and that the problem converges within 2 iterations, and for every optimization iteration, only 1 iteration is needed to converge the block.

Table 7.11: Balloon design optimization, input and results for initial structure

Inputs

Quantity	Value	Unit
z	30	km
v	6	m/s
m_t	200	kg
m_{req_0}	1	kg
V_0	4500	m^3

Results

Quantity	Value	Unit
v^*	0.1	m/s
m_t	9317.3	kg
m_{req_0}	1358.5	kg
V_0	7606.5	m^3

7.4 Marine system

The Platform for Expanding AUV exploration to Longer ranges, or PEARL [59] is a concept for an offshore autonomous surface vehicle that will serve as a platform for other autonomous vehicles, like autonomous underwater vehicles (AUVs) or unmanned aerial vehicles(UAV). Current ocean exploration missions rely significantly on autonomous vehicles to reach hostile and challenging underwater environments. Many of the interesting ocean exploration destinations are far from the shore, and for AUVs going to significant depths and gathering large amounts of data the autonomy is limited to several hours. Additionally, missions are often planned in a reactive manner, where new data informs the destination for the next sampling activity, and untethered vehicles require recovery of the vehicle for recovering the data. As a result, the deployment and recovery of the autonomous vehicles themselves is still mainly operated from ships that navigate to the area of interest for exploration, manually launch AUVs, then recover them, offload the data gathered, and either recharge or swap the energy storage of the AUVs. Missions relying on this concept of operations involving ships are expensive to operate, costing up to $30,000/day [60], and mission durations can range over multiple weeks.

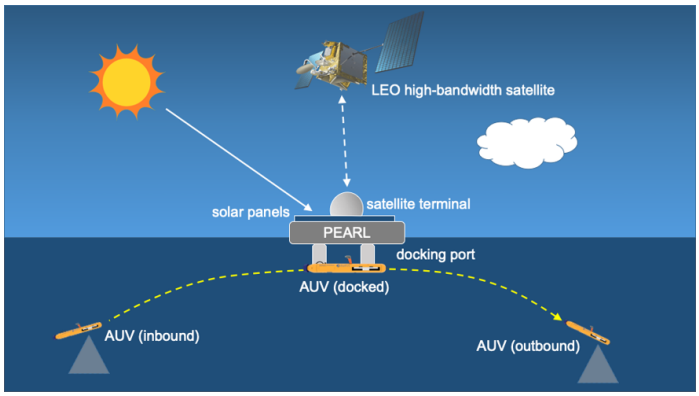


Figure 7-11: PEARL concept of operations with one AUV docking, recharging and offloading data, and undocking to continue its mission.

PEARL stands as a concept to automate the operations that so far have been carried out by crews by offering energy transfer and data-relay functionalities, not only

to AUVs, but potentially other autonomous vehicles like UAVs, or even surface autonomous vehicles(SAVs) like PEARL itself, that might be smaller and with a lower autonomoy than PEARL. Fig. 7-11 shows a use-case example with an AUV. A rendering of major elements of a PEARL prototype that has been built and tested in both fresh-water and salt-water under friendlier environmental conditions in terms of weather and sea, is shown in Fig. 7-12. Although physical prototypes of PEARL have been built, little analysis, design or optimization has been done to justify the current sizing decisions, beyond limited preliminary work [61]. In this case study we focus on the key preliminary sizing design of six major disciplines for conceptual modeling purposes: *geometry, power, mass, hydrodynamics, propulsion and communications*.

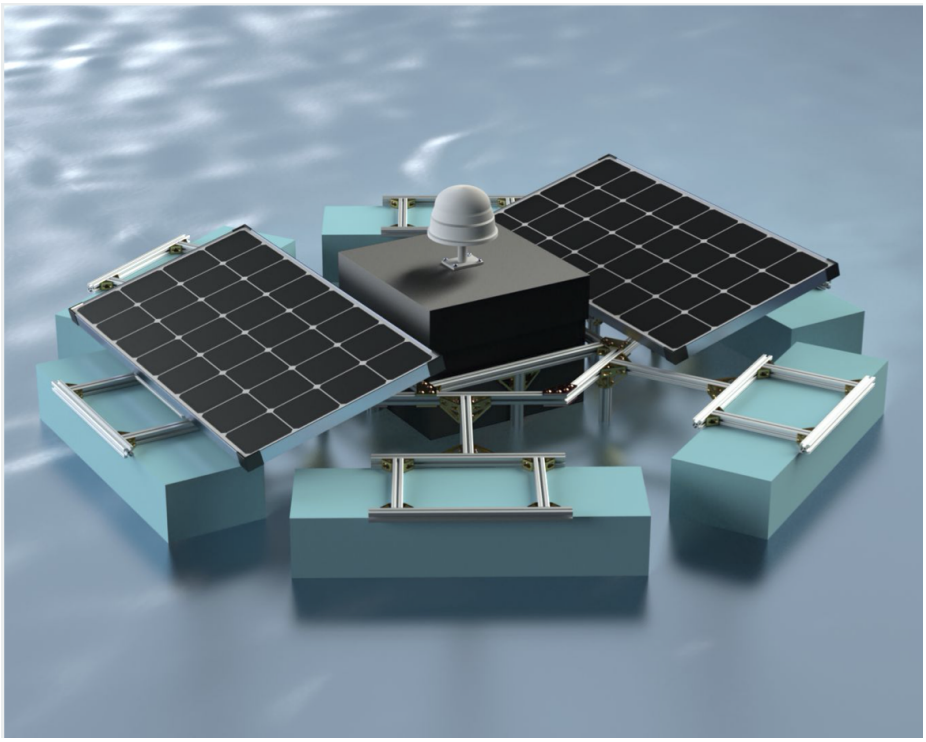


Figure 7-12: Rendering of a PEARL prototype, showing solar panels, satellite communications terminal payload box and flotation devices. Multiple elements like a damping plate and the propulsion subsystem are underwater.

We give a *naive* multidisciplinary formulation of PEARL. We call it *naive* as the focus was on modeling each discipline, but not in a way that would necessarily imply a better structure for answering multiple questions. We then show the following

capabilities based on the method presented in the thesis:

1. Transformation from the naive formulation to multiple valid formulations that can be solved and comparison in performance.

2. Reconfiguration of the original problem for generating design candidates.

3. Reconfiguration of the original problem for specific design questions.

7.4.1 Initial multidisciplinary formulation

The conceptual model presented here focuses on the use case of charging a number N of AUVs over a mission time $t_{mission}$. The model is used for sizing the batteries, the solar panels, the structure, the antenna and the engine of PEARL. These different subsystems are shown in Fig. 7-13. Although in reality most of the components will be off-the-shelf, we assume for purposes of this case study that the variables to be sized can be selected over a continuous range of values. For the solar panels the main sizing variable is the solar panel area A_s, for the battery it is the capacity C, for the antenna, the antenna diameter D_r, and for the engine, the power P_p.

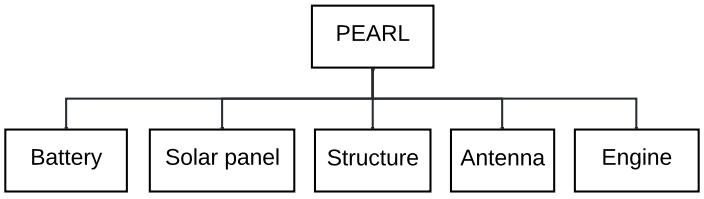


Figure 7-13: PEARL components sized

The sizing model has been artificially subdivided in six "disciplines", some of which correspond to the sizing of a specific subsystem, but others which are based more on the engineering field used for modeling. These disciplines are *geometry*, *communications*, *power*, *hydrodynamics*, *mass budgets* and *propulsion*. The couplings across the disciplines are visualized in Fig. 7-14.

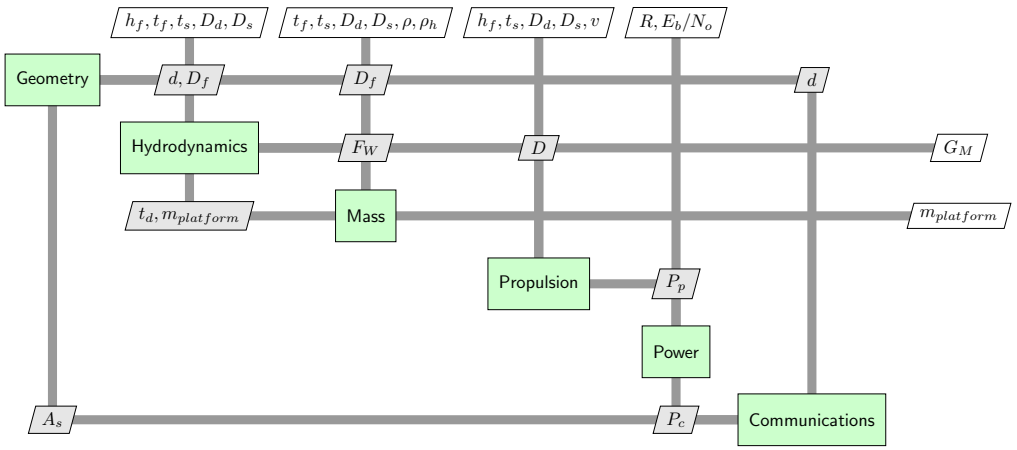


Figure 7-14: XDSM showing variable coupling among PEARL modules. The order of the modules is based on the order the modules are presented in this thesis.

Geometry

For simplifying purposes, PEARL is assumed to have a geometry consisting of a cylindrical float section of diameter D_f and thickness t_f, a cylindrical damping plate of diameter D_d and thickness t_d, and a cylindrical structural section of diameter D_s and length t_s. This intermediary section serves as a structural connection between the flotation and damping element.

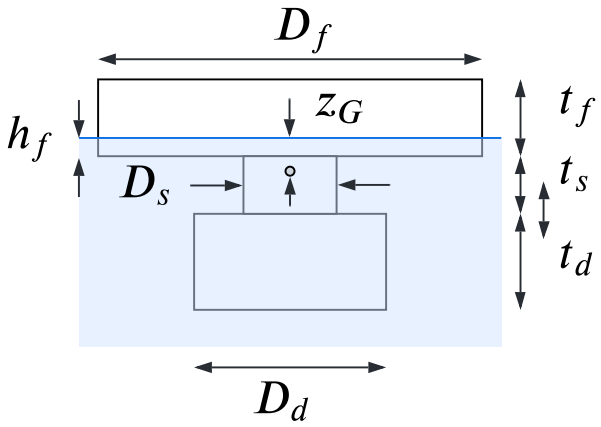


Figure 7-15: PEARL simplified geometry

Key quantities that are important for modeling purposes are also shown in Fig. 7-15: h_f the distance between the waterline and the bottom of the flotation element, and z_G

denotes the distance between the waterline and the center of mass along the z-axis. A center of mass below the waterline has a positive z_G value. For the multidisciplinary formulation, we assume that A_s is a given (by the power module), and we use a simplifying assumption that the surface area of the floating element is driven by the size of the solar panel and the antenna diameter D_r, which we define as being a percentage α of the total floater diameter.

$$D_f = \sqrt{\frac{4A_s}{\pi(1-\alpha)}} \tag{7.44}$$

$$D_r = \alpha D_f \tag{7.45}$$

Hydrostatics and heave response

The hydrodynamic model estimates two key performance properties of a marine system: the metacentric height, and the natural frequency of heave. These are values that we require to be within certain bounds: for static stability the metacentric height needs to be strictly positive, and for dynamic stability the natural frequency needs to be outside the natural frequency of the most common waves encountered at sea.

We start with modeling the buoyancy force F_B as a function of the height h_f of PEARL over the waterline.

$$F_B = \rho \forall \tag{7.46}$$

Where ρ is the density of saline water, $\forall$ the displaced volume, m_{total} the total mass of PEARL and g the gravitational constant. The displaced volume is given by

$$\forall = \frac{\pi}{4}\left(t_d D_d^2 + t_s D_s^2 + h_f D_f^2\right) \tag{7.47}$$

We then use h_f to calculate the center of buoyancy above the keel, KB. Due to the simplified geometry, the center of buoyancy is the center of gravity of the three rectangles with positions measured from the keel and areas:

element	damping	neck	floating
center of mass	$t_d/2$	$t_d + t_s/2$	$t_d + t_s + h_f/2$
area	$t_d D_d$	$t_s D_s$	$h_f D_f$

We then recover KB, the geometric center of the immersed volume as the weighted average of the centers of mass of each element:

$$KB = \frac{t_d D_d t_d/2 + t_s D_s(t_d + t_s/2) + h_f D_f(t_d + t_s + h_f/2)}{t_d D_d + t_s D_s + h_f D_f} \tag{7.48}$$

We can similarly get KG, the center of mass above the keel by using t_f in place of h_f in the previous calculation which gives:

$$KG = \frac{t_d D_d t_d/2 + t_s D_s(t_d + t_s/2) + h_f D_f(t_d + t_s + t_f/2)}{t_d D_d + t_s D_s + t_f D_f} \tag{7.49}$$

Finally, BM the distance from the centre of buoyancy to the centre of gravity is given by the following sizing relationship:

$$KM = I/\forall \tag{7.50}$$

Where I is the second moment of area of the water plan area and is given by

$$I = \frac{\pi}{4}\left(\frac{D_f}{2}\right)^2 \tag{7.51}$$

The metacentric height GM is then given by

$$GM = KB + BM - KG \tag{7.52}$$

This model therefore has two key variables that are coupled with other modules: the buoyancy force, which is used in the mass module, and the metacentric height, which

will be used as a design constraint.

Propulsion

The propulsion model calculates the power required to propel PEARL as a function of vehicle speed. At steady state speed v_s, the thrusting force F_T must equal the drag force F_D generated by PEARL:

$$F_D = \frac{1}{2} C_D v_s^2 \rho S_w \tag{7.53}$$

Where S_w is the wetted surface area of the platform, C_D is the drag coefficient of PEARL, and ρ the density of saline water. Due to the geometry of PEARL we model S_w as the sum of the wetted surface of each of the three elements of the geometry:

$$S_d = \pi(D_d/2)^2 - (D_s/2)^2 + (D_d/2)^2 + 2(D_d/2)t_d) \tag{7.54}$$

$$S_s = 2\pi(D_s/2)t_s \tag{7.55}$$

$$S_f = \pi((D_f/2)^2 - (Ds/2)^2 + 2(Df/2)hf) \tag{7.56}$$

$$S_w = S_d + S_s + S_f \tag{7.57}$$

From the steady state conditions we get $F_T = F_D$, and the power required for propulsion is then given by

$$P_{prop} = F_T v_s \tag{7.58}$$

We arbitrarily set $C_D = 1$, as a square plate at 90 degrees to the flow has $C_D = 1.17$.

Power budget

The energy budget model sums up all energy consumed by the system and sizes the solar panels to generate the required energy. It also sizes batteries for PEARL to store the energy required charging an AUV.

The total energy required is the sum of the energy required for each subsystem:

$$E_{hotel} = P_{hotel} t_{mission} \tag{7.59}$$

$$E_{service} = P_{service} t_{service} \tag{7.60}$$

$$E_{comms} = P_{comms} t_{comms} \tag{7.61}$$

$$E_{prop} = P_{prop} t_{prop} \tag{7.62}$$

$$E_{req} = E_{hotel} + E_{services} + E_{comms} + E_{move} \tag{7.63}$$

Where P_{hotel} is the fixed estimated power draw of all components that are not captured in higher detail at the conceptual modeling stage. The required energy needs to matched the produced energy by the solar panels, resulting in

$$P_{solar} = E_{req} / t_{recharge} \tag{7.64}$$

Where $t_{recharge}$ is the duration of the day, or the recharge session, at the mission location, and P_{solar}, the power that must be generated by a solar panel. From this in term we can calculate the solar panel area required:

$$A_s = P_{solar} / \left(\eta Q \cos\left(\bar{\theta}\right) I_{\text{deg}} (1 - d_{\text{deg}})^L \right) \tag{7.65}$$

Where η is the solar cell efficiency, Q is the incident solar irradiance, $\bar{\theta}$ the average solar angle, I_{deg} the inherent degradation, which describes the fraction of solar panel area that is actually solar cells, d_{deg} the solar cell degradation per year, L the lifetime of the solar panels and A_s the solar panel area.

Finally, the battery capacity required can be sized as follows:

$$C = \frac{E_{req}}{(DOD) N \eta_{battery}} \tag{7.66}$$

Where DOD is the depth of discharge of the battery, $\eta_{battery}$ the transmission efficiency between the battery and the load.

Mass budget

The mass of the platform $m_{platform}$ must have a weight equal and opposite to the buoyancy force determined in the hydrodynamics module:

$$m_{platform} = F_B/g \tag{7.67}$$

where $g = 9.81\,\mathrm{m/s^2}$ is the gravitational constant.

Masses of each subsystem are given below:

$$m_{battery} = NE_{battery}/\mu_{battery} \tag{7.68}$$

$$m_{solar} = \rho_S A_s \tag{7.69}$$

$$m_{comms} = \rho_C D_c^2 \tag{7.70}$$

$$m_{prop} = \rho_P P_{move} \tag{7.71}$$

$$m_{structure} = \frac{\pi}{4}\left(\rho_d t_d D_d^2 + \rho_f t_s D_s^2 + \rho_f h_f D_f^2\right) \tag{7.72}$$

where ρ_f is the density of the material of the flotation element, ρ_d the density of the material of the neck and the dampening plate. ρ_S, ρ_C, ρ_P are different constants for sizing the mass of the different parameters as a function of their key properties. We use the mass of the platform given from the buoyancy force to size the thickness of the damping structure t_d:

$$m_{structure} = m_{platform} - m_{battery} - m_{solar} - m_{comms} - m_{prop} \tag{7.73}$$

$$t_d = \frac{4}{\pi} \cdot \frac{m_{structure} - \rho_f D_f^2 t_f - \rho_f D_s^2 t_s}{\rho_d D_d^2} \tag{7.74}$$

Communications

We use the same sizing equations as in the first case study, except this time we are sizing the diameter of the receiving antenna. We use the link budget equation similar

to the spacecraft study, except in this case the variable we are sizing is the diameter of the antenna.

$$E_b/N_o = \frac{P_{trans} \cdot G_{trans} \cdot T \cdot d}{L_{other} \cdot k \cdot T_{sys} \cdot b} \eta \left(\frac{D_r}{4r} \right)^2 \tag{7.75}$$

$$r = \sqrt{h^2 + 2R_e h} \tag{7.76}$$

Where E_b is the energy per bit, N_o is the noise spectral density, P_{trans} is the transmitter power, L_{lumped} represented the lumped losses gain in the link, T_{sys} is the system noise temperature, k is Boltzmann constant, R is the data rate, D_r is the size of the platform antenna, G_{trans} the transmitter gain, r is the distance from the satellite to the furthest most distant point within its field of view, which is taken as the worse case constraint for the design, which is a function of h the altitude of the orbit of the satellite constellation and R_e the radius of the Earth.

Parameters

Table 7.12 summarizes key parameters used in the model, except for the communication system, where the parameters from the communications module in the first case study are reused.

Table 7.12: Summary of fixed parameters for the current satellite design example.

Quantity	Value	Description
α	0.05	Fraction of PEARL dedicated to antenna
ρ_f	$700\,\mathrm{kg/m^3}$	Density of flotation element
ρ_d	$2700\,\mathrm{kg/m^3}$	Density of dampening element
ρ	$1023.6\,\mathrm{kg/m^3}$	Sea water density
Q	$1367\,\mathrm{W/m^2}$	Solar constant
I_{deg}	0.9	Inherent degradation
$\bar{\theta}$	55 deg	Solar constant
t_{move}	1 hr	Time spent moving
$t_{service}$	12 hr	Time spent recharing AUVs
$t_{recharge}$	12 hr	Time when solar panels can charge
t_{comms}	1 hr	Communication time
$t_{mission}$	24 hr	Mission time
$\eta_{battery}$	0.85 J/K	Battery efficiency
$\mu_{battery}$	200 Whr/kg	Battery energy mass density
N	1	Number of recharging AUVs
E_{AUV}	1.9 kWh	Energy required for recharging AUV

7.4.2 Design analysis problem

The HSM of the initial formulation is given in Fig. 7-16, showing multiple feedback couplings in the upper right corner. Note that this is not a valid formulation, but we can make it valid by applying multiple residual transformation based on the feedback variables: $m_{platform}$, t_d, A_{solar} and $m_{battery}$.

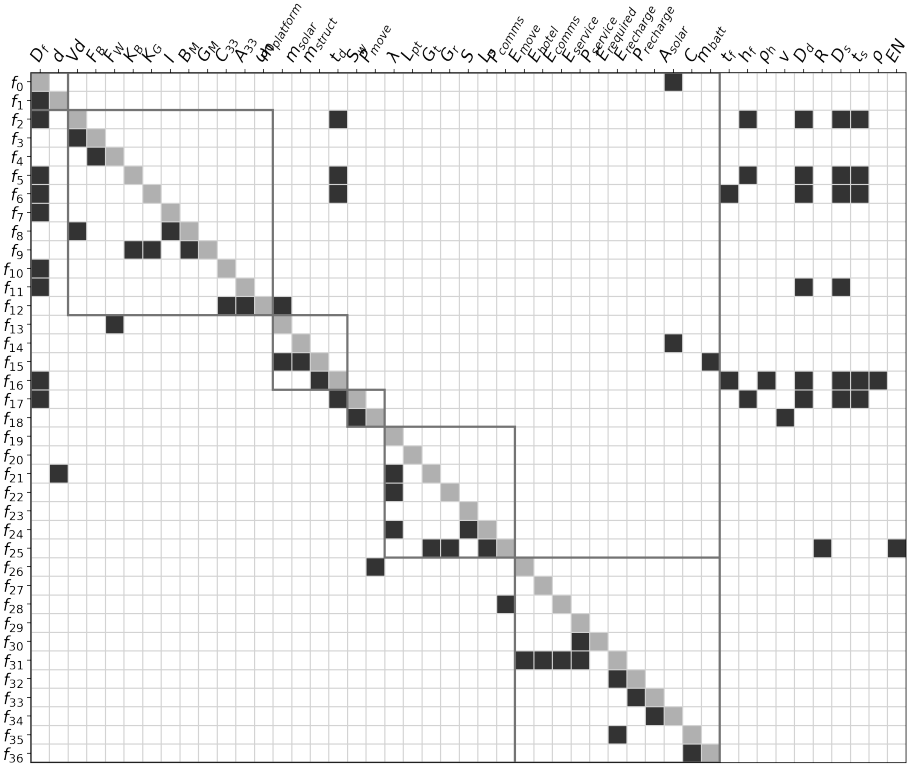


Figure 7-16: HSM of initial multidisciplinary formulation

We show the results from running the design analysis problem:

Table 7.13: Analysis results

Quantity	Value	Unit	Quantity	Value	Unit
D_s	0.25	m	D_f	3.4	m
D_d	0.3	km	$m_{platform}$	1695.5	kg
t_f	0.2	kg	t_d	10	cm
t_s	0.1	kg	m_{struct}	1410	kg
h_f	0.18	m	A_{solar}	8.7	m^2
R	10	Mbit/s	G_M	4	m
EN	10	dB			
v	1	m/s			

After re-parametrizing the problem we get the following results:

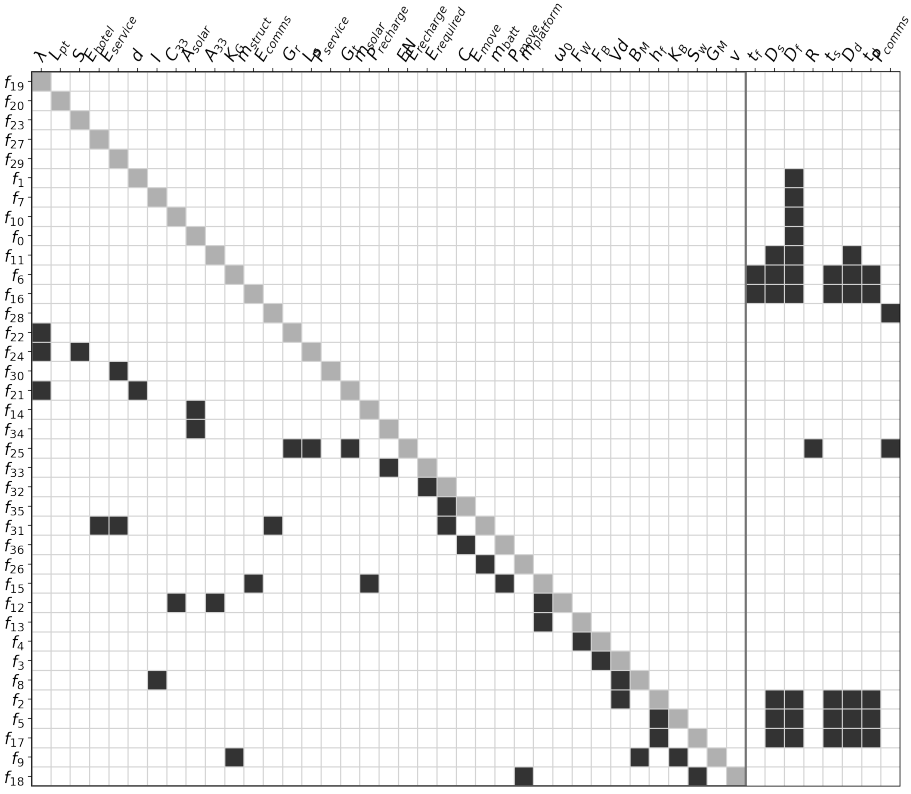


Figure 7-17: HSM of restructured problem used for optimization

7.4.3 Design Optimization problem

We aim to minimize the platform mass $m_{platform}$. The design variables were restricted to the geometric design variables D_f, D_s, D_d and t_f, t_s, t_d, and the remaining input variables P_{comms} and R were fixed. With the new formulation it turns out we need to add further constraints, for the optimizer not to search over invalid design points. These were discoverd to be:

$$h_f \leq 0.9t_f \tag{7.77}$$

$$D_s \leq 0.9D_f \tag{7.78}$$

$$D_s \leq 0.9D_d \tag{7.79}$$

$$P_{move} \geq 0 \tag{7.80}$$

The optimization was carried out with an internal point method, as the SLSQP method was failing to converge for the baseline structure of the problem.

The results from the optimization are shown in Table 7.14, where the lower, initial guessing value as well as optimal and upper values for the different design variables are shown. On the right hand side the value of key quantities are also shown for reference.

Table 7.14: PEARL design optimization results

Design variables

Quantity	Lower	Value	**Optimal**	Upper	Unit
D_f	0.1	3.713	**1.98**	10	m
D_s	0.1	0.15	**0.19**	10	m
D_d	0.1	1.	**0.21**	10	m
t_f	0.1	0.1	**0.1**	0.1	m
t_s	0.1	0.2	**10**	10	m
t_d	0.1	0.2	**0.1**	10	m
P_{comms}		50			W
R		10			Mbit/s
α		0.2			

Key results

Quantity	Value	Unit
$m_{platform}$	585	kg
m_{struct}	429	kg
v	0	m/s
G_M	1.288	m
h_f	0.09	m
EN	14.2	dB

We can see that the design variables t_f, t_d are as small, and t_s as large, as allowed by the boundary constraints, meaning that these constraints are active for the current parameters, and there is a good chance that they could be removed as design variables from the problem all together as a result. The constraint Eq. (7.79) is tight, meaning that one of the variables from this constraint could also be eliminated from the problem. This means that in the end only two variables, D_f and D_s or D_d, take non trivial values, which are governed by the model we have given.

The resulting design is large: with 2 meter diameter platform. The suspicion is that this value is driven majorly by the charging requirement of the AUV of 1.9kWh. Further sensitivity analysis would need to be carried out to confirm this. The platform

is also fairly heavy: this is highly sensitive to the density parameters used for the lumped mass estimation. A more refined model should be developed to give more realistic mass estimates. As a result of the design having a large diameter, but low thickness t_f, it is a very statically stable design, which can be seen from the metacentric height of 1.3 m.

The timing performance of the optimization of the different structures are shown in Table 7.15:

Table 7.15: Marine design optimization runtime with different structures

Structure	Formulation	Function iterations	Runtime (s)	Objective (kg)
Baseline	Block reduced	134	208.2	585.304
Baseline	Bordered reduced	3002	2499.6	2796.9
Restructured	Bordered reduced	134	22.59	585.304

The restructured problem was optimized one order of magnitude faster with respect to a block reduced version of the baseline problem (MDF architecture) and two order faster with respect to a bordered reduced (partial SAND/IDF architecture). Most likely this comes from the fact that for these optimization runs, the variables were started at random guesses with value 1, which might guide the optimizer in the wrong direction.

Chapter 8

Conclusion

Understanding what and why did not work may be more instructive than celebrating our successes.

Mikhail Leonidovich Gromov

8.1 Summary

In this thesis, we describe multidisciplinary problems from a different perspective than how they have usually been described in the literature: as problems with functions satisfying specific properties, mainly *sparsity*, *invertibility* - with respect to one or multiple variables, and as a single function or as a subset of the functions involved in the problem, and *reducibility*. Whenever the last property is present, it gives a candidate for introducing elimination functions, whose form can be given explicitly through feed forward through the inverted functions. These functions reduce the number of variables of the original problem we are trying to solve. The main advantages that such a reduction in size hopes for are:

1. increased performance in turns of computational time
2. reduce the number of initial guesses required
3. increase robustness to initial guesses (which is mainly a corollary of the second item)

The scope of the thesis is limited to conceptual design, as the elementary functions used in conceptual design often exhibit invertibility with respect to multiple variables. This gives a much larger set of possible multidisciplinary formulations, which in turn means a greater chance of finding a formulation with properties that can be exploited for elimination purposes.

In order to find such elimination functions, we introduced a new formalism for describing multidisciplinary design problems, both for analysis, design, and optimization. This formalism relies on a joint function and directed bipartite graph. We use this representation to embody the different choices of inversion that can be made on the problem and also encode the natural choice of inversion given in the original formulation of the problem. For a representation to be a valid formulation, we introduced the condition of the directed graph being acyclic. Next we described a set of transformation of the representation that results in equivalent *multidisciplinary* problems: *reduction* "undoes" *inversions*, and *inversions* either turn functions into *disciplines*, or change the variable with which a function is inverted. Through these transformations, it is simple to tansform invalid(cyclic) formulations into valid formulations. We also showed that this formalism can be used for representing multiple of the existing multidisciplinary architecture formulations, and that based on a representation that contains information of user defined inversions (which are most often due to custom solvers for a subset of the functions used in the model), we can derive the different MDO architecture formulations through a set of transformations of the representation.

Finally, this thesis investigated minimal representations that can be generated based on any representation. Two minimal valid representations are investigated: one with the minimum number of end components in the graph (which translates into a specific elimination structure), and the other with the smallest merged component based on strongly connected components of the original directed graph. Finding such representations amounts to solving a combinatorial problem we formulate as a binary integer program and solve exactly through cutting planes methods. The main novelty we add

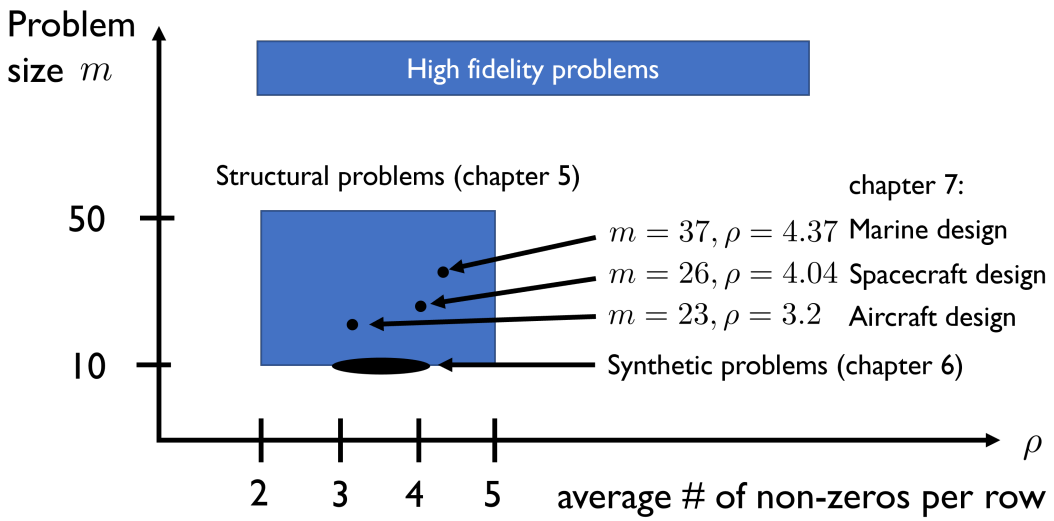


Figure 8-1: Big picture of where the results fit within the world of design problems

is to allow for inversions of functions with respect to input variables; this essentially changes what we would consider the inputs and outputs of the entire system.

We have shown that these methods can both robustify the solutions and increase the computational performance occasionally through both synthetic and real conceptual design problems. However, there is not enough evidence to claim that this method can consistently outperform non-optimized disciplinary formulations; yet this should be expected according to the "no free lunch" theorem [62]. Fig. 8-1 shows a summary of the key problem parameters investigated in this thesis for conceptual design problems. Other application areas will help populate this picture and help better understand the potential benefits of restructuring conceptual design problems.

8.2 Contributions

This thesis gives the following set of original contributions:

1. A new *formalism for describing multidisciplinary problems* and their formulation, which has precise mathematical semantics and from which feasibility and optimization formulations can easily be derived. It can be used for describing the same information that has conventionally been encoded in data flow graphs and

process graphs, and translating between one representation and the other can also be done mechanically. This formalism goes beyond the traditional concept of *discipline* and treats a discipline as an invertible function and a multidisciplinary problem as one composed of sparse, invertible functions. Properties of the representation, particularly the acyclicity of the directed graph, can be used to reduce the problem to smaller formulations mechanically.

2. A set of *isomorphic transformations* that can be applied to pre-process the problem. The transformations operate on the new representation without changing the problem. Some of these transformations can transform the graph from irreducible to reducible and as a result, reduce the size of the problem solved.

3. Two new binary integer formulations on the graph to find transformations that generate minimally reducible representations and as a result, the smallest problem formulation possible based on the set of transformations developed in this thesis. We show that the formulation can quickly solve sparse problems as long as keeping the dimensionality below 100 variables. This exact method can serve as a benchmarking algorithm for heuristic methods that we expect to scale much better with the size of the problem.

4. A new method for generating random synthetic problems based on polynomials that aim to imitate the sizing relationships encountered in conceptual design. The problem is scalable and therefore makes it possible to increase the size of the problem arbitrarily to different benchmark methods as a function of the size of the problem.

5. Formulation of three conceptual design problems, two of which are original in this thesis: the sizing relationships for a high altitude balloon concept and a marine platform for autonomous marine systems. The formulations generated have significant coupling between the intuitive *disciplines* used. These problems can be used as sample problems for future research work in this area.

6. Although not discussed in much detail in the body of the thesis, a major con-

tribution behind all of the work in this thesis is the computational package that was developed to both formulate, encode the representation developed, solve the binary integer problems, and execute the multidisciplinary formulations. Developed in the Python programming language, and on top of numerous scientific packages, notably `openMDAO`, `numpy`, `sympy`, `scipy`, `networkx` and `graphviz`. The package is for now called `minimdo` and can be found at `https://github.com/norheim/designresolver`.

8.3 Limitation and future work

The methods developed in this thesis come with certain limitations, however:

The first is the types of models whose structure can be exploited. The major advantages of this method were aimed at first-order, low-fidelity models, which commonly rely on elementary functions that are often invertible with respect to many of the variables. Although, in theory, one could always *invert* a function locally, due to the implicit function theorem, applying this idea could hide solutions or result in numerical problems. The simplest example to portray this ideas is the function $h(x_1, x_2) = x_1 - x_2^2$. Although technically we can invert the function with respect to x_2 when solving $h(x_1, x_2) = 0$, for certain values of x_1, namely negative inputs, there are no real-valued solutions for x_1. This could then impede the solver that uses the value of x_1 in a calculation downstream.

The second limitation is a bit of an extension from the first: many higher fidelity models involve solving large systems of, very-often, linear equations. Although the methods can, in theory, be applied to linear systems (which are invertible with respect to all variables), numerical issues can quickly appear. Methods exist to limit this in the case of linear cases, which could also be applied to nonlinear cases. We think this method is better suited for smaller systems, where the variables in the graph node either represent lumped values (like a vector that results from solving a system), or a variable from an elementary function.

An exciting realization from this work is that conceptual design equations are highly sparse and not as coupled as thought initially: even sparser than most of the test problems generated synthetically. The only way to reduce the sparsity is by merging most of the feedforward relationships into smaller components; however, at this point, the size of the system decreases significantly. As a result, it might be pretty intuitive for a person to find the formulations we showed in this thesis through some minimal brute force search. It would be helpful to find a non-synthetic problem that exhibits less sparsity but is still significant and where manual effort would most likely be in vain.

Another unexpected downside of transforming the formulation is that it often results in unexpected values for some of the inputs. For example, most engineering quantities usually are positive, but while transforming the expressions from the case studies, sometimes negative values were produced for specific inputs. This could result in numerical errors or domain problems for the inverted functions.

Some of these limitations might be addressed through more research; however, there is a set of other topics that future work could help address better.

First, the optimization formulations developed in Chapter 5 take a significant time to solve problems with more than a few tens of variables. There is a suspicion that much of this time is spent proving optimality instead of finding improved results; this is very often the case with binary and integer optimizations. It could be that the method could be stopped preemptively and still generate close to optimal results. However, this is only speculation for now. Although the first instinct was to formulate the problem as integer problems, it would be interesting to investigate whether we could decompose the minimum reduction and inversion problem into an assignment problem on one end and the feedback arc set problem on the other; since we can solve the assignment problem very efficiently, we might be able to leverage off the shelf feedback arc set solvers, which is an area of a fair amount of research at the moment, and where good heuristic methods have been developed.

Second, a fair amount of research has gone into optimizing the structure of linear

matrices to help solve linear equations. For this type of problem, it is known that optimizing the structure of the equations to minimize feedback, for example, is not, in general, an advantageous strategy and can lead to numerical challenges. Although in this thesis we focused on inverting nonlinear functions, at the Jacobian level, we suspect that the problem of restructuring linear and nonlinear functions is analogous. Most numerical methods rely on using the Jacobian during the solution method. This would imply that any method similar to tearing is generally a bad idea and that the minimum reduction and inversion can result in numerically unstable problems when solving nonlinear equations. During optimization, it is also worth wondering if, in exchange for reducing the problem's variable size, the Dulmage-Mendelsohn decomposition leads to numerically unstable optimization problems. We, therefore, see the methods developed in this thesis to have their advantage from the point of view of helping the engineer generate different structures of the problem, as opposed to necessarily generating structures with guarantees of better performance.

The formalism developed in this thesis was mainly applied to the original research question and in the context of conceptual design. However, it is much more general. Combined with the transformations, it can be used to generate representations of all the MDO architectures while giving a new language and mathematical tools to justify these architectures. The XDSM has become the de facto representation for MDO architectures, but we hope that the formulation graphs will become the new way of giving the mathematical formulations.

This thesis also stands as an enabler to a topic that is also critical in the early stages of conceptual design: linking the conceptual design models to the engineering requirements that drive the design in the first place. Design constraints, such as the spacecraft lifetime constraint given in Section 7.1 are often derived from such requirements. These requirements are normally encoded in natural language. However, the maturing and quickly evolving field of Natural Language Processing would make it possible to extract sentence structures that map to such constraints. Given a design model, this thesis, combined with such a parser (which might be automated or semi-

automated), would allow us to quickly explore the design space and rearrange design models with degrees of freedom when subject to different requirements. Vice-versa, rearrangement of the problems might quickly reveal unboundedness problems in the model, which would hint at the need for additional constraints, either from other modeling or additional requirements.

Another direction of future work that could benefit from the work in this thesis is speeding up the model building. Given that the output assignment of a design relationship does not need to be specified, we can store the sizing relationships in a library, similar to how it is done in Modelica language. From this, we could significantly reduce the time it takes to build and generate useful designs based on conceptual design models.

Appendix A

miniMDO code snippets

A.1 Syntax for specify sizing relationships

[16]:
```
from compute import Var, Par
from datastructures.api import Model, adda
```

[17]:
```
m = Var('m', unit='kg')
# no need to specify values for variables declared with Var
# if specified it will be treated as a guessing variable for solver
# unit field can also be left empty
g = Par('g', 9.81, unit='m/s^2')
```

We store all sizing relationships in a Model object

[20]:
```
model = Model()
# model.root is used to access the nesting level
F = adda(model.root, 'F', m*g); F
```

[20]:
$F = 9.81\ \mathrm{kg} \cdot \mathrm{m/s^2}\ (m = 1)$

Note that if we update the value of m, the value of F does not change automatically; think of variables as temporary storage. This means that we could have out of synch information like below, if we manually set the value of F.

[21]:
```
F.varval = 20; F
```

[21]:
$F = 20\ \mathrm{kg} \cdot \mathrm{m/s}^2\ (m = 1)$

A.2 Data structure for storing the formulation

[39]:
```
from datastructures.api import addf, addsolver, setsolvefor
```

[40]:
```
model = Model()
m = model.root
Pbus = Var('P_{bus}')
A = Var('A', 0.5)
solver = addsolver(m)
Pcomms = adda(solver, 'P_{comms}', 483.3*A-Pbus)
addf(solver, Pbus-10*Pcomms**0.5)
setsolvefor(solver, [Pbus])
h = Var('h', 400e3)
D = Var('D', 0.1)
R = adda(m, 'R', Pcomms*D**2/h**2*1e18/(8e6))
C = adda(m, 'C', 2500*D**2+12000*(A+1)+100*Pbus);
```

Once we have built a model through the syntax above we can recover the structure of the model and the nested formulation as follows:

[41]:
```
edges, tree = model.generate_formulation()
```

Input edges:

[42]:
```
edges[0]
```

[42]:
```
{0: ('A', 'P_{bus}'),
 1: ('P_{comms}', 'P_{bus}'),
 2: ('D', 'P_{comms}', 'h'),
 3: ('A', 'D', 'P_{bus}')}
```

Output edges:

[43]:
```
edges[1]
```

[43]:
```
{0: ('P_{comms}',), 1: (None,), 2: ('R',), 3: ('C',)}
```

Component tree (ordered)

[44]:
```
tree[0]
```

[44]:
```
OrderedDict([(0, 2), (1, 2), (2, 1), (3, 1)])
```

Solver tree (unordered)

Order is determined based on order of component

[45]:
```
tree[1]
```

[45]:
```
{2: 1}
```

Variable tree (unordered)

Order is determined based on order of component

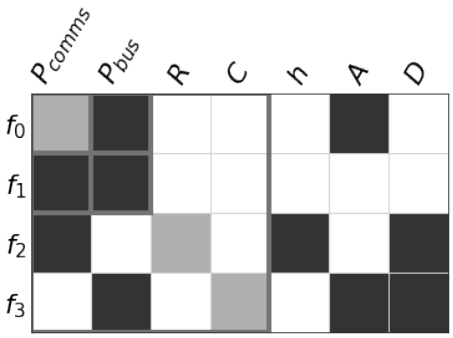
[46]:
```
tree[2]
```

[46]:
```
{'P_{bus}': 2}
```

A.3 Visualizing the hierarchical structure matrix

[47]:
```
from datastructures.rendering import render_incidence
```

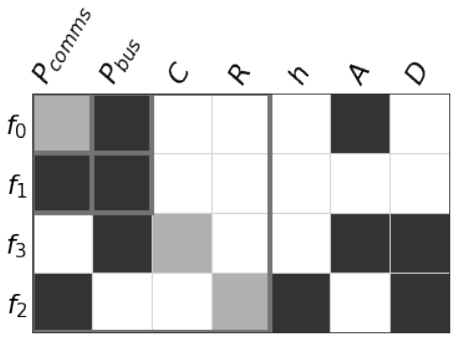
[49]:
```
render_incidence(edges, tree, hideticks=True, rawvarname=True, patchwidth=4,␣
↪fontsize=22);
```



Order visualized in HSM is based on order in which sizing relationships were added this can be modified by changing the tree. Below we flip f_2 and f_3 in the matrix

[58]:
```
new_tree = {0:2,1:2,3:1,2:1},tree[1],tree[2]
```

[59]:
```
render_incidence(edges, new_tree, hideticks=True, rawvarname=True, patchwidth=4,␣
↪fontsize=22);
```



A.4 Syntax for computing with components

[38]:
```
from compute import Var
from datastructures.execution import Component
```

A.4.1 Component handles unit conversions

[2]:
```
Pcomms = Var('P_{comms}', unit='W')
D = Var('D', unit='m')
h = Var('h', unit='km')
R = Var('R', unit='kW')
sizing_relationship = Pcomms*D**2/h**2*1e18/(8e6)
```

[3]:
```
sizing_relationship
```

[3]:
$$\frac{125000000000.0 D^2 P_{comms}}{h^2}$$

[4]:
```
component = Component.fromsympy(sizing_relationship, tovar=R)
```

A.4.2 Component attributes

[5]:
```
component.inputs, component.indims
```

[5]:
```
(('h', 'P_{comms}', 'D'), (1, 1, 1))
```

[6]:
```
component.outputs, component.outdims
```

[6]:
```
(('R',), (1,))
```

A.4.3 Evaluate components

[7]:
```
component.evaldict({'h':400, 'P_{comms}':10, 'D':10})
```

[7]:
```
[DeviceArray(0.78125, dtype=float64)]
```

A.4.4 Evaluate gradients through automatic differentiation

[8]:
```
component.graddict({'h':400, 'P_{comms}':10, 'D':10})
```

[8]:
```
{('R', 'h'): DeviceArray(-0.00390625, dtype=float64),
 ('R', 'P_{comms}'): DeviceArray(0.078125, dtype=float64),
 ('R', 'D'): DeviceArray(0.15625, dtype=float64)}
```

A.4.5 Component manual creation

Demonstration with matrix/vector quantities

[27]:
```
import numpy as np
A = np.random.rand(10,10)
b0 = np.random.rand(10)
x0 = np.random.rand(10)
fx = lambda x,b: (b@(A@x-b),) # The @ is numpys matrix multiplication
component = Component(fx, inputs=('x','b'), outputs=('y',), indims=(10,10),␣
↪outdims=(1,))
```

[28]:
```
component.evaldict({'x':x0, 'b':b0})
```

[28]:
```
(4.391333423109129,)
```

[29]:
```
component.graddict({'x':x0, 'b':b0})
```

[29]:
```
{('y',
  'x'): DeviceArray([2.18488828, 1.43761381, 2.01779949, 1.61428775, 2.76793662,
                     1.87935127, 2.02227283, 0.88950607, 1.8922517 , 2.036412  ],
dtype=float64),
 ('y',
  'b'): DeviceArray([1.0146121 , 1.13650587, 0.33435548, 1.81286402, 1.94674412,
                     1.05914187, 1.0388783 , 0.71057041, 0.0901676 , 1.00682255],
dtype=float64)}
```

Check against analytical derivatives

[39]:
```
b0@A # dy/dx
```

[39]:
```
array([2.18488828, 1.43761381, 2.01779949, 1.61428775, 2.76793662,
       1.87935127, 2.02227283, 0.88950607, 1.8922517 , 2.036412  ])
```

[40]:
```
A@x0-2*b0 # dy/db
```

```
[40]: array([1.0146121 , 1.13650587, 0.33435548, 1.81286402, 1.94674412,
             1.05914187, 1.0388783 , 0.71057041, 0.0901676 , 1.00682255])
```

A.5 Restructuring introduction

A.5.1 Starting model

[1]:
```
from compute import Var
from datastructures.api import Model, adda
```

[2]:
```
model = Model()
m = model.root
Pbus = Var('P_{bus}')
A = Var('A', 0.5)
Pcomms = adda(m, 'P_{comms}', 483.3*A-Pbus)
adda(m, Pbus, 10*abs(Pcomms)**0.5)
h = Var('h', 400e3)
D = Var('D', 0.1)
R = adda(m, 'R', Pcomms*D**2/h**2*1e18/(8e6))
C = adda(m, 'C', 2.5*D**2+12*(A+1)+0.1*Pbus)
```

We restructure with extended tearing

A.5.2 Restructuring interface

[3]:
```
from datastructures.graphutils import all_variables, all_edges
from datastructures.tearing import dir_graph, min_arc_set_assign
from datastructures.operators import reformulate
```

[8]:
```
edges, tree = model.generate_formulation()
not_output = ['h', 'D', 'R'] # This forces the solution from the thesis to come␣
↪out
eqnidxs = list(edges[1].keys())
varidxs = all_variables(*edges)
graph_edges_minassign = all_edges(*edges)
edges_left_right = list(dir_graph(graph_edges_minassign, eqnidxs, {}))
xsol,_ = min_arc_set_assign(edges_left_right, varidxs, eqnidxs,␣
↪not_output=not_output)
```

[9]:
```
outset_initial = {comp: var[0] for comp,var in edges[1].items()}
outset_opt = {right:left for left, right in edges_left_right
```

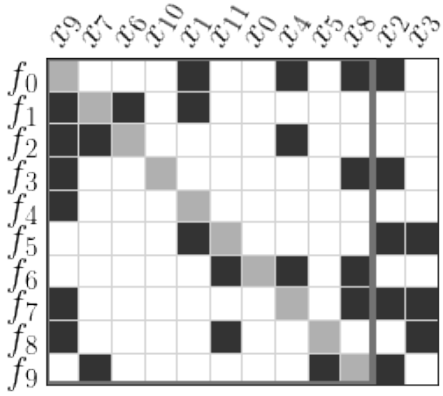
```
                    if (left,right) in edges_left_right and xsol[left, right] > 0.
↪5}
```

[10]:
```
edges_minassign, tree_minassign = reformulate(edges, tree, outset_initial,
                                              outset_opt, solveforvars=False)
```

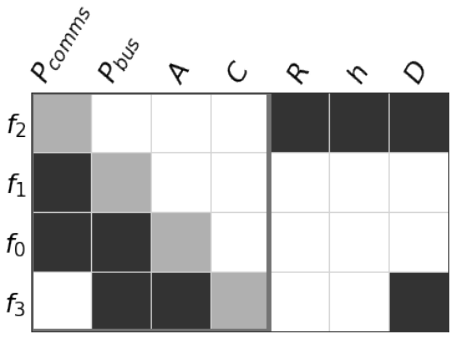
A.5.3 Visualize HSM

[12]:
```
from datastructures.rendering import render_incidence
```

[14]:
```
render_incidence(edges, tree, hideticks=True, rawvarname=True, patchwidth=4,␣
↪fontsize=22);
```



[13]:
```
render_incidence(edges_minassign, tree_minassign, hideticks=True,␣
↪rawvarname=True, patchwidth=4, fontsize=22);
```



[1]:
```
from testproblems import random_problem_with_artifacts
from datastructures.operators import eqv_to_edges_tree, reformulate, invert_edges
from datastructures.graphutils import flat_graph_formulation, Node, COMP, VAR
from datastructures.rendering import render_incidence
from datastructures.tearing import min_max_scc2, min_arc_set_assign,␣
↪min_arc_set, feedbacks
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
plt.rcParams['text.usetex'] = True
```

A.6 Restructuring algorithms

[2]:
```
m,n,seed,sparsity = 10, 12, 229, 4.
kwargs = random_problem_with_artifacts(m,n,seed,sparsity, independent_of_n=True)
```

[3]:
```
eq_incidence,outset,eqnidxs,edges_varonleft,varidxs = map(kwargs.get,␣
↪["eq_incidence","outset","eqnidxs","edges_varonleft","varidxs"])
```

A.6.1 Calculated row density

[4]:
```
np.mean([len(val) for val in eq_incidence.values()])
```

[4]:
```
4.0
```

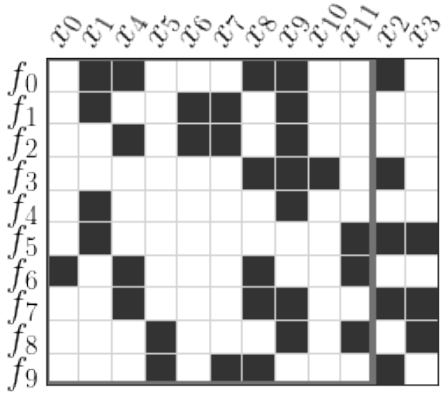
[5]:
```
vizargs = {'figsize':(4,4), 'rotation':60, 'hideticks':True, 'fontsize':22,␣
↪'patchwidth':4}
```

A.6.2 Undirected formulation

We see all sizing relationships as "equality constraints"

[6]:
```
edges_undir,tree_undir,_ = eqv_to_edges_tree(eq_incidence, n_eqs=m)
edges_dir,tree_dir,outset_adj = eqv_to_edges_tree(eq_incidence, outset, n_eqs=m)
tree_undir[2].update({val:1 for val in outset_adj.values()})
```

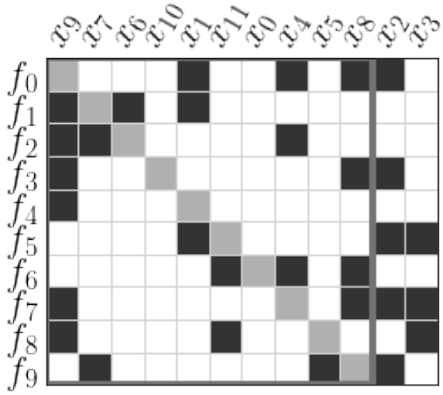
[7]:
```
render_incidence(edges_undir, tree_undir, **vizargs);
```



A.6.3 Directed formulation

Unstructured based on randomness

[8]:
```
render_incidence(edges_dir, tree_dir, **vizargs);
```

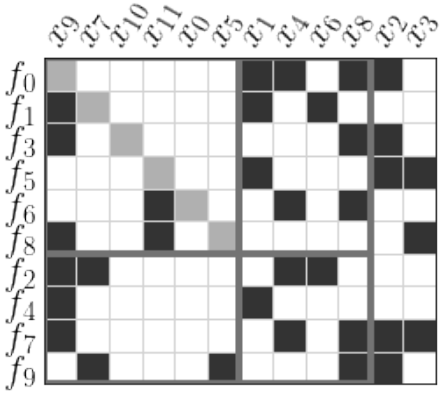


A.6.4 Polynomial restructuring algorithms

Reformulated based on feedback from unstructured

[9]:
```
D = flat_graph_formulation(*edges_dir)
fvars, fcomps = feedbacks(D, [Node(idx,COMP) for idx in eqnidxs])
outset_f = {comp:var for comp,var in outset.items() if Node(comp,COMP) not in␣
↪fcomps}
edges_fb,tree_fb,_ = eqv_to_edges_tree(eq_incidence, outset_f, n_eqs=m)
tree_fb[2].update({v.name: 1 for v in sorted(fvars, key=lambda x: x.name)})
```

[10]:
```
render_incidence(edges_fb, tree_fb, **vizargs);
```

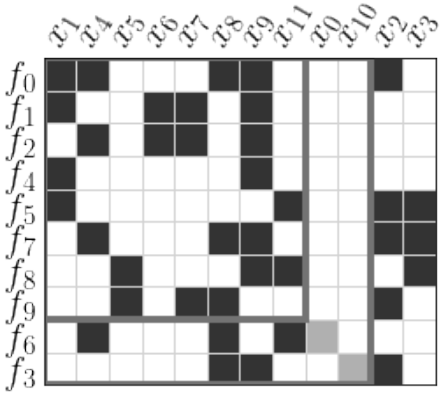


Condensation (Tarjan's algorithm)

Gives a topological sorting of the strongly connected components

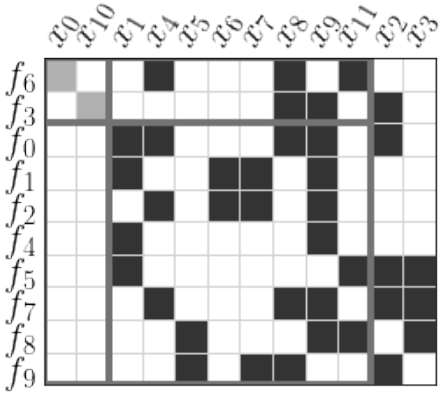
Block reduce (subproblems)

[11]:
```
formulation_scc = reformulate(edges_dir, tree_dir, root_solver_name=1)
render_incidence(*formulation_scc, **vizargs);
```



Border reduce (no-subproblem)

[12]:
```
formulation_scc = reformulate(edges_dir, tree_dir, root_solver_name=1, mdf=False)
render_incidence(*formulation_scc, **vizargs);
```



Fine Dumlage-Menelsohn decomposition

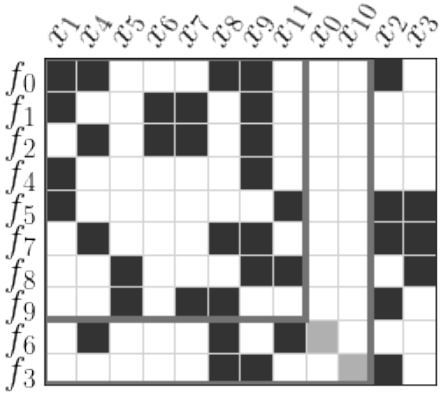
Returns the same formulation as Tarjan's algorithm

[13]:
```
Ein_no_source = {comp:tuple(var for var in allvars if var in outset.values())␣
↪for comp,allvars in eq_incidence.items()}
G = flat_graph_formulation(Ein_no_source,{},{},{VAR: 'x_{{{}}}', COMP:␣
↪'f_{{{}}}'})
matching = nx.bipartite.maximum_matching(G)
```

```
m_out = {key: matching[Node(key, COMP)].name for key in eqnidxs}
edges_dir_m,tree_dir_m,_ = eqv_to_edges_tree(eq_incidence, m_out, n_eqs=m)
```

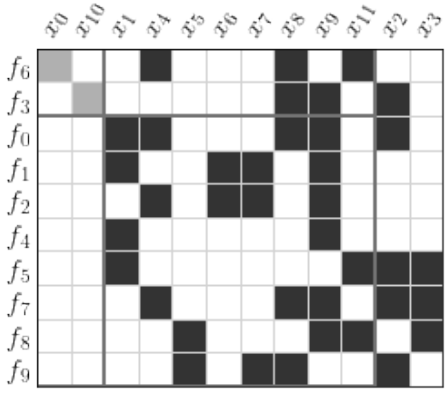
Block reduce (subproblems)

[14]:
```
formulation_scc_m = reformulate(edges_dir_m, tree_dir_m, root_solver_name=1)
render_incidence(*formulation_scc_m, **vizargs);
```



Border reduce (no-subproblem)

[15]:
```
formulation_scc_m = reformulate(edges_dir_m, tree_dir_m, root_solver_name=1,␣
↪mdf=False)
render_incidence(*formulation_scc_m, figsize=((4,4)), hideticks=True);
```



A.6.5 Optimization-based algorithms (NP-complete)

Minimum Feedback Arc Set (FAS)

[16]:
```
cycles, elimset, model = min_arc_set(edges_varonleft, outset, varidxs, eqnidxs)
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2023-02-25
```

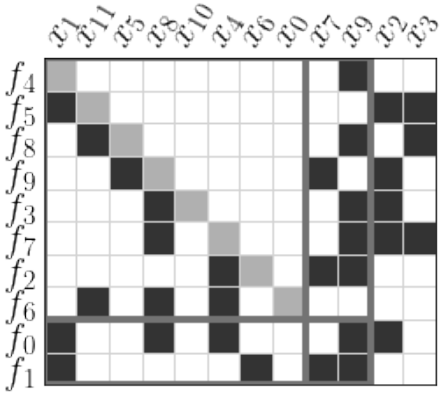
[17]:
```
elimset, {outset_adj[elt] for elt in elimset}
```

[17]:
```
({0, 1}, {7, 9})
```

[18]:
```
outset_mfs = {key:val for key,val in outset_adj.items() if key not in elimset}
```

Compare to condensation (Tarjan's algorithm) results. Size of problem is reduced to 2.

[19]:
```
formulation_mfs = reformulate(edges_dir, tree_dir, outset_adj, outset_mfs,␣
↪root_solver_name=1, solveforvars=2)
render_incidence(*formulation_mfs, **vizargs);
```

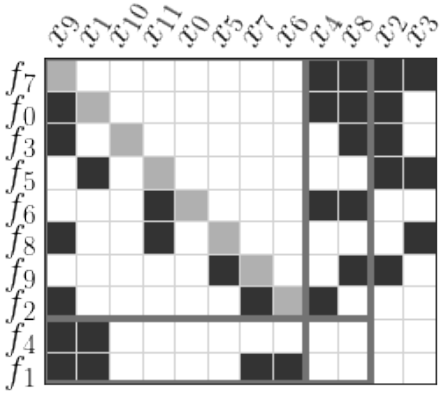


Regular tearing

In this problem regular tearing does not do any better than FAS

[20]:
```
fixed_inputs = set(varidxs)-set(outset.values())
```

[21]:
```
xsol, model = min_arc_set_assign(edges_varonleft, varidxs, eqnidxs,␣
↪not_output=fixed_inputs)
outset_tear = dict((right, left-m) for left, right in edges_varonleft if␣
↪xsol[left, right] > 0.5)
formulation_tear = reformulate(edges_dir, tree_dir, outset_adj, outset_tear,
                               not_outputs={elt-m for elt in fixed_inputs},
                               root_solver_name=1)
render_incidence(*formulation_tear, **vizargs);
```



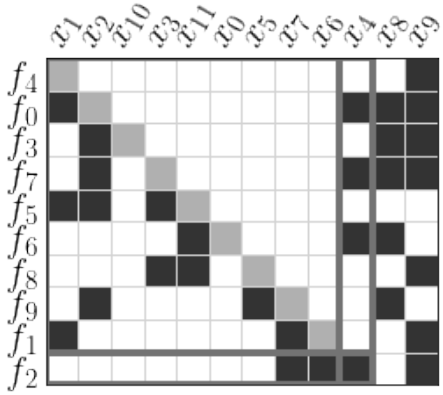
Extended tearing

We change the inputs to be x_8 and x_9 and reduce the problem size further to 1

[22]:
```
xsol, model = min_arc_set_assign(edges_varonleft, varidxs, eqnidxs)
outset_minassign = dict((right, left-m) for left, right in edges_varonleft if␣
↪xsol[left, right] > 0.5)
```

[23]:
```
edges_minassign=invert_edges(edges_undir[0], edges_undir[1], outset_minassign)
```

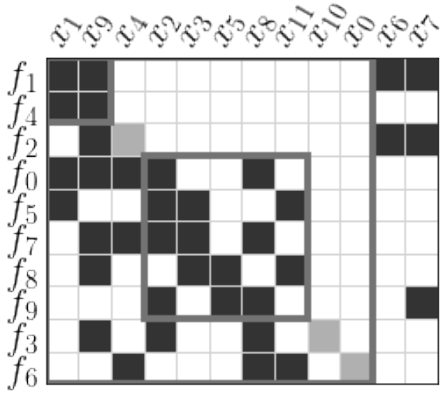
[24]:
```
formulation_minassign = reformulate(edges_undir, tree_dir, outset_adj,␣
↪outset_minassign, root_solver_name=1)
render_incidence(*formulation_minassign, **vizargs);
```



Reduced subproblems

Compare results to condensation (Tarjan's algorithm) results with block reduce

[25]:
```
xsol, model = min_max_scc2(edges_varonleft, varidxs, eqnidxs)
outset_minscc = dict((right, left-m) for left, right in edges_varonleft if␣
 ↪xsol[left, right] > 0.5)
formulation_minscc = reformulate(edges_dir, tree_dir, outset_adj, outset_minscc,␣
 ↪root_solver_name=1)
render_incidence(*formulation_minscc, **vizargs);
```



Starting model

[1]:
```
from compute import Var, Par
from datastructures.api import Model, adda, addeq,addineq, addobj, setsolvefor,␣
↪merge, OPT
```

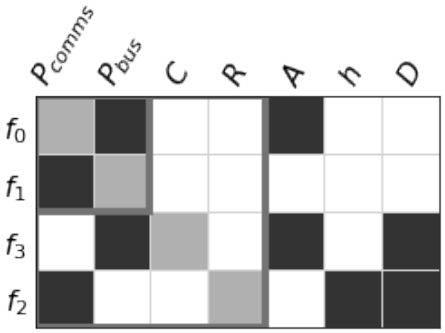
[55]:
```
model = Model()
m = model.root
Pbus = Var('P_{bus}')
A = Var('A', 0.5)
Pcomms = adda(m, 'P_{comms}', 483.3*A-Pbus)
adda(m, Pbus, 10*abs(Pcomms)**0.5)
h = Var('h', 400e3)
D = Var('D', 0.1)
R = adda(m, 'R', Pcomms*D**2/h**2*1e18/(8e6))
C = adda(m, 'C', 2.5*D**2+12*(A+1)+0.1*Pbus)
```

Reduce size of problem through condensation strategy

[56]:
```
formulation = model.generate_formulation()
```

[57]:
```
from datastructures.operators import reformulate
from datastructures.rendering import render_incidence
```

[58]:
```
_,new_tree = reformulate(*formulation, root_solver_name=1)
formulation_scc = formulation[0], new_tree
render_incidence(*formulation_scc, rawvarname=True, figsize=((4,4)),␣
↪rotation=60, hideticks=True, patchwidth=4);
```



Transform components based on new formulation graph

[59]:
```
from datastructures.transformations import transform_components
```

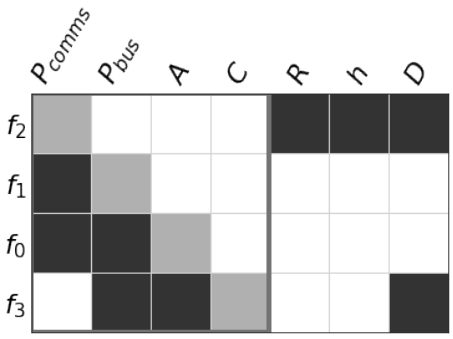
[60]:
```
edges = formulation[0]
new_edges = formulation_scc[0]
new_comps = transform_components(edges, new_edges, model.components, model.
↪idmapping)
```

Generate optimization problem

[61]:
```
addobj(m, C)
addeq(m, -R+1)
setsolvefor(m, [D,A], {D:[0.5,10], A:[0.04,1]}) # we pick only optimize over a␣
↪subset of the input variables
```

[62]:
```
# we need to merge the reformulation with the added information
formulation_scc_opt = merge(model.generate_formulation(), *formulation_scc,␣
↪copysolvers=False)
```

[63]:
```
render_incidence(*formulation_scc_opt, rawvarname=True, figsize=((4,4)),␣
↪rotation=60, hideticks=True, patchwidth=4);
```



A.7 Generate workflow

[64]:
```
comp_options = model.comp_options
var_options = model.var_options
solvers_options = model.solvers_options
idmapping = model.idmapping
nodetyperepr = model.nametyperepr
components = model.components+new_comps
```

[65]:
```
solvers_options[1] = {'type': OPT, 'optimizer': 'SLSQP'}
solvers_options[2] = {'solver': 'N', 'maxiter':20}
```

Workflow generation

[66]:
```
from datastructures.workflow import order_from_tree
```

[67]:
```
edges, tree = formulation_scc_opt
sequence = order_from_tree(tree[0], tree[1], edges[1])
```

[68]:
```
sequence
```

[68]:
```
[(SOLVER, 1, None),
 (SOLVER, 2, 1),
 (COMP, 0, 2),
 (COMP, 1, 2),
 (COMP, 3, 1),
 (COMP, 2, 1),
 (ENDCOMP, [4, 5], 1)]
```

Solver Options

Populate the solvers with default values if not set, based on graph formulation

[69]:
```
from datastructures.workflow import default_solver_options
```

[70]:
```
solvers_options = default_solver_options(tree, solvers_options)
```

[71]:
```
solvers_options
```

[71]:
```
{1: {'type': OPT, 'optimizer': 'SLSQP', 'designvars': ('D', 'A')},
 2: {'solver': 'N',
  'maxiter': 20,
  'type': SOLVE,
  'designvars': ('P_{comms}', 'P_{bus}')}}
```

openMDAO specific workflow

Based on solver options generate openMDAO specific workflow

[72]:
```
from datastructures.workflow import mdao_workflow
```

[73]:
```
wf = mdao_workflow(sequence, solvers_options, comp_options, var_options)
```

[74]:
```
wf
```

[74]:
```
[(OPT,
  1,
  None,
  {'optimizer': 'SLSQP', 'designvars': ('D', 'A')},
  {'D': [0.5, 10], 'A': [0.04, 1]}),
 (SOLVE,
  2,
  1,
  {'solver': 'N', 'maxiter': 20, 'designvars': ('P_{comms}', 'P_{bus}')},
  {}),
 (EXPL, 0, 2),
 (EXPL, 1, 2),
 (EXPL, 3, 1),
 (EXPL, 2, 1),
 (OBJ, 4, 1),
 (EQ, 5, 1)]
```

- OPT = specifies design variables to be declared in openMDAO
- SOLVE = groups with nonlinear solvers in openMDAO
- IMPL = implicit component in openMDAO
- EXPL = explicit component in openMDAO
- OBJ = objective function for optimization

Detailed openMDAO workflow

This contains the actual functions, and the explicit gradients that will be given to openMDAO

[75]:
```
from datastructures.workflow import get_f
from datastructures.graphutils import namefromsympy
from datastructures.workflow_mdao import mdao_workflow_with_args
```

[76]:
```
lookup_f = get_f(components, edges)
namingfunc = namefromsympy(nodetyperepr)
wfmdao = mdao_workflow_with_args(wf, lookup_f, namingfunc)
```

[77]:
```
wfmdao
```

[77]:
```
[(OPT,
  None,
  's1',
  ('D', 'A'),
  {'optimizer': 'SLSQP'},
  {'D': [0.5, 10], 'A': [0.04, 1]}),
 (SOLVE, 's1', 's2', {'solver': 'N', 'maxiter': 20}, {}),
 (EXPL,
  's2',
  'f0',
  ('A', 'P_bus'),
  ('P_comms',),
  <bound method Component.evaldict of (('A', 'P_bus'), 0, ('P_comms',),
'None')>,
  <bound method Component.graddict of (('A', 'P_bus'), 0, ('P_comms',),
'None')>),
 (EXPL,
  's2',
  'f1',
  ('P_comms',),
  ('P_bus',),
  <bound method Component.evaldict of (('P_comms',), 1, ('P_bus',), 'None')>,
  <bound method Component.graddict of (('P_comms',), 1, ('P_bus',), 'None')>),
 (EXPL,
```

```
  's1',
  'f3',
  ('A', 'D', 'P_bus'),
  ('C',),
  <bound method Component.evaldict of (('A', 'D', 'P_bus'), 3, ('C',), 'None')>,
  <bound method Component.graddict of (('A', 'D', 'P_bus'), 3, ('C',),
'None')>),
 (EXPL,
  's1',
  'f2',
  ('h', 'P_comms', 'D'),
  ('R',),
  <bound method Component.evaldict of (('h', 'P_comms', 'D'), 2, ('R',),
'None')>,
  <bound method Component.graddict of (('h', 'P_comms', 'D'), 2, ('R',),
'None')>),
 (OBJ,
  's1',
  'f4',
  ('C',),
  'obj4',
  <bound method Component.evaldict of (('C',), 4, ('obj4',), 'None')>,
  <bound method Component.graddict of (('C',), 4, ('obj4',), 'None')>),
 (EQ,
  's1',
  'f5',
  ('R',),
  'eq5',
  <bound method Component.evaldict of (('R',), 5, ('eq5',), 'None')>,
  <bound method Component.graddict of (('R',), 5, ('eq5',), 'None')>)]
```

Assemble the formulation in openMDAO

[78]:
```
from datastructures.assembly import build_archi
```

[91]:
```
prob, mdao_in, groups = build_archi(edges, tree, wfmdao, namingfunc, idmapping)
```

A.8 Execute model without running optimization

[92]:
```
components
```

[92]:
```
[(('A', 'P_{bus}'), 0, ('P_{comms}',), '483.3*A - P_{bus}'),
 (('P_{comms}',), 1, ('P_{bus}',), '10*Abs(P_{comms})**0.5'),
 (('h', 'P_{comms}', 'D'), 2, ('R',), '125000000000.0*D**2*P_{comms}/h**2'),
 (('A', 'D', 'P_{bus}'), 3, ('C',), '12*A + 2.5*D**2 + 0.1*P_{bus} + 12'),
 (('C',), 4, (None,), 'C'),
 (('R',), 5, (None,), '1 - R')]
```

[93]:
```
components[0].graddict({'A':10, 'P_{bus}':10})
```

[93]:
```
{('P_{comms}', 'A'): DeviceArray(483.3, dtype=float64),
 ('P_{comms}', 'P_{bus}'): DeviceArray(-1., dtype=float64)}
```

[94]:
```
components[2].graddict({'h':h.varval, 'P_{comms}':10, 'D':10})
```

[94]:
```
{('R', 'h'): DeviceArray(-0.00390625, dtype=float64),
 ('R', 'P_{comms}'): DeviceArray(78.125, dtype=float64),
 ('R', 'D'): DeviceArray(156.25, dtype=float64)}
```

[95]:
```
components[3].graddict({'D':10, 'P_{bus}':10, 'A':10})
```

[95]:
```
{('C', 'A'): DeviceArray(12., dtype=float64),
 ('C', 'D'): DeviceArray(50., dtype=float64),
 ('C', 'P_{bus}'): DeviceArray(0.1, dtype=float64)}
```

[117]:
```
prob.set_val('D', 0.67)
prob.set_val('A', 0.1)
prob.run_model()
```

```
==
s2
==
NL: Newton Converged in 4 iterations
```

Display results

[115]:
```
from datastructures.postprocess import print_outputs
print_outputs(model, prob, namingfunc, rounding=3)
```

[115]:
```
<pandas.io.formats.style.Styler at 0x28b44477f70>
```

A.9 Optimize model

[116]:
```
prob.run_driver();
```

```
==
s2
==
NL: Newton Converged in 0 iterations

==
s2
==
NL: Newton Converged in 0 iterations

==
s2
==
NL: Newton Converged in 5 iterations

==
s2
==
NL: Newton Converged in 4 iterations

==
s2
==
NL: Newton Converged in 4 iterations
```

```
==
s2
==
NL: Newton Converged in 2 iterations

==
s2
==
NL: Newton Converged in 3 iterations

==
s2
==
NL: Newton Converged in 3 iterations

==
s2
==
NL: Newton Converged in 3 iterations

==
s2
==
NL: Newton Converged in 2 iterations
Optimization terminated successfully    (Exit mode 0)
            Current function value: [15.30093085]
            Iterations: 8
            Function evaluations: 9
            Gradient evaluations: 7
Optimization Complete
---------------------------------
```

Bibliography

[1] J. T. Allison, *Optimal Partitioning and Coordination for Decomposition-based Design Optimization*. PhD thesis, University of Michigan, 2008.

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

[3] T. M. Coffee, "A modular programming language for engineering design," Master's thesis, Massachusetts Institute of Technology, 2008.

[4] O. de Weck and K. Willcox, "Multidisciplinary system design optimization." Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu/. License: Creative Commons BY-NC-SA., Spring 2010.

[5] D. V. Steward, "On an approach to techniques for the analysis of the structure of large systems of equations," Master's thesis, Stanford University, 1962.

[6] P. Y. Papalambros and D. J. Wilde, *Principles of optimal design: modeling and computation*, ch. 3, pp. 91–154. Cambridge university press, 2000.

[7] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimization and engineering*, vol. 8, no. 1, pp. 67–127, 2007.

[8] J. R. R. A. Martins and A. Ning, *Engineering Design Optimization*. Cambridge University Press, 2021.

[9] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*, vol. 6. Athena Scientific Belmont, MA, 1997.

[10] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[11] J. E. Dennis, Jr, J. M. Martínez, and X. Zhang, "Triangular decomposition methods for solving reducible nonlinear systems of equations," *SIAM Journal on Optimization*, vol. 4, no. 2, pp. 358–382, 1994.

[12] A. Baharev, H. Schichl, and A. Neumaier, "Tearing systems of nonlinear equations i. a survey," tech. rep., University of Vienna, 2016.

[13] H. Elmqvist and M. Otter, "Methods for tearing systems of equations in object-oriented modeling," in *Proceedings ESM*, vol. 94, pp. 1–3, 1994.

[14] G. R. Shubin, "Application of alternative multidisciplinary optimization formulations to a model problem for static aeroelasticity," *Journal of Computational Physics*, vol. 118, no. 1, pp. 73–85, 1995.

[15] R. J. Balling and J. Sobieszczanski-Sobieski, "Optimization of coupled systems-a critical overview of approaches," *AIAA journal*, vol. 34, no. 1, pp. 6–17, 1996.

[16] J. Sobieszczanski-Sobieski and R. C. Goetz, "Synthesis of aircraft structures using integrated design and analysis methods," *Res. in Computerized Structural Analysis and Syn.,*, 1978.

[17] J. R. R. A. Martins and A. B. Lambe, "Multidisciplinary design optimization: A survey of architectures," *AIAA Journal*, vol. 51, pp. 2049–2075, September 2013.

[18] T. C. Wagner and P. Y. Papalambros, "A general framework for decomposition analysis in optimal design," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 97690, pp. 315–325, American Society of Mechanical Engineers, 1993.

[19] N. F. Michelena and P. Y. Papalambros, "A hypergraph framework for optimal model-based decomposition of design problems," *Computational optimization and applications*, vol. 8, no. 2, pp. 173–196, 1997.

[20] Z. Lu and J. R. R. A. Martins, "Graph partitioning-based coordination methods for large-scale multidisciplinary design optimization problems," in *Proceedings of the 14th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, (Indianapolis, IN), September 2012. AIAA 2012-5522.

[21] E. J. Cramer, J. E. Dennis, Jr, P. D. Frank, R. M. Lewis, and G. R. Shubin, "Problem formulation for multidisciplinary optimization," *SIAM Journal on Optimization*, vol. 4, no. 4, pp. 754–776, 1994.

[22] R. Pandi Perumal, H. Voos, F. Dalla Vedova, and H. Moser, "Comparison of multidisciplinary design optimization architectures for the design of distributed space systems," in *Proceedings of the 71st international astronautical congress 2020*, 2020.

[23] M. Spivak, *Calculus on manifolds: a modern approach to classical theorems of advanced calculus*. CRC press, 2018.

[24] P. Fritzson and P. Bunus, "Modelica-a general object-oriented language for continuous and discrete-event system modeling and simulation," in *Proceedings 35th Annual Simulation Symposium. SS 2002*, pp. 365–380, IEEE, 2002.

[25] H. Elmqvist, *A Structured Model Language for Large Continuous Systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology (LTH), 1978.

[26] A. Bahareva, H. Schichla, and A. Neumaiera, "Tearing systems of nonlinear equations ii. a practical exact algorithm," 2016.

[27] A. L. Dulmage and N. S. Mendelsohn, "Two algorithms for bipartite graphs," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 1, pp. 183–194, 1963.

[28] R. S. Mah, *Chemical process structures and information flows*. Elsevier, 2013.

[29] N. Alexandrov and R. Lewis, "Reconfigurability in mdo problem synthesis, part 1 and part 2," tech. rep., Tech. rep., Papers AIAA-2004-4307 and AIAA-2004-4308, 2004.

[30] S. Tosserams, A. Hofkamp, L. Etman, and J. Rooda, "A specification language for problem partitioning in decomposition-based design optimization," *Structural and Multidisciplinary Optimization*, vol. 42, no. 5, pp. 707–723, 2010.

[31] D. J. Pate, J. Gray, and B. J. German, "A graph theoretic approach to problem formulation for multidisciplinary design analysis and optimization," *Structural and Multidisciplinary Optimization*, vol. 49, no. 5, pp. 743–760, 2014.

[32] I. van Gent and G. La Rocca, "Formulation and integration of mdao systems for collaborative design: A graph-based methodological approach," *Aerospace Science and Technology*, vol. 90, pp. 410–433, 2019.

[33] I. van Gent, G. La Rocca, and M. F. Hoogreef, "Cmdows: a proposed new standard to store and exchange mdo systems," *CEAS Aeronautical Journal*, vol. 9, no. 4, pp. 607–627, 2018.

[34] F. Gallard, C. Vanaret, D. Guénot, V. Gachelin, R. Lafage, B. Pauwels, P.-J. Barjhoux, and A. Gazaix, "GEMS: A Python Library for Automation of Multidisciplinary Design Optimization Process Generation," in *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018.

[35] D. V. Steward, "The design structure system: A method for managing the design of complex systems," *IEEE transactions on Engineering Management*, no. 3, pp. 71–74, 1981.

[36] T. R. Browning, "Applying the design structure matrix to system decomposition and integration problems: a review and new directions," *IEEE Transactions on Engineering management*, vol. 48, no. 3, pp. 292–306, 2001. Publisher: IEEE.

[37] T. R. Browning, "Design structure matrix extensions and innovations: a survey and new opportunities," *IEEE Transactions on engineering management*, vol. 63, no. 1, pp. 27–52, 2015.

[38] R. J. Lano, *A technique for software and systems design*. TRW series on software technology ; v. 3, Amsterdam ;: North-Holland Pub. Co., 1979.

[39] M. Grötschel, M. Jünger, and G. Reinelt, "A cutting plane algorithm for the linear ordering problem," *Operations research*, vol. 32, no. 6, pp. 1195–1220, 1984.

[40] R. Fletcher and J. Hall, "Ordering algorithms for irreducible sparse linear systems," *Annals of Operations Research*, vol. 43, no. 1, pp. 15–32, 1993. Publisher: Springer.

[41] J. J. Alonso, P. LeGresley, E. van der Weide, J. R. R. A. Martins, and J. J. Reuther, "pyMDO: A framework for high-fidelity multi-disciplinary optimization," in *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, (Albany, NY), September 2004. AIAA 2004-4480.

[42] J. S. Gray, J. T. Hwang, J. R. R. A. Martins, K. T. Moore, and B. A. Naylor, "OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization," *Structural and Multidisciplinary Optimization*, vol. 59, pp. 1075–1104, April 2019.

[43] F. Gallard, C. Vanaret, D. Guénot, V. Gachelin, R. Lafage, B. Pauwels, P.-J. Barjhoux, and A. Gazaix, "Gems: A python library for automation of multidisciplinary design optimization process generation," in *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018.

[44] R. S. Krishnamachari, *A decomposition synthesis methodology for optimal systems design*. University of Michigan, 1996.

[45] R. S. Krishnamachari and P. Y. Papalambros, "Optimal hierarchical decomposition synthesis using integer programming," *Journal of Mechanical Design*, vol. 119, pp. 440–447, Dec. 1997.

[46] S. Tosserams, L. Etman, and J. Rooda, "Augmented Lagrangian coordination for distributed optimal design in MDO," *International journal for numerical methods in engineering*, vol. 73, no. 13, pp. 1885–1910, 2008. Publisher: Wiley Online Library.

[47] D. V. Steward, "Partitioning and tearing systems of equations," *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, vol. 2, no. 2, pp. 345–365, 1965.

[48] A. Baharev, H. Schichl, A. Neumaier, and T. Achterberg, "An exact method for the minimum feedback arc set problem," *Journal of Experimental Algorithmics (JEA)*, vol. 26, pp. 1–28, 2021.

[49] D. V. Steward, *Systems analysis and management: structure, strategy, and design*. Petrocelli books, 1981.

[50] A. Baharev, H. Schichl, A. Neumaier, and T. Achterberg, "An exact method for the minimum feedback arc set problem," *Journal of Experimental Algorithmics (JEA)*, vol. 26, pp. 1–28, 2021. Publisher: ACM New York, NY, USA.

[51] D. B. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM Journal on Computing*, vol. 4, no. 1, pp. 77–84, 1975.

[52] N. P. Tedford and J. R. R. A. Martins, "Benchmarking multidisciplinary design optimization algorithms," *Optimization and Engineering*, vol. 11, pp. 159–183, February 2010.

[53] D. Zhang, B. Song, P. Wang, and Y. He, "Performance evaluation of mdo architectures within a variable complexity problem," *Mathematical Problems in Engineering*, vol. 2017, 2017.

[54] E. L. Allgower and K. Georg, *Introduction to numerical continuation methods*. SIAM, 2003.

[55] P. Breiding and S. Timme, "HomotopyContinuation.jl: A Package for Homotopy Continuation in Julia," in *International Congress on Mathematical Software*, pp. 458–465, Springer, 2018.

[56] J. Norheim, "Satellite component selection with mixed integer nonlinear programming," in *2020 IEEE Aerospace Conference*, pp. 1–9, IEEE, 2020.

[57] J. Norheim and O. de Weck, "Co-optimizing spacecraft component selection, design, and operation with minlp," in *2021 IEEE Aerospace Conference (50100)*, pp. 1–10, IEEE, 2021.

[58] W. J. Larson and J. R. Wertz, "Space mission analysis and design," tech. rep., Torrance, CA (United States); Microcosm, Inc., 1992.

[59] M. N. Haji, J. Tran, J. Norheim, and O. L. de Weck, "Design and testing of auv docking modules for a renewably powered offshore auv servicing platform," in *International Conference on Offshore Mechanics and Arctic Engineering*, vol. 84386, p. V06BT06A024, American Society of Mechanical Engineers, 2020.

[60] T. Podder, M. Sibenac, and J. Bellingham, "Auv docking system for sustainable science missions," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 5, pp. 4478–4484, IEEE, 2004.

[61] M. Haji, J. Norheim, and O. L. de Weck, "A framework for the design of renewably powered offshore auv servicing platforms," in *Ocean Sciences Meeting 2020*, AGU, 2020.

[62] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.