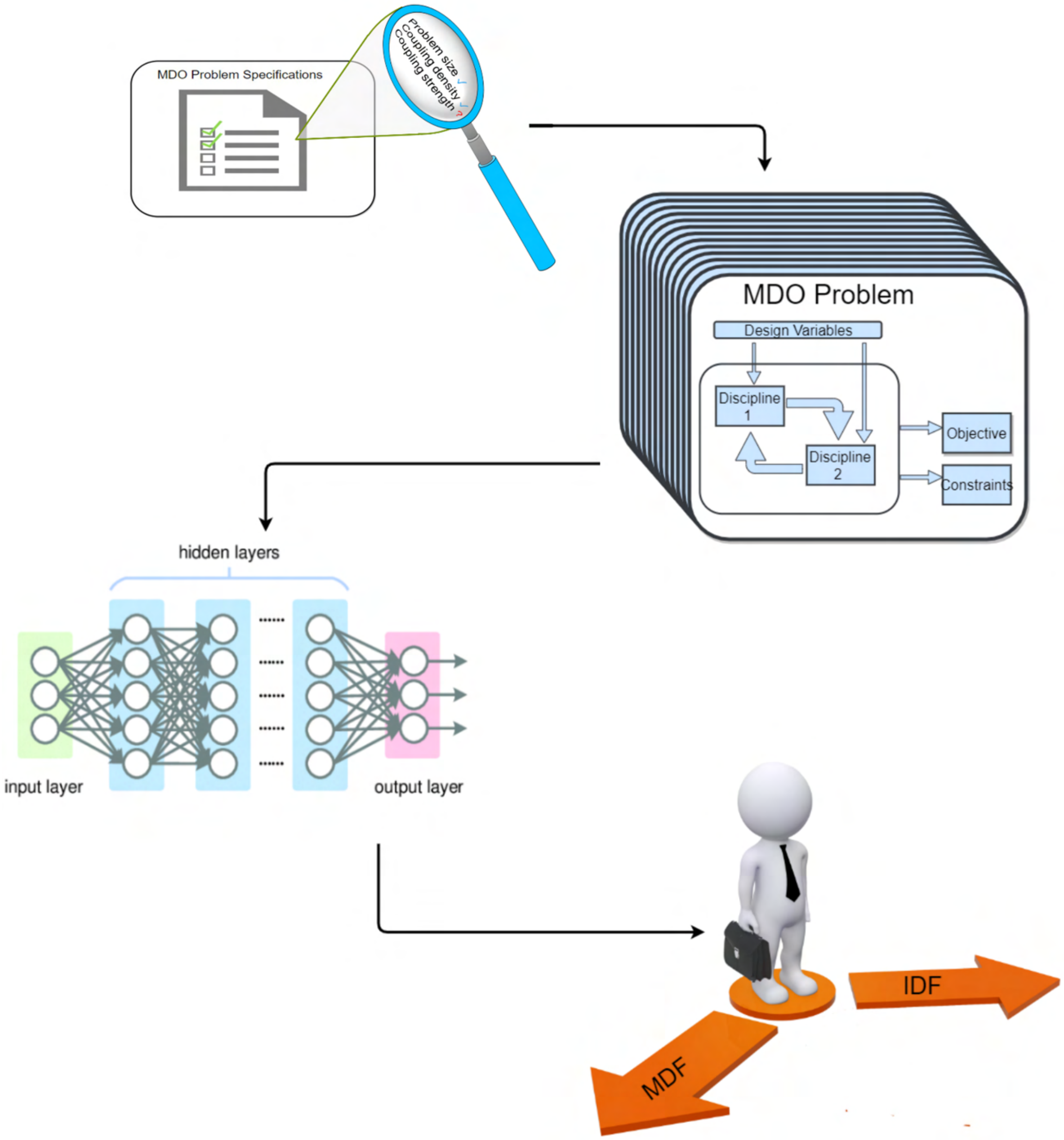
Advisory System for MDO Architecture Selection in the MDO System Formulation Stage

A. Marik



Advisory System for MDO Architecture Selection in the MDO System Formulation Stage

by

Arnab Marik

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday, October 13, 2021 at 12:30 PM.

Student number:	4622405	
Project duration:	January, 2019 – September 2021	
Thesis committee:	Prof. dr. ir. G. La Rocca,	TU Delft, Committee Chair
	Dr. M. F. M. Hoogreef,	TU Delft
	Dr. O. A. Sharpanskykh,	TU Delft

TU Delft

Summary

Machine learning is not a new phenomenon. Having evolved from pattern recognition and computational learning theory, it can be defined as a set of techniques that can learn from an existing repository of data, to make certain predictions on a similar set of data without being specifically programmed to do so. Due to the exponential increase in computing power, Machine Learning is being increasingly used for operations in medical, financial, and engineering fields. Some of its applications include surrogate modeling of computationally expensive analysis tools and recommendation systems that can suggest commercials based on a user's search history or automatically filter spam messages from important ones. Multidisciplinary Design Optimization(MDO) is a field that can benefit from a machine learning based recommendation system.

MDO problems deal with the optimization of coupled systems. By considering a specific optimization approach, referred to as an MDO architecture, MDO enables the concurrent optimization of multiple disciplines to arrive at optimal solutions. The structure of an MDO problem can be represented by internal problem parameters such as number of disciplines, number of design variables, and parameters that define the nature of interdisciplinary couplings. The performance of each optimization approach, in terms of solution time, depends on the above-mentioned internal problem parameters and also on execution environment-related parameters such as the number of processor cores used in the optimization process. One of the challenges in the formulation of MDO systems is the selection of the most appropriate MDO architecture, in terms of optimization time, for a given MDO problem.

This has lead to research in the field of benchmarking the performance of MDO architectures. The existing studies follow a similar template. Firstly, by considering a particular class of MDO problems, called scalable problems, a repository of MDO problems is created. Scalable MDO problems are formed in a way that allow the user to alter one or more internal problem parameters to create a particular iteration of an MDO problem. The repository of scalable MDO problems is used to test the performance of MDO architectures. Based on the assessment, certain observations are made regarding the effect that each problem parameter has on the performance of MDO architectures.

A couple of drawbacks are observed from the existing research in this field. Firstly, the benchmarking results obtained from one set of MDO problems cannot be validated on new MDO problems. This is because internal problem parameters used for generating the repository of MDO problems are not defined in a generalized manner. Secondly, not every problem parameter is considered in the same study. This is because the choice of parameters is dependent on the structure and the disciplinary origin of the MDO problems. The focus of this thesis was to resolve the above two drawbacks and make a better comparative study of MDO architectures. Based on the comparative study, it was required to create an advisory system or a prediction model, that could recommend the most appropriate MDO architecture for a given MDO problem, by applying a suitable machine learning algorithm on a database of MDO problems.

To achieve the above goal, a new benchmarking process was developed. This process was inspired by a recently proposed scaling methodology that allowed the user to create a scaled or "transformed" version of an existing MDO problem, by defining a set of internal problem parameters. Unlike earlier studies, the benchmarking process was flexible such that the parameters could be defined in a normalized manner, and the obtained results could be applied to predict the relative performance of MDO architectures on new MDO problems, irrespective of their structure and disciplinary origin. Additionally, the problem agnostic nature of the benchmarking process allowed for a wider choice of problem parameters compared to earlier comparative studies.

The benchmarking process was applied to a well-known MDO problem from literature to create a database of MDO problems. The problems in the database were executed using two MDO architectures, Multiple Discipline Feasible(MDF)(using either Gauss-Seidel or Jacobi convergence scheme) and Individual Discipline Feasible(IDF). Using the outcome of the two executions, a label value was defined that indicated the relative cost of optimization for an MDO problem using IDF and MDF architectures. This label value was calculated and attached to each entry in the database of scaled problems. Following this, an appropriate machine learning algorithm was applied on the database to create a prediction model that could predict the label value(cost ratio of optimization) for a new MDO problem. The prediction model was validated on test MDO problems that were extracted from the literature. The feature-set of the machine learning model was adjusted to improve the predictive performance on the new test problems. The improved prediction model was further tested on a real aircraft based MDO problem. The model was able to predict, with more than eighty percent accuracy, the relative performance of the two MDO architectures on the aircraft based problem.

The impact of the thesis is that, contrary to earlier works, this thesis made significant steps towards a general form of a prediction model that could predict the relative solution cost of two MDO architectures, on any given MDO problem, irrespective of its structure and disciplinary origin.

Future works involve creating a more accurate prediction model by introducing more MDO problems at the training stage. By including optimization settings(as part of the set of problem parameters), the trade-off between solution time and objective accuracy can be brought into consideration in the training process of the prediction model. Additionally, by making use of more complicated machine learning algorithms, a prediction model can be trained that uses as input the entire structure of the MDO problem and not just a set of parameters.

Delft, September 28, 2021

Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial Neural Network
BLISS	Bi-Level Integrated System Synthesis.
CCF	Custom Cost Function
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
CO	Collaborative Optimization
CSSO	Concurrent Subspace Optimization
CST	Class Shape Transformation
DOE	Design of Experiments
DSM	Design Structure Matrix
EVRS	Explained Variance Regression Score
GASEN	Genetic Algorithm based Selective Ensemble
GSE	Global Sensitivity Equations
IDF	Individual Discipline Feasible
KNN	K Nearest Neighbour
LDA	Linear Discriminant Analysis
MDF-GS	Multiple Discipline Feasible - Gauss Seidel
MDF-J	Multiple Discipline Feasible - Jacobi
MDO	Multidisciplinary Design Optimization
MDOIS	Multidisciplinary Design Optimization based on Independent Subspaces
NLBGS	Non Linear Block Gauss Seidel
NLBJ	Non Linear Block Jacobi
OSA	Optimum Sensitivity Analysis
PDE	Partial Differential Equation
QDA	Quadratic Discriminant Analysis
SARF	Scalable Analytic Replacement Function
SLSQP	Sequential Least Squares Programming
SNARC	Stochastic Neural Analog Reinforcement Calculator
SQP	Sequential Quadratic Programming
SSBJ	Super Sonic Business Jet
SVM	Support Vector Machines

List of Symbols

n_x	Size of each design variable
n_y	Size of each coupling variable
d	Coupling density
n_p	Number of processors
n_c	Number of constraint variables
ρ	Coupling strength
N_x	Total size of design space
N_y	Total size of coupling variable space
R	Normalized cost ratio of IDF over MDF architecture
R_t	Normalized ratio of execution time of IDF over MDF architecture
E	Modulus of elasticity

1

Introduction

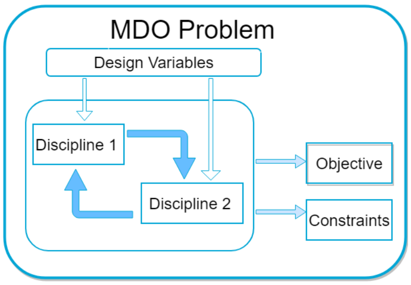


Figure 1.1: MDO problem representation

Conventional product design processes rely on the independent and parallel designing of product components, without taking into account the effect that one component has over the other. This prevents the design optimization of the product from the perspective of multiple disciplines and leaves a portion of the design space unexplored. Due to the constant requirement to build more efficient, sustainable, and high-performance products, especially in the aerospace and automotive sector, there is a push to include more integrated design methods at earlier stages of design. Multidisciplinary Design Optimization(MDO) is a design method that enables interaction between disciplines to maximize performance requirements of products[1]. The mutual interaction between disciplines, referred to as a coupling, enables designers to simultaneously alter design parameters across multiple disciplines[2]. With MDO, designers get the ability to formulate design choices and to analyze them from the perspective of multiple disciplines[3] which leads to a more accurate final solution compared to single discipline optimization methods. A representation of a two discipline MDO problem is shown in Figure 1.1. Alongside the above benefits, MDO also presents new obstacles beyond those found in single discipline optimization methods. Because of the presence of coupled disciplines, MDO problems pose tougher computational challenges compared to the sum of computational effort spent in single discipline optimizations[4]. This has lead to the development of multiple MDO solution strategies, called MDO architectures. Each MDO architecture proposes a unique way to resolve the interdisciplinary coupling present within an MDO problem. This is shown using Figure 1.2.

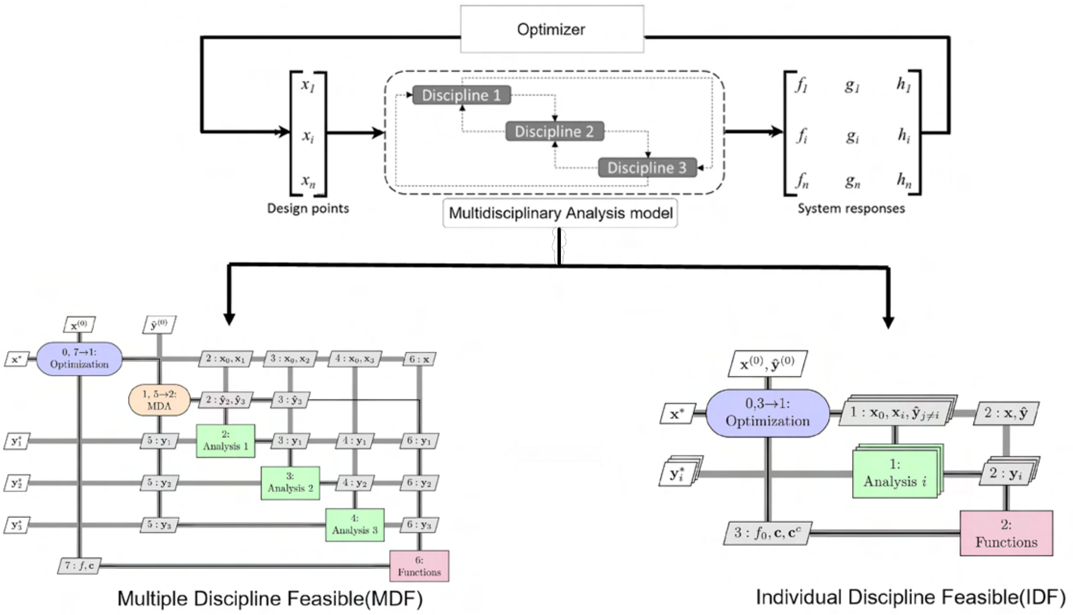


Figure 1.2: Two MDO architectures - MDF and IDF[16]

Figure 1.2 shows an MDO problem containing three coupled disciplines. An Optimizer or optimization algorithm is also shown at the top. The vector of x_i components represents the design points and the system responses g_i and h_i represent inequality and equality constraints respectively. The optimizer handles an MDO problem in the same manner as it handles a single discipline optimization problem. The MDO architecture works at a lower level, and creates a unique version of the **Multi-Disciplinary Analysis(MDA)** model, depending on the method used to resolve the interdisciplinary couplings. Thus, an MDO architecture creates a unique version of a given optimization problem, while preserving the optimization objective and the design constraints.

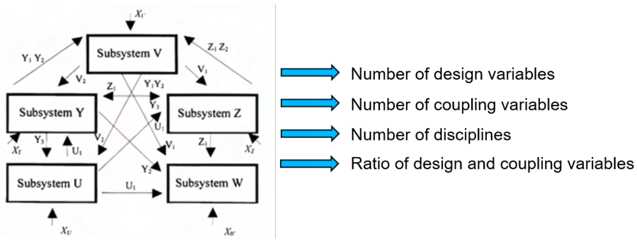

Subsystem V
Subsystem Y
Subsystem Z
Subsystem U
Subsystem W
Number of design variables
Number of coupling variables
Number of disciplines
Ratio of design and coupling variables

Figure 1.3: MDO problem parameters

The formulation of an MDO problem can be represented by a number of **internal problem parameters** as shown in Figure 1.3. Internal parameters, such as the number of design variables or the number of disciplines affect the performance of MDO architectures in terms of computational cost/solution time. One of the challenges in setting up an MDO system is the selection of the most appropriate MDO architecture for a given MDO problem, in terms of the computational cost/solution time. This challenge has led to research in the area of benchmarking MDO architectures with respect to a range of problem parameters[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. The main objective of such studies has been the comparison of MDO architectures, using a repository of MDO problems. The repository of MDO problems is created such that it represents a range of problem parameters. There are two ways of creating this repository. The first is to assemble several random MDO problems from literature, each representing a particular combination of problem parameters[9]. This method allows only a qualitative analysis of MDO architectures and the exact contribution of each problem parameter cannot be analyzed. Another disadvantage to this method is the limited number of MDO problems that are available in the existing literature, which makes the comparative study less elaborate. A better method is to implement the idea of scalable problems[5, 8]. Scalable problems work by extracting certain internal parameters as rules from a given MDO problem. An example of a rule is the ratio of design and coupling variables present in the MDO problem. A scalable MDO problem is built in a way such that each value of the rule generates a particular iteration of the MDO problem. By assigning a range of values to each extracted parameter/rule, it becomes possible to create a repository of a large number of MDO problems, so that the contribution of individual problem parameters can be analyzed. Following this, conclusions are made regarding the effect that problem parameters have on the relative solution cost of MDO problems. An example of this process is given in Figure 1.4

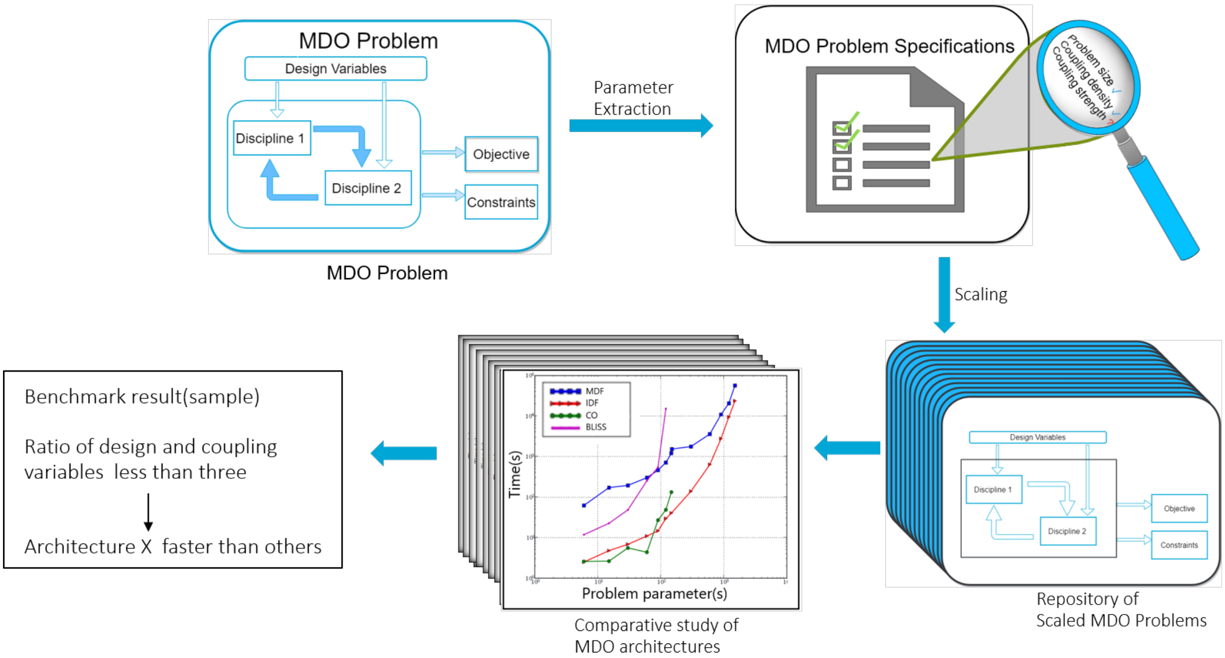


Figure 1.4: Comparative study of MDO architectures - Example

The existing research in this field (using scaled problems) provides evidence that internal problem parameters such as the ones shown in Figure 1.3 have an impact on the comparative performance of MDO architectures. However, there are two shortcomings as discussed below:

1. **Every problem parameter is not analyzed in the same comparative study**
   The scalable problems used in the existing comparative studies belong to one of two classes. One class of scalable problems allows the user to alter the size of the problem in terms of the number of design, coupling, and constraint variables[9, 6, 14]. The second class of scalable problems allows the user to alter the sensitivity of coupling variables with respect to each other[5, 8, 10]. In such problems, a parameter called **coupling strength** or **degree of coupling** is used. The definition of coupling strength varies between existing papers, but the common purpose is to define a single value that captures the sensitivity of interdisciplinary couplings for the entire MDO problem. The value of coupling strength is calculated either by applying sensitivity analysis tools on an MDO problem[15] or by selecting a parameter within the MDO problem definition as an indicator of coupling strength[5]. In an aero-structural optimization problem, for instance, the wing sweep might be considered as an indicator of coupling strength because it's a quantity that alters the sensitivity between coupled structural and aerodynamic disciplines[8]. Within the first class of scalable problems[9, 6, 14], when the number of design or constraint variables are altered, the coupling strength(defined using a particular sensitivity analysis method) also changes. Analyzing the effect of altering the coupling strength independent of the problem size based parameters is not possible in such studies. The same drawback applies to scalable problems that allow the user to alter the coupling strength[5, 8, 10]. Such MDO problems don't give the user the flexibility to make changes in the number of design or coupling variables. The existing studies, therefore, fail to define an all-encompassing scalable problem that allows the user to simultaneously vary the size of the MDO problem as well as the coupling strength.
2. **The parameters used for comparison are not defined in a generalized manner**
   Another drawback of existing studies is that the problem parameters are not defined in a generalized manner. For instance, a comparative study that creates a three discipline scalable problem to analyze the effect of the number of coupling variables on the performance of MDO architectures cannot be validated on a new MDO problem with four disciplines, unless there is a definite pattern to the distribution of those coupling variables. The same applies to parameters like coupling strength that are not normalized using a logical value. Therefore, the results obtained from a particular comparative study cannot be validated on new MDO problems.

A recent investigation in this field was conducted in 2017 by Vanaret et.al.[16]. The scaling methodology proposed in this investigation, the Scalable Analytic Replacement Function(SARF) solves the above drawbacks to a large extent as explained below.
Instead of a proposing a new scalable MDO problem, the method proposes a scaling methodology that allows the user to create a scaled version of an existing MDO problem. The method allows the user to define problem size related parameters in a normalized manner, irrespective of the number of disciplines used in the MDO problem. The method also proposes an additional normalized parameter called the **density of coupling** which allows the user to randomly allocate a certain percentage of interdisciplinary couplings as active . Although the SARF method does not propose a parameter to alter the coupling strength of the MDO problem, the coupling strength remains largely unaffected when an MDO problem is scaled according to the SARF method. The effect of altering the coupling strength can therefore be analyzed by scaling MDO problems of varying coupling strengths and adding it to the repository of scaled MDO problems. Because of these advantages, the research paper by Vanaret et. al.[16] is used as a starting point in this thesis.
In the application part of the investigation by Vanaret et.al.[16], the SARF methodology is used to compare two MDO architectures, Multiple Discipline Feasible(MDF)(using a particular convergence scheme) and Individual Discipline Feasible(IDF). A well-known test case from the NASA MDO test suite[17], the Supersonic Business Jet(SSBJ) [18] MDO problem is used for the comparative study. Two problem size related parameters, related to the number of design variables and the number of coupling variables are used to create a repository of SSBJ based scaled MDO problems and the optimization cost of MDF and IDF architectures is compared. The process is shown in Figure 1.5[16].

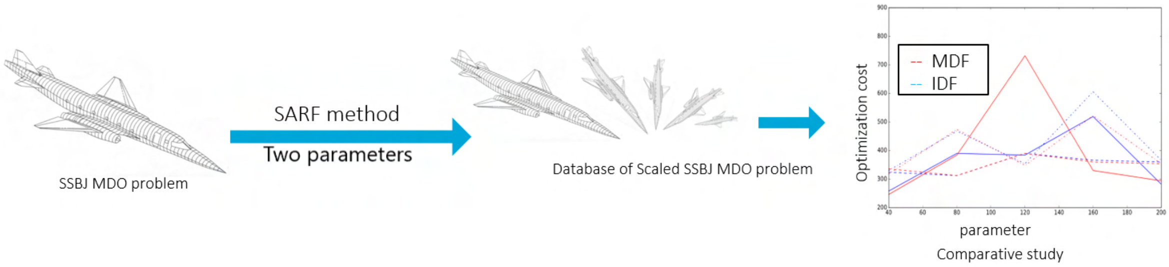


Figure 1.5: Benchmark process using SARF method

Based on the comparative study, it is concluded by Vanaret et. al.[16], that a direct correlation cannot be observed between problem parameters and performance of MDO architectures. However, the following drawbacks can be seen from the comparative study:

1. In the comparative study, each scaled problem(created using a unique combination of problem parameters) is given only one run. However, the SARF scaling method leads to a certain amount of randomness in the scaled problem formulation(because of the coupling density factor). Therefore, it is required to run each scaled problem multiple times and consider the average of the optimization cost to make a fair comparison.
2. Only two parameters are considered (number of design variables and the number of coupling variables) in the comparative study even though the SARF method offers more scaling options such as parameters related to the number of constraints and coupling density.

The implementation of the SARF method on test cases, based on the above-listed shortfalls, can be called rushed and unconvincing. The conclusion made in the paper, that a direct correlation cannot be observed between internal problem parameters and performance of MDO architectures, needs to be revisited by accounting for the above drawbacks. This forms the premise for this thesis project. The intention is to conduct a more thorough comparison of single-level architectures, MDF(using two convergence schemes) and IDF, by implementing the complete dashboard of options provided in the SARF methodology. This is to be followed by the development of a prediction model of MDO architecture by applying a suitable machine learning algorithm on a large database of MDO problems, that have been created using the SARF methodology.

The research question to be answered through this thesis is:

Is it possible to create a prediction model that can recommend the better MDO architecture between MDF and IDF in terms of solution cost, for a given MDO problem, solely based on features that have been extracted from the problem formulation, without executing the problem?

The following three sub-questions can be formed out of the above question:

1. What are the drawbacks of existing comparative studies of MDO architectures and how does the thesis propose to resolve them?
2. How is the prediction model developed and validated?
3. How is the prediction model tested on new MDO problems?

The goal of the thesis is to create a prediction model that can predict the relative performance of MDF and IDF architecture for a previously unseen MDO problem, based on certain pre-execution features of the problem. To fulfill this goal the following steps and sub-steps can be defined:

1. Perform a reproducibility study of the comparative analysis performed by Vanaret et.al[16].
   (a) Apply the SARF method on the Super Sonic Business Jet(SSBJ) MDO problem by selecting the same range and sparsity of parameter values as used in the original study by Vanaret et. al.[16] to create a repository of SSBJ based scaled MDO problems. Perform the comparative analysis of the two MDO architectures.

   (b) Repeat the comparative analysis by including more data points(parameter values) and providing more runs to each data point to find contradictions with the established result in the paper by Vanaret et.al.[16].
   (c) Acquire the motive for pursuing a more elaborate comparative analysis of MDF and IDF architectures based on the SARF method and developing a machine learning based prediction model.

2. Create a prediction model of MDO architecture based on the outcome of the SSBJ based scaled MDO problems.

   (a) Consider the implementation of the SARF Method on the SSBJ problem. Identify the parameters, both internal to the problem as well as related to the optimization setup, that can be used for building the prediction model.
      i. Identify the normalized **predictive parameters**, i.e, the problem parameters that influence the cost of optimization for the two architectures to create a feature set for training a machine learning algorithm.
      ii. Define a normalized **cost parameter** that can account for the relative performance of MDF and IDF architectures on the scaled MDO problems. This forms the outcome of the prediction model.
   (b) Generate a database of SSBJ based scaled MDO problems. The database should contain predictive parameter columns, and a single cost parameter column representing the predicted outcome for each entry in the database.
   (c) Apply four machine learning algorithms onto the database of scaled problems and compare each algorithm's fitting accuracy by selecting an error metric. The best performing algorithm is the first prediction model created in the thesis.

3. Validate the prediction model on new MDO problems.

   (a) Consider three new MDO problems from existing literature. Scale the new problems using the SARF method such that a database of scaled test problems can be formed.
   (b) Apply the prediction model to predict the outcome(cost parameter) of the test cases.
   (c) Improve the prediction on the new problems by identifying a new predictive parameter and re-training the machine learning model. The re-trained machine learning model is the second and final prediction model created in the thesis.

4. Test the re-trained prediction model on original, unscaled MDO problems.

   (a) Introduce two new original, unscaled MDO problems, including an aircraft based MDO case. Also, import the original versions of the existing scaled MDO problems.
   (b) Define a method to extract features from the unscaled MDO problems to create a database that is compatible with the prediction model.
   (c) Apply the re-trained prediction model on the database to predict the cost parameter(relative performance of MDF and IDF architecture) on the unscaled problems and verify the predictions with literature(existing comparative studies).

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

By executing a class of scalable MDO problems with varying characteristics such as number of design variables, number of coupling variables, and the coupling strength between disciplines a database is to be constructed which should enable a suitable machine learning algorithm to train on, using the problem characteristics as features. The trained algorithm should then be able to analyze a new MDO problem and recommend the more effective solution strategy between MDF and IDF.

The thesis is structured into four chapters. Chapter 2 focuses on the Literature Review performed before the thesis project. The topics of MDO and machine learning are discussed. The current state of MDO as a field is reviewed along with a discussion on single level MDO architectures and scalable MDO problems. Examples of comparative studies of MDO architectures from the existing literature are reviewed. This is followed by a discussion on machine learning that covers the current state and outlook of machine learning, examples of machine learning applied to aerospace, and the method of implementing a machine learning based prediction model.

Chapter 3 of the thesis is a reproducibility study of the investigation by Vanaret et.al.[16], which includes a detailed explanation of the SARF methodology, and a repetition of the comparative study made in the original paper by Vanaret et.al.[16]. The reproduced results are compared with the original paper. The effect of making a more robust comparative study(more values between parameter ranges as well as multiple runs to each parameter combination) is looked at. Based on the outcome, a motive is obtained for building a prediction model based on executing a range of MDO problems with the SARF method.

Chapter 4 of the thesis deals with the construction of a machine learning based prediction model of MDO architecture. The data for building the prediction model is extracted by applying the SARF scaling methodology on the SSBJ MDO problem. Based on existing examples of estimation models from literature[19, 20, 21, 22], the idea of **build, validate, and test** is applied to create a prediction model of MDO architecture. The major steps in this process include a visual analysis based feature definition, creation of a database of SSBJ based scaled MDO problems, application of machine learning algorithms to create prediction models, testing the best performing model on new scaled MDO problems, and feature engineering of a new parameter to improve the prediction accuracy.

Chapter 5 of the thesis deals with the testing of the prediction model on original, unscaled MDO problems. For applying the prediction model on unscaled MDO problems, a feature extraction method is defined so that a compatible database of original MDO problems can be created. The prediction obtained by the machine learning model on unscaled MDO problems is verified using examples from earlier comparative studies. Following this, the steps to deploy the prediction model on a given MDO problem are explained.

2
Literature Review

The goal of this thesis is to create a machine learning assisted prediction model, that can recommend the faster MDO architecture(between MDF and IDF) for a given MDO problem. The literature review addresses the topics in MDO and machine learning that are relevant to the goal of this thesis. Firstly, a brief introduction is made on single-level MDO architectures, followed by a review of the existing research where MDO architectures are compared with respect to problem parameters. The drawbacks of the existing research are discussed. The paper by Vanaret et.al.[16] is discussed as the last example and a connection is made between the literature review and the upcoming chapters.

2.1. MDO in Aerospace - History and Outlook

The roots of MDO are found in structural optimization[2]. Very early work on iterative and gradient methods for optimum design of structures was conducted by Sved et al.[23] in 1968, who suggested an alternative solution to Schmit's[24] 3 bar truss constrained optimization problem, which was able to find a global optimum. This early work in structural engineering paved the way for the concept of constrained non-linear optimization to be extended to aerospace applications by including other disciplines such as aerodynamics (aerofoil and wing shape), aircraft performance(mission profile) and propulsion (engineering thermodynamics)[2]. The idea of an **MDO approach** or **architecture** was pioneered by J.Sobieszczanski Sobieski who, in his 1971 technical report[25], discussed the need for the development of a new method for optimization of large multidisciplinary engineering systems. The development of the scheme was to be driven by two factors. One of them was the need to bring mathematical optimization methods to bear on large engineering optimization problems, while the other was the opportunity to take advantage of the recent computer technology developments such as approximate analysis and variable linking, analytical generation of gradient information and piecewise linear optimization of non-linear problems[25]. The development of MDO architectures is based on two major contexts, one being a method to account for the level of coupling present between disciplines, and the other being a method to carry out the overall optimization problem[26]. In this process, two classes of MDO architectures have evolved, monolithic and distributed. Monolithic architectures are based on solving a single optimization problem. Monolithic architectures differ in the management of the coupling between disciplines to achieve multidisciplinary feasibility[26]. Distributed architectures work by decomposing an optimization problem into a set of smaller optimization problems, or sub-problems, that have the same solution when reassembled[26], which enables better handling of certain organizational aspects of an engineering optimization problem. From the context of the current thesis project, only monolithic architectures are discussed in this literature review(section 2.2).

2.2. Monolithic MDO Architectures

Monolithic architectures are single-level MDO formulations where the decision making is centralized and performed by a single optimizer[5]. Following the naming scheme provided in the paper by Martins et al.[26], single-level MDO architectures have four basic variations:

1. MDF(MultiDisciplinary Feasible): Also referred to as NAND(Nested Analysis and Design), MDF uses a single system-level optimizer. The single system-level optimizer supplies the system analyzer with

a design vector, consisting of design variables and the Multi Disciplinary Analyzer(MDA) supplies the optimizer with appropriate response functions in terms of objective and constraints[5, 26]. This approach is desirable if the disciplines are weakly coupled and the multidisciplinary analysis is not computationally expensive[5, 26]. For running an optimization problem with the MDF architecture, it is also required to specify the convergence scheme for the MDA routine. A few convergence schemes exist such as Gauss-Seidel, Jacobi and Newton. There are existing studies that compare the advantages and disadvantages of these convergence schemes[8, 27]. In this thesis, two convergence schemes are used, Gauss-Seidel and Jacobi. While being usually slower than Gauss-Seidel for most applications[28], Jacobi enables parallel evaluation of disciplinary outputs when multiple processors are available.

2. IDF(Individual Discipline Feasible): IDF differs from MDF in the way that the optimizer coordinates the interactions between disciplines, which enables more parallelization compared to MDF-Jacobi. In IDF, the interdisciplinary responses are decoupled from each other and replaced with coupling variable copies. The original coupling variables are implicit functions of the design variables and the coupling variable copies[26]. The performance of IDF does not depend on the strength of coupling present in the MDO problem[5]. For highly coupled problems, IDF is considered to be a more effective architecture than MDF. This comes at the cost of greater centralization[5, 29]. Unlike MDF, the design point is not feasible at every iteration[30].

3. SAND(Simultaneous Analysis and design): Like IDF, SAND is a centralized approach where the residuals of governing equations are also included in the problem statement[5, 26], so additional auxiliary constraints are required to ensure zero residuals at problem convergence[5, 26]. The system optimizer is loaded with three sets of decision variables, coupling variables and the state variables[5] but there are no separate target and response groups, so consistency constraints are eliminated.

4. AAO(All At Once): AAO is the most general form of describing an MDO problem, which includes design variables, response variables, coupling variables and coupling variable copies[5]), as well as consistency constraints and residuals of the governing equations directly in the problem statement[26]. This results in a very large problem formulation.[31]. This also makes AAO the most centralized of all the four monolithic approaches.

Figure 2.1[26] shows the four monolithic approaches from the fully centralized AAO to fully nested MDF approach.

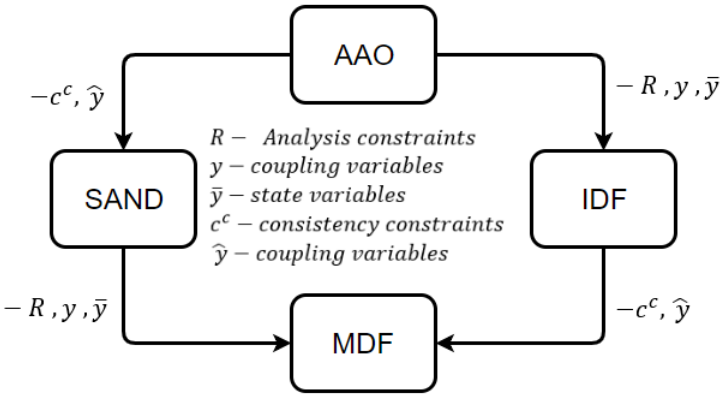

AAO
$-c^c, \hat{y}$
$-R, y, \bar{y}$
SAND
IDF
$R -$ Analysis constraints
$y -$ coupling variables
$\bar{y} -$ state variables
$c^c -$ consistency constraints
$\hat{y} -$ coupling variables
$-R, y, \bar{y}$
$-c^c, \hat{y}$
MDF

Figure 2.1: Four Monolithic Approaches

2.3. Benchmarking MDO architectures with Scalable Problems

The idea of scalable problems and their usage in testing the performance of MDO architectures in terms of solution cost was discussed in the introduction. This part of the Literature Review is used to look at the existing research in this field, where scalable problems are used to benchmark MDO architectures with respect to internal problem parameters (Figure 1.3). This section is divided into two parts. The first part deals with the existing studies that compare MDO architectures from the standpoint of coupling strength. In the second part, the comparative studies are considered which compare the performance of MDO architectures with respect to problem size related parameters, such as the number of design variables and the number of coupling variables.

2.3.1. Coupling Strength

MDO problems differ from a standard constrained nonlinear programming problem due to the presence of disciplinary boundaries. Interdisciplinary couplings are a result of these boundaries. Coupled systems may be viewed as simultaneous systems of nonlinear equations, which are solved with iterative methods such as the fixed point Iterations[5] or with substitute variables. As mentioned in the introduction, coupling strength is a term used to represent the sensitivity of interdisciplinary couplings with respect to each other in an MDO problem. Many researchers[5, 29, 10] have used coupling strength between disciplines as a parameter to test the performance of different MDO architectures. Two such papers are discussed below:

1. A method to test the performance of MDO architectures with respect to coupling strength was conducted by Allison et al.[5], who tested IDF and MDF formulations on the design optimization of a turbine blade. This is an analytic problem that allows for the variation of coupling strength between two of its disciplines, Structural and thermal analysis via coupling variables **dilated length** and **temperature profile** as shown in Figure 2.2[5]. Coupling strength can be adjusted by the modulus of elasticity E. A smaller value of E leads to a more compliant blade by increasing elongation and exposed surface area, which results in a strongly coupled system[5]. Allison et al.[5] tested the dependence of computation time on the coupling strength for MDF and IDF optimization architectures and the result is plotted in Figure 2.3[5].

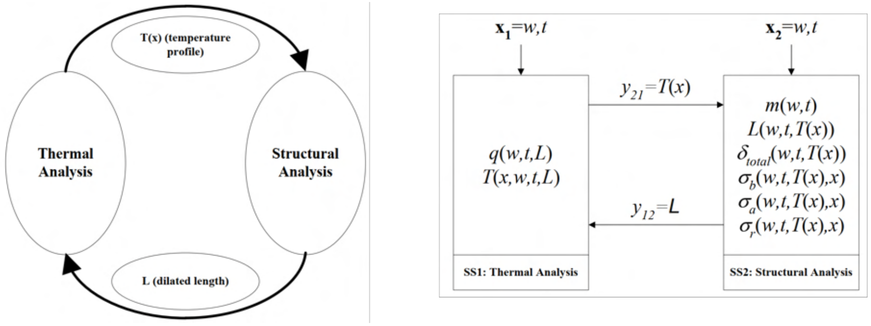


Figure 2.2: Coupling between thermal and structural analysis

Figure 2.3 shows the performance of IDF and MDF(using fixed point iterations as MDA convergence scheme) with respect to coupling strength [5]. It can be observed that weakly coupled systems are more effectively solved with MDF, in terms of computation time while strongly coupled systems require excessive iterations, possibly due to inner analysis loops of MDF[5]. The inherently decoupled IDF approach expectedly maintains a nearly constant computation time, irrespective of the strength of coupling in the problem.

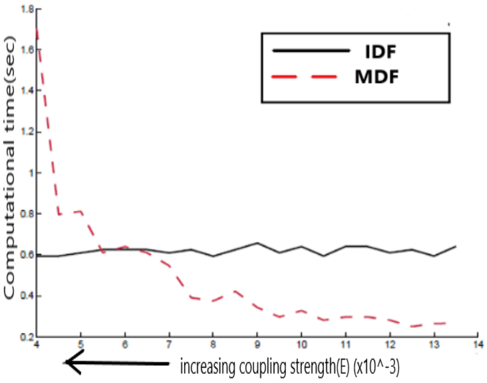


Figure 2.3: Computation time vs coupling strength (E) for MDF and IDF

2. A study was conducted by Balling et al.[29], who tested multiple single-level and distributed architectures on a three discipline system, given as closed-form mathematical expressions and shown in Figure 2.4[29]. The structure of the problem is discussed briefly here. The constraint functions and the coupling functions can be constructed for discipline 1 as follows:

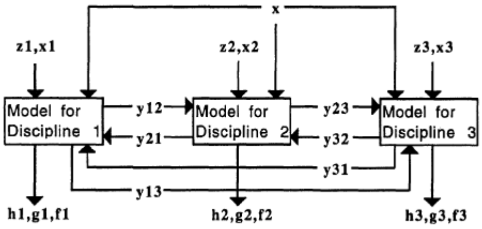


Figure 2.4: Coupled system with 3 disciplines

$$g_{1i} = \prod_{j=1}^{\eta v} v_{1j}^{a_{1ij}} - 1; \qquad g_{1i} = \prod_{j=1}^{\eta v} v_{1j}^{b_{1ij}} - 1$$

$$y_{12i} = \prod_{j=1}^{\eta v} v_{1j}^{c_{1ij}} - 1; \qquad y_{13i} = \prod_{j=1}^{\eta v} v_{1j}^{d_{1ij}} - 1$$

Where vector v_1 contains the values of the input variables for discipline 1 while a_1, b_1, c_1, d_1 are randomly generated exponent values. The above constraints form a convex feasible region as shown in Figure 2.5 [29].

A non-linear, quadratic form for the disciplinary objective function is used as follows:

$$f_1 = 1/2 \sum_{j=1}^{\eta v} e_{1j} (v_{1j})^2$$

$$f = f_1 + f_2 + f_3$$

where the disciplinary optimality equation gives the formulas for the coefficient e_{1j} in terms of $a1_{ij}$, b_{1ij}, c_{1ij}, d_{1ij}.

The problem allows the user to alter the strength of coupling between disciplines using a factor A_{coup}. How the factor A_{coup} integrates into the problem formulation is complicated and therefore not discussed here. A non-zero value of A_{coup} indicates some level of coupling between disciplines.

Balling et al.[29] tested the AAO, IDF, MDF, CSSO(Concurrent Subspace Optimization) and CO(Collaborative Optimization) architectures using different coupling strength values A_{coup}. Figure 2.6[29] shows the convergence plot for a highly coupled system where $A_{coup} = 1$.

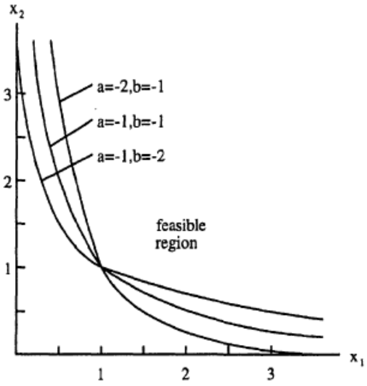


Figure 2.5: Curves of the equation $x_{1\,a}\, x_{2\,b} - 1 = 0$

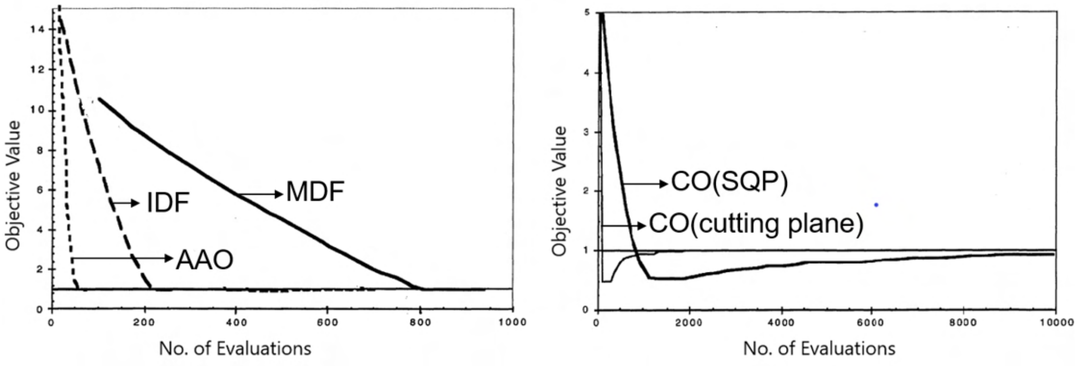


Figure 2.6: Normalized objective vs no. of evaluation(for a highly coupled MDO system)
(left: single-level architectures, right: Collaborative Optimization)
Feasible solution not found with CSSO

It can be seen in Figure 2.6(right) that CO approach uses two algorithms, SQP and cutting plane while the single-level approaches use only the SQP algorithm. The relevant observation that can be made from the figure is that the multi-level CO approach(using both SQP and cutting plane algorithms) takes an order of magnitude higher number of function evaluations compared to the single-level approaches, which reflects their inefficiency. Among the single-level architectures shown in Figure 2.6(left), AAO requires the least number of evaluations, followed by IDF and MDF. Since the disciplines are highly coupled ($A_{coup} = 1$), it is no surprise that the AAO and IDF approaches are faster than the MDF approach, which requires more function evaluations due to the presence of system analyzer[29]. This trend is similar to one observed by Allison et al.[5], where higher coupling strength prefers a more centralized MDO approach.

A point of difference between the comparison made by Allison et al.[5] and Balling et al.[29] is that Allison et al.[5] used computation time while Balling et al.[29] used function evaluations. This could be because Allison et. al.[5] defined a real-world MDO problem based on turbine blade elongation while Balling et al.[29] defined a problem based on mathematical disciplines. While solving engineering problems, the time for an approximation process and a function call can be more dominant than the time of optimization [9]. Therefore, comparison of MDO methods on mathematical problems is made on function calls rather than solution time.

Looking at the above two studies from the context of the drawbacks discussed in the Introduction, it can be seen that the results obtained from the comparative plot by Allison et. al.[5] cannot be used to make predictions on other problems. This is because the coupling strength metric, represented by the modulus of elasticity, is not normalized with a logical value. Additionally, the modulus of elasticity is considered to be indicative of the coupling strength based on background knowledge of the problem. Such a quantity might not be available on different MDO problems. The study by Balling et al.[29] looks at the computational effort utilized by MDO architectures for solving a highly coupled MDO problem. The coupling strength factor is again, not defined in a normalized manner. From the context of MDF and IDF, the only takeaway is that for a highly coupled problem, IDF is nearly four times faster than MDF.

While both the studies show IDF to be faster for strongly coupled MDO problems, the studies are not suitable to be applied on new MDO problems. Additionally, for both the papers it can be seen that there is no provision to study the effect of problem size (design/ coupling/ constraint vectors) on the performance of MDO architectures.

2.3.2. Problem Size

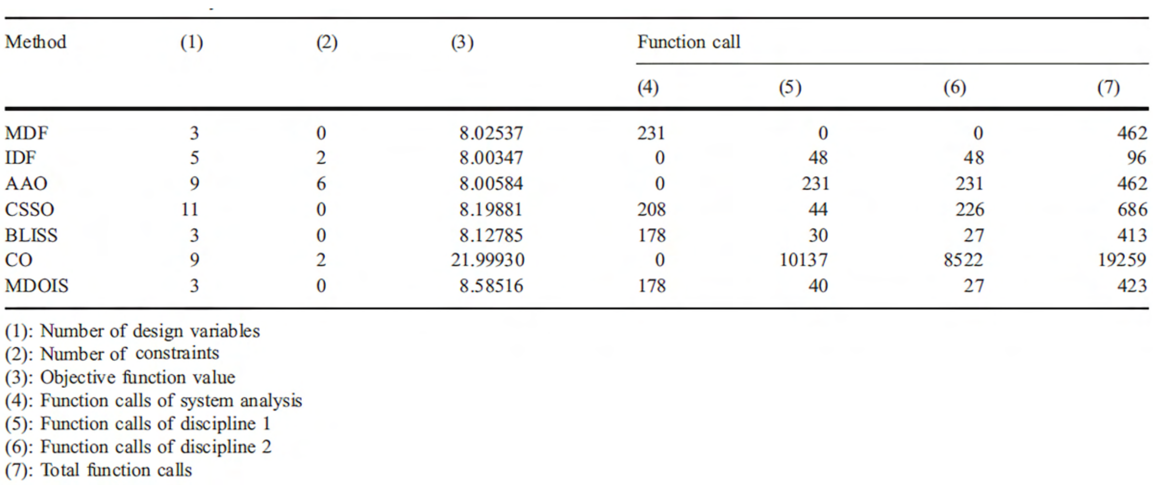
The problem size for an MDO problem can be represented in terms of the number of disciplines, the number of global/local design variables, the number of coupling variables and the number of global/local constraints. There are existing papers, which compare the performance of MDO architectures with respect to problem size related parameters. In this section, three of these papers are discussed.

1. A study was conducted by Shin et al.[9], who tested and compared MDO methods using seven mathematical problems from the iSIGHT manual[32] and the NASA MDO test suite[17] . One of the problems is shown below:

   Find: b_1, b_2, b_3
   To minimize: $z_1 + z_2$
   Subject to the following constraints:
   $g_1 = z_3 \geq 0 \quad ; \quad g_2 = z_6 \geq 0$
   and the following residual equations:

   $$z_1 = b_1^2 + b_2 + b_3 - 4 - 0.2z_4 \quad ; \quad z_2 = z_1 + (b_1 - 2)^2$$
   $$z_3 = z_1/8 - 1 \quad ; \quad z_4 = b_1 + b_3 - 2 + \sqrt{z_1}$$
   $$z_5 = b_3 - 2 + \exp - z_4 \quad ; \quad z_6 = 1 - z_4/10$$

   The problem is solved using the MDF, IDF, AAO, CSSO, BLISS, CO and MDOIS approaches. The results are tabulated below followed by the observations:



Method	(1)	(2)	(3)	Function call			
				(4)	(5)	(6)	(7)
MDF	3	0	8.02537	231	0	0	462
IDF	5	2	8.00347	0	48	48	96
AAO	9	6	8.00584	0	231	231	462
CSSO	11	0	8.19881	208	44	226	686
BLISS	3	0	8.12785	178	30	27	413
CO	9	2	21.99930	0	10137	8522	19259
MDOIS	3	0	8.58516	178	40	27	423

(1): Number of design variables
(2): Number of constraints
(3): Objective function value
(4): Function calls of system analysis
(5): Function calls of discipline 1
(6): Function calls of discipline 2
(7): Total function calls

Table 2.1: Performance of 7 approaches in terms of function calls

It can be observed in Table 2.1[9] that IDF has the least number of function calls while MDF has the most. BLISS has the best objective function value, while CO is the worst performing approach both in terms of number of function calls and objective value. To vary the problem size, Shin et al.[9] constructed six more mathematical examples. The example problems simulated the effect of using only local design variables, using only common design variables and using both local and common design variables. Based on the multiple problem formulations and testing, Shin et al.[9] made the following conclusions:

(a) MDF is recommended when system analysis is simple, but when it's difficult to construct a system analysis, then IDF/ AAO are more effective[9].

(b) It is difficult to apply CO to MDO problems because of the larger number of function calls and worse convergence[9].

(c) In the above set of problems, the GSE(Global Sensitivity Equations) and OSA(Optimum sensitivity Analysis) are analytically calculated whereas, in engineering problems, finite difference might be the only resort, which increases the function calls. But if sensitivity information can be easily calculated then BLISS might be a competitive multi-level method.

Because the comparison is made on the basis of a repository of imported MDO problems(each having different problem size in terms of the number of design and constraint variables) and not scalable problems, the above paper does not provide a direct comparison of the performance of MDO architectures with respect to well-defined problem size parameters. The paper also does not discuss the effect of coupling strength. The following paper by Tedford[6], relies on using a scalable mathematical problem and gives a direct comparison of the performance of single-level MDO approaches with respect to problem size (in terms of the number of design and coupling variables).

2. Tedford[6] created a scalable problem to investigate the effects of altering the problem dimensionality (in terms of local and coupling variables) on MDO architectures. The optimization problem statement for the scalable problem setup is given as follows:

$Minimize: \sum Z^2 + \sum y_i^2$
w.r.t: z, x_i
with $i = 1, 2, ...N$
With the disciplines defined as follows:
$Discipline_i:$
$given: z, x_i$(from system-level); y_j(from other disciplines)
$solve: y_i = -1/C_{yi}(C_z Z + C_{xi} x_i - C_{yj} y_j)$

Two separate investigations are performed. In the first investigation the number of disciplines, global design variables and coupling variables are held constant and the number of local design variables associated with each discipline is varied. In the second, the number of coupling variables is varied, keeping the other variables constant. Figure 2.7[6] shows the results of the optimization time taken by three approaches MDF, IDF and SAND.

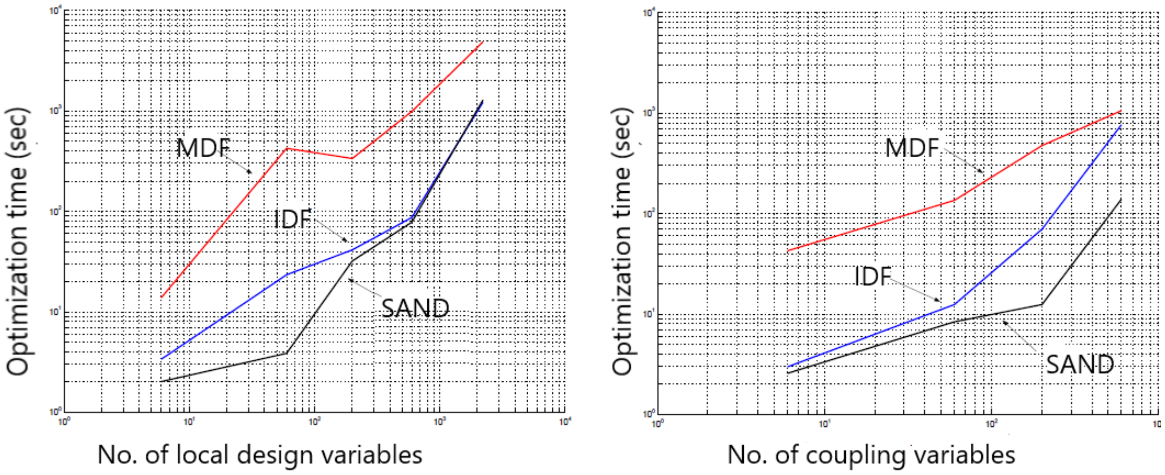


Figure 2.7: Scalable problem convergence times - left: Design variable investigation, right: coupling variable investigation

It can be observed that in both cases, SAND and IDF consistently outperform MDF. Though the state variables are identical for each investigated problem, SAND's use of state variables and residual constraints seems to allow it to converge slightly more rapidly than the structurally similar IDF[6]. It can be seen that the slope of the MDF line is less than that of IDF. Tedford claimed in his study, that provided results can be obtained beyond the 600 coupling variable mark, MDF may begin to outperform IDF. His claim remained unsubstantiated until 2010, when an improvement to the above analysis was made by Tedford and Martins[7] in another paper. In this paper, a more thorough investigation was conducted on the same scalable problem by including more data points, spread across a larger range of problem size parameters as shown in the logarithmic plot of Figure 2.8[7].

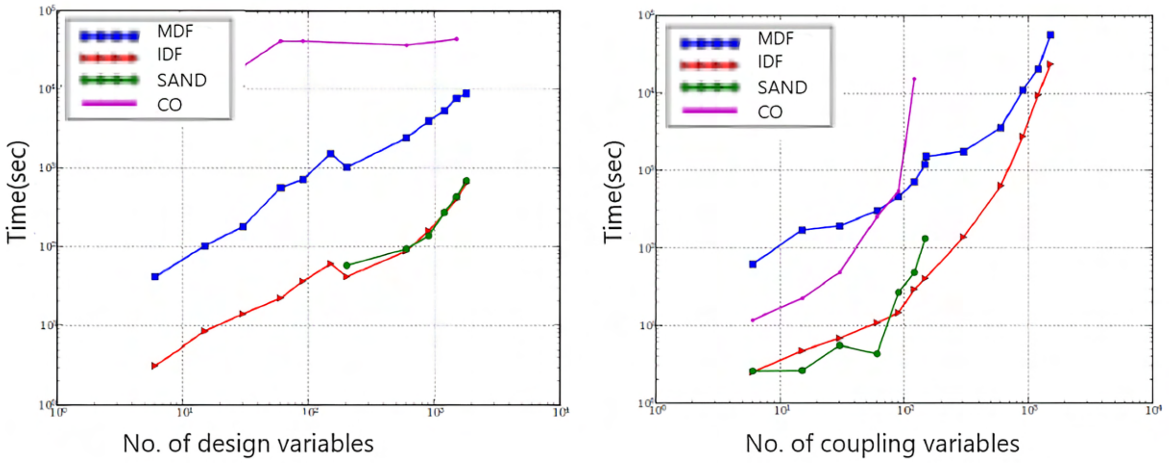


Figure 2.8: Scalable problem convergence times - left: Design variable investigation, right: coupling variable investigation

The results obtained by Tedford and Martins[7] (Figure2.8) can be seen as a refined version of the earlier result by Tedford[6] (Figure2.7). Altering the number of design variables seems to have a lesser impact on the relative performance of MDF and IDF architectures as compared to the coupling variables. For an extraordinarily large number of coupling variables($> 10^3$), IDF loses its solution time advantage over MDF. As the number of coupling variables increase, so does the number of consistency constraints that have to be satisfied at every optimizer iteration. This reduces the effectiveness of IDF at higher dimensionality of coupling variables[7].

The above papers[6, 7] give a good example of a scalable problem that is used to make a direct comparison of MDF and IDF architectures. The scalable problem allows the user to independently alter the number of design and coupling variables. However, it can be seen from the structure of the problem that the number of disciplines is fixed to the number of coupling variables. This reduces the applicability of the results to MDO problems where the number of disciplines is independent of the number of coupling variables. Additionally, there is no provision to change the coupling strength of the problem.

The research papers by Tedford and Martins[7] serve as a starting point for the work done by Vanaret et al.[16] who suggested that the above study as well as other existing scalable problems are limited to disciplines that contain simple mathematical expressions, and are not inspired from real MDO problems that are defined by aerospace or automobile engineers. These problems often include disciplinary outputs(couplings/constraints/objective) that have higher-order convexities with respect to the design space which cannot be emulated using mathematical expressions. Vanaret et al.[16] suggested the need for:

(a) An analytic replacement function that is not only scalable with design and coupling variable parameters but also able to preserve the mathematical structure of a real MDO problem.

(b) An inclusive cost function that gives an accurate estimate of the Optimization cost, by taking into account the various architecture-specific sub-steps within an optimization procedure.

The scaling methodology suggested by Vanaret et. al.[16] is discussed next.

3. Vanaret et al.[16] proposed to benchmark different MDO architectures by replacing an existing MDO problem with a SARF based problem(previously mentioned in the introduction) that can be scaled in terms of the problem size, yet captures the structure and behavior(convexity of outputs w.r.t inputs for each discipline) of the original problem[16]. The process is explained using Figure 2.9

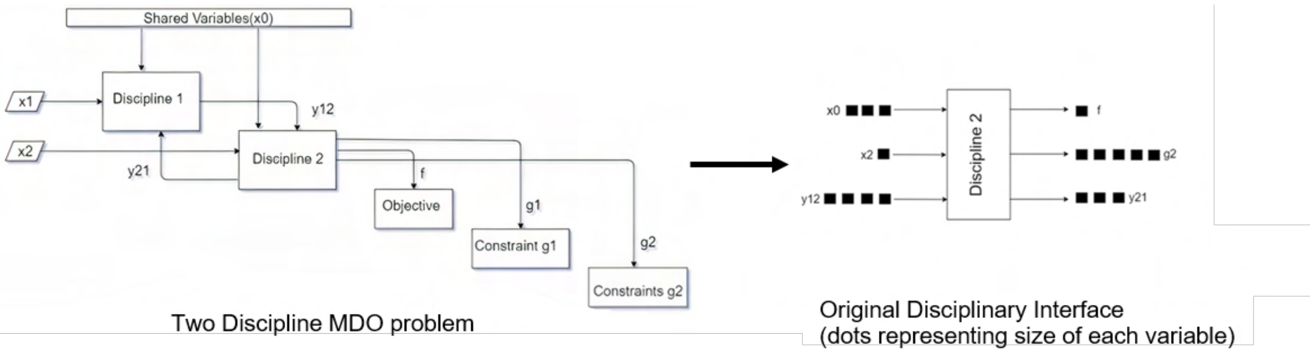


Figure 2.9: A two discipline MDO problem

Figure 2.9 shows a two discipline MDO problem with shared(x_0) and local(x_1, x_2) design vectors along with vectors representing constraints(g_1, g_2) and coupling(y_{12}, y_{21}) functions. The objective is represented by f. Figure 2.9 also shows the formalization of the inputs and outputs for Discipline 2 in the form of a disciplinary interface. In the SARF method, each input and output for a discipline is considered to be a vector. Each "square dot" represents the size of the corresponding vector. Since y_{12} is a vector of size four, it conveys that four response variables from Discipline 1 serve as inputs to Discipline 2. The SARF scaling process for Discipline 2 is demonstrated in Figure 2.10 and 2.11

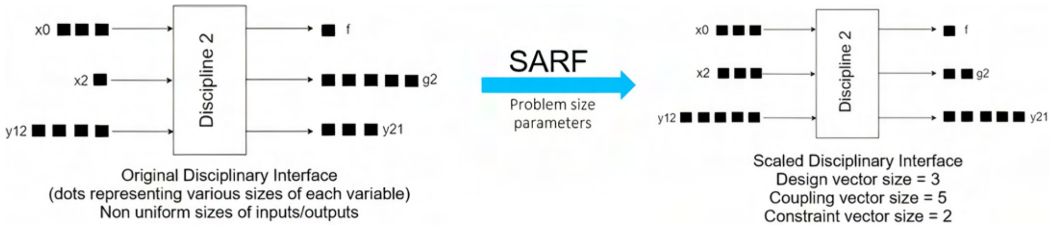

x0
x2
y12
Discipline 2
f
g2
y21
Original Disciplinary Interface
(dots representing various sizes of each variable)
Non uniform sizes of inputs/outputs
SARF
Problem size parameters
Scaled Disciplinary Interface
Design vector size = 3
Coupling vector size = 5
Constraint vector size = 2

Figure 2.10: SARF scaling process(Problem size) for Discipline 2

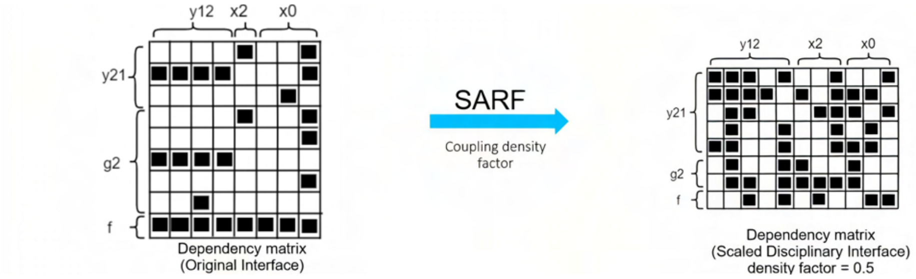


Figure 2.11: SARF scaling process(Dependency matrix) for Discipline 2

The SARF scaling process scales each discipline by defining the following parameters:

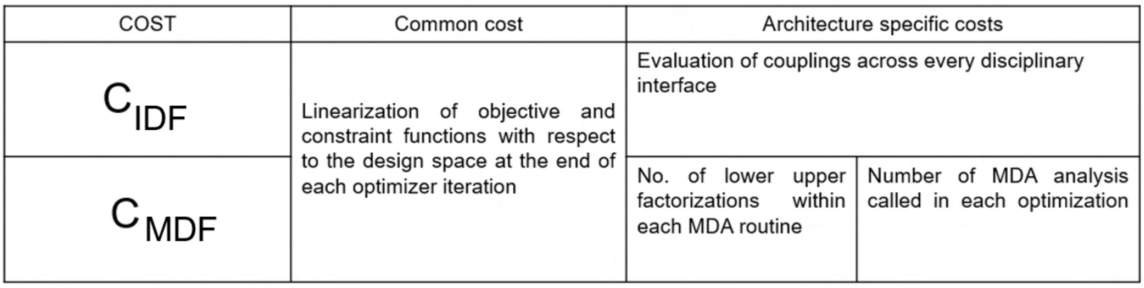
(a) Three parameters to alter the size of input, coupling and constraint vectors.

(b) One parameter to alter the **coupling density**(explained below) within each disciplinary interface.

Looking at Figure 2.10(left), it can be seen that the original disciplinary interface for Discipline 2 has non-uniform sizes of inputs and outputs. The SARF method allows the user to define three problem size parameters to set the size of design vectors, coupling vectors and constraint vectors. In this case, the parameter values are three, five and two respectively. It can be seen that within the scaled disciplinary interface(Figure 2.10(right)), both input and output coupling vectors have a size of five. Similarly, each design variable vector(shared or local) receives a dimension of three, and constraint vectors g_1 and g_2 receive a dimension of two. The SARF method proposes another factor that gives the user the ability to set the percentage of dependencies between inputs and outputs for the interface. Figure 2.11(left) shows the dependency between inputs(along the abscissa) and outputs(along the ordinate) for the original disciplinary interface for Discipline 2. A "square dot" in this case represents the existence of a dependency. The SARF method allows the user to define a **coupling density factor** for the scaled disciplinary interface for Discipline 2. For a coupling density factor of 0.5(Figure 2.11(right)), this results in fifty percent of the dependencies being existent. The distribution of dependencies is random. For problems with more than two disciplines, the same values of the above defined parameters are applied. Since the problem size related parameters are defined for the whole problem the same parameter value applies to each design, coupling, and constraint variable irrespective of the number of disciplines. Similarly, the same coupling density factor is used to generate a random dependency matrix for each disciplinary interface. Therefore, unlike the scalable problems used in the previous studies[6, 7, 9], the scaling parameters used by the SARF method work independent of the number of disciplines. More details on this method, including the way the scaled outputs are calculated, are discussed in the next chapter.

As far as coupling strength is concerned, the SARF method is similar to the previous scaling processes related to problem size in the way that it does not explicitly offer a parameter to alter the coupling strength of the MDO problem. However, due to the mathematical structure of the original problem being preserved, the coupling strength of the scaled MDO problem does not get affected by the SARF scaling process. The feature regarding the coupling strength retention is discussed in detail in the next chapter. Due to the coupling strength retention and the problem agnostic nature of the SARF method, several MDO problems can be simultaneously scaled to create a repository of SARF based scaled MDO problems representing a range of both problem sizes and coupling strength.

For benchmarking MDF-GS and IDF approaches, the SARF method proposes a cost criterion(C) as shown below:

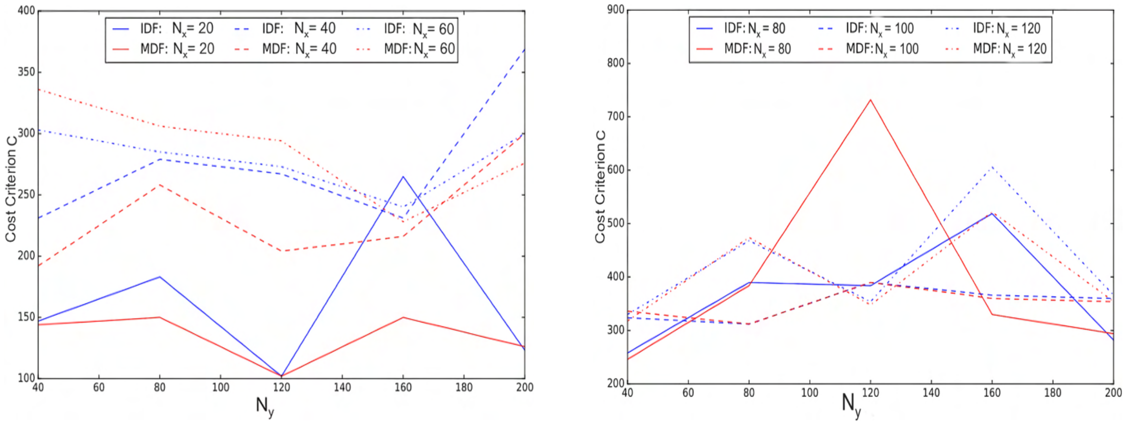


COST	Common cost	Architecture specific costs	
C_{IDF}	Linearization of objective and constraint functions with respect to the design space at the end of each optimizer iteration	Evaluation of couplings across every disciplinary interface	
C_{MDF}		No. of lower upper factorizations within each MDA routine	Number of MDA analysis called in each optimization

Figure 2.12: Cost Criterion for MDF-GS and IDF architectures

The cost criterion consists of common as well as architecture-specific costs. The mathematical details of the cost criterion are discussed in the next chapter.

A comparative study of MDF-GS and IDF architectures was performed by Vanaret et.al[16] by considering a scaled version of the SSBJ problem(introduced in Chapter 1). Out of the above discussed four parameters, the comparison uses two, representing the size of design and coupling vectors. The benchmark uses two symbols N_x and N_y representing the cumulative size of design and coupling vectors respectively. So, for the scaled version of the two discipline problem shown above(Figure 2.9), $N_x = size(x_0) + size(x_1) + size(x_2) = 12$, $N_y = size(y_{12}) + size(y_{21}) = 10$. Vanaret et al.[16] applied the above cost criterion to a repository of SSBJ based scaled MDO problems and the results are plotted in Figure 2.13 [16]



(a) N_x = [20; 60], N_y= [40; 200]

(b) N_x = [80; 120], N_y= [40; 200]

Figure 2.13: Estimated cost for MDF-GS and IDF optimizations on the SSBJ test case

The above plot does not show a definite trend in the cost criterion with N_x and N_y. There are some values of (N_x, N_y) which show dips in the cost criterion. For some values of (N_x, N_y) such as (20,160), the cost is exceptionally high. Vanaret et. al.[16] concluded that a clear trend cannot be observed between the cost criterion and the problem parameters. However, there are three shortcomings in this analysis:

(a) It can be observed that the cost function has only been calculated for a few (N_x, N_y) pairs.

(b) There is a certain amount of randomness in the SARF based scaled problem formulation due to the density factor, which randomly distributes the dependencies. This can be accounted for by having multiple runs for each combination of problem parameters.

(c) Other parameters such as the coupling density factor and constraint vector size are available in the SARF methodology but not considered in the comparative study.

> It can be said that the SARF based scaling methodology proposed by Vanaret at.al.[16] is better than the earlier scalable problems discussed in this literature review, but the comparative study performed using the SARF method is limited and not conclusive. This opens the possibility for further investigation. The next chapter(Chapter 3) consists of a reproducibility study of the research paper by Vanaret et. al.[16]. In this chapter, firstly, the SARF method is discussed in detail. Following this, by accounting for the above stated shortcomings, the conclusion made in the original paper([16]) regarding the lack of correlation between the problem parameters and solution cost of MDO architectures is challenged.

The final part of the literature review is a brief introduction to machine learning.

2.4. Machine Learning - History and Outlook

Machine learning is a new buzzword in the technological space, but the underlying concepts have been in use for a long time. One of the earliest series of papers titled **Method of least squares**[33] was published by Gauss and Legendre, which showed the application of what is now referred to as **Linear Regression**. Towards the end of the 18th century, this approach was initially applied in observational astronomy to quantify and calibrate the cosmic distance scale necessary for the study of the large scale structure of the universe[34]. In 1913, Andrey Markov, a Russian mathematician best known for his work on stochastic(random) processes, invented a set of techniques known as **Markov chains**, which he used to analyze poems[35]. In 1950, Alan Turing proposed a learning machine that could learn and become artificially intelligent. The scope of application to real-world problems was still restricted, until a year later when Marvin Minsky and Dean Edmonds made the first neural network machine called SNARC. Finally, in 1952, Arthur Samuel, also known as **The Pioneer in Machine Learning** began development on some of the very first machine learning programs and created a program that could play checkers[36]. Until 1930, Linear regression was mainly used for predicting quantitative values such as an employee's salary[37], the distance between celestial objects, and stock prices. For predicting qualitative values, such as the rate of default within Loan applicants, the mortality rate for a particular disease, or the outcome of a football match, other models had to be developed that could predict responses with categorical variables. These are called classification models. Among these models, logistic regression and linear discriminant analysis are the ones used most widely[37]. More computationally intensive methods, such as **K nearest neighbours, Decision trees and Support Vector Machines(SVM)** also exist, which can be used for both classification and regression tasks. In the past 20 years, a particular set of techniques called **Artificial Neural Networks(ANN)** have become popular as the first choice method for processing most types of machine learning tasks. These techniques can be referred to as a digital form of the original SNARC neural network build by Minsky, replacing potentiometers and electro-mechanical components with digital weights[1]. These methods serve the same purpose as the previous ones i.e. solving certain classification or regression tasks, but they have a basis in biology, in the sense that they attempt to mimic biological neurons with an artificial neuron, known as the perceptron.

2.5. Machine Learning application in Aerospace

In the field of aerospace, machine learning is used for a wide variety of applications. At airports, machine learning algorithms are being used to improve and accelerate threat and anomaly detection in areas such as baggage scanning and queue monitoring at security checkpoints[2]. Machine learning algorithms can be applied on large telemetry data, accumulated over thousands of aircraft/spacecraft missions to ensure better safety and fault diagnosis[38]. In areas that are closer to aircraft design, machine learning is used for meta-modeling of analytical disciplines[22, 39, 19, 20, 21] and cost estimation/ performance prediction of airplane components[21]. Two such papers are discussed below.

2.5.1. Drag coefficient prediction of wing configurations

An example of neural network based meta-modeling was proposed by Secco et. al.[39]. This approach uses artificial neural networks(ANN) for predicting aerodynamic drag coefficients of wing configurations to replace a full potential code with viscous corrections. The first step is to generate a database using inputs for the original analytical discipline and drag prediction. The full potential code comprises of forty inputs, that are used to define the geometry of the wing, the shape of the airfoil at three span locations, and flying conditions. Table 2.2 shows the range of these inputs.

[1] https://historyof.ai/snarc/

[2] https://www.airport-suppliers.com/supplier/aurora-ai/

Type of input	Input	Range
Flying Conditions (3 inputs)	Mach Number	0.2-0.8
	Altitude	0-13000m
	Angle of attack	-3° – 6°
Wing Geometry (10 inputs)	Wing Aspect Ratio	7- 11.5
	Wing Taper Ratio	0.2 - 0.6
	.	.
	.	.
	.	.
	Inboard Wing Dihedral	0° – 5°
	Wing Area	50-200
Airfoil Shape (27 inputs)	Maximum relative thickness(root)	0.1- 0.16
	Leading edge radius(root)	0.005 - 0.2
	.	.
	.	.
	.	.
	Location of the maximum relative thickness(tip)	0.2 - 0.46
	Maximum camber(tip)	0-0.03

Table 2.2: List/Range of inputs

The inputs and their ranges shown in Table 2.2 represent the training data on which the neural network algorithm is to be trained. In creating a database for training a machine learning algorithm, each input is assigned to a *feature* or a *predictor* column while the quantity to be predicted, which is the overall drag coefficient in this case, is assigned to a *label* column. By considering each unique combination of input values as a "data point", the database for training a machine learning algorithm is built. Table 2.3 represents a training database consisting of 140000 rows and forty-one columns, of which forty are feature columns used to train the neural network model, while the last column represented by the drag coefficient is the predicted quantity. Once the database is created, it is required to normalize each input variable within the [-1, 1] interval. According to Secco et. al.[39], creating a normalized database is mandatory for running the neural network with the particular optimization algorithm used in the study (Nguyen-Widrow weights initialization algorithm). Following the normalization, it is required to split the database into training and test sets. By considering a particular ratio, called the train-test split ratio, a certain number of rows are separated from the database, such that the machine learning model can be trained on them. The test rows are used to test the predictive capability of the machine learning model. This is shown in Figure 2.14

Case	Features						Label
	Mach Number	Altitude	Angle of Attack		Maximum Camber(tip)	Leading Edge Radius(tip)	Drag Coefficient
1	0.2	10000	3°	....	0.01	0.005	0.015
2	0.3	12000	3°	....	0.02	0.007	0.016
3	0.4	12000	3°	....	0.02	0.005	0.017
4	0.5	12500	3°	....	0.03	0.008	0.018
5	0.2	10000	3°	....	0.03	0.009	0.019
.	.	.	.	....	.	.	.
.	.	.	.	....	.	.	.
.	.	.	.	....	.	.	.
.	.	.	.	....	.	.	.
139999	0.8	13000	5°	....	0.03	0.2	0.024
140000	0.8	13000	5°	....	0.03	0.2	0.028

Table 2.3: Database for training neural network(forty features and one label)

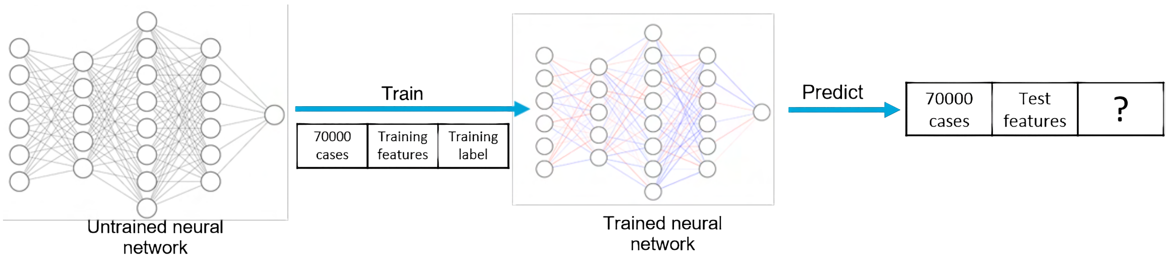


Figure 2.14: Fitting process of neural network

Figure 2.14 shows a train-test split made using a 1:1 train-test split ratio. It must be noted that the neural network shown in Figure 2.14 is just an example and does not represent the actual structure of the neural network model used by Secco et. al.[39]. From the point of view of the thesis, it is not required to review the mathematical details of the optimization process applied within the training of a neural network. The relevant parts to be included in the review are the practical choices made in creating the machine learning model, such as the options selected while fitting the neural network model and the method used to test the fitting accuracy. One of the requirements in implementing a neural network on a database is the selection of the right size of the network in terms of the number of layers and the number of neurons(shown with empty spheres) in each layer. For instance, the example neural network shown in Figure 2.14 consists of three hidden layers in the form of 5 - 8 - 6 neurons. Hidden layers signify the middle layers of the model. In a simple regression problem such as this, the last layer is always a single neuron that outputs the prediction while the first layer always has the same number of neurons as the inputs or features present in the database(should be forty for this problem, but only six are shown). In the paper by Secco et. al.[39], the number of hidden layers is fixed at two and the right number of neurons are discovered using an experimental *grid search* approach. The number of neurons used on the first layer varies between 20 and 100, in steps of 20. The same interval is used for the neurons of the second layer. The performance of each iteration of the neural network is tested using a Mean Square Error (MSE) metric, which is a non-dimensional metric used to measure the prediction accuracy of machine learning models. The MSE is calculated using two arrays of data, representing the predicted test label values(drag coefficients predicted by the neural network model) and the actual test label values(obtained from the full potential solver). The result of the grid search operation is shown in Figure 2.15[39]

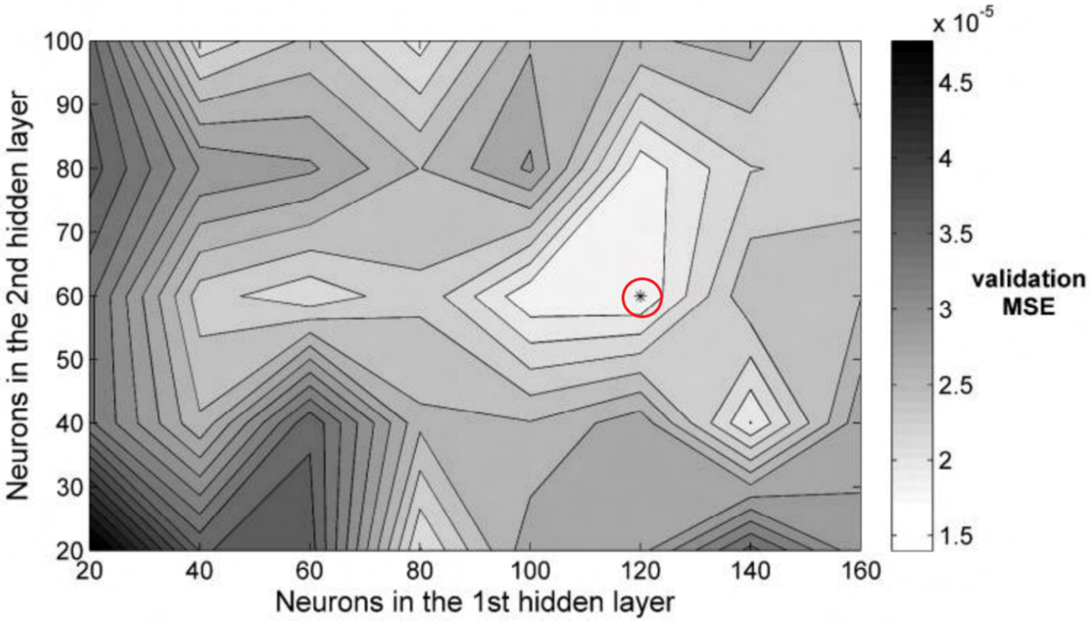
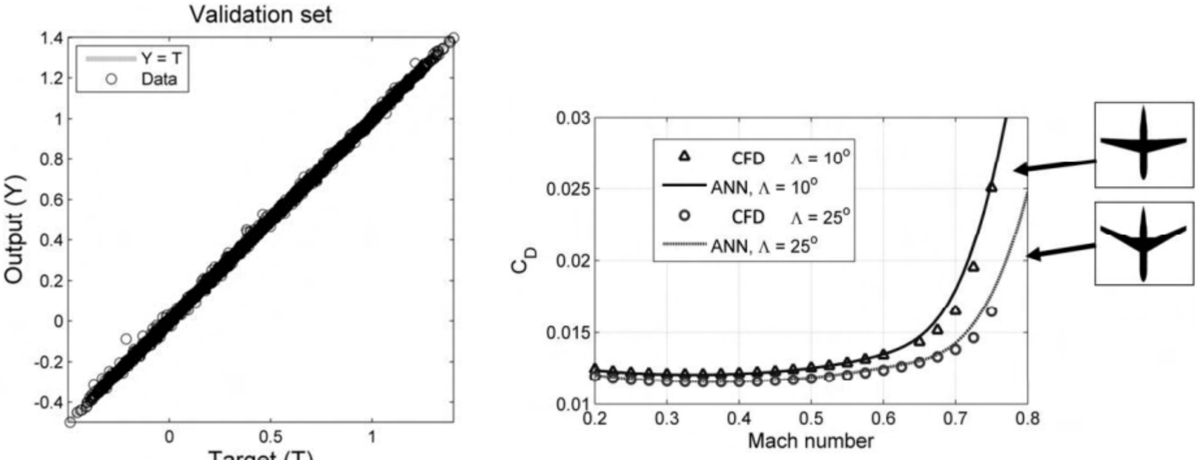


Figure 2.15: Grid Search to find lowest MSE

It can be seen from Figure 2.15[39] that the lowest MSE is obtained for the neural network representing

120 - 60 hidden layer configuration(circled in red). The performance of the corresponding neural network is visualized in two ways. One way is to make a plot of the predicted vs actual test labels as shown in Figure 2.16a[39]. The Target(T) represents the actual test label values while Output(Y) represents the predicted test label values. It can be seen that the prediction accuracy is extremely high for the neural network model as all test points lie very close to the line Y=X(line of perfect prediction) Figure 2.16b[39] shows a second way to plot the performance of the neural network based prediction model. Using certain Mach Number and geometry Inputs the drag prediction from the potential solver and neural network model are compared.



(a) Predicted vs actual test labels

(b) Comparison of Drag prediction from solver and the ANN meta model

Figure 2.16: Visualize performance of optimized neural network

The above study by Secco et. al. [39] presents a workflow of implementing a machine learning based prediction model, including feature definition, database creation, model fitting, and performance testing. A similar workflow is used for implementing machine learning based prediction models later in the thesis (Section 4.2.2.1). The final paper to be looked at in the literature review is about a machine learning based cost estimation tool of airplane components by Loyer et.al.[21]

2.5.2. Estimation of manufacturing cost using machine learning

Loyer et al.[21] proposed a machine learning based method to predict the manufacturing cost of jet engine components during a preliminary design stage. As shown by the neural network representation in Figure 2.17, the data contains six inputs or predictors, two of which are related to geometry(span and chord), two to material properties(machinability, cost rate) and one to the economics of the product (production volume). Additionally, a new feature can be seen as $span * chord$. This is an example of feature engineering, where the background knowledge of the prediction problem is applied to create a feature that helps create a more accurate machine learning model. This paper shows the application of machine learning algorithms on a database which is much smaller than the database used in the earlier paper(Table 2.3), both in terms of number of columns(6 vs 41) and rows(254 vs 140,000). This shows that machine learning algorithms are applicable in a wide variety of data-sets. In this thesis, the database used for building the prediction model(Table 4.4) has a size which is intermediate between the size

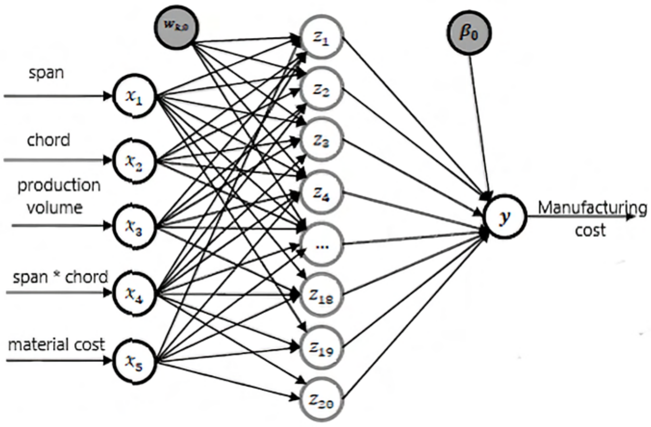


Figure 2.17: Neural network to predict manufacturing cost of components

of the databases used in the above two papers, both in terms of rows and columns. A relevant part of this study is the comparison of various machine learning approaches, that are applied on the database. Loyer et al.[21] looked at five machine learning algorithms:

1. Multiple Linear Regression(MLR)
2. Generalized Additive Models(GAM)
3. Artificial Neural Networks(ANN)
4. Support Vector Regression(SVR)
5. Decision Trees(DT)

Figure 2.18 shows the accuracy of the predictions made by the five algorithms. The figure maps the outcome predicted by the algorithms(predicted cost) vs the actual cost for the five algorithms tested. The points are mapped and a least-squares curve is drawn through them. Ideally, the points should be along $y = x$ line. The SVR(Support Vector Regression) and ANN(Artificial Neural Network) lines seem to be the most aligned with the $y = x$ line, which gives proof of their higher accuracy. A similar process is adopted in the thesis as well(Section 4.2.2), where four machine learning algorithms are applied on a created database of MDO problems and their predictive performance is compared as shown in Figure 2.18.

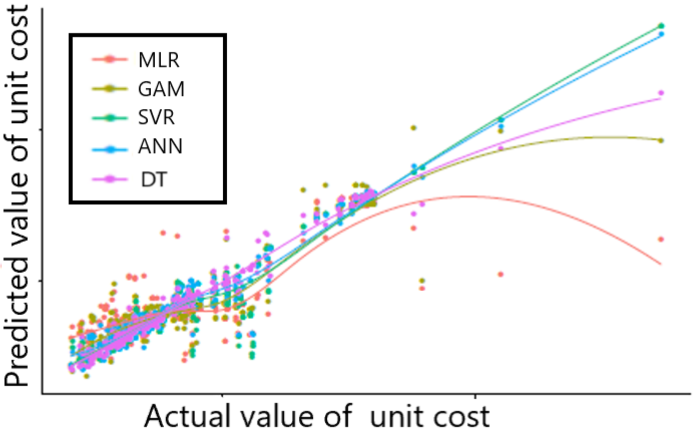


Figure 2.18: Predicted cost vs Actual cost

3

Reproducibility Study - Scalable Analytic Replacement Function

As discussed in the Introduction and the Literature Review, the starting point for the thesis is a reproducibility study of the research paper by Vanaret et. al.[16]. The methodology, called Scalable Analytic Replacement Function(SARF) proposes to create a scalable version of a computationally expensive MDO model, which allows the user to independently vary the dimensions of input and output variables and their dependencies on each other so that the performance of MDO architectures can be compared with respect to such parameters. However, the conclusion made in the paper that a direct correlation cannot be observed between problem parameters and performance of MDO architecture has some pitfalls and requires a better inspection. This was explained in Chapter 2, Section 3. This chapter is used to explain the SARF method and conduct a critical analysis of the conclusions made in the paper by Vanaret et. al.[16]. Based on the critical analysis, a motive is to be gained for creating a prediction model of MDO architecture by expanding on the feature-set provided in the SARF scaling method. Following are the sub-steps within the reproducibility study:

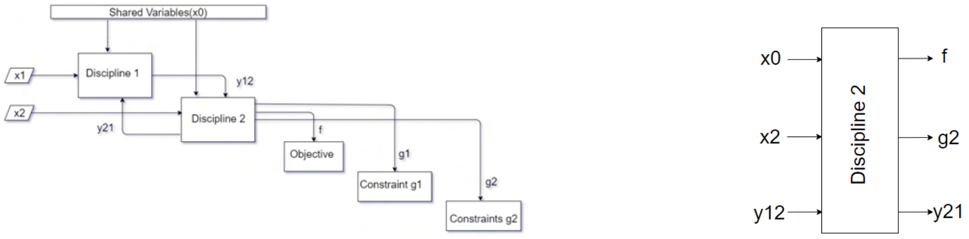
1. An explanation of the SARF methodology used for creating a scaled MDO problem.(Section 3.1)
2. Demonstrating the implementation of the SARF methodology on a test case, the well known SSBJ(Super Sonic Business Jet) benchmark problem.(Section 3.2)
3. Setting up the cost criterion/optimization parameters for performing the critical analysis.(Section 3.3)
4. Performing a critical analysis of the observations provided in the study by Vanaret et al.[16], challenging the existing conclusion(Section 3.4) and acquiring the motive for further investigation(Section 3.5)

The following subsections are used to explain the above points.

3.1. SARF Methodology

(Though a brief introduction to the SARF method was previously provided in the Literature Review, an in-depth explanation of the SARF method and the scaling process is required to explain certain points that are relevant to this thesis but were not taken up in the original paper.)

An MDO problem can be seen as an interaction of multiple disciplinary interfaces. An example of a two discipline MDO problem is shown in Figure 3.1a. Figure 3.1b shows the disciplinary interface for Discipline 2.

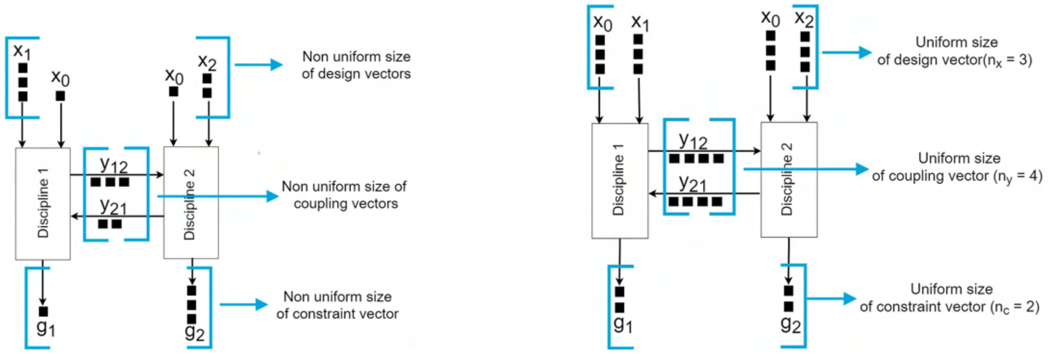


(a) Coupled two discipline Problem

(b) Disciplinary Interface

Figure 3.1: Disciplinary interface

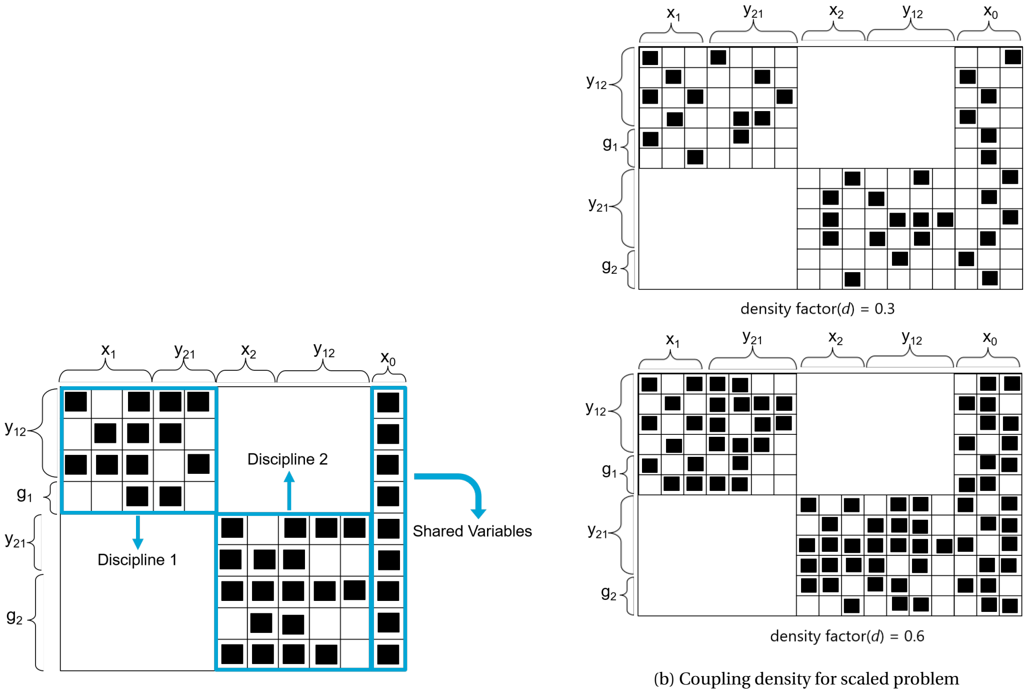
The meaning of g, x, and y used in Figure 3.1b is standard and can be inferred from Figure 3.1a. The SARF methodology is used to create a scalable version of an MDO problem, which allows the user to parametrically change the dimensions of inputs and outputs for every disciplinary interface. Figure 3.2a represents the dimension of every input/output of the coupled two discipline MDO problem shown in Figure 3.1a. The number of squares beside every input and output represents its dimension. Therefore, in Figure 3.2a, y_{12} is a vector of size three. To create a scaled version of this problem, the SARF methodology defines a set of parameters that allow the user to independently vary the dimensions of variables used in the MDO problem. In the paper by Vanaret et.al.[16], these parameters are mentioned as n_x, n_y and n_c, representing the dimension of input, coupling and constraint variables respectively. By setting the values of the parameters as $n_x = 3$, $n_y = 4$, and $n_c = 2$, the scaled disciplinary interface is shown in Figure 3.2b. Within the scaled disciplinary interface, each design vector is assigned the same size which is determined by the parameter n_x. The same applies to each coupling vector through n_y and to each constraint vector through n_c. Apart from the dimensions of variables, the SARF methodology also allows the user to vary the dependence of each component of an output vector w.r.t each component of an input vector by defining a coupling density factor. This is explained using Figure 3.3



(a) original disciplinary interface

(b) scaled disciplinary interface

Figure 3.2: Original/scaled disciplinary interface



(a) Coupling density for the original problem

(b) Coupling density for scaled problem (density factor $d = 0.3$ / 0.6)

Figure 3.3: Effect of density factor d

Figure 3.3a is a matrix representation of the original disciplinary interface shown in Figure 3.2b, showing the dependencies between individual elements of every input and output vector, placed along the horizontal and vertical axes respectively. In case of the dependency matrix, each of the "black squares" represents an existing dependency. Three blocks are shown, representing Discipline 1, Discipline 2, and shared variables respectively. For the scaled MDO problem, a coupling density factor(d) is defined to alter the existing dependencies between elements of the input and output vectors. Considering the "box" formed by one element of the input space and one element of the output space to be a placeholder, the coupling density factor randomly allocates "black squares" to a certain percentage of placeholders. As shown in Figure 3.3b, for a coupling density factor of 0.3, about thirty percent of the boxes are randomly filled. In this case, the created matrix is quite sparse and the level of dependence between the elements of input and output vectors is low. When the coupling density factor is increased to 0.6, there is a greater chance of dependence between elements of input and output vectors. Therefore, the coupling density factor gives the user the ability to assign a particular percentage of dependencies between a scaled discipline's inputs and outputs as active. This assignment is random for each version of the scaled problem. Therefore, two versions of a scaled problem created using the same combination of n_x, n_y, n_c, and d have different solutions.

There are three steps in implementing the SARF methodology as shown below:

1. One-dimensional restriction
2. Scaling and Interpolation
3. Extrapolation

These steps are explained in the following subsections.

3.1.1. One-dimensional Restriction

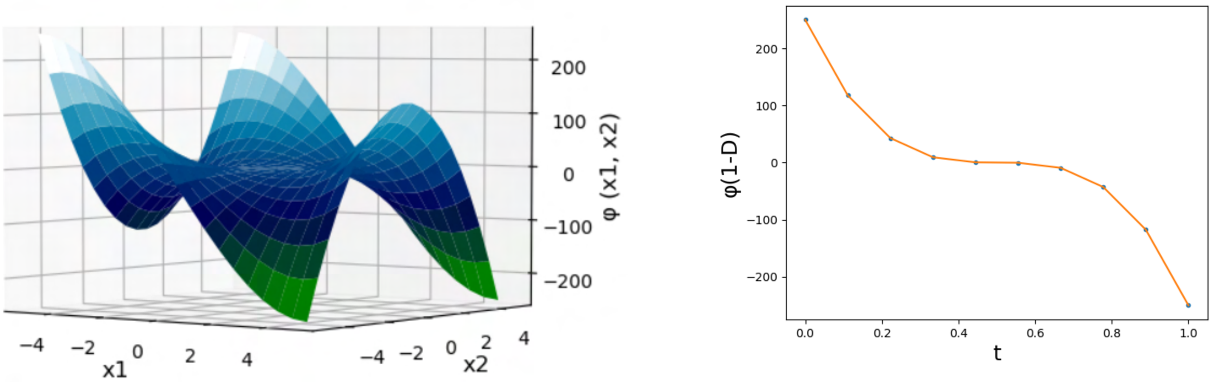
As the name suggests, the first step in the SARF methodology[16] requires restriction of each output of each disciplinary interface(couplings/ constraint/ objective) along a particular diagonal in design space. Considering a coupling to be a function of the form $\phi : R^n \to R^m$, where the lower and upper bounds of every dimension in the n dimensional domain exist as: $[\underline{x_1}, \overline{x_1}], [\underline{x_2}, \overline{x_2}], [\underline{x_3}, \overline{x_3}].....[\underline{x_n}, \overline{x_n}]$.
An interpolated function can be defined by using the lower and upper bounds of every dimension and defining a parameter t such that :

$$\phi^{(1d)}(t) = \phi(\underline{x_1} + t(\overline{x_1} - \underline{x_1}),\ \underline{x_2} + t(\overline{x_2} - \underline{x_2}), ..., \underline{x_n} + t(\overline{x_n} - \underline{x_n})) \tag{3.1}$$

Consider a two dimensional surface defined by the equation $\phi(x_1, x_2) = x_1^3 - 3x_1x_2^2$. For $(x_1, x_2) \in [-4, 4]^2$, the generated surface is represented in Figure 3.4a. Now, the one-dimensional restricted function can be evaluated as follows:

$$\begin{aligned}\phi^{(1d)}(t) &= \phi(-4 + t(4 - (-4)), -4 + t(4 - (-4))) \\ &= \phi(-4 + 8t, -4 + 8t) \\ &= (-4 + 8t)^3 - 3(-4 + 8t)(-4 + 8t)^2 \\ &= -2(-4 + 8t)^3\end{aligned} \tag{3.2}$$

For $t \in [0, 1]$, $\phi^{(1d)}(t)$ is shown in Figure 3.4b.



(a) $\phi(x_1, x_2) = x_1^3 - 3x_1x_2^2$

(b) 1-D restriction for ϕ $\phi^{(1d)}(t) = -2(-4 + 8t)^3$

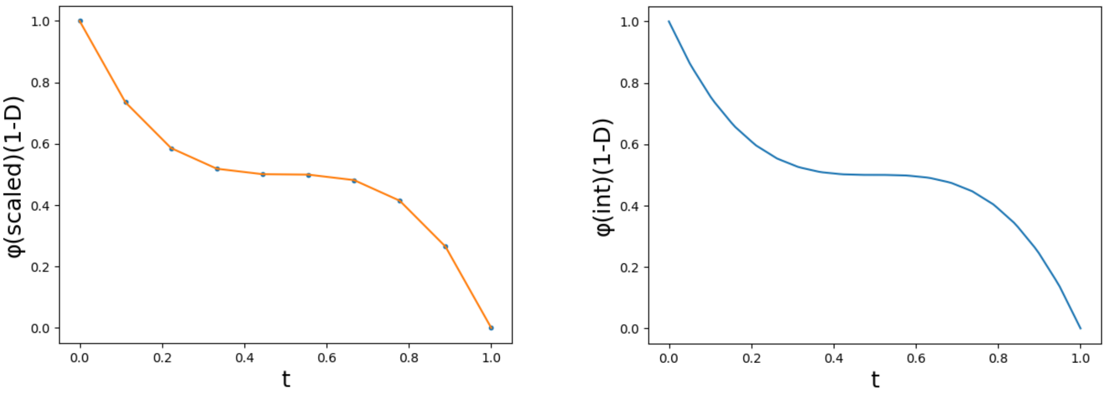
Figure 3.4: one-dimensional restriction

3.1.2. Scaling and Interpolation

Following one-dimensional restriction, the second step is to scale the one-dimensional function ($\phi^{(1d)}(t)$) using maximum and minimum values over the uni dimensional domain, noted as $min(\phi^{(1d)}(t))$ and $max(\phi^{(1d)}(t))$. The scaled function can be defined in the following manner:

$$\phi^{(scaled)}(t) = \frac{\phi^{(1d)}(t) - min(\phi^{(1d)}(t))}{max(\phi^{(1d)}(t)) - min(\phi^{(1d)}(t))} \tag{3.3}$$

The original saddle surface as well as the one-dimensional function shown in Figure 3.4b, are created out of a coarse set of points because they require the evaluation of the original discipline for each point. The scaled function $\phi^{(scaled)}(t)$ is therefore further required to be approximated using a polynomial spline to form a continuous, interpolated output $\phi^{(int)}(t)$ that can be calculated inexpensively. Figure 3.5a and Figure 3.5b shows a scaled and interpolated form(using a third order spline) of the one-dimensional example function shown in Figure 3.4b

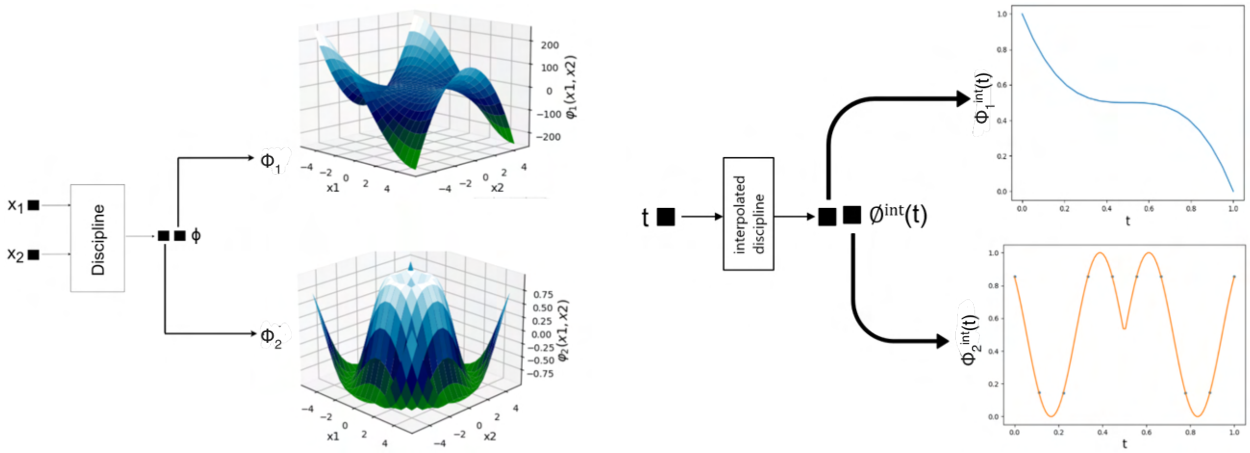


(a) $\phi^{(scaled)}(t)$

(b) $\phi^{(int)}(t)$

Figure 3.5: Scaled and Interpolated Function

The above two steps explain the derivation of one-dimensional, scaled interpolated function $\phi^{(int)} : [0,1] \rightarrow [0,1]$ from a multi dimensional function $\phi : R^n \rightarrow R^m$. For the above example, the value of n and m are two and one respectively. A second example is shown for $n, m = 2, 2$ in the figure below:



(a) $\phi : R^2 \rightarrow R^2$

(b) $\phi^{(int)} : [0,1]^1 \rightarrow [0,1]^2$

Figure 3.6: Original($\phi : R^2 \rightarrow R^2$) and Interpolated Function($\phi^{(int)} : [0,1]^1 \rightarrow [0,1]^2$)

3.1.3. Extrapolation

After the calculation of ϕ^{int}, the final step is to use the interpolated function to generate the extrapolated functions $\phi^{(ext)}$. Using the example of the discipline shown in Figure 3.6, the extrapolation process is shown in Figure 3.7

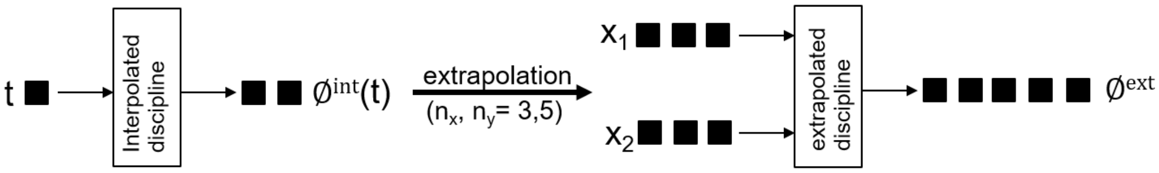
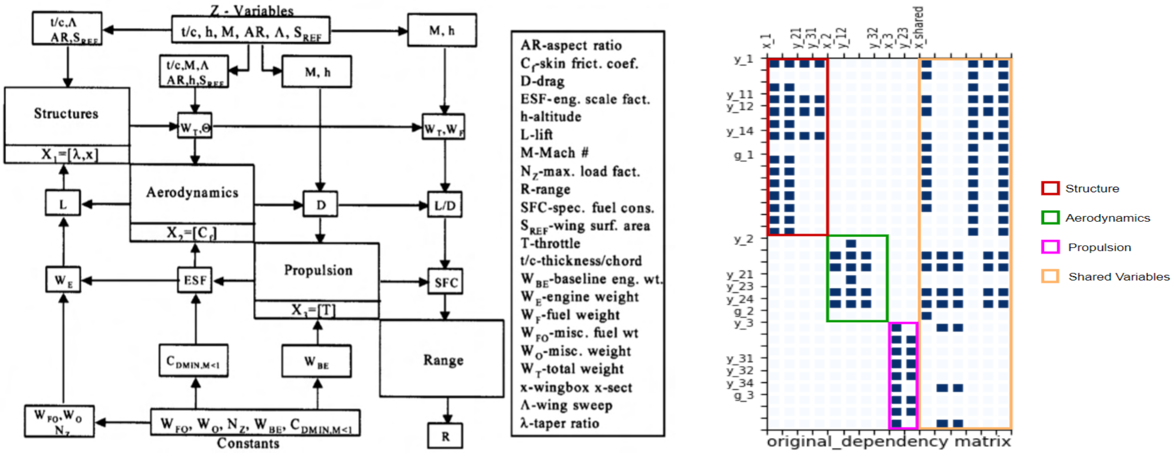


Figure 3.7: Extrapolation step for discipline ϕ of Figure 3.6

Figure 3.7 shows the extrapolated discipline created using parameter values $(n_x, n_y = 3, 5)$. Both x_1 and x_2 receive a dimension of three, while ϕ^{ext} has a dimension of five. The extrapolated function, in keeping with the SARF methodology, allows the user to independently vary the sizes of the input and output vectors for every disciplinary interface. This is achieved using two data structures namely, the input-output dependency matrix and the component dependency table which are explained in the following two sections. The method used to estimate the extrapolated outputs is explained thereafter.

Input-Output Dependency Matrix

The input-output dependency matrix is used to show the dependency(or lack thereof) of an output vector component with respect to components of an input vector across a disciplinary interface. Dependency matrix for a disciplinary interface was previously introduced in Section 3.1. An MDO problem typically contains two or more disciplinary interfaces. By diagonally stitching the dependency matrices for every interface, an input-output dependency matrix can be drawn for the MDO problem. An example of the dependency matrix is shown for the Super Sonic Business Jet(SSBJ) MDO problem in Figure 3.8b[16]. This is a well known range maximization problem modeled as a coupled system of four disciplines - Structures, Aerodynamics, Propulsion, and Aircraft Range. The formulation of the MDO problem is given using the data flow diagram, shown in Figure 3.8a[18]. As can be seen in the data flow diagram, there are ten design variables of which six are shared(global) variables. Similarly, the sizes of local and coupling vectors can also be observed in Figure 3.8a[18]. The dependencies between design variables and couplings can be looked up by analyzing the expressions used within the individual disciplines. The problem also includes local constraints, the sizes, and dependencies of which can be looked up from the original problem description. Using the above information, the dependency matrix for the original model can be constructed as shown in Figure 3.8b[16]. The horizontal axis represents the entire design space in terms of local/shared variables as well as coupling vectors. Each row on the vertical axis represents a disciplinary output in terms of a coupling or constraint. Each square block represents the existence of dependency between a particular component of a disciplinary output and an input vector. The naming scheme for the inputs/outputs is simply derived by assigning numbers 1, 2, 3, and 4 to the Structures, Aerodynamics, Propulsion, and Range disciplines respectively. The coupling from Structures to Aerodynamics is hence named y_{12}. As can be seen in Figure 3.8a[18], $y_{12} = [W_T, \theta]$ is a two dimensional coupling vector. Correspondingly, two consecutive rows are dedicated to y_{12} in the dependency matrix. The dependency matrix can also be looked at as a conjunction of four blocks, three of which represent the Structure, Aerodynamics, and Propulsion disciplines. The fourth block represents the dependency of the outputs of the three disciplines with respect to the shared variables. There is no dedicated block for the Range discipline as the output is the objective itself, which cannot be represented along the input space on the x-axis.



(a) Data flow diagram for SSBJ Problem

(b) Dependency Matrix(original model)

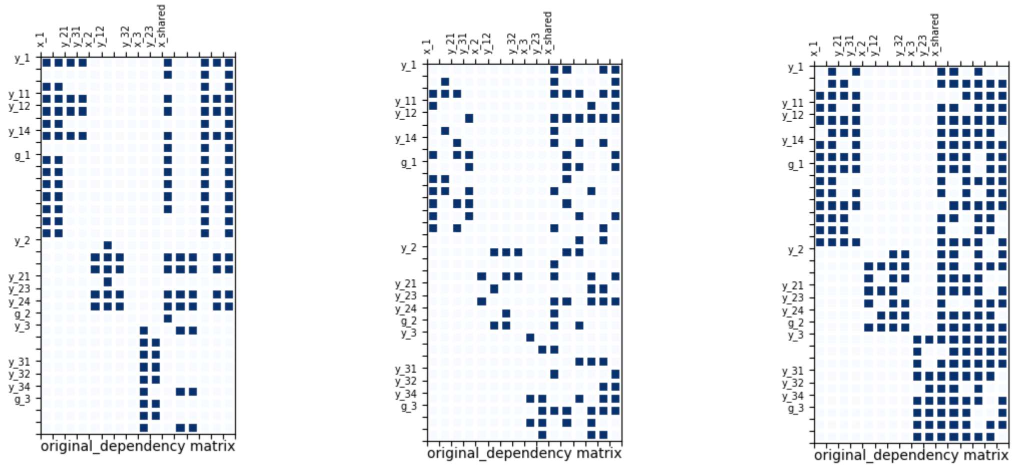
Figure 3.8: Super Sonic Business Jet(SSBJ) MDO Problem

As can be seen in Figure 3.8b[16], the original SSBJ problem has a variable size for every input and output vector. For instance, constraint g_1 has a dimension of seven while the other output variables in the Structural block have a dimension between one and three. Also, the distribution of dependencies between every input and output component is different for every block. The dependency within the Propulsion block is high as most of the blocks are filled, while the dependency for the Structural and Aerodynamic blocks are lower. As mentioned in section 3.1, to create a family of scaled problems, a set of parameters can be defined as follows:

- density factor d to assign the level of dependency between design variables and couplings(to be applied uniformly across each block of the problem).
- Parameters n_x and n_y that assign the number of components for each design and coupling variable respectively(In the original paper, the value of n_c(size of constraint vector) is considered same as n_x and therefore the same simplification is used here).

By making use of the above-mentioned parameters, the dependency matrix for the original problem can be reconstructed to create a family of scaled MDO problems.

Figure 3.9b shows a dependency matrix with the same problem dimensions as the original problem but an user defined coupling density factor(d = 0.4) that has been uniformly applied on the individual blocks. Figure 3.9c shows the same dependency matrix with higher coupling density factor(d = 0.8). The dependency matrix for the original problem is shown in Figure 3.9a for comparison.



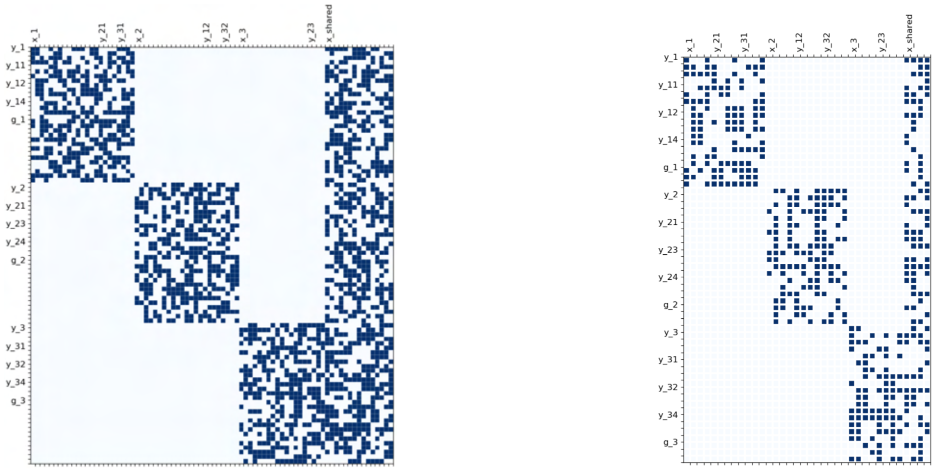
(a) original model

(b) original model(d= 0.4)

(c) original model(d= 0.8)

Figure 3.9: Effect of density factor on the original SSBJ problem

While using parameters n_x and n_y to alter the dimensions of the problem, it must be ensured that a consistent family of scaled problems is created. For this purpose firstly a large random dependency matrix is generated using a specific density factor, which is then scaled down to the required dimensions. This ensures that the distribution of dependency between input and output vectors is preserved[16]. Figure 3.10a represents the large dependency matrix for the parameters $(n_x, n_y, d) = (15, 2, 0.5)$. This matrix can be scaled down to the dimensions $(n_x, n_y, d) = (4, 4, 0.5)$ as shown in Figure 3.10b. For this example(and also in the paper by Vanaret[16]), the size of the local constraint vectors are considered to be the same as that of the design variables. It must be noted that n_x and n_y are parameters that assign a particular size to each design and coupling variable respectively, across all disciplines as seen in Figure 3.10b

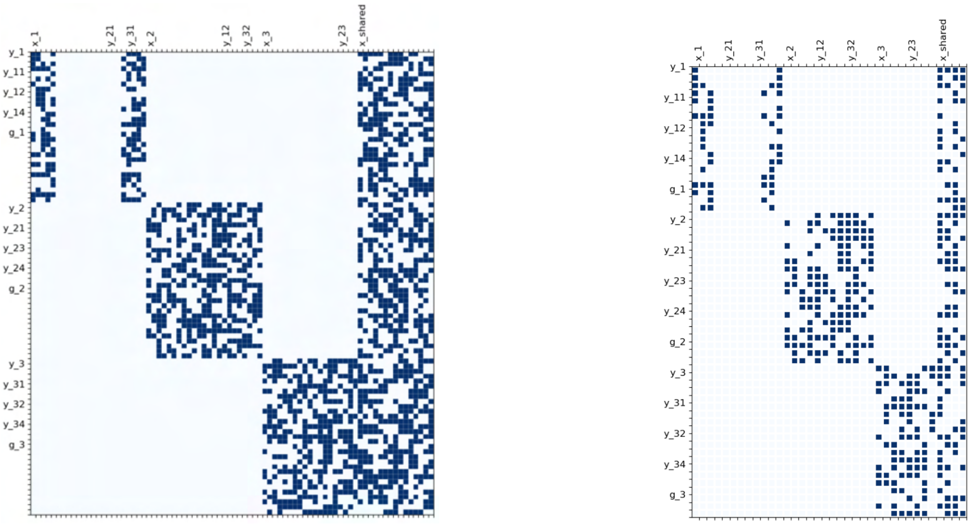


(a) Large matrix(n_x, n_y, d) = (15,2,0.5)

(b) Scaled matrix(n_x, n_y, d) = (4,4,0.5)

Figure 3.10: Large dependency matrix scaled to desired dimension

The exact procedure used to scale down the large dependency matrix is not discussed in the original paper. A custom process is defined to scale down the large dependency matrix as shown in Appendix A.9. The working of the process can be verified from Figure 3.11. Figure 3.11a represents a large dependency matrix, wherein a bias has been applied on the structural block to keep the elements in the middle of the design space devoid of dependencies. When scaled down to the required dimensions, the same bias is conserved as shown in Figure 3.11b.



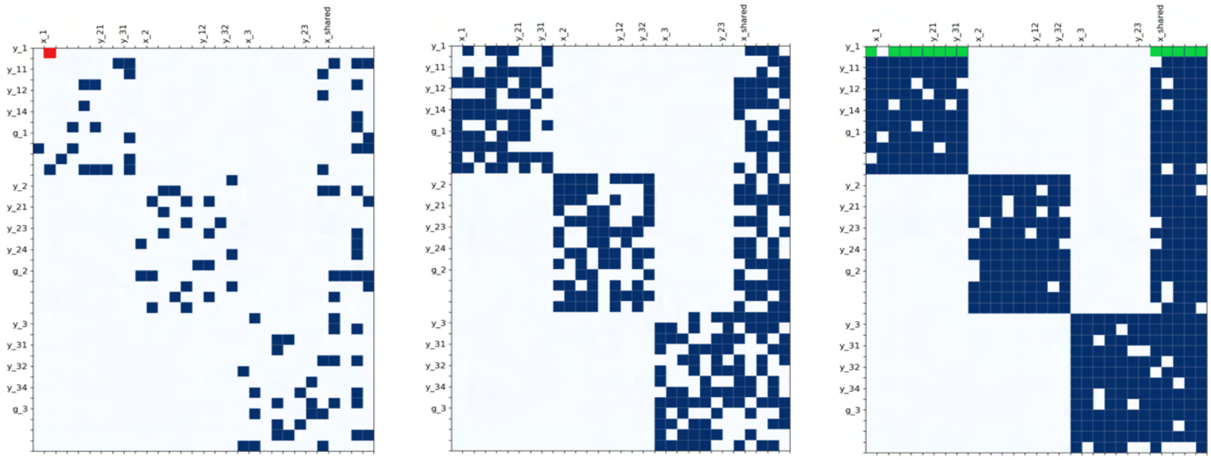
(a) Large matrix(biased)(n_x, n_y, d) = (15,2,0.5)

(b) Scaled matrix(biased)(n_x, n_y, d) = (4,4,0.5)

Figure 3.11: Large dependency matrix(biased) scaled to desired dimension

The combined effect of the coupling density and the input/output dimension parameters is shown in

Figure 3.12. By scaling inputs and outputs independently($n_x = 5$, $n_y = 2$), Figure 3.12a to 3.12c show the effect of increasing the coupling density d on the scaled dependency matrix.



(a) (n_x,n_y,d) = (5,2,0.2)

(b) (n_x,n_y,d) = (5,2,0.6)

(c) (n_x,n_y,d) = (5,2,0.9)

Figure 3.12: Effect of density factor on scaled SSBJ problem

The takeaway from the format of the dependency matrix is that the scaled outputs are dependent on randomly defined combinations of the original inputs. For example, in the sparse matrix of Figure 3.12a, $y_1(0)$ is dependent only on $x_1(1)$(shown in red). For the dense matrix of Figure 3.12c, $y_1(0)$ is dependent on all possible inputs(shown in green) except $x_1(1)$. The allocation of dependencies is repeated for all the scaled output vectors. The random distribution of dependencies can be seen as an on/off switch as far as the dependency between any particular input and output vector element is concerned. However, over a large number of scaled outputs, this results in the preservation of the convexity/linearity of the disciplinary outputs w.r.t their inputs while going from the original to the scaled problem. This can also be noted in the next section(Section 3.2, Figure 3.21), where it is seen that the scaled disciplinary outputs retain their convexity. A direct consequence of convexity retention is that interdisciplinary sensitivities do not get affected when any of the scaling parameters are changed. Therefore, if a parameter is defined that depends only on the interdisciplinary sensitivities, the value of this parameter must not change for any version of the scaled MDO problem. This is an important aspect of the SARF methodology which is utilized later in the thesis(Chapter 4 Section 4.2.4).

Component Dependency Graph

Until now, two data structures have been developed, a library containing interpolated functions for disciplinary outputs and an input-output dependency matrix representing the dependency of output components on input components. Before extrapolated outputs can be calculated, a component dependency graph needs to be defined as shown below:

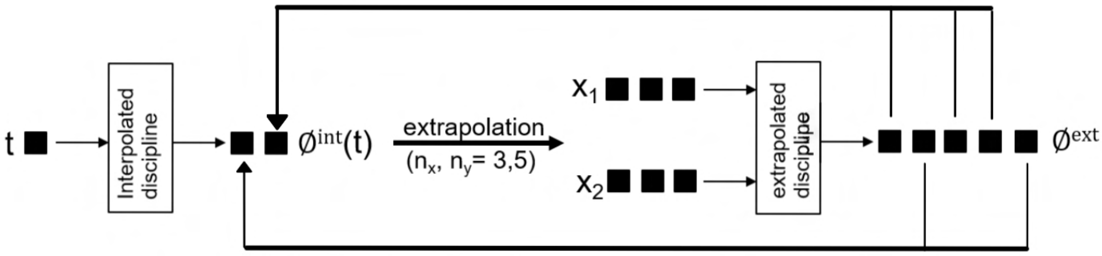


Figure 3.13: Random mapping of components between extrapolated and interpolated outputs

Figure 3.13 shows the example of the extrapolation process previously used in Figure 3.7. The interpolated discipline has a uni-dimensional input and a two dimensional output ϕ^{int}. Using parameters $n_x, n_y = 3,5$, the discipline is extrapolated. The extrapolated discipline has two input vectors, each of size three and a five dimensional output ϕ^{ext}. For constructing the component dependency graph, each extrapolated output

component needs to be randomly mapped to a particular interpolated output component. In the example of Figure 3.13, the 1^{st}, 3^{rd} and 4^{th} extrapolated components are mapped to the 2^{nd} interpolated component while the 2^{nd} and 5^{th} extrapolated components are mapped to the 1^{st} interpolated component. The mapping information is stored in the form of a dictionary as shown below:
$\phi^{ext}: \{2,1,2,2,1\}$
The component dependency graph is constructed by assembling the mapping information for each output component within the MDO problem. Along with the input-output dependency matrix and the repository of interpolated functions, the component dependency graph is used to calculate the extrapolated outputs as shown in the next section.

Extrapolated Output

Considering a coupling vector y with original dimension m and scaled dimension n_y, each of the n_y components of the extrapolated coupling vector can be evaluated in the following manner:

$$\phi_i^{(ext)}(x) = \frac{1}{|S_i|} \sum_{j \in S_i} \phi_{k_i}^{(int)}(x_j) \tag{3.4}$$

where:
x represents the design space shown along the horizontal axis of dependency matrix,
$i \in \{1,2,....n_y\}$ is one of the n_y components of the extrapolated output,
$k_i \in \{1,2,...m\}$ represents the index of the original interpolated component(ϕ^{int}), mapped from every extrapolated component i, extracted from the component dependency graph.
S_i represents the corresponding row of the dependency matrix,
$|S|$ represents the "sum of dots" along the corresponding row i.e. the total number of dependencies for the i^{th} component of the extrapolated coupling.

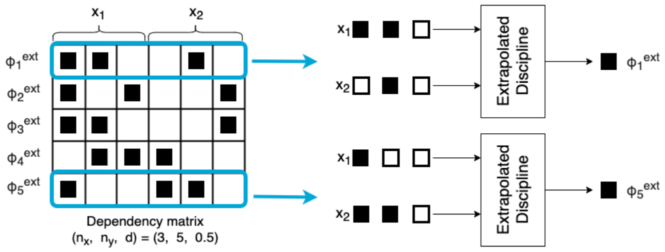


Figure 3.14: Dependency matrix and extrapolated disciplinary interface

The above calculation is demonstrated using the earlier example of Figure 3.13 and Figure 3.6. The original discipline contains two inputs and two outputs. For the extrapolated discipline shown in Figure 3.13, n_x and n_y are 3 and 5 respectively. This translates to two input vectors each of size three and extrapolated vector ϕ^{ext} of size five. Now, considering the density factor to be 0.5, Figure 3.14 shows the input-output dependency matrix and extrapolated disciplinary interface for two(out of the five) extrapolated output components, ϕ_1^{ext} and ϕ_5^{ext}. Based on the randomly generated dependency matrix, the output vector components are dependent on certain input components shown with filled squares. For calculating the values of the output components, the component dependency mapping and the library of interpolated functions has to be called. Based on the mapping shown in Figure 3.13, ϕ_1^{ext} is mapped to ϕ_2^{int} and ϕ_5^{ext} is mapped to ϕ_1^{int}. Using equation 3.4, ϕ_1^{ext} and ϕ_5^{ext} can be calculated as follows:

$$\phi_1^{ext} = \tfrac{1}{3}[\phi_2^{int}(x_1[0]) + \phi_2^{int}(x_1[1]) + \phi_2^{int}(x_2[1]]$$

$$\phi_5^{ext} = \tfrac{1}{3}[\phi_1^{int}(x_1[0]) + \phi_1^{int}(x_2[0]) + \phi_1^{int}(x_2[1]]$$

The values of the interpolated functions $\phi_1^{int}(t)$ and $\phi_2^{int}(t)$ are known from the library of uni-dimensional interpolated functions shown in Figure 3.6b.

This extrapolation procedure, when applied to constraint functions, always extrapolates the constraint outputs between zero and one. However this leads to permanently inconsistent constraint functions. In the SARF methodology, this problem is tackled by translating each extrapolated constraint $g_i \leq 0$ using a threshold value $\tau_i \in [0,1]$ as shown below:

$$g_i - \tau_i < 0 \tag{3.5}$$

To calculate this threshold value, it is required to define three constants $\mu(0,1)$, α_i and p at run-time, each having a random value between zero and one. These constants are explained below:

1. p determines what percentage of constraints are activated at the initial point.
2. α_i determines the extent to which inactive constraints are satisfied.
3. $\mu(0,1)$ is a random value value between 0 and 1, chosen with uniform probability.

Considering $g_i^{(0)}$ to be the value of the constraint output at initial point, the threshold τ_i can be defined as follows:

$$\tau_i = \begin{cases} g_i^{(0)} & if\ \mu(0,1) \leq p \\ g_i^{(0)} + (1 - g_i^{(0)})\alpha_i^{(0)} & otherwise \end{cases} \tag{3.6}$$

The above definition allows a certain percentage p of constraints to be initially active while the remaining constraints are initially satisfied but inactive.
The next section is used to show the implementation of the SARF method on the SSBJ problem.

3.2. Implementation of SARF methology on SSBJ Problem

(Refer to Appendix A and Appendix A.8 for the code and UML regarding the implementation of the SARF methodology)

Having explained the SARF method, it is now required to implement the SARF methodology on the test SSBJ problem which was previously introduced in section 3.1.3.1. The SSBJ(Super Sonic Business Jet) problem is a well known MDO benchmark problem proposed by NASA in 1998[18]. It is a range maximization problem consisting of four disciplines, Structures, Aerodynamics, Propulsion, and Performance, each defined by analytical expressions. Additionally, there are local constraints attached to Structures, Aerodynamics, and Propulsion blocks. The numbering scheme used to denote the disciplines and couplings has been mentioned in section 3.1.3.1. The XDSM for the original SSBJ problem is shown in Figure 3.15 and Figure 3.16. The abbreviations used in the XDSM representation are same as in Figure 3.8a.

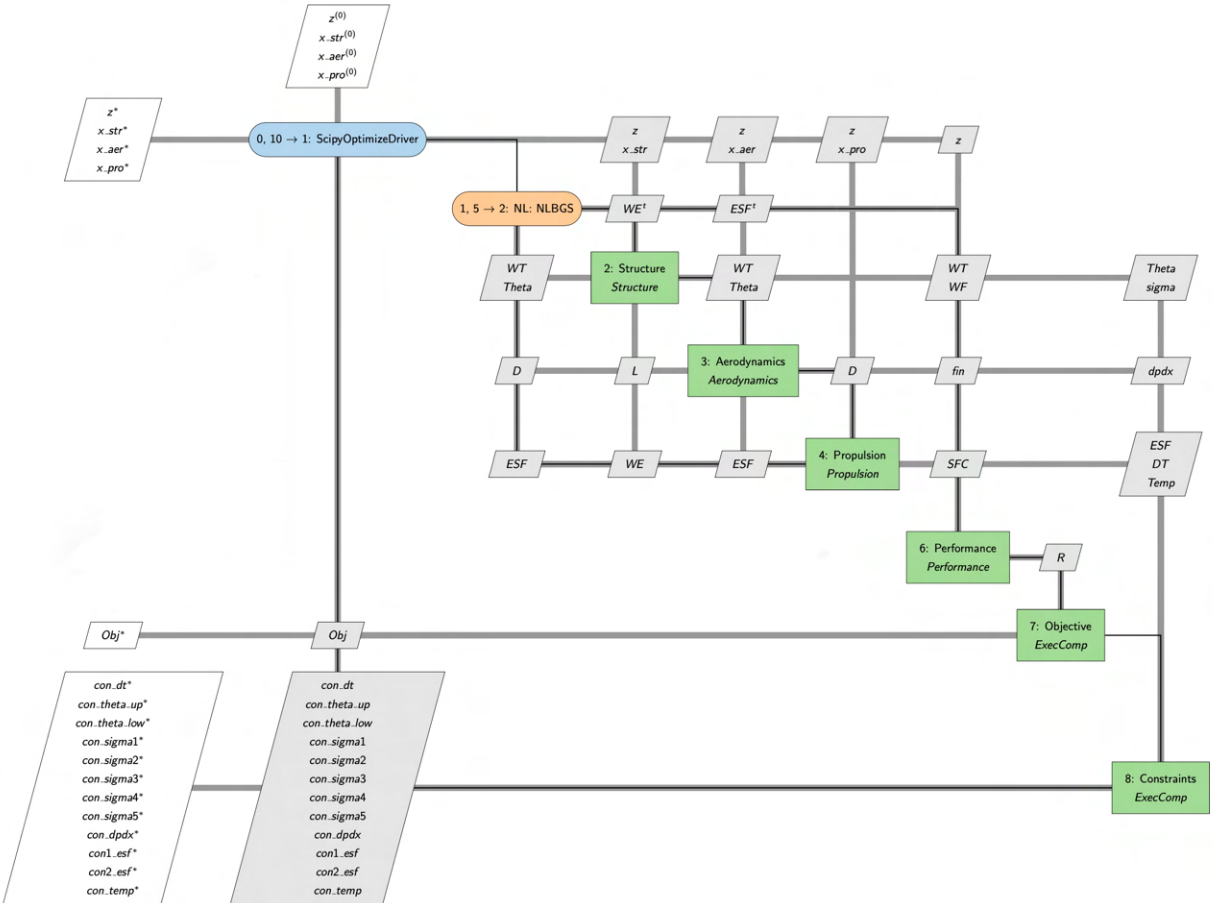


Figure 3.15: XDSM for Original SSBJ problem(MDF-GS)

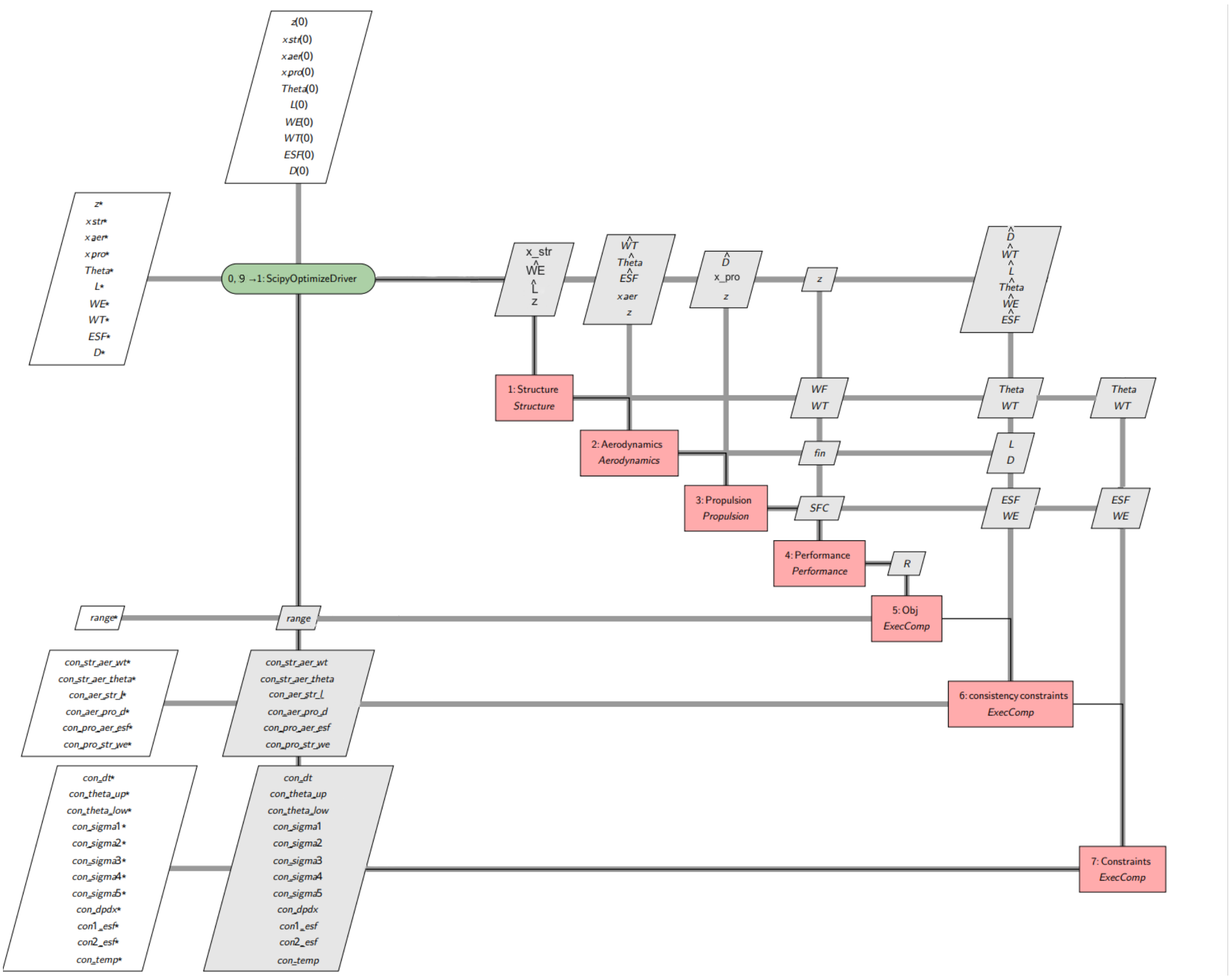
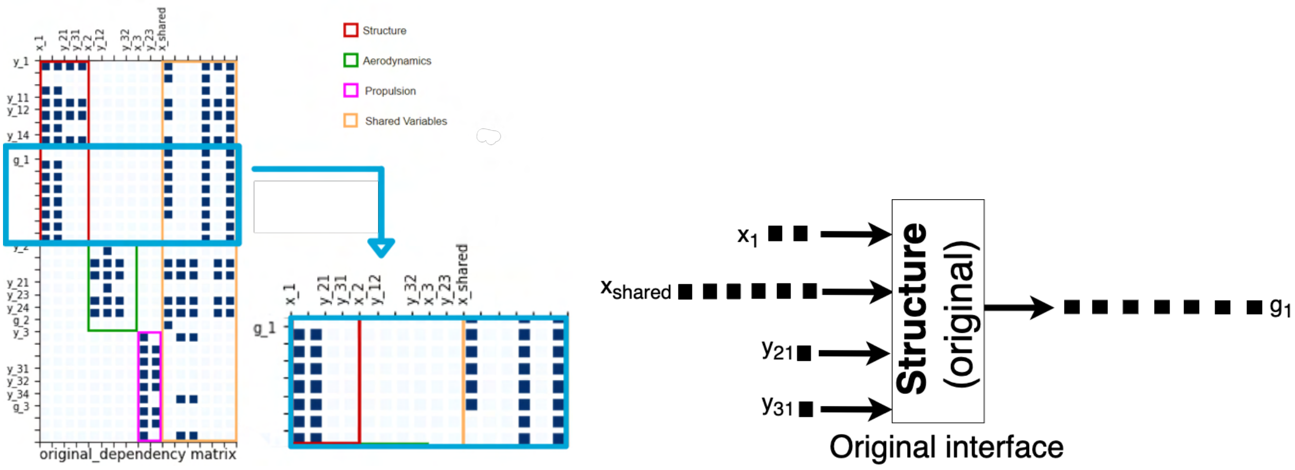


Figure 3.16: XDSM for Original SSBJ problem(IDF)

Within the MDF-GS implementation, the NLBGS solver is used to run MDA routines over three coupled disciplines Structures, Aerodynamics, and Propulsion. The Performance discipline has no feedback, therefore it is not a part of the MDA analysis. There are a total of five coupling vectors each of unitary dimension except $y_{12} = \{Theta, W_T\}$, which has a dimension of two. For IDF, this results in six consistency constraints.

The sub-steps of the SARF methodology can be applied sequentially to create a scaled version of the SSBJ Problem. The dependency matrix for the original SSBJ problem is shown in Figure 3.17a. The inputs and outputs for constraint g_1 is shown in Figure 3.17b.



(a) dependency matrix for g_1 constraint

(b) Disciplinary interface for constraint $g_1 : R^{10} \rightarrow R^7$

Figure 3.17: dependency matrix and disciplinary interface for constraint g_1

Combining the first two steps, it is required to uni-dimensionalize, scale, and interpolate every disciplinary interface(couplings, constraints, and objective). Considering the example of g_1, which is a seven-dimensional constraint vector attached to structure discipline, the transition is shown in Figure 3.18.

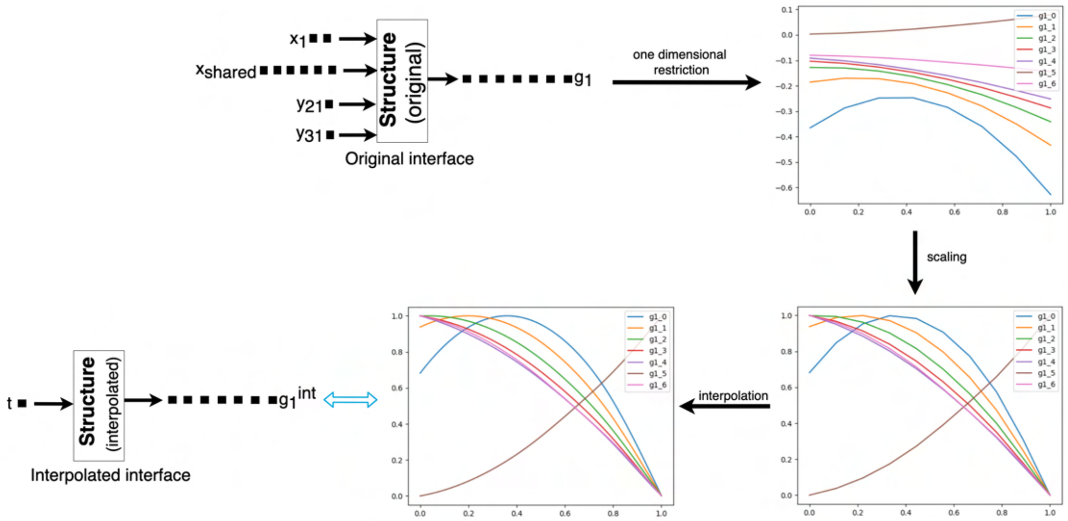


Figure 3.18: Interpolation of disciplinary interface

The above transition is followed by the generation of the dependency matrix as explained in section 3.1.3.1. Each scaled version of the SSBJ problem is constructed using a particular value of (n_x, n_y, n_c, d). For a value of $(n_x, n_y, n_c) = (2, 2, 2)$, the following flow diagram shows the effect of altering the coupling density on the N^2 diagram for the SSBJ problem(using MDF).

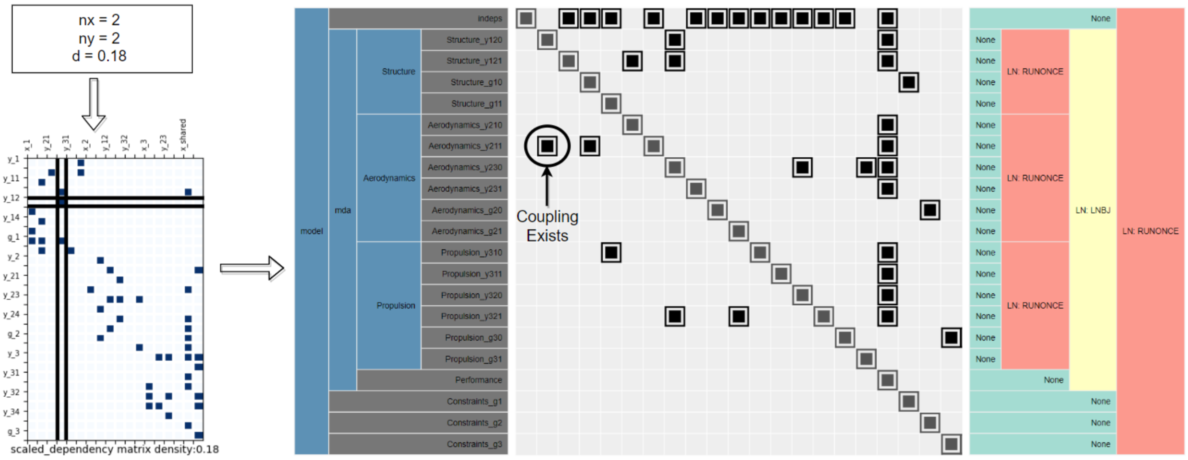


Figure 3.19: Coupling between $y21_1$ and $y12_0$

Figure 3.19 and Figure 3.20 show the effect of low($d = 0.18$) and high($d = 0.95$) coupling densities. In Figure 3.19($d = 0.18$), a sparse density matrix is generated. Within that, the connective link that exists between $y21_1$ and $y12_0$ is shown using the filled square that exists at the junction of the intersecting black lines. The corresponding coupling in the N^2 diagram is encircled. In Figure 3.20, a dense matrix(d = 0.95) is shown with a few missing connections. The missing connection between $y21_1$ and $y12_1$ is shown using the empty square at the junction of the intersecting black lines, and the corresponding lack of coupling is displayed in the N^2 diagram.

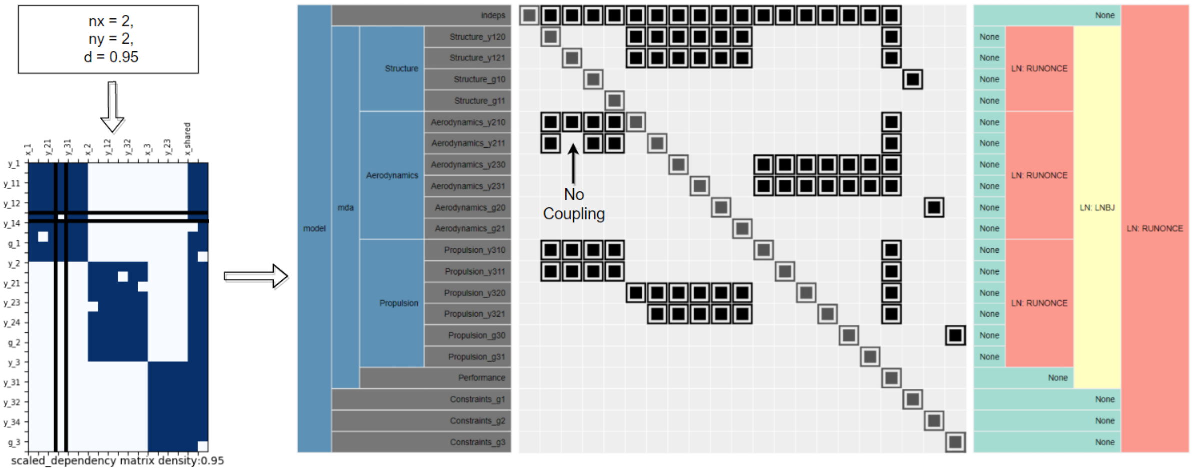


Figure 3.20: No coupling between $y21_1$ and $y12_1$

As explained in Section 3.1.3.2, along with the dependency matrix, a component dependency graph is also required to be generated so as to evaluate each component of the extrapolated output according to a random component of the original output. By considering a particular value of $(n_x, n_y, n_c, d) = (5, 8, 5, 0.5)$, the interpolated constraint $g_1^{(int)}$ from Figure 3.18 can be extrapolated as shown in the following flow diagram(Figure 3.21):

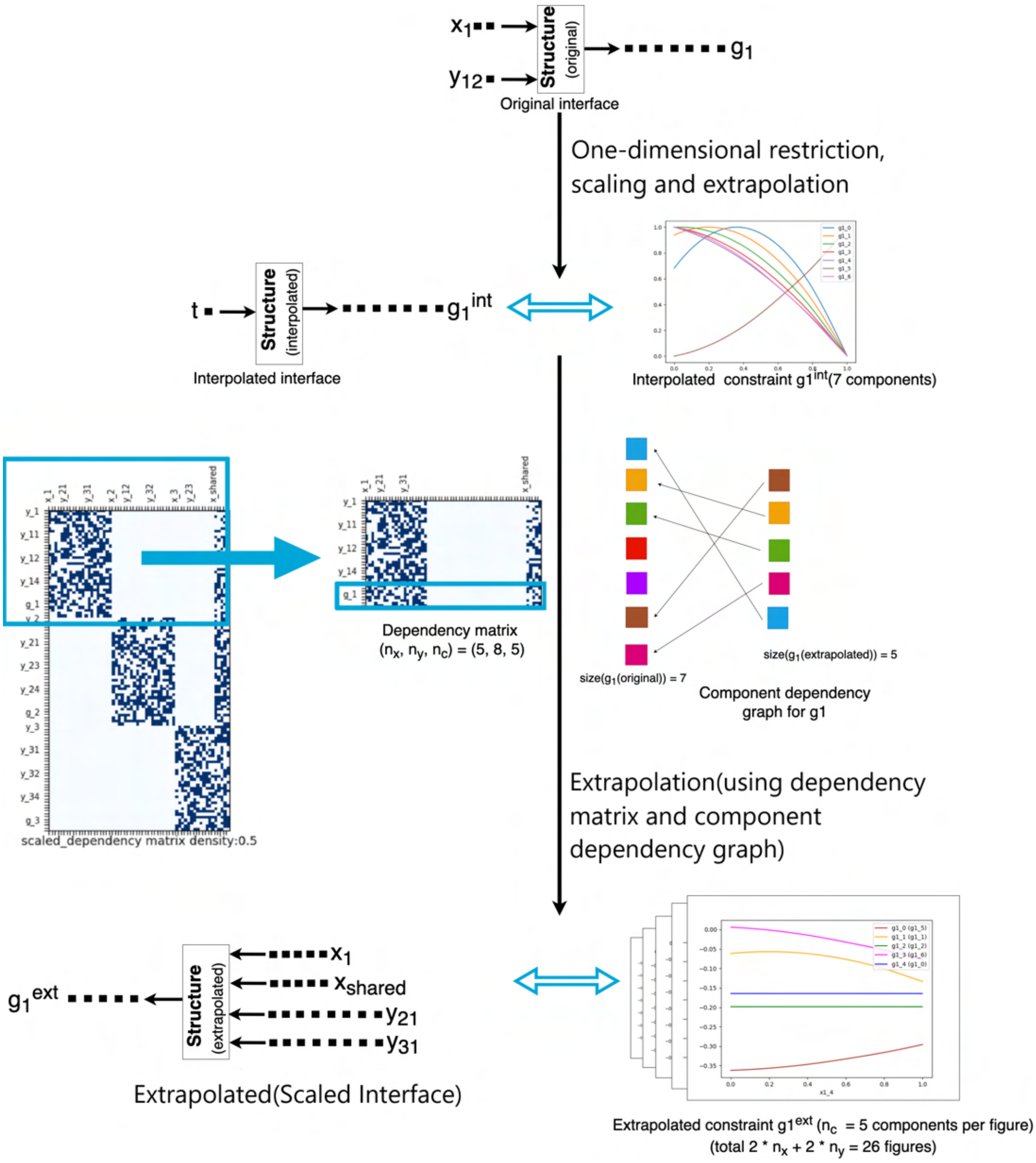


Figure 3.21: Flow diagram for extrapolation of constraint $g_1, (n_x, n_y, n_c, d) = (5, 8, 5, 0.5)$

The extrapolated constraint g_1^{ext} has n_y components. The input components for the extrapolated constraint can be observed in the horizontal label of the dependency matrix. There are four input vectors, x_1, y_{21}, y_{31} and x_{shared} , comprising of n_x, n_y, n_y and n_x components respectively. The output vector g_1^{ext} comprises of n_c components. The extrapolated constraints are evaluated using three entities, the library of interpolated constraints, the dependency matrix, and the component dependency graph. The overlapping extrapolated constraint plots shown in the flow diagram(Figure 3.21) represent the individual contribution of some of the input components(keeping the other components constant). Comparing the plots of interpolated and extrapolated constraint(g_1^{int} and g_1^{ext}), the convexity of interpolated and extrapolated components can be seen to be preserved. The straight lines in the extrapolated component graph represent the missing dependency between a particular input and output component.

3.3. Pre-processing / Optimization settings

Prior to running a critical analysis of the comparative study performed by Vanaret et.al.[16], it is required to establish the following:

1. Define and justify the cost criterion used to compare MDF and IDF architectures.
2. Define the optimization and tolerance parameters to be used in the forthcoming analysis.

The above points are taken up in the following two subsections:

3.3.1. Cost criterion for Comparing MDF-GS and IDF Architectures

A general comparison of MDO architectures can be made by noting the total time or discipline calls. However, within the SARF methodology, expensive discipline calls are replaced by interpolated libraries using cubic splines, which enables the testing of architectures across a large array of multiple parameters. Therefore, instead of total time/discipline calls to compare MDF-GS and IDF architectures, the SARF methodology proposes a cost function that factors in the major sub-steps within the two architectures that require computational effort. Owing to the difference in approaches of the MDF-GS and IDF methods, these sub-steps also vary accordingly, but they can be categorized into two major components, linearization, and disciplinary evaluation. An optimization approach using a gradient-based algorithm like SQP requires linearization of objective and constraint functions that are performed at every optimizer iteration[16, 26]. This directly translates to the calculation of partial derivatives of each disciplinary coupling with respect to all components of the design space(Vanaret et. al.[16], Eq. 3). In IDF, this step requires more calculations owing to the additional consistency constraints and design variables. However, to fulfill the linearization of objectives and constraints in MDF-GS, it is also required to solve a linear system at the end of every MDA iteration[16]. For iteratively solving a linear system given by equation $Ax = b$, matrix A is required to be decomposed into lower and upper triangular matrices[40]. The cost build-up for MDF-GS therefore includes the number of lower upper factorizations within each MDA routine as well as the number of MDA analyses invoked during the optimization process. These two factors also account for the cost of disciplinary evaluation in MDF-GS. In case of IDF, the optimizer directly evaluates the couplings across every disciplinary interface, so the cost for IDF includes the number of coupling evaluations over each discipline. The cost breakup for MDF-GS and IDF approaches can be stated as follows:

$$C_{IDF} = \sum_{i \in disciplines} n_{eval_i} + c_{lin} \sum_{i \in disciplines} n_{lin_i} \tag{3.7}$$

$$C_{MDF-GS} = c_{lin} \sum_{i \in disciplines} n_{lin_i} + c_{LU} n_{LU} + c_{MDA} n_{MDA} \tag{3.8}$$

where: n_{eval_i} represents the number of disciplinary evaluations in IDF, n_{lin_i} represents the number of linearizations(calculation of partial derivatives) over all disciplines, n_{LU} and n_{MDA} represent the number of lower upper factorizations and the number of MDA routines respectively in MDF.

The weight factors used in the above equations are listed below:

- c_{lin} is the time ratio between a linearization and each disciplinary analysis.
- c_{LU} is the time ratio between a lower upper factorization and a disciplinary analysis.
- c_{MDA} is the time ratio between an MDA analysis and a disciplinary analysis.

3.3.2. Optimization and Tolerance Parameters

As the final preparatory step before running the comparative study, it is required to adjust the various optimization and tolerance parameters to be used in OpenMDAO for the forthcoming analysis. This is shown in Figure 3.22

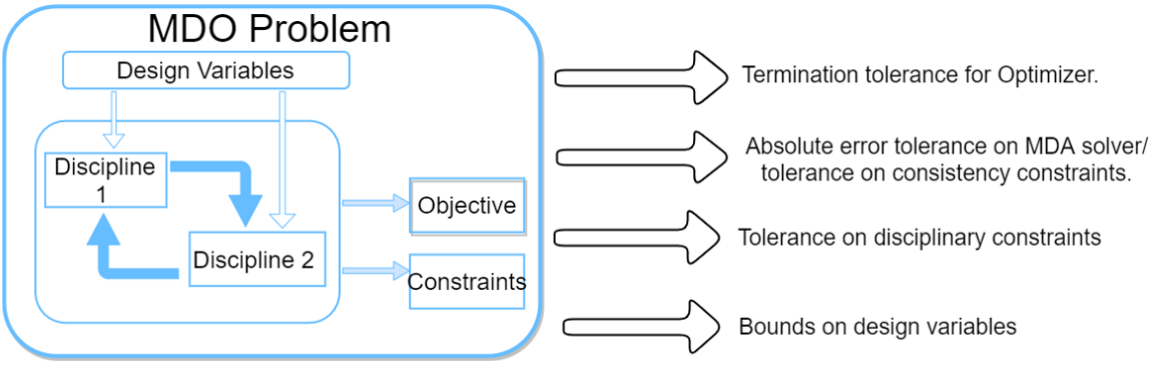

MDO Problem
Design Variables
Discipline 1
Discipline 2
Objective
Constraints
Termination tolerance for Optimizer.
Absolute error tolerance on MDA solver/ tolerance on consistency constraints.
Tolerance on disciplinary constraints
Bounds on design variables

Figure 3.22: Pre-processing settings for SSBJ based scaled problem

The original paper does not provide a value for the above mentioned tolerance parameters. However, the above parameters are critical to ensure that a fair comparison can be made between MDF-GS and IDF architectures in the comparative study. Since the comparison of architectures is to be made in terms of the cost criterion(which is representative of solution effort), it must be ensured that the final objective is the same(up to a certain number of decimals) for both MDF-GS and IDF architectures for each scaled problem. Additionally, it must be ensured that "redundant" iterations do not occur for a particular architecture. Redundant iterations are those where optimizer iterations continue to occur without any significant changes in the objective value. The reason for extra iterations is required to be investigated such that changes can be made in the tolerance settings and additional iterations can be clipped off. It must also be ensured that the overall time required to create the database of scaled problems is as low as possible without compromising on the above points.

Therefore a tuning study of optimization settings is required, before performing the comparative study of the investigation by Vanaret et.al.[16]. The tuning process is explained in detail in Appendix B.

3.4. Critical Analysis of Existing Results

In the application part of the paper by Vanaret et. al.[16], the SARF scaling methodology was applied on the SSBJ MDO problem to test the performance of MDF-GS and IDF architectures. Using a particular range of problem parameters, a repository of SARF based scaled SSBJ problems was created and a comparative analysis was carried out to check for correlations between problem parameters and performance of MDO architectures. In this section, the objective is to repeat the comparative analysis carried out by Vanaret et. al.[16] using the same set of parameters, make a side by side comparison of the findings with those provided in the original paper and arrive at the motive for further investigation. The following table (Table 3.1) shows the range of problem parameters used for the comparison.

Parameter	Range of values
n_x	5 - 30
n_y	5 - 40
d	0.4
n_c	Same as n_x

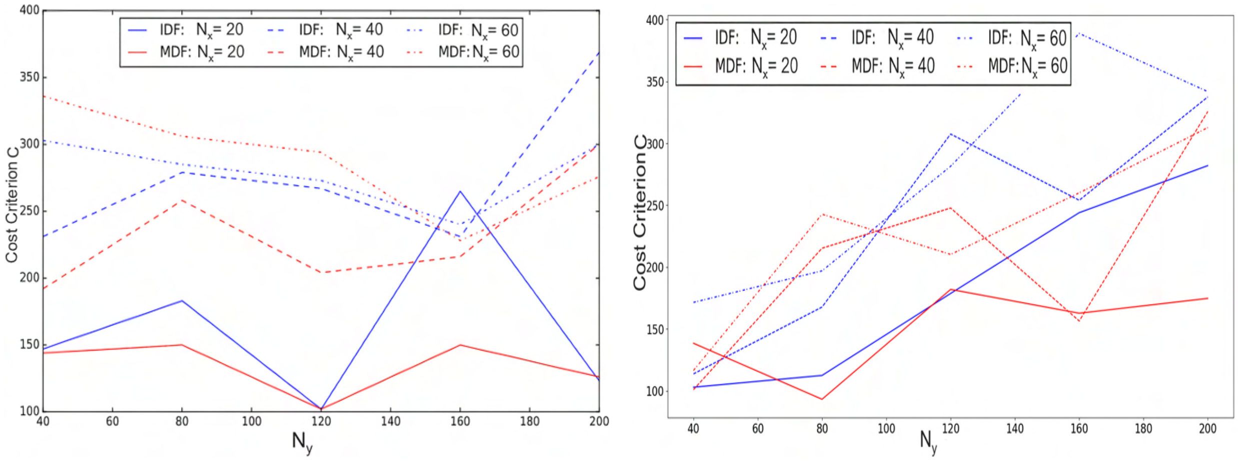
Table 3.1: Range of parameters used for comparison

coefficient	value
c_{lin}	0.5
c_{MDA}	2
c_{LU}	2

Table 3.2: Coefficients used within cost buildup

The comparison is made using the cost criterion C defined in Section 3.3.1. The value of coefficients(time ratios) used within the cost criterion are shown in Table 3.2. Figure 3.23 and 3.24 represent a side by side comparison of the plots made in the original paper and the one made by the candidate using similar conditions.

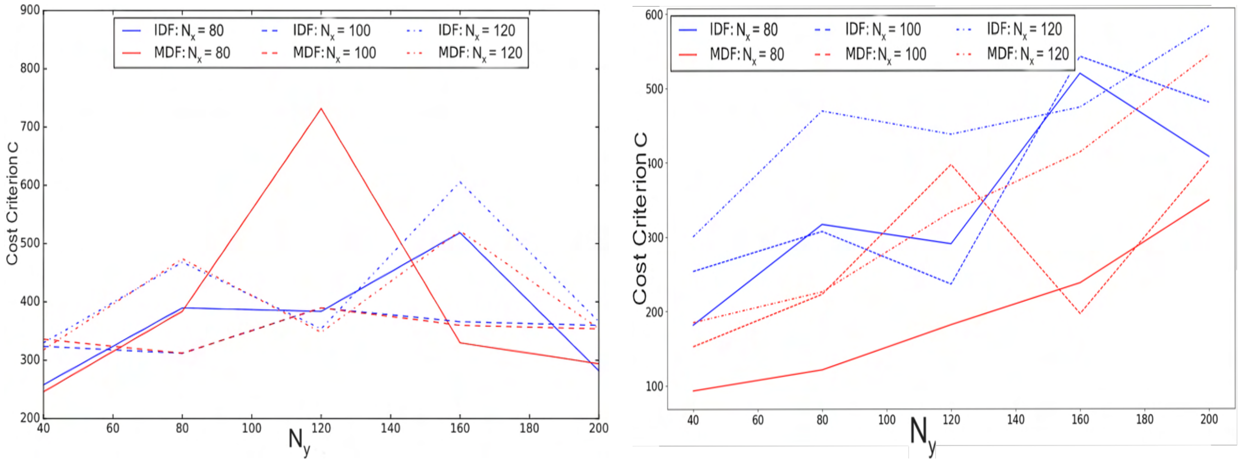
Two figures are used to cover the full range of parameters given in Table 3.1. It can be noticed that N_x, N_y are used in place of n_x, n_y, thereby indicating the total number of design variables and the total number of couplings. For the SSBJ problem, it can be made out from the scaled dependency matrix (Figure 3.12), that for uniform size of design and coupling variables, $N_x = 4 * n_x, N_y = 5 * n_y$.



(a) Original paper

(b) Reproducibility study

Figure 3.23: Estimated cost for MDF-GS and IDF optimizations on the SSBJ test case with $N_x \in [20,60]$ and $N_y \in [40,200]$



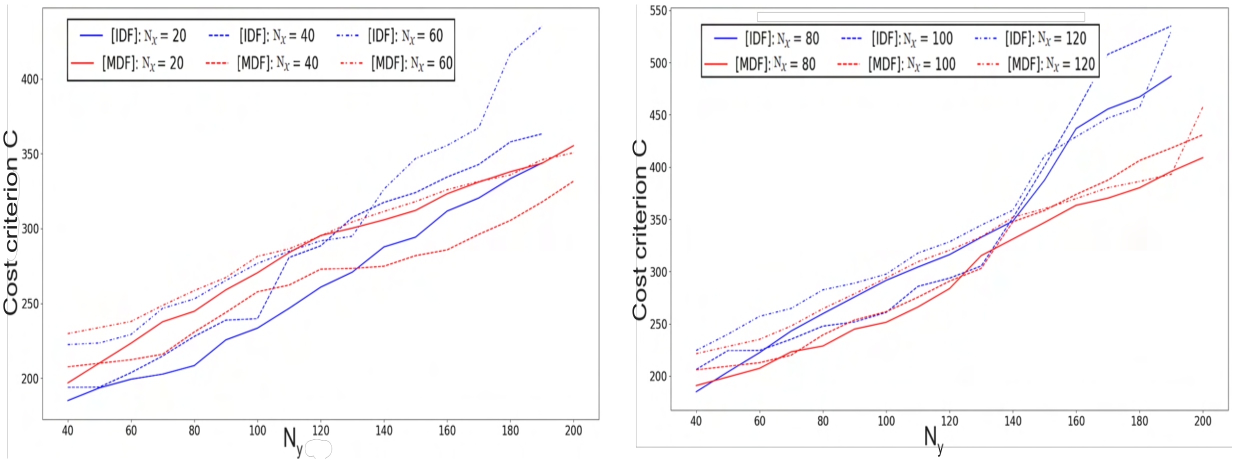
(a) Original paper

(b) Reproducibility study

Figure 3.24: Estimated cost for MDF-GS and IDF optimizations on the SSBJ test case with $N_x \in [80,120]$ and $N_y \in [40,200]$

Looking at Figure 3.23a[16] and Figure 3.24a[16], it can be observed that the sample points used for comparative study in the original paper do not produce a noticeable trend. For each value of N_x, there does not seem to be any particular range of N_y that shows a preference for a particular architecture. Looking at MDF-GS and IDF architectures individually, the overall cost of optimization does not show a gradual rise or fall with an increase in the number of design variables(N_x). There are certain peaks/ dips in the cost criterion (for eg. at $(N_x, N_y = 80, 120)$, $(N_x, N_y = 20, 120)$) that are left unexplained in the paper.
The results from the reproducibility study are shown in Figure 3.23b and Figure 3.24b. Firstly, it can be seen that the results in the reproducibility study are different from the ones obtained in the original plot. This is because of the coupling density factor(d). Even though the same factor($d = 0.4$) is used for both problems, the solution is different because of the random method of dependency allocation used in the SARF method(as explained in Section 3.1). An observation regarding a particular range of N_x, N_y values favoring any particular architecture still cannot be made. However, the reproducibility study represents a more consistent trend in the cost criterion for both MDF-GS and IDF approaches. For both architectures, there seems to be a gradual increase in the cost criterion for each value of N_x when N_y is increased, although unexpected peaks/dips do occur in the cost criterion at points such as $N_y = 160$ (across a range of N_x values).

It can be said that a more thorough investigation is required to plot the cost criterion vs problem size in a more deterministic way. This can be done by including more sample points and multiple runs on each sample point in the reproducibility study. Applying multiple runs on the same (N_x, N_y) value amounts to running different MDO problems by generating a new dependency matrix and component dependency graph at every run. Considering a denser set of parameter values does not generate a trend on its own, but it should add clarity to new trends that emerge out of multiple runs. By considering a denser set of N_y values within the same lower and upper bounds($N_y \in [40, 200]$), and considering the median cost ratio of seven runs, the same comparison is repeated and the results are plotted in Figure 3.25.



(a) $N_x \in [20,60]$, $N_y \in [40,200]$

(b) $N_x \in [80,120]$, $N_y \in [40,200]$

Figure 3.25: Estimated cost for MDF-GS and IDF optimizations on the SSBJ test case(multiple runs, more sample points)

For making the above plots, the median cost ratio of seven runs is considered in place of the average cost ratio because it eliminates the effect of an extreme point. Figure 3.25 shows that for the entire range of N_x values tested, a (more) consistent rise in cost criterion with increase in N_y can be observed. More importantly from the context of comparing MDF-GS and IDF architectures, it can be seen that at higher values of $N_y(> 140)$, MDF-GS looks to be the preferred architecture with consistently less cost of optimization compared to IDF, specially at higher values of N_x. For lower overall problem size (eg. $N_x = 20, N_y = 40$), IDF seems to perform slightly better compared to MDF-GS.

3.5. Outcome of Critical Analysis and Further Investigation

Applying the SARF methodology with more runs and a denser data set leads to the emergence of trends that are not found in the original paper. Adding multiple runs accounts for the randomness of the SARF Method and leads to certain regions of data points where one architecture seems to be faster than the other, an observation that challenges the conclusion of "no visible trend" made in the original paper. The denser data set is like an enhancement, which adds clarity to the trends by increasing the number of runs. Therefore, performing a more robust implementation of the comparative study by Vanaret. et.al[16] leads to the emergence of patterns that connect the relative performance of MDF-GS and IDF architectures with certain ranges of problem parameters representing the size of design and coupling variables. In other words, based on Figure 3.25a and Figure 3.25b, a prediction can be made about the relative performance of MDF-GS and IDF architecture on the scaled SSBJ problem without executing the problem. The extended reproducibility study can be considered an example of a small scale prediction model of MDO architecture.

The extended reproducibility study can be thought of as a starting point for building a larger prediction model, where more parameters are included in the investigation and more MDO problems are used to test the existence of the above-mentioned patterns. The SARF methodology allows the user to vary the size of the local constraint vector(n_c) and the coupling density factor(d) but the effect of these parameters on the choice of MDO architecture is not studied in the original paper by Vanaret et. al.[16]. Instead, the number of constraints is fixed to the number of design variables and only one value of density factor is tested($d = 0.4$) which is the approximate density factor of the original SSBJ problem. These parameters can be included in the larger prediction model. The SARF methodology provides two coupling related parameters, n_y, and d which represent the number of coupling variables and the coupling density. However, as discussed in the literature review, the coupling strength is a separate parameter, which can be estimated through sensitivity analysis of the couplings present in the problem[15, 8]. The difference between coupling strength and coupling density was discussed in Section 3.1.3.1. By selecting an existing method to calculate coupling strength based on literature, a parameter can be defined to assign a particular value of coupling strength for each scaled problem. Along with internal problem parameters, it is also required to include parameters related to the optimization environment such as the number of processor cores. This should allow the prediction model to see the effect of parallel processing on the relative performance of MDO architectures. As explained later, centralized processes like IDF(which don't use Multi-Disciplinary Analysis Solver) run faster when parallel processing is utilized. For MDF, the Jacobi convergence scheme is required for parallel processing to work (as opposed to Gauss-Seidel which was used in the reproducibility study and the original paper by Vanaret et. al.[16]), but the speed improvements are not expected to be as vast as IDF.

The process adopted by Vanaret et.al[16] to make a two-dimensional plot for each parameter pair is not going to work when the effect of many parameters is to be analyzed simultaneously. For instance, when looking at the combined effect of coupling density and the number of processors, other parameters have to be considered constant. How the "considered" features react with the "constant" ones to affect the performance of MDF(GS/jacobi) and IDF architectures cannot be seen with a two-dimensional plot. This is where a machine learning based analysis can be useful. The next chapter is about the construction of a generalized prediction model, based on machine learning assisted analysis of problem parameters.

4

Prediction Model of MDO Architecture

A machine learning system works within the frame work of **build, validate** and **test**. However, employing a blindfold approach by assigning certain parameters as features and a particular quantity as a label often leads to an unreliable prediction model. Every potential feature to be included in a machine learning based prediction model needs to undergo a review process so that the benefit of including such a feature can be assessed:

1. Firstly, it should be studied whether a given feature has a tangible effect on the outcome of the prediction model. A feature should not only be able to independently bias the decision of a prediction model, but also establish a definite trend in the outcome. If a perceivable trend is not apparent, such a feature would harm the prediction capability of the model.
2. Secondly, every selected feature should support the idea of generalizability, i.e each feature should be repeatable to new MDO problems without compatibility issues. For instance, the coupling density factor parameter in the SARF methodology proposes one parameter to assign the density of coupling across all disciplines. Now, proposing features to assign individual density factors for each discipline will generate a more accurate model, but the model may not be repeatable to a new MDO problem with a different number of disciplines.

The SARF methodology provides four generalized parameters for creating a scaled problem as shown below:

1. n_x represents the size of each design variable
2. n_y represents the size of each coupling variable
3. d represents the density of coupling
4. n_c represents the size of each constraint variable

The two discipline problem in Figure 4.1 shows the way in which the parameters n_x, n_y and n_c define a scaled problem. The SSBJ problem is selected as the MDO problem for the review process of problem parameters. Initially, three out of the above four features are used, while the constraint vector size(n_c) is considered to be the same as the design vector size(n_x). By assigning some initial ranges to the above features, a repository of SSBJ based scaled MDO problems is created. By executing this repository of scaled SSBJ problems with MDF and IDF architectures, a three feature - one label database is created, where the label represents the relative cost of executing an MDO problem with the two architectures(precisely defined in next section). Since there are only three features, the review process can be done through a visual analysis using a three dimensional scatter plot. Following that, the research is extended by sequentially adding two more parameters, constraint vector size(n_c) and the number of processor cores(n_p). The visual analysis is used to answer the following questions related to the implementation of each feature:

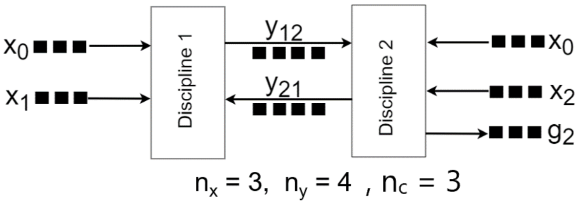


Figure 4.1: n_x, n_y, n_c - Effect on scaled MDO problem

1. What is the ideal range and sparsity that should be assigned to each feature such that the database can be generated within a reasonable time and also the patterns generated by each feature are fully captured?
2. What should be the mode of implementation of each feature column(categorical/numerical/ordinal)?

Once there is clarity on these questions, a normalized database of SSBJ based scaled MDO problems can be generated and a machine learning based prediction model can be built and validated.

4.1. Visual Analysis - SSBJ problem

This section is used to perform an exploratory analysis of the contribution of each problem parameter on the comparative performance of MDF(using gauss-seidel convergence scheme) and IDF architectures. There are three major subsections here, the first deals with the problem size and coupling density parameters proposed by the SARF methodology(n_x, n_y, d). This is followed by a subsection where the size of local constraints is chosen as a separate parameter and its effect on the relative performance of MDF and IDF architectures is analyzed. Lastly, the effect of parallel processing, i.e using multiple processor cores for MDO optimization is analyzed(using Jacobi convergence scheme in place of NLBGS for MDF).

4.1.1. Problem size and Coupling density

To define the problem size, SARF methodology uses two parameters n_x and n_y, representing the size of the design and coupling vectors respectively, and a density factor d to determine the level of dependency between disciplines. Taking a cue from the investigation performed by Vanaret et. al.[16], the size of the design and coupling vectors can be selected such that within a reasonable time, a sufficiently large database of scaled problems can be executed. Considering a particular value of (n_x, n_y, d) as a data point, seven runs are required at each data point to allow the emergence of a trend, a point previously discussed in the reproducibility study of Section 3.4. Initially, the following range of parameters is considered as shown in table 4.1.

Parameter	Range of values
n_x	2 - 16
n_y	2 - 16
d	0.3 - 0.7

Table 4.1: Initial range of parameters

$\mathbf{n_x}$	$\mathbf{n_y}$	d	Cost Criterion[MDF])	Cost Criterion[IDF])
2	7	.7	158	100
2	10	.7	198	134
2	15	.7	228	154

Table 4.2: Sample Database for analysis

A random section of the generated database is shown in table 4.2. This table includes the above problem parameters and the cost criterion for MDF and IDF optimizations. For making a comparison based on the cost criterion, it is required to non-dimensionalize the cost outputs and define a cost ratio, so that the advantage of using one architecture over the other can be seen over a range of problem parameters. The following steps are undertaken to define the cost ratio(R) over each data point:

1. Snip the part of the database containing all the runs for a particular data point and calculate the median cost ratio for the data point.
2. Based on the cost definition proposed by Vanaret[16](Equation 3.7 and 3.8), take the sum of the total cost for both MDF and IDF and divide IDF over MDF to get an initial cost ratio R_i, which could range from zero to infinity. When greater than one, R_i signifies MDF to be faster than IDF. Between zero and one, it signifies that IDF is faster.
3. The maximum and the minimum values of the factor R_i, over the entire test database is calculated as max(R_i) and min(R_i).
4. Using max(R_i) and min(R_i), the cost ratio(R) is interpolated separately as shown below (and in Figure 4.2). Therefore a negative value of R indicates IDF gives a quicker solution while a positive value means that MDF is faster.

$$R = \begin{cases} (R_i) * \frac{(R_i-1)}{(max(R_i)-1)}, & for \qquad R_i > 1 \\ -(R_i) * \frac{(R_i-1)}{(1/min(R_i)-1)}, & for \qquad R_i < 1 \end{cases}$$

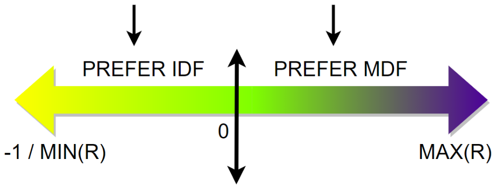


Figure 4.2: Interpolation for factor R

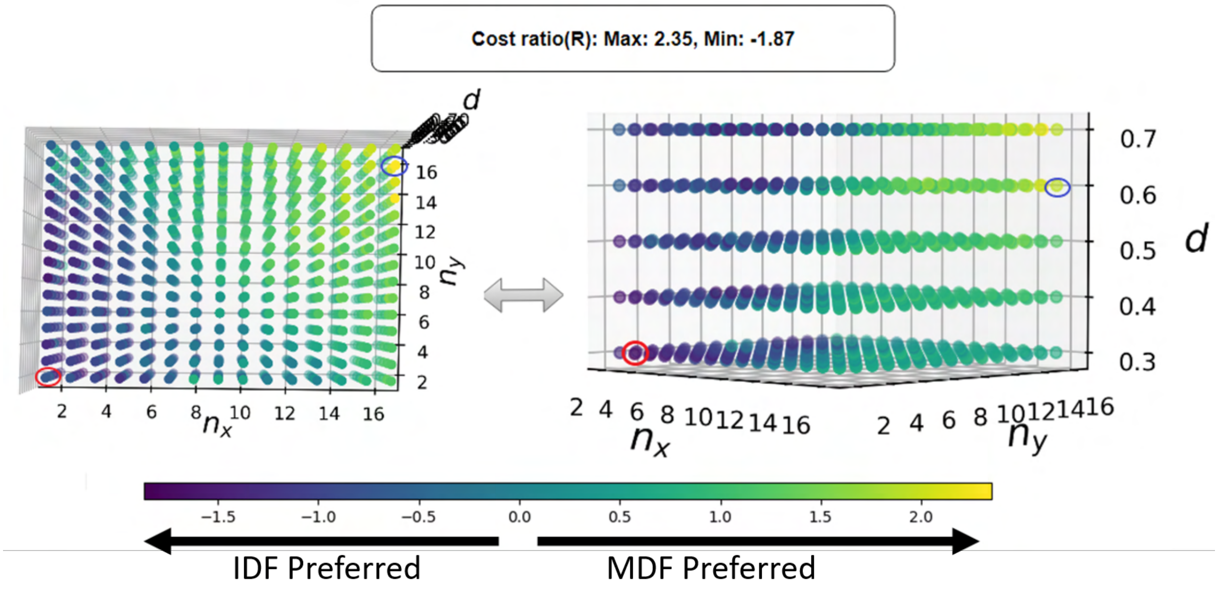


Figure 4.3: Scatterplot - cost ratio vs (n_x, n_y, d)

The cost ratio R replaces the last two columns of Table 4.2 with a single label. Considering the range of parameters shown in Table 4.1, a three dimensional scatter plot can be made as shown in Figure 4.3. Since every "dot" represents a particular data point or a particular value of (n_x, n_y, d), Figure 4.3 represents a total of (13 * 13 * 5 = 845) data-points. Every data point is run seven times for a total of 5915 runs. A **viridis** colormap is used to represent the cost ratio(R) as a function of n_x, n_y and d, plotted along the three axes. The maximum positive and negative values of the R is also indicated in the figure. It shows that for the dot representing the darkest blue close to the bottom left end of the color spectrum(circled in red), IDF is faster than MDF by about 187.4 percent while for the lightest green at the other end(circled in blue), MDF is faster than IDF by about 235.2 percent. Two viewpoints are used, one along the n_x - n_y plane and the other along an asymmetric viewpoint between n_x - d plane and n_y - d plane. For every value of coupling density, as the problem size is increased, the value of the R becomes progressively positive indicating preference towards MDF. This phenomenon is more apparent with an increment in n_y which indicates the size of the coupling variables used in the scaled problem. This suggests that for a higher number of coupling variables, MDF might be a better option. A possible reason for this might be the large number of consistency constraints that are required to be satisfied while running a scaled problem with IDF architecture.

When looked at the same plot from the viewpoint of increasing the coupling density, it looks like the shift from IDF to MDF with an increase in problem size is slightly more apparent at higher coupling densities but the effect is not highly conclusive at this stage. Given the methodology used in forming the replacement function, it is not beneficial to use values of coupling density closer to 1.0 because it reduces the likelihood of generating independent combinations of the components of the output[16]. Also selecting coupling density lower than 0.3 has practical limitations in the way that density matrix becomes too sparse to generate a correlation between every coupling and at least one component of the design space.

Parameter	Range of values
n_x	16 - 24, 30, 31
n_y	16 - 24, 30, 31
d	0.3 - 0.7

Table 4.3: Extended range of parameters

The above range of parameters can be extended to include greater problem sizes. Taking into consideration the higher wall time of optimization for greater problem sizes, both parameters related to problem size, n_x and n_y, are increased in steps of two , as shown in the Table 4.3. Additionally, to account for extremely large problem sizes, two more rows and columns, representing problem sizes $(n_x, n_y \in [30,31],[30,31])$ are also included. This set of extensions leads to the creation of separate zones with varying density of data-points as shown in Figure 4.4(x-y plane). By superimposing the data-points of Table 4.3 on the previous set(Table 4.1), Figure 4.4 allows the user to check the continuance of the trends observed in Figure 4.3 into higher parameter ranges.

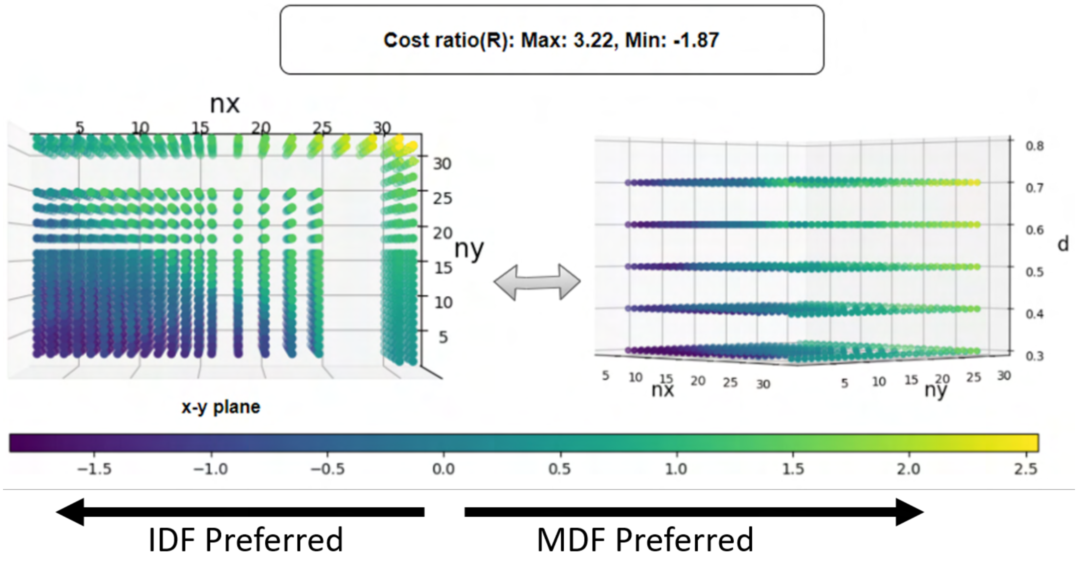


Figure 4.4: Scatterplot - cost ratio(R) vs (n_x, n_y, d) (Extended parameter range)

Looking at Figure 4.4, a clear gap can be observed between the zones where MDF and IDF are preferred. For smaller values of $n_y(< 5)$ at lower coupling densities(d), IDF seems to be the preferred architecture. At higher values of d, preference towards IDF is restricted to very small problem sizes($(n_x, n_y < 5)$ Preference towards MDF architecture is observed at higher values of n_y, irrespective of the coupling density factor d. Compared to the smaller parameter ranges(Figure 4.3), at all values of d, the shift from IDF to MDF is more apparent when moving towards very high problem sizes. The minimum value of the R stays the same as in Figure 4.3(-1.87) since it represents the left end of the spectrum that shows the best case scenario for IDF. At the other end of the spectrum, the maximum value of the cost ratio is 2.55, which corresponds to the data-point $(n_x, n_y, d) = (31, 30, 0.7)$.

Based on the above two investigations, it can be said that an increment in any of the three parameters investigated so far leads to a shift of preference from IDF to MDF architecture. The shift from IDF to MDF can also be represented by a secondary plot created in Figure 4.5, which uses a three-color representation instead of a color bar. The purple zone indicates cost ratios that are lower than -0.5(IDF > 50 percent faster than MDF), green indicates a cost ratio between -0.5 and 0.5 while yellow indicates cost ratio beyond 0.5(MDF > 50 percent faster than IDF). Given the implementation of the SARF methodology, since n_y and d together determine the overall number of dependencies, it can be said that increasing the number of coupling dependencies leads to MDF being

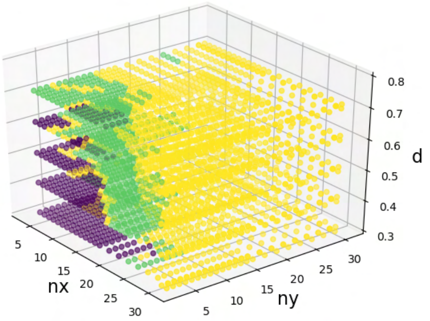
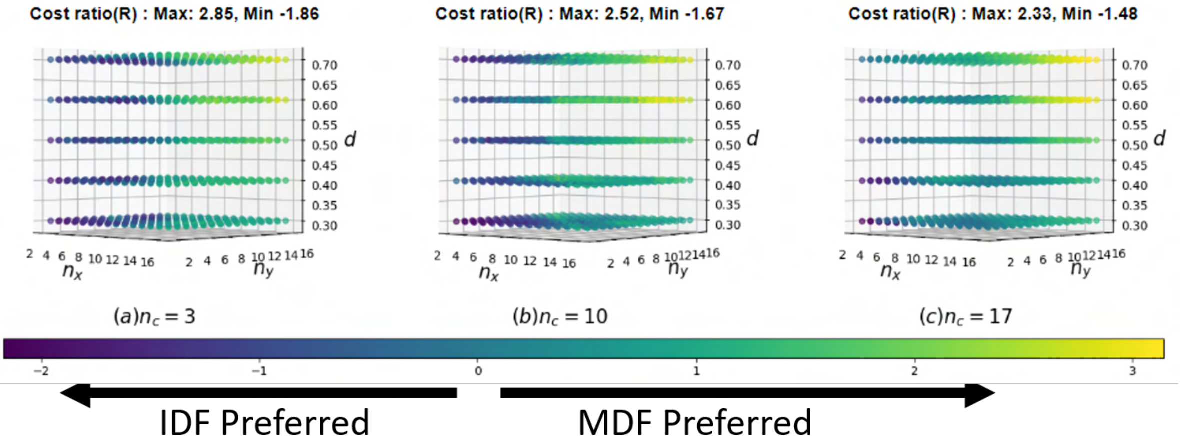


Figure 4.5: Binning cost ratios into three categories (Extended parameter range)

the more preferred architecture. An argument can be made here that the existing results are dependent on the SARF methodology. This is especially true for the extended range plotted in Figure 4.5, which shows some extremely high problem sizes. However, it must be noted that the nature of the original SSBJ problem resembles a coupling density factor of around 0.4[16]. Also, considering the averages of the non uniform values of n_x and n_y, the original SSBJ problem can be considered to be an MDO problem of small size($n_x, n_y < 5, 5$). The data-points that resemble these values indicate a preference for IDF. When the original SSBJ problem is executed in OpenMDAO, the cost ratio also comes out to be negative(-1.35, discussed in more detail in Chapter 5). The data-points in Figure 4.5 that represent MDF as the preferred architecture have been derived by scaling the original SSBJ problem to much higher dimensions($n_x, n_y > 15$). Such high values of problem size are not indicative of real-world MDO problems. Also, looking at Figure 4.5, there is a green zone of data points that do not indicate a clear preference between MDF and IDF ($n_x, n_y \in [5, 15], [5, 15]$, all coupling densities). These ranges require additional investigation using a machine learning model.

4.1.2. Constraint size

The above investigation was carried out by considering the size of disciplinary constraint vectors to be equal to the size of the design vector(n_x). A second investigation can be conducted by decoupling the sizes of the design and the constraint vectors so that the sole effect of altering the disciplinary constraints can be studied. A singular parameter n_c is defined to uniformly assign the number of disciplinary constraints outputted by each discipline. Instead of selecting a range of values for n_c, to reduce the time of analysis, n_c is added as a parameter that takes up a value representing the relative number of constraints with respect to another feature such as n_x. In this manner, a four feature database is created such that the design variables, couplings, constraints and the density of interdisciplinary coupling can be independently represented. For this investigation, the range of earlier parameters is considered to be the same as shown in Table 4.1. Concerning the range of n_x, n_c is assigned to one of [3, 10, 17], representing three levels of local constraint sizes. The database is executed for each value of n_c with MDF and IDF architectures and the results are plotted in Figure 4.6



$(a) n_c = 3$ $(b) n_c = 10$ $(c) n_c = 17$

Figure 4.6: Scatterplot - cost ratio(R) vs (n_x, n_y, d) / varying constraint sizes

Before these results are discussed, it must be noted that in the SARF methodology, every disciplinary interface is represented by a child class of OpenMDAO's $ExplicitComponent$ class, which provides the methods to handle the inputs and output of each coupling of the discipline. The disciplinary constraints are computed by initializing a separate $ExplicitComponent$ class. Since the classes representing the constraint outputs are not placed within an iterative solver scheme, the constraints are not required to be evaluated within any analysis routine and are handled only by the optimizer. Therefore varying the size of the constraint vector is not expected to create a decisive bias between the performance of MDF and IDF architectures. However, there is still some merit in including the constraint vector size as a feature in the prediction model as shown below.

The analysis of the trends found in Figure 4.6 is more subtle than that of Section 4.1.1. Firstly, Figure 4.6 (a) is considered. It can be seen that for a value of $n_c = 3$, the trends observed in the scatter plot with respect to all three axial features are similar to that of Figure 4.3. Looking at the spectrum from left to right, the shift from IDF to MDF is visible. The maximum advantage that IDF has over MDF is also similar at 186%. However, the highest positive value for the cost ratio is 286%, which is more than the case shown in Figure 4.3(where n_c was fixed to n_x). This could because the system optimizer is not loaded with the satisfaction of a large number of constraint variables, which allows the emergence of a wider gap between the two architectures, solely due to the different routes taken by them for solving the interdisciplinary couplings.

Moving towards Figure 4.6 (b) and (c), for higher values of n_c, the clarity of the zones preferring MDF and IDF architectures is much reduced. The maximum and minimum values of the cost ratio(R) is also reduced in magnitude. For a value of n_c = 17, as shown in Figure 4.6 (c), a large part of the spectrum(except the lower left and upper right extremes), seems to suggest a value of R close to zero which leads to a prediction model suggesting no difference between the performance of MDF and IDF architectures. The plot of Figure 4.6(c) is shown with greater clarity in Figure 4.7 by binning the cost ratios into three categories(using same method as Figure 4.5). It can be seen that green portion covers the largest range of (n_x, n_y) values(Cost ratio$R \in [-0.5, 0.5]$). Based on the above analysis, it can be said that n_c is a feature that does not affect the relative performance of the two architectures but at certain values, it leads to an inconclusive prediction model. At the same time, without the inclusion of this feature, the model might overestimate the advantage that one architecture has over another. Therefore, it is relevant to include a feature related to the constraint size in the prediction model.

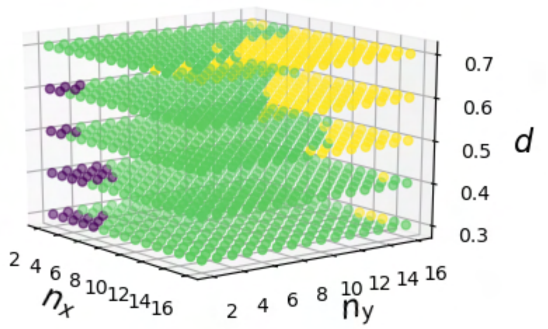


Figure 4.7: Binning cost ratios into three categories for Figure 4.6(c)

4.1.3. Number of Processors

Parallel processing, i.e. usage of multiple processors, is another parameter that can be taken into account while selecting an MDO architecture. There are some existing works in this field that discuss the benefits of using parallel processing approaches at different "algorithmic levels" of an MDO problem like the subsystem solver level, system optimization level and multi-point optimization. Costiner et al.[41] reckoned in their paper, "Just like any complex industrial design task, an optimization can be parallelized and practically solved only when they can be split into weakly coupled subtasks layered on several levels". This seems to suggest that IDF should take better advantage of parallel processing. The same idea is given by Cramer et al.[31] who, while introducing the IDF approach as a decoupled version of MDF in 1994, suggested: "IDF has the advantage of coarse-grained parallelism naturally suited to a heterogeneous computing environment[31]." Within this investigation, it is intended to study the merit of parallel processing as a potential feature within the prediction model. In other words, it is required to study the effect parallel processing has on the performance of MDF and IDF architectures. Within OpenMDAO parallel processing is implemented by defining a $ParallelGroup$ class. The components needed to be executed in parallel are then added to this class using $add\_subsystem$ method. In MDF, parallelism is implemented at the solver level by instantiating a $ParallelGroup$ class for the MDA solver which enables the parallel evaluation of the three coupled disciplines(Structure, Aerodynamics and Propulsion). The XDSM for this configuration, generated using a scaled SSBJ problem $(n_x, n_y, n_c = 2, 2, 2)$ is shown in Figure 4.8, where disciplines to be evaluated in parallel are represented using a parallel group. Since parallel processing is only used within the MDA routine, only the coupled disciplines are shown inside the parallel group while the Performance discipline and the constraints are not. Instead of NLBGS, Non-Linear Block Jacobi(NLBJ) is used as the MDA solver because the sequential nature of NLBGS does not lead to performance gains over single-core optimization. In the case of IDF, the subsystems to be evaluated in parallel can be directly added to the optimizer. The XDSM for the IDF configuration is shown in figure 4.9.

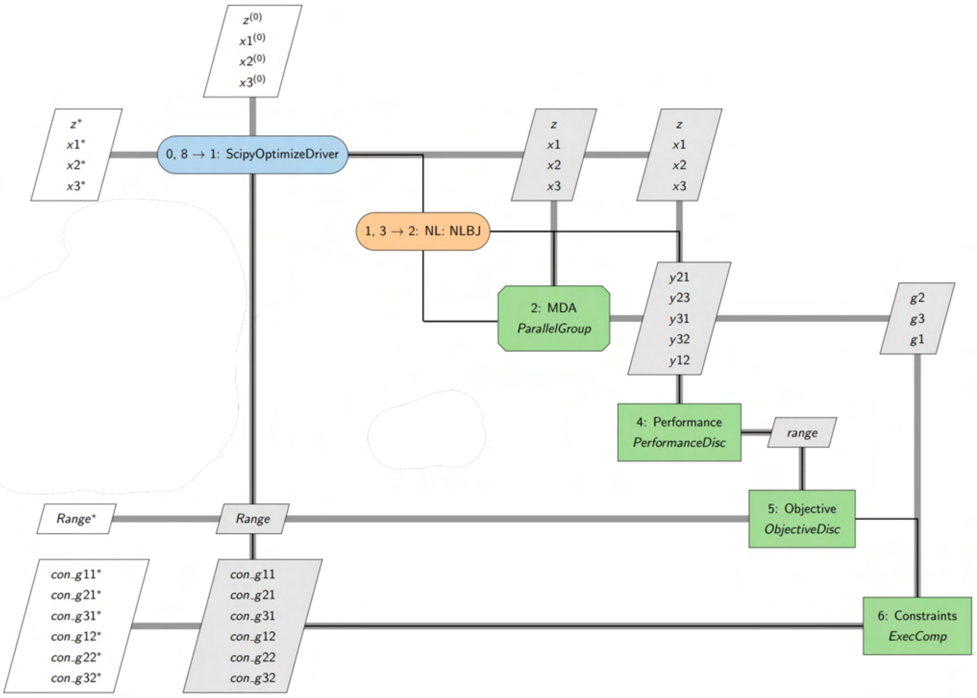


Figure 4.8: XDSM for scaled SSBJ problem $(n_x, n_y, n_c = 2, 2, 2)$ (MDF-Jacobi)

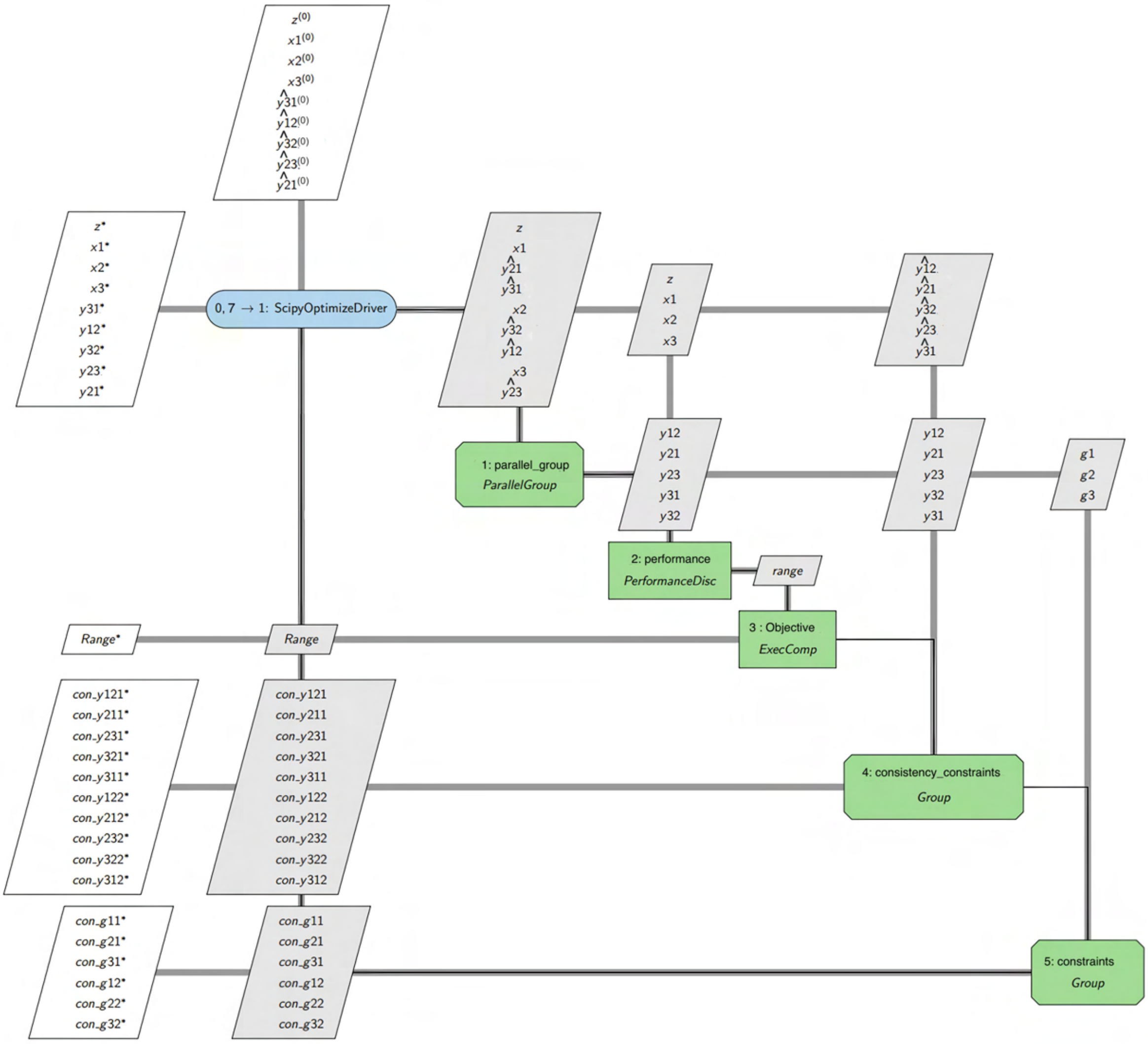


Figure 4.9: XDSM for scaled SSBJ problem $(n_x, n_y, n_c = 2, 2, 2)$ (IDF)

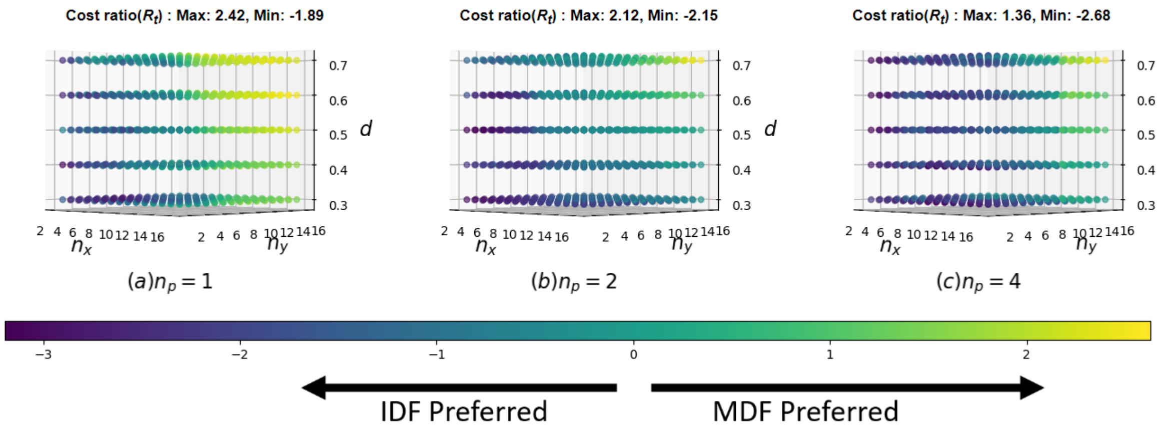
In this form of parallel processing, for both MDF and IDF, there is a direct relation between the number of processors and the number of computational blocks. To run an optimization process in parallel, OpenMDAO requires the presence of Openmpi4py and Petsc4py packages, along with a working build of MPI(Message Passing Interface). The following command can be called in the terminal, specifying the script to be run and the number of processors to be used.

$$-mpirun \quad -n \quad (number \quad of \quad processors) \quad python \quad script.py$$

In this way, parallel processing is enabled in the optimization process by specifying the number of processor cores that are available to the optimizer. In this section, it is required to analyze the effect of varying the number of processors on the relative performance of MDF and IDF architectures.

Modern multi-core desktop processors are usually configured to have a core count in multiples of two. The test system for this analysis consists of a processor with four physical cores. By defining a feature called the number of processors (n_p), that can take a value one of $[1,2,4]$, the investigation can be carried out in the form of three plots, each representing the number of processor cores(n_p). For $n_p = 4$, there is an adequate number of processor cores for both MDF and IDF. However, for $n_p \in [1,2]$, the computational blocks to executed in parallel are more than the available cores. In such cases, the blocks to be executed in parallel are selected randomly by the Petsc4py library. Therefore, each setting for n_p is expected to have an effect on the optimization process and the relative performance of MDF and IDF architectures.

The plot representing $n_p = 1$ is drawn to represent a basic case with a Jacobi convergence scheme. Since there are existing studies [28] to show that block Gauss-Seidel is usually faster than block Jacobi over a variety of convergence problems, it is important to plot this basic case so that the sole effect of altering the number of processors can be properly studied. Also for this investigation, the same range of the basic feature values (n_x, n_y, d) are used as shown in Table 4.1. The constraint size parameter is fixed at $n_c = 5$ to avoid the biases incurred due to changing constraint sizes. Just as before, every data point is given seven runs to allow the trends to emerge. It must also be noted that for this investigation, wall time ratio(R_t) is used instead of the cost ratio(R) as the comparison parameter because the iterative counters used in the original cost criterion(discussed in Chapter 3, Section 3.3.1) do not change due to the usage of multiple processors.



$(a) n_p = 1$ $(b) n_p = 2$ $(c) n_p = 4$

Figure 4.10: Scatterplot - time ratio(R_t) vs (n_x, n_y, d) / number of processors

From Figure 4.10(a), it can be observed that switching to the Jacobian iteration(NLBJ) MDA method for MDF and using a different cost metric produces a mild difference compared to Figure 4.3 (where the same range of feature values were tested using NLBGS iteration scheme). The maximum and minimum values of the cost ratio are found to be nearly similar. Increasing the processor count to two($n_p = 2$), it can be seen in Figure 4.10(b) that there is an expansion in the zone that indicates IDF as the preferred architecture. Across all values of coupling density, a larger range of problem sizes indicate IDF as the faster of the two architecture. Specifically looking at lower coupling densities($d \in [0.3, 0.4]$), it seems that IDF is the preferred architecture for the majority of the problem sizes. Increasing the processor count to four ($n_p = 4$), leads to an increment in this trend and except for regions that represent a very high number of couplings ($n_y > 12, d \in [0.7, 0.8]$), the majority of the spectrum seems to indicate IDF as the preferred architecture.

Based on the above analysis it can be concluded that allocating individual processor cores to computational blocks has a tangible effect on the preference of MDO architectures, in that increasing the number of cores leads to greater improvement in the solution times required by IDF. But the effect has so far only been observed in the context of the three basic problem parameters. How this feature interacts with constraint size as a feature or any other feature (to be included later) can be better understood with a machine learning model.

4.2. Building a Prediction model

In the previous section, some of the parameters that could influence the performance of MDO architectures were visually analyzed. Testing for the influence of individual parameters, a range of feasible values for each parameter was used to construct a database within a reasonable time. For the problem size related parameters, it was analyzed as to within what ranges, a parameter could have an influential say on the decision of a recommendation system, beyond which the effect would taper off. Apart from the problem size and coupling density related parameters proposed by Vanaret et. al[16], two other parameters were also identified, i.e number of constraints(n_c) and the number of processors(n_p).

For building a machine learning assisted prediction model of MDO architecture, it is required to use a data science based approach from the ground up. There is no specific protocol for building a supervised machine learning model. The approach varies according to the volume and type of features used in the model, and the overall sectioning of the data-set used to train, validate, test and deploy the machine learning model. In the current scenario, the SSBJ MDO problem has been used so far as the standard for implementing the reproducibility study for the SARF methodology, as well as for the visual analysis of the potential features. Therefore the initial version of the machine learning model is created on a database of scaled SSBJ Problems. Following this, additional MDO problems are introduced to the machine learning model for validation and testing.

The following list shows the major steps involved in the creation and deployment of the machine learning based prediction model:

1. Generation of a normalized data-set of features and label(Section 4.2.1).
2. Testing the fitting algorithms provided by a python based machine learning package to look for the best possible fit(Section 4.2.2).
3. Validating the model based on the SSBJ Problem with scaled versions of three other MDO problems from the NASA MDO test suite and testing for fit accuracy(Section 4.2.3).
4. Making possible readjustments to model parameters / adding additional features to make a better prediction model(Section 4.2.4).
5. Testing the well-trained model on new MDO problems(besides SSBJ and the three validation problems) and driving conclusions(Chapter 5).

Four of the above five points are discussed in the following sections, the final point being taken up in the next chapter.

4.2.1. Feature Definition - Generation of a Normalized Database

The first step towards creating a machine learning model is the generation of a large database with a number of normalized distinct feature columns and one label column. For the feature columns concerning problem size, namely n_x, n_y, based upon the visual analysis, the variable sparsity range of values is selected as shown below:

$$n_x \in \{range(2,16,1), range(18,24,2), 30\} \tag{4.1}$$

From 2 to 16, the values are selected in steps of one and from 18 to 24 the values are selected in steps of two, with 30 being an extreme value. The same applies to n_y. This is an example of a categorical feature column containing a set of non-continuous values. The normalizing factor for n_x and n_y is selected as the total number of disciplines. The total number of disciplines is a logical choice because the parameters allocate the same size of design and coupling variable to each discipline. For implementing the number of constraint variables (n_c), a categorical feature is adopted based on a constraint factor f_{n_c}, $f_{n_c} \in \{0,1,2,3\}$ which allocates a value of n_c relative to the overall range of design variables(n_x) used in the database. Therefore, n_c can be defined as function of the user specified constraint factor(f_{n_c}), comprising of one of four outcomes as shown below:

$$n_{ci} = min(n_x) + \frac{f_{n_c} * (max(n_x) - min(n_x))}{3} \quad ; \qquad i \in \{0,1,2,3\} \tag{4.2}$$

Applying the above translation, for $\{min(n_x), max(n_x)\} = \{2,24\}$, the values to be entered in the constraint feature columns are $\{2,9,16,24\}$. For non-integral values of n_{ci}, a **floor** function can be applied to restrict the outcome to only integral values. In this manner, the relative effect of the constraint variables with respect to design variables can be better studied by the machine learning algorithm. Since effectively only four values are assigned to the feature, there is only a moderate increase in the time required to create the database. Further, the number of constraints entered in the feature column can be normalized with respect to the total number of disciplines. The feature column representing the coupling density factor is already non-dimensional. Coupling density can be seen as a numerical feature containing continuous values between $[0.3, 0.7]$. Finally, the number of processors can be implemented as an ordinal feature column, that takes on a value within $\{.25, .5, 1\}$ representing one, two and four cores.

The label column is the outcome of every case that gets entered into the database. For the feature rows representing $n_p = 1$, the MDF convergence scheme used is Gauss-Seidel and the cost ratio(R) is calculated using cost criterion discussed in Section 3.3.1. For $n_p = 2,4$, the MDF convergence scheme is Jacobi and the cost ratio is calculated using the wall time of optimization(R_t). For the scaled SSBJ problem, using the range of values given above, the normalized data set is sequentially generated as shown in Table 4.4.

Case	Features					label
	Design ($\mathbf{n_x}$)	Coupling ($\mathbf{n_y}$)	Constraint ($\mathbf{n_c}$)	Coupling Density (**d**)	Processors ($\mathbf{n_p}$)	Cost or Wall time ratio (**R** or $\mathbf{R_t}$)
1	2/3	2/3	2/3	.3	1/4	-1.89
2	3/3	2/3	2/3	.3	1/4	-1.86
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
31740	30/3	30/3	24/3	.7	4/4	1.36

Table 4.4: Initial data-set for training(SSBJ Problem)

4.2.2. Testing Machine Learning Algorithms on SSBJ Database

Within this section, the objective is to discuss the approach of building a prediction model out of the SSBJ based database of scaled MDO problems (Table 4.4) using Python's scikit-learn package. The first step is to split the database into training and test sets. This was previously discussed in the literature review(Section 2.5.1). According to the train test split ratio, a certain percentage of case rows are separated into training cases and the remaining rows are used as test cases. The two data sets are further divided into features and label sets. In pythonic terms, this translates to fragmenting the data-set into four individual Pandas DataFrames[1] each representing training features, training labels, test features and test labels respectively. This is shown in Table 4.5. 33% of the rows(both features and labels) that fall under training cases are used in training the machine learning model and 66% of the rows(only features) are used for testing the prediction capability of the trained model.

Cases	Features (5 columns)	Label (1 column)
Training cases (10580 cases)	Training Feature DataFrame	Training Label DataFrame
Test cases (21160 cases)	Test Feature DataFrame	Test Label DataFrame

Table 4.5: Split the SSBJ based database
(1:2 train-test split ratio)

[1] https://www.geeksforgeeks.org/python-pandas-dataframe/

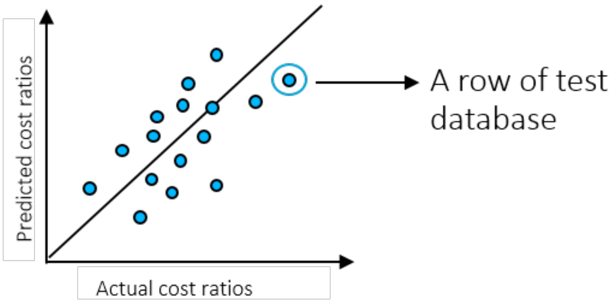


Figure 4.11: Test method-Predicted vs Actual cost ratios

Following this, a method is defined for testing the prediction capability of a trained machine learning model. Looking at the testing methodology proposed by Secco et.al.[39](Section 2.5.1) and Loyer et.al[21](Section 2.5.2), the method is based on a two-dimensional plot that compares the prediction made by the machine learning model on each test row vs the actual prediction. An example of this plot is shown in Figure 4.11. For a perfect prediction, all data points should fall along the $y = x$ line. Now, in the paper by Secco et.al.[39], the mean square error(MSE) metric was used to measure the predictive performance of the machine learning algorithms. MSE can be defined as shown below:

$$MSE = \frac{1}{n}\sum_{i=1}^{i=n}(y_{pred(i)} - y_{test(i)})^2 \tag{4.3}$$

In equation 4.3, y_{pred} and y_{test} refers to the predicted and actual labels. The above MSE term can be considered as the objective of the minimization problem that lies at the core of the machine learning model. It can also be used as a performance metric to measure the prediction capability of a machine learning algorithm. However, MSE is not a non-dimensional metric and does not provide a clear understanding of the "goodness of fit" of the model. For measuring the quality of fit in a qualitative manner, a number of non-dimensional metrics can be used. One such metric is the Explained Variance Regression Score($EVRS$)[2]. Unlike MSE, EVRS takes the form of a proportion, thereby explaining the lack of fit of the model using a dimensionless factor between zero and one[37]. Considering $Var(x)$ to be the variance of data set x, EVRS can be defined as:

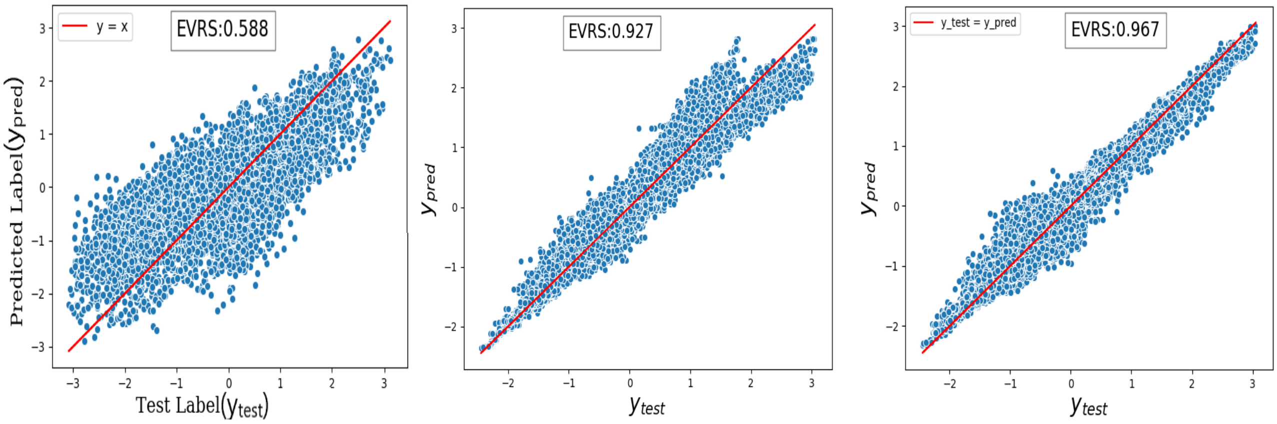
$$EVRS = 1 - \frac{Var(y_{test} - y_{pred})}{Var(y_{test})} \tag{4.4}$$

The above EVRS score is used to test the predictive capability of the machine learning models used in this thesis. The following machine learning algorithms are used for building and testing the prediction model of MDO architecture:

1. Linear Regression
2. K Nearest Neighbours
3. Decision Trees
4. Deep Neural networks

Looking at the spectrum of machine learning algorithms listed above, a hierarchy of complexity in the mathematical background of each algorithm can be observed. At the heart of each machine learning algorithm lies a minimization problem. A detailed discussion of the mathematical background behind every machine learning algorithm is beyond the scope of this thesis. Within this thesis, the understanding of each algorithm is restricted to the model centric parameters involved in the fitting process. The purpose is to analyze the fitting capability of various machine learning algorithms when applied to the custom data set(Table 4.4). The fitting process for each algorithm is discussed in detail in this thesis. However, to keep the story compact, the discussion of the first three algorithms is moved to Appendix C. In this section, only the predictive performance of the first three algorithms on the SSBJ based database of Table 4.5 is shown. The predictive plots of Figure 4.12 are made using the same template as the example plot shown in Figure 4.11.

[2] https://scikit-learn.org/stable/modules/model_evaluation.html#explained-variance-score



(a) Linear Regression

(b) K Nearest Neighbours

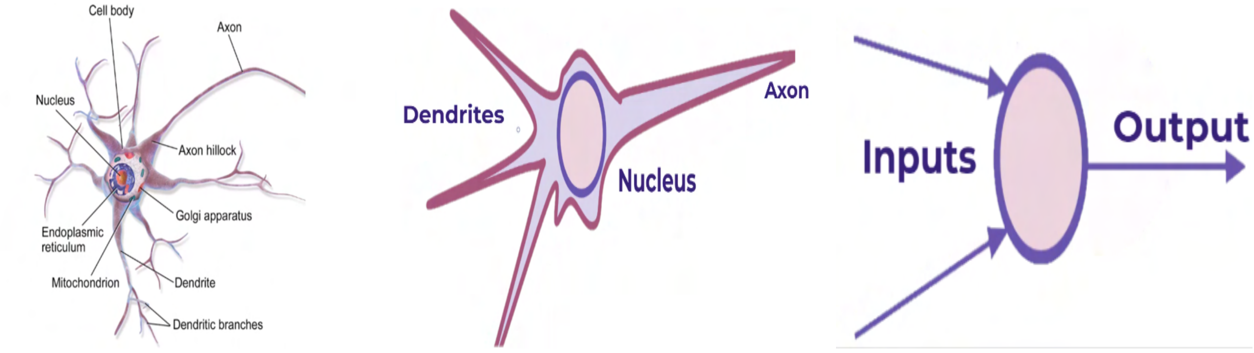
(c) Decision Tree

Figure 4.12: Predictive Performance of three machine learning models on the SSBJ based database of Table 4.5

Looking at the predictive plots of Figure 4.12, the test label(y_{test}), plotted along the x-axis, represents the actual label values for the entries in the test database. The predicted label(y_{pred}), plotted along the y axis, represents the corresponding label values that are predicted by the machine learning algorithms. The prediction accuracy is also shown for each plot in terms of EVRS values. It can be seen that the prediction accuracy increases when moving from linear regression to K Nearest Neighbours and then to Decision Trees. The fitting process for the three algorithms can be retrieved from Appendix C. The final algorithm to be tested is neural networks. This is discussed in detail in the following section.

Neural Networks

Neural networks are a class of machine learning algorithms that aim to mimic biological natural intelligence, using a perceptron model, which is the digital equivalent of a biological neuron. A neuron is generally represented by biologists as an elongated cell with a complicated structure as shown in Figure 4.13a [3]. From the context of neural networks, a simplified version of the neuron can be drawn as shown in Figure 4.13b [4]. A perceptron model can be derived from the simple description of the biological neuron, comprising of three main components, dendrites or the input terminal, a nucleus or the processing core and axon or the output terminal. The perceptron model is shown in Figure 4.13c



(a) Biological Neuron

(b) Simplified Neuron

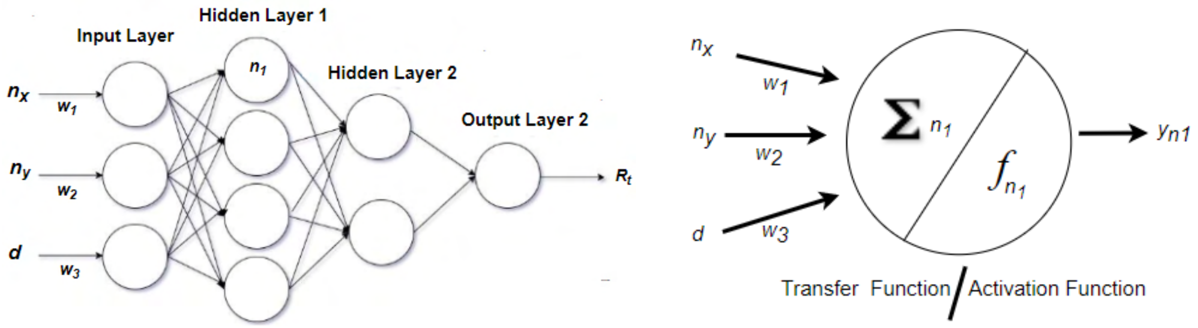
(c) Perceptron model

Figure 4.13: Derivation of Perceptron

The perceptron serves as the elementary building block for a neural network model. A neural network model is usually represented as a web of interconnected nodes sectioned into layers. An example of a simple neural network can be shown using a reduced version of the SSBJ database as shown in Figure 4.14a. The first layer of nodes, referred to as the input layer, is used to feed in the feature values(n_x, n_y, d). The last layer (shown by a singular node for regression problems), represents the output of the neural network. The layers in the middle are used for training the model.

[3] https://en.wikipedia.org/wiki/Neural_circuit

[4] https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7



(a) Simplified Neural Network

(b) Singular Node Representation

Figure 4.14: Neural Network - Node Representation

Figure 4.14b shows the first node of **Hidden Layer 1**, depicted as n_1. Now, there are three inputs to this node, namely $\{n_x, n_y, d\}$. Additionally, there are also certain weights attached to each input. The output of the node, referred to as transfer function($\sum n_1$) can be defined as:

$$\sum n_1 = n_x * w_1 + n_y * w_2 + d * w_3 + b_{n_1} \tag{4.5}$$

where $w \in \{w_1, w_2, w_3\}$ are the weights applied onto each feature value and b_{n_1} is a bias term added to the transfer function. The output of the transfer function is then passed through an activation function[5] to limit its value. Some commonly used activation functions are ReLU(Rectified Linear Unit) and Sigmoid Function. The output of the node is then used as an input to nodes of the following layer. The output of the final layer, comprising of a singular node is the predicted output for the set of values$\{n_x, n_y, d\}$. Neural networks are considered to be powerful estimation tools. Given an appropriate number of layers and nodes, a neural network can be shown to emulate any convex function[42].
It can be seen from the above explanation that every node receives a set of weights attached to each input and also a bias term. For the entire neural network, these weights and bias terms can be framed into individual matrices. Now, to build a machine learning model out of a multiple layer neural network(also referred to as a deep neural network), it is required to adjust these weights and bias values using training cases. The training process is conducted in the following manner:

1. Initiate by feeding in rows from the training database, assigning certain random values to the weight and bias matrix.
2. Calculate the output of the neural network and measure the error from the actual output.
3. Using two techniques called **Gradient Descent** and **Backpropagation**[6], gradually reduce the error metric by adding more training cases into the neural network.

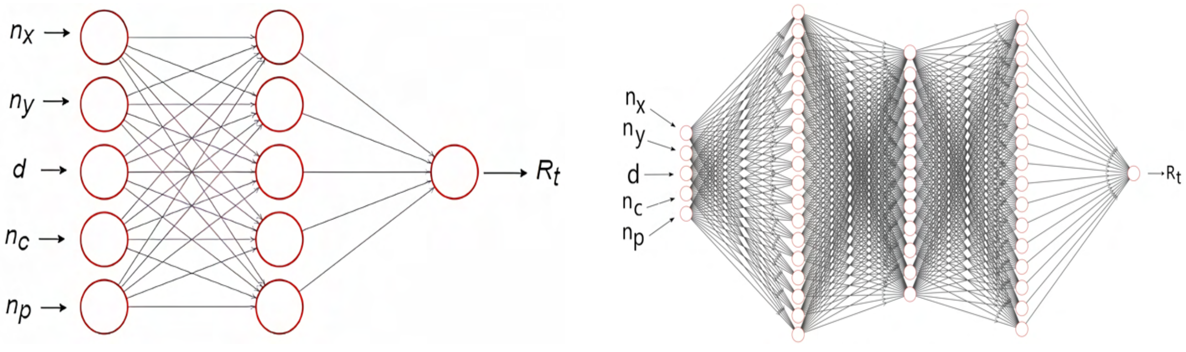
The weights and biases attached to each node, therefore, act as design variables for an optimization problem at the core of the training process, which aims to minimize the error between the projected and the actual output for training cases.

From the context of this thesis, as mentioned before, it is of our interest to understand the way certain user-defined hyper-parameters alter the prediction capability of neural networks. Neural networks are usually considered to be very powerful estimation tools, but at the same time they involve many internal hyper-parameters that require adjustment according to the database in question[7]. One method is to consider all parameters at the same time and perform a multi-dimensional grid search, but that would make the process too complicated. A better option is to broadly classify the parameters into two categories:

1. Parameters that alter the size of the neural network, in terms of the number of hidden layers as well as the number of processing nodes within each layer.
2. Parameters that alter the learning rate of the deep learning algorithm.

[5] https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/
[6] https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/
[7] https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

The first step in hyper-parameter training is size optimization, i.e, to come up with a neural network with the required number of neurons and hidden layers, that is best suited for the SSBJ based database. Shown in Figure 4.14a, the hidden layers of a neural network provide a certain level of abstraction between the feature and label values[43], which makes it an experimental task to narrow in on the appropriate size of the neural network that can effectively simulate a particular database. A neural network of reasonable size should be able to learn the representations that emerge out of the selected database without over-fitting. To settle on a ballpark value for the number of nodes in each hidden layer, references can be taken from existing literature[43, 44, 45]. A recommended norm is to select the same number of nodes in each layer as the number of features in the database[8]. Therefore, for the SSBJ dataset with five features, as a starting point, each hidden layer is provided with five processing nodes. As for the total number of hidden layers, a definitive thumb rule is difficult to obtain. Based on empirical studies by Jeff Heaton[9], one such thumb rule is proposed as "The number of hidden neurons should be between the size of the input and output layers." This should suggest, that for uniform hidden layers(i.e layers having the same number of nodes), the total number of hidden layers should be no more than one. However, it must be noted that the above suggestion is meant only for classification problems, while a specific suggestion for regression problems is not to be found in the literature. Additionally, while proposing the above thumb rule, Heaton also stated that any such rule can only be considered to be just a starting point for a more detailed, data set specific investigation. Therefore, considering the initial number of layers to be one and the initial number of nodes per layer to be five, a systematic grid search is conducted across the number of layers and nodes per layer to arrive at the optimum size of the neural network.



(a) Neural Network - Initial Size

(b) Neural Network - Optimal Size

Figure 4.15: Neural Network - Grid Search Optimization

The outcome of the grid search operation is shown in Figure 4.15. Three hidden layers comprising of different nodes can be observed. Following the selection of the ideal size of the neural network, it is now required to adjust the hyper-parameters that control the learning rate of the neural network. There are many hyper-parameters in this context, such as the activation function for each node, the optimizing algorithm/error metric, or the learning rate of the neural network optimizer. In the current scenario, for simplicity, two parameters are selected that directly affect the speed and stability of the training process, namely, **batch size** and **epochs**. Batch size refers to the number of randomized samples from the training data set that are inducted into the model at one time before error gradients are estimated and model weights are updated. One training epoch conveys that the learning algorithm has made one pass through the training data set, where examples were separated into randomly selected “batch size” groups[10]. The right combination of batch size and epoch is required to effectively control the speed and stability of training the neural network. To find the appropriate value of the above two parameters, one method is to perform a grid search across various epochs and batch sizes. However, given the large ranges of values of these parameters that can be found online, [46, 47, 48], a systematic grid search would be too time taking. Instead, a heuristics-based approach is applied to settle on the appropriate values for these parameters.

According to Bengio et. al.[47], batch size as a hyper-parameter impacts the training time more than it affects the predictive capability of the neural network model. A recommended norm is to select batch sizes in the

[8] https://www.udemy.com/course/python-for-data-science-and-machine-learning-bootcamp/

[9] https://www.heatonresearch.com/2017/06/01/hidden-layers.html

[10] https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

form of 2^n, such as one made by Luschi et. al. [48], who proved using multiple data sets, that selecting a batch size between two(2^1) and thirty-two(2^5) gets reasonable results. Other studies suggest selecting a batch size in order of hundreds[11]. From the existing literature[46, 47, 48], it can be inferred that unless lower batch sizes create excessive noise in the fitting process(in terms of the update to model weights), the batch size should be selected as low as possible, provided that the training time does not get prohibitively large. For narrowing down the appropriate batch size for the SSBJ model, firstly, the number of epochs or training cycles is fixed ($epochs = 2000$). Next, an order of batch sizes is selected(in powers of two) and the fitting accuracy of the neural network is tracked for each consecutive training cycle, or each epoch, up to the total number of epochs. This plot is represented in in Figure 4.16.

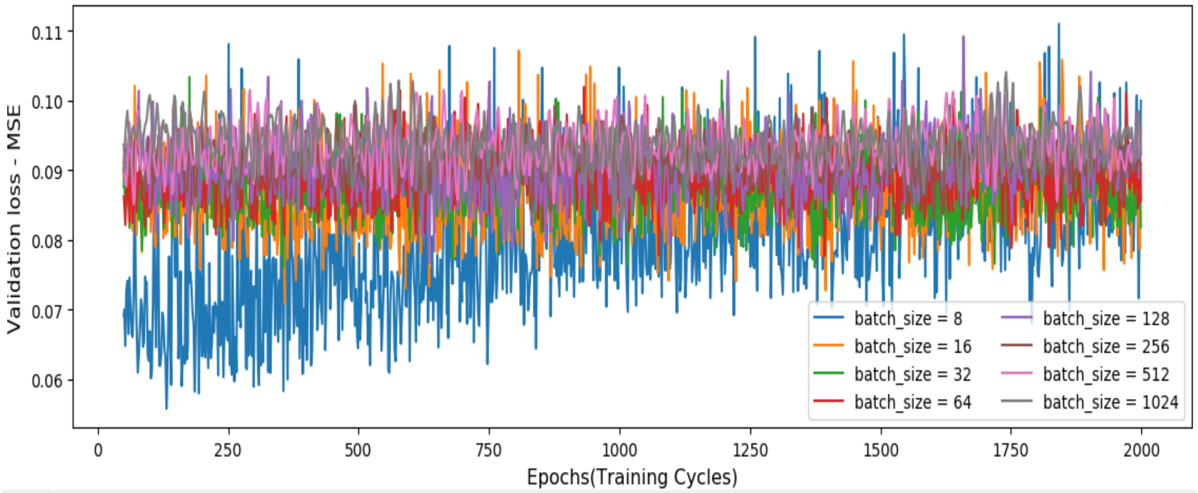


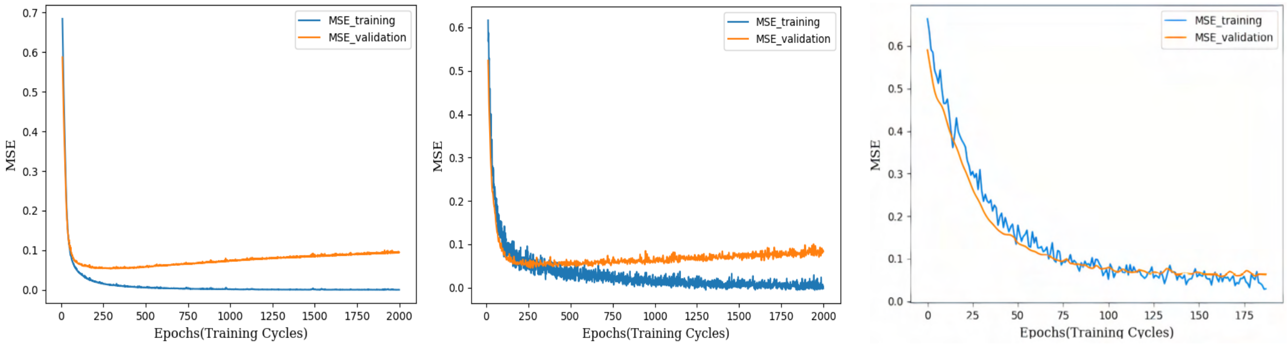
Figure 4.16: Effect of batch size on model training

Looking at Figure 4.16, the quantity plotted along the y-axis is the validation loss, which is the mean square error(MSE), calculated on the validation data points, i.e data points within the training data set that are not used for the particular epoch or training cycle. Instead of a normalized metric like EVRS, scikit-learn uses MSE as a loss function for updating weights at the end of each epoch. MSE represents the predictive performance of the model at the end of a particular epoch or training cycle. The first few epochs(< 100) have been removed from the plot for clarity. Looking at Figure 4.16, two primary observations can be made. Firstly, every batch size has a certain band or range wherein its predictive accuracy lies, the highest predictive accuracy being that of the lowest batch size,i.e $batch\_size = 8$. Secondly, for the lowest batch size, the updates to the error metric are very noisy, which is generally undesirable for stable training of the neural network[47]. Therefore the next best, $batch\_size = 16$ is chosen as the default batch size for building the neural network.

The final step in hyper-parameter tuning is to select the appropriate number of epochs. Using the size of the neural network shown in Figure 4.15b, and a batch size of sixteen, a plot is made to track the training and test accuracy of the neural network for a range of epoch values. This plot is shown in Figure 4.17a.

It can be seen in Figure 4.17a, that as the epochs or training cycles proceed, the error on the training and test data set decreases until a certain point, beyond which the training error continues to decrease while the test error begins to creep up. This is the result of overfitting due to the selection of a high number of epochs/-training cycles. This issue of selecting the right epoch can be resolved in two ways. One way is to apply an **Early Stopping** method on the training process. The method calls for real-time monitoring of the validation error as training proceeds. A **Patience** factor is to be defined and supplied as a keyword argument to this method, which provides a limit on the number of epochs the training can proceed without showing any improvement(reduction) on the validation loss. A generally accepted value for Patience is around 25. In this manner, the right value of the epoch parameter is selected by eliminating redundant epochs. The other way to resolve the issue of overfitting is **Dropout**[12]. Compared to Early Stopping, this is a less direct way of resolving the issue of optimum epoch selection in the sense that it targets a reduction in the validation loss. Dropout is

[11] https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/
[12] https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/



(a) Large Epoch value

(b) Effect of Dropout

(c) Effect of Dropout plus Early Stopping

Figure 4.17: Selection of optimum Epoch

often used in conjunction with Early Stopping[13]. It is a regularizing technique for neural networks. The idea is to randomly drop a certain percentage of nodes from a specific hidden layer of the neural network at each epoch. The dropped nodes lose both their input and output extensions. The fact that some of the nodes can be effectively dropped in the next epoch prevents the node units from co-adapting too much and results in the nodes taking lesser responsibility for the inputs[49] While defining the individual layers, Dropout can be added in as a factor for that particular layer. To keep things simple, each hidden layer is assigned the same dropout factor. The dropout factor is usually set between 0.1 and 0.5[50]. For the current model, a dropout factor of 0.2 works the best.

For improving the predictive accuracy of the basic neural network model shown in Figure 4.17a, both Dropout and Early Stopping methods are consecutively applied and the result is shown in Figure 4.17b and 4.17c. Through direct(Early Stopping) and indirect(Dropout) means, the number of effective epochs is reduced from 2000 to 182. Based on the above investigation, the final values of the hyper-parameters are now known. Apart from the ones discussed above, there are other parameters that are not investigated, but are also required to be supplied to the neural network model, such as the Optimizer Algorithm, Learning Rate and the Activation Function for each node of the neural network. Suitable values for these parameters are derived from literature and shown in Table 4.6. Using the hyper-parameter values in Table 4.6, the neural network model is built upon the SSBJ dataset of Table 4.5. Using EVRS as the normalized metric, the predictive accuracy of the optimized neural network model can be plotted as shown in Figure 4.18. The EVRS score is found to be 0.962 which is similar to the score obtained by the Decision Tree model(Figure 4.12c).

Type	**hyper-parameter**	**Value**
Optimized	Hidden Layer Nodes	{18, 12, 16}
	Batch Size	16
	Epochs	2000
	Dropout Rate	0.2
	Early Stopping Patience	25
Default	Optimizer	'ADAM'[1]
	Activation Function	'ReLU'[2]
	Loss Function	'MSE'
	Learning Rate	0.001

Table 4.6: Optimized hyper-parameters

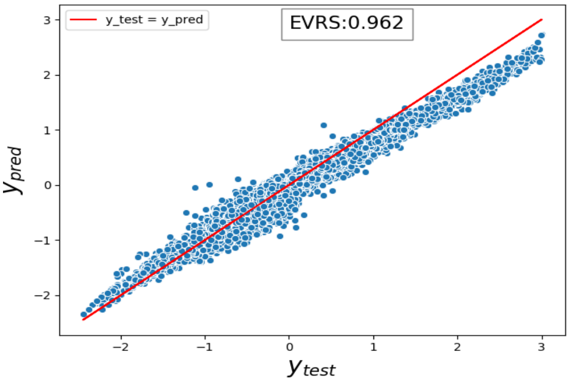


Figure 4.18: Performance evaluation of Neural Network

Selecting the most effective Algorithm

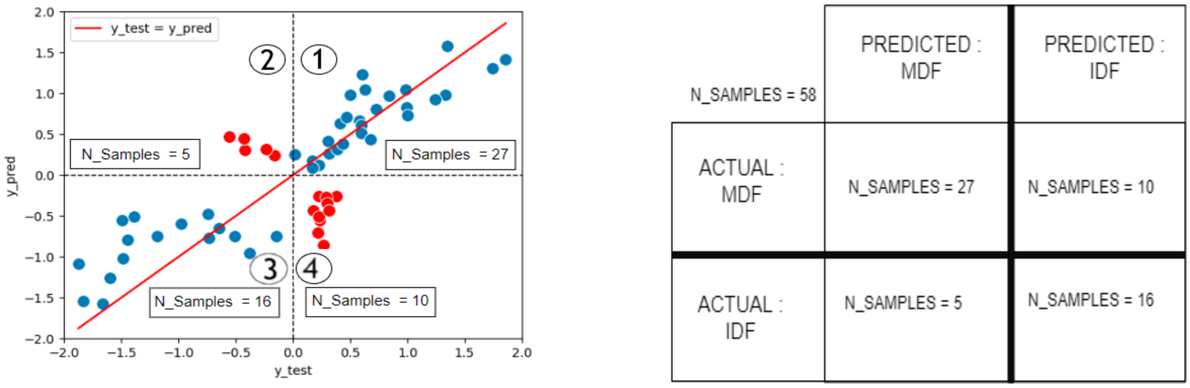
Based on the fitting process in the above section, it can be seen that in terms of EVRS, decision trees(0.967) and neural networks(0.962) have higher fitting accuracy compared to Linear Regression(0.588) and KNN(0.927). To get a better sense of the comparative effectiveness of decision trees and neural networks, the performance

[13] https://datascience.stackexchange.com/questions/30555/regularization-combine-drop-out-with-early-stopping

[2] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[1] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

of both algorithms has to be looked at from a context beyond the one provided by a regression metric such as EVRS. To explain this, a plot is made that shows a randomly generated sample distribution of points representing tested(actual) and predicted labels. This plot is shown in Figure 4.19a.



(a) Four Classes of Predicted Space

N_SAMPLES = 58	PREDICTED : MDF	PREDICTED : IDF
ACTUAL : MDF	N_SAMPLES = 27	N_SAMPLES = 10
ACTUAL : IDF	N_SAMPLES = 5	N_SAMPLES = 16

(b) Confusion Matrix

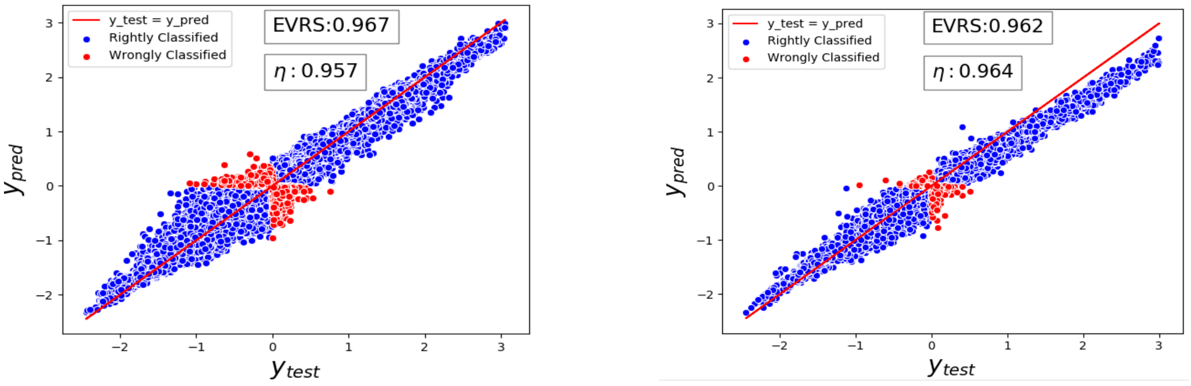
Figure 4.19: Derivation of Confusion matrix

It can be noticed from Figure 4.19a that the representative points have been divided into four quadrants that are also numbered according to the general convention. As defined earlier, the label in use for the regression analysis, i.e y_{pred} along y axis and y_{test} along x axis, is indicative of the cost ratio of optimization using MDF and IDF architectures. A positive cost ratio indicates a preference towards MDF while a negative cost ratio indicates a preference towards IDF. Therefore, the points that lie in the first and third quadrants are those that are accurately predicted by the machine learning algorithm to be having a preference towards MDF and IDF respectively. Quadrants two and four represent an incorrect classification. Based on this plot, a classification report, in the form of a confusion matrix[14] can be tabulated shown in Figure 4.19b. Based on this confusion matrix, the **classification accuracy(η)** of the algorithm used to create the Predicted space of Figure 4.19a can be calculated as follows:

$$Classification\ Accuracy(\eta) = \frac{n_{Q1} + n_{Q3}}{n_{Q1} + n_{Q2} + n_{Q3} + n_{Q4}} = \frac{27 + 16}{27 + 16 + 5 + 10} = .746 \tag{4.6}$$

where n_{Qi} represents the total number of points that fall in Quadrant i.

The classification accuracy metric derived above can be used to narrow down the more effective machine learning model when the fitting capability of the models are found to be nearly identical using a regression metric like EVRS. Using this metric, a side by side comparison of the predictive capability of neural network and decision trees is shown in Figure 4.20



(a) Classification Accuracy(Decision Tree)

(b) Classification Accuracy(Neural Network)

Figure 4.20: Comparison of Neural Network/Decision Tree

[14] https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

The EVRS metric represents the absolute prediction capability of the machine learning model(in terms of predicting the label) while the classification accuracy metric (η) is indicative of the final consequence of each prediction. The EVRS metric, in case of neural network model(0.962) is slightly lower than that of decision trees(0.967). As can be seen in Figure 4.20b, this can be partly attributed to the under-prediction provided by the neural network model for certain points where $y_{test} > 2$. However, despite this under-prediction, the final decision of the neural network based prediction model does not change because these points lie in a zone where the cost ratio is quite high(>2), which indicates a strong preference towards MDF. The classification accuracy, on the other hand, is higher for the neural network based model. Looking at Figure 4.20a, lower classification accuracy can be attributed to the higher spread of values for the red points, i.e points that have been wrongly classified. Most of these points are found in the ranges representing $y_{test} \in [-1, 1]$. From this analysis the following conclusion can be drawn:

While the prediction accuracy obtained by decision trees on the SSBJ data set is higher than neural networks, the better final outcome (of the prediction system) provided by Neural network makes it more effective than Decision Tree.

The machine learning model created using neural network is found to be most appropriate for building the prediction system. Until now, the model has been trained and tested only on the scaled SSBJ data set. In keeping with the set of objectives laid out at the beginning of Section 4.2, it is now required to test the predictive capability of the model on new MDO problems. This process is taken up in the following section.

4.2.3. Verify Prediction Model - Test MDO problems

In the previous section, a neural network based prediction model was created on a database of SSBJ based scaled MDO problems, consisting of five features and one label(Table 4.4). The algorithm was trained on the train split of the SSBJ based database and the performance was tested on the test split of the same database. In keeping with the idea of of **build, validate and test**, the earlier plots(Figure 4.20b) can be seen as **validate**. The prediction capability of the model now needs to be tested on new test problems. For this purpose, scaled versions of the following MDO problems, taken from the NASA MDO test suite[17] are introduced:

1. The propane combustion MDO Problem[17, 16]
2. Design optimization of a Speed Reducer[17, 51]
3. The Heart Dipole Problem[17],[15]

The three MDO problems mentioned above are used quite extensively throughout the field of MDO to benchmark MDO architectures[17, 16, 52, 53]. These problems primarily differ in their problem structure in terms of the presence of shared/ local variables, global/local constraints and the level of interdisciplinary coupling. Therefore, these problems resemble a challenging test for the prediction model. In this section, the neural network based predictive model, built upon the SSBJ based database, is tested out on the databases created using the above three MDO problems. Before applying the SSBJ based neural network model on the test problems and analyzing the predictive performance, it is required to discuss the individual MDO problems and the changes that are required to be made to the structure of the problem, as a pre-processing step such that the SARF based scaling process can be applied on it. The details of the test MDO problems,including the SARF-based scaling process are discussed in Appendix D

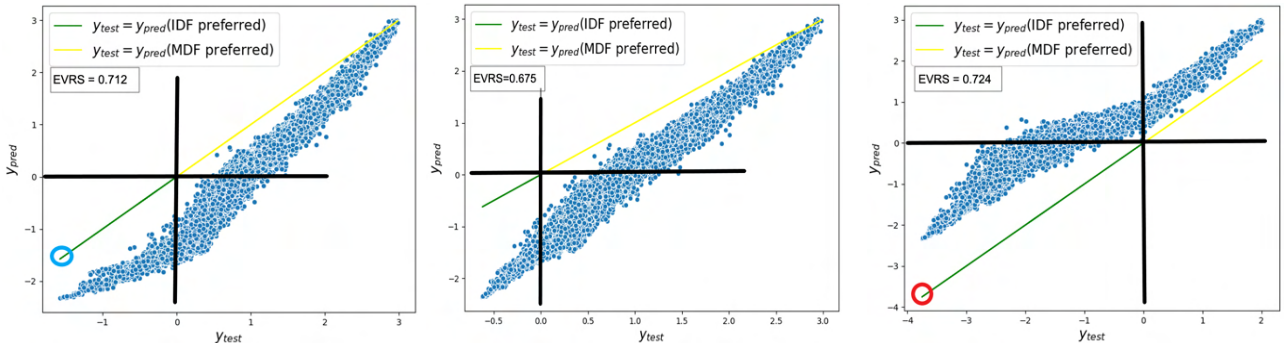
This section is used to test out the predictive performance of the SSBJ based neural network model on scaled versions of the three MDO problems discussed above. The database, for each problem, is firstly required to be created. This is done on the basis of the template shown in Table 4.7(η stands for the normalizing factor). While building the SSBJ based database, the normalizing factor η for the size related parameters(n_x, n_y, n_c) was selected to be the number of disciplines. Using the same rule, for the propane combustion problem and the speed reducer problem, the database is built by considering the value of η to be three. For the heart dipole problem, the value of η is two. Based on the range of values provided in Table 4.7, the databases for the three problems are constructed.

Parameters	**Range**
n_x	$2/\eta$ - $30/\eta$
n_y	$2/\eta$ - $30/\eta$
d	0.3 - 0.7
n_p	1/4 - 4/4
n_c	$\{2/\eta, 9/\eta, 16/\eta, 24/\eta\}$

Table 4.7: Template for test databases

[15] https://www.aere.iastate.edu/bloebaum/ii-c-1-heart-dipole-problem/

Following this, the neural network model, trained solely on the SSBJ based database, is tested on the three new problems. The results of the testing process are displayed in Figure 4.21. The analysis is made only on the basis of prediction accuracy(EVRS) and not classification accuracy(η).



(a) Predicted vs actual cost ratio (Propane Combustion)

(b) Predicted vs actual cost ratio (Speed Reducer)

(c) Predicted vs actual cost ratio (Heart Dipole)

Figure 4.21: Predictions made by SSBJ based neural network model

Figure 4.21 plots the predicted vs the actual cost ratios(R) for the three scaled MDO problems. The line ($y_{pred} = y_{test}$) represents the actual values of the cost ratio(R) for the three databases of scaled problems. The green part of the line represents the points that favour IDF($y_{test} < 0$) and the yellow portion of the line represents the points that favour MDF($y_{test} > 0$). The blue dots represent the predictions made by the neural network model trained on the SSBJ database. Looking at the predictive performance of the trained neural network model on the three databases, a significant amount of over/under-prediction can be observed. The observations from the above three plots are discussed below:

For the database generated using the scaled propane combustion problem (Figure 4.21a), the predicted value of the cost ratio is lower across the majority of the values tested. The inaccuracy in this prediction is reflected by an EVRS score of 0.712. The neural network model predicts the cost ratio to be consistently lower than the actual value. Trained solely on the SSBJ based database, the model wrongly estimates the scaled propane combustion problem to give faster solutions when executed using the IDF architecture. Additionally, looking at the left endpoint of the line $y_{pred} = y_{test}$, the extreme negative value of the actual cost ratio is only about -1.45(circled in blue), which indicates that even in the best-case scenario, IDF has a lesser advantage over MDF for the scaled propane combustion problem, as compared to the scaled SSBJ problem. The prediction is found to be fairly accurate only at the extreme left end of the yellow line, which indicates that for a very small percentage of data points representing a distinct advantage for MDF($y_{test} \sim 3$), the cost ratio is accurately predicted by the neural network model.

The predictions made on the scaled speed reducer problem (Figure 4.21b) shows an extension of the trends seen in Figure 4.21a. Looking at the actual cost ratios represented by the line $y_{pred} = y_{test}$, the green portion of the line is comparatively much smaller than the yellow portion, which indicates that a very small percentage of points are found to be representing a negative cost ratio(and therefore preferring IDF). This leads to a significant under-prediction made by the neural model. Compared to the scaled propane combustion problem, a higher level of under-prediction can be observed between the predicted and actual cost ratios for $y_{test} < 0$, which is reflected in the lower EVRS score of 0.675.

The predictions observed for the scaled heart dipole problem (Figure 4.21c) are opposite to the previous two problems. While the EVRS score of 0.724 is higher than the other two plots, the SSBJ based neural network model makes an over-prediction on the cost ratio across the entire range of y_{test} values. This implies that the model inaccurately predicts the scaled heart dipole problem, across all data points, to be optimized faster using the MDF architecture. This over-prediction is more pronounced for the data points on the left side of the plot. Additionally, looking at the left end of the line $y_{pred} = y_{test}$, that represents the true values of the cost ratios for the scaled heart dipole problem, the extreme negative value of y_{test} is about -3.8(circled in red), which represents an overwhelming preference towards IDF. The SSBJ based neural network model is unable to predict such extreme values.

Based on the above observations, it can be concluded that the cost ratios for the scaled SSBJ problem do not align alongside the test problems. The misalignment is emphasized by the extreme values of the cost ratios used to train the neural network model. For the SSBJ based database, the lowest and highest values of the cost ratios are 2.98 and -2.35 respectively. However, for the other three problems, the actual lower and upper bounds are different. For instance, in the heart dipole problem, the lower bound on the actual cost ratios is -3.8. This is an extreme point, a label value that lies outside the range of label values used to train the prediction model. This label cannot be predicted by the SSBJ based neural network, unless it is retrained on a data point with such a label value. Additionally, there is a varying degree of misalignment between the neural network based prediction and the actual cost ratio values when going from one scaled problem to the other. This is apparent from the EVRS factors that are calculated for the three plots. In terms of compatibility, the scaled speed reducer problem seems to be furthest away from the SSBJ problem, followed by the scaled propane combustion problem and the scaled heart dipole problem. This indicates that the existing set of features is not enough to fully define a scaled MDO problem. There must be an internal feature, not used yet for the training process, which leads to the existing incompatibility.

The process of resolving the above discussed incompatibilities can be looked at from the context of machine learning as well as MDO. This is discussed in the next section.

4.2.4. Feature Engineering - Resolve Incompatibility and Retrain model

The objective is to create a machine learning model that has high and consistent accuracy of prediction across multiple scaled MDO problems. It must be noted that the same template has been followed to build the training database(using scaled SSBJ MDO problem) and the test databases(using scaled versions of the three test MDO problems). The values of five normalized features that are entered within the database are the same for each MDO problem. In such a case it is not possible to simply re-train the existing neural network model by appending a certain fraction of the test databases to the SSBJ based database and allowing the neural network to train on the new data points. This leads to multi-collinearity, which indicates the presence of duplicate entries within the database with same feature values but a different label. An example of this is shown below:

Source	n_x	n_y	d	n_p	n_c	Label
SSBJ	2/3	2/3	.3	2/3	2/3	-2.34
Propane Combustion	2/3	2/3	.3	2/3	2/3	-1.45

Table 4.8: Collinearity due to same feature - different label

Therefore, to make the databases compatible with each other, it is required to introduce a separate differentiating feature within each database. This is the point where the background knowledge of MDO, scalable problems and the SARF methodology has to be applied to define a factor that can, in a meaningful way, account for the abnormalities observed in the predictive plots of Figure 4.21a to 4.21c. The following subsections are dedicated to this process.

Derive the Differentiating Feature - Coupling Strength

Within the Literature Review, it was seen that the existing literature fails to propose an all-encompassing scalable problem that would allow the user to independently alter features related to both problem size and coupling strength(Section 2.3.1 and Section 2.3.2). The same holds true for the SARF based scaling method. However, despite not proposing a method(parameter) to alter the coupling strength of the problem, it was (indirectly) claimed in the paper by Vanaret et.al.[16] that the SARF scaling method preserves the coupling strength between the original and scaled MDO problems. A possible explanation for this was given while deriving the dependency matrix in the previous chapter(Chapter 3, Section 3.1.3.1), where it was claimed that the value of a parameter depending on only the interdisciplinary sensitivities does not change while going from original to the SARF based scaled problem. If a numerical estimate of the coupling strength for a particular problem can be obtained, then it can be used to create a differentiating feature to justify the incompatibility between the SSBJ based database and the database of the three test MDO problems. The feature can be added as an additional column, with each database receiving a unique value of the feature. This should resolve the issue of multi-collinearity and pave the way for creating a combined database of multiple MDO problems, which can then be used to train a more efficient neural network model that can differentiate between MDO problems of varying coupling strength. The process of obtaining a numerical estimate of the coupling strength can be understood from some of the previous works in this area. The existing literature[16, 8, 27, 5, 54] is searched to look for a generic method, that can be used to estimate the coupling strength for any given MDO problem. One such method is derived using the research performed by Chauhan et. al.[27] as discussed in the following subsection.

Numerical Estimate of Coupling Strength

For studying the relationship between local sensitivities and convergence criteria of MDA based approaches, Chauhan et.al[8] looked into linear block Gauss-Seidel convergence criteria. For a linearized multi disciplinary system comprising of n components, it is possible to create an equation representing the updated vector of couplings at the end of the k^{th} block Gauss–Seidel iteration as a function of the coupling values at $(k-1)^{th}$ iteration and the local sensitivities of couplings with respect to each other[8 *eq.15*]. This equation is reworked by Chauhan et. al.[8] to form the following equation:

$$\begin{bmatrix} \Delta v^{(1)} \\ \Delta v^{(2)} \\ \vdots \\ \Delta v^{(\eta)} \end{bmatrix}_{k+1} = \underbrace{\begin{bmatrix} I & 0 & \cdots & 0 \\ -\frac{\partial v^{(2)}}{\partial v^{(1)}} & I & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ -\frac{\partial v^{(\eta)}}{\partial v^{(1)}} & \cdots & -\frac{\partial v^{(\eta)}}{\partial v^{(\eta-1)}} & I \end{bmatrix}^{-1} \begin{bmatrix} 0 & \frac{\partial v^{(1)}}{\partial v^{(2)}} & \cdots & \frac{\partial v^{(1)}}{\partial v^{(\eta)}} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \frac{\partial v^{(\eta-1)}}{\partial v^{(\eta)}} \\ 0 & 0 & \cdots & 0 \end{bmatrix}}_{G} \begin{bmatrix} \Delta v^{(1)} \\ \Delta v^{(2)} \\ \vdots \\ \Delta v^{(\eta)} \end{bmatrix}_{k} \tag{4.7}$$

where $\frac{\Delta v^x}{\Delta v^y}$ refers to the local sensitivity of the coupling v^y with respect to v^x.

The above matrix equation directly relates the vector representing the change in coupling output values from one Gauss-Seidel iteration to another through a transformation matrix G, referred to as the iteration matrix. This is the multiplier that allows the linear Gauss-Seidel process to arrive at the next iteration of Δv values. The above matrix equation(Equation.4.7), is in the form of a standard set of linear equation $b = Ax$ where A represents the Iteration matrix, while b and x represent the δv values for the k^{th} and $(k+1)^{th}$ iterations respectively. Given this form, two conclusions can be made from Equation 4.7:

1. By computing the spectral radius[16] of this iteration matrix G, it is possible to get an idea of the rate of convergence of the block gauss-seidel iterative process for a particular iteration[8, 55]. The smaller the spectral radius, the lesser the coupling strength, and hence faster the convergence of the linearized MDA system around the iteration point[8].

2. In the absence of a relaxation strategy(like Aitkens relaxation), the spectral radius of the iteration matrix G should be smaller than one for moving towards a better solution while computing Δv^{k+1} from Δv^k[8].

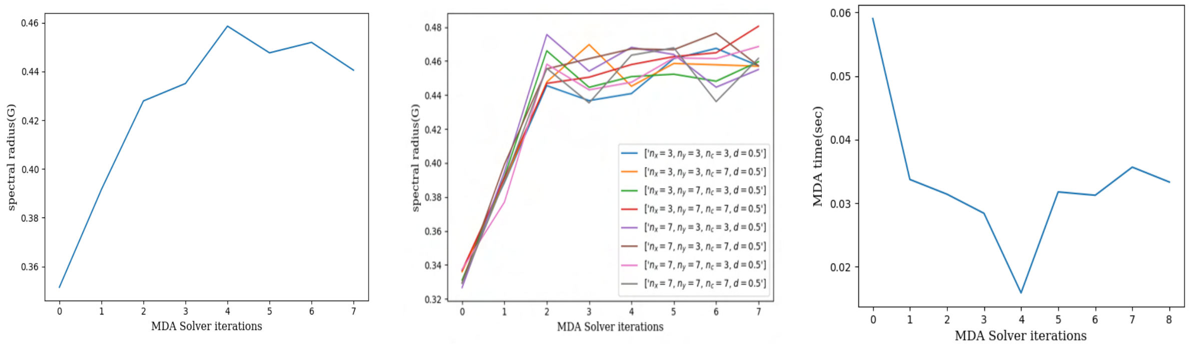
The above two points are useful in their own way. The first gives a reason to link a certain quantifiable parameter such as the spectral radius with the coupling strength of an MDO problem around an iteration point. The second point gives the possible range of coupling strength values that can be used to normalize the coupling strength parameter. Now, within the OpenMDAO environment, there is no direct option to compute the iteration matrix of Equation 4.7, as OpenMDAO executes linear block gauss-seidel iterations in a matrix-free environment[17]. A workaround method is to use OpenMDAO's Direct Solver as the choice of linear solver in place of Gauss-Seidel. As opposed to an iterative solver, Direct solvers make use of the Jacobian matrix, assembled in memory, for computing an inverse or a factorization that can be used to solve the linear system[18]. The Jacobian can be calculated for each solver iteration within the memory by converging the $DirectSolver$ MDA routine once for the MDO problem. The Jacobian matrix is then separated into lower and upper matrices to create the iteration matrix G(with the lower matrix containing the diagonal blocks). The spectral radius of the iteration matrix G is obtained by computing its largest absolute eigenvalue. This process does not require the execution of any MDO optimizer iterations. Therefore, coupling strength, when computed with the above mentioned method, can still be termed as a pre-execution feature, which can be

[16] https://mathworld.wolfram.com/SpectralRadius.html

[17] https://stackoverflow.com/questions/61559186/how-to-compute-the-iteration-matrix-for-nth-nlbgs-iteration/6159963761599637

[18] http://openmdao.org/twodocs/versions/latest/theory_manual/total_derivs/setup_linear_solvers.html

calculated for any MDO problem, without running optimizer iterations.
The above discussed methodology is implemented in the unscaled SSBJ problem. By placing a pickled counter within the $\_linearize$ method present in OpenMDAO's $DirectSolver$ class, the spectral radius of the iteration matrix(G) is calculated and stored for each iteration. Figure 4.22a shows the plot of the spectral radius vs each Solver iteration for the unscaled SSBJ problem.



(a) Spectral Radius vs Direct Solver Iterations(original problem)

(b) Spectral radius vs Direct Solver Iterations(scaled problems)

(c) Wall time vs Direct Solver Iterations(original problem)

Figure 4.22: Spectral radius of iteration matrix and wall time vs solver iterations(Original and scaled SSBJ problem)

The original SSBJ problem, when solved using the MDF-Direct Solver scheme, completes in nine iterations. Upon scaling the original SSBJ problem using a range of parameters, the coupling strength(ρ) at each direct solver iteration is recalculated and plotted as shown in Figure 4.22b. The corresponding values of the spectral radius is in the same ballpark as that of the original SSBJ problem(Figure 4.22a). This confirms the claim made by Vanaret et. al.[16] that the coupling strength between disciplines does not get altered from the scaling process.
For estimating the coupling strength(ρ) of the entire problem, a weighted mean based procedure is adopted that takes into consideration the amount of time taken by the solver to converge a particular MDA iteration. A time-averaged method to calculate the coupling strength for a particular MDO problem is proposed as shown below:

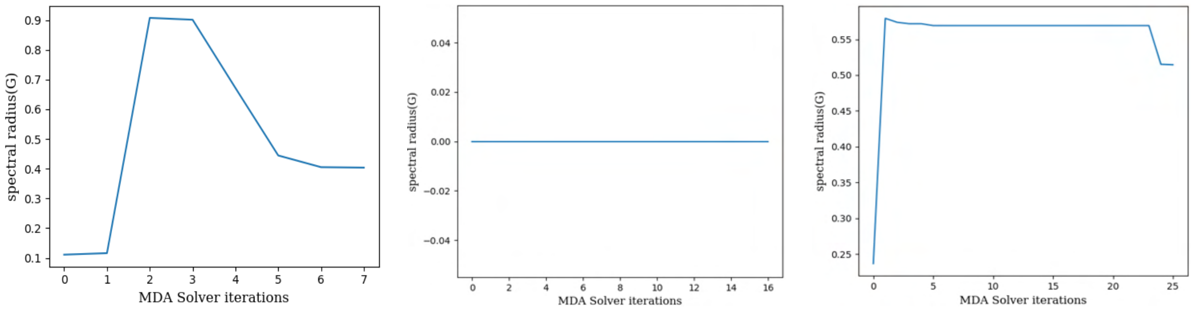
$$\rho = \frac{\sum t_i * s_i}{\sum t_i} \tag{4.8}$$

where t_i and s_i are the time and spectral radius values recorded at each iteration. For calculating the above derived ρ value the total time consumed in each iteration(for the original SSBJ problem) is also measured and plotted as shown in Figure 4.22c.

The above equation is justified using the following points:

1. Instead of considering just the spectral radius at convergence, it is required to take into account the spectral radius obtained at individual solver iterations such that the overall effort spent in resolving the interdisciplinary couplings can be taken into account.
2. The spectral radius represents the rate of convergence, based on the iteration matrix that is assembled at the beginning of each $DirectSolver$ iteration. However, each solver iteration requires a certain amount of time as shown in Figure 4.22c. The factor t_i is used as weights to account for the time spent in each $DirectSolver$ iteration.

Using the above mentioned process, the coupling strength(ρ) of the unscaled SSBJ problem is found to be 0.422 while that of the scaled SSBJ problem is found to be in range of 0.362 - 0.483. Upon inspection it can be seen that there are a few outliers within this range and the mean value of the scaled SSBJ problem is 0.432 which is close to the coupling strength of the unscaled problem. The above process is repeated for the three test problems and the results are shown in Figure 4.23 and 4.24.

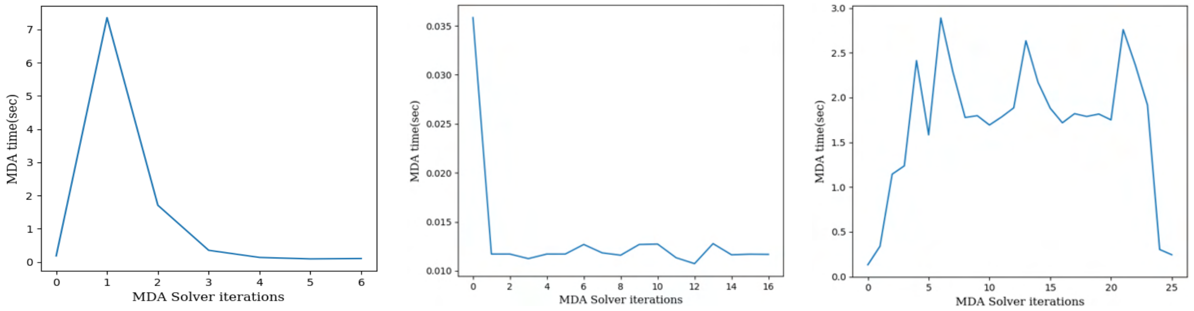


(a) Propane Combustion Problem

(b) Speed Reducer Problem

(c) Heart Dipole Problem

Figure 4.23: Spectral Radius vs Direct Solver Iterations



(a) Propane Combustion Problem

(b) Speed Reducer Problem

(c) Heart Dipole Problem

Figure 4.24: Spectral Radius vs Direct Solver Iterations

Based on the time-averaged method shown in Equation 4.8, a numerical estimate is obtained about the coupling strength of the SSBJ test problem as well as the three test problems and shown in Table 4.9. For the speed reducer problem , the spectral radius of the iteration matrix stays constant at zero throughout the direct solver iterations(Figure 4.23b), thereby the coupling strength also becomes zero. This can be attributed to the fact that the speed reducer problem does not involve any bi-directional couplings and relies only on unidirectional flow of information, from the first three disciplines to the fourth discipline. This can be seen from the XDSM representation(using MDF) for the speed reducer problem shown in Figure D.6. The Propane combustion problem represents a moderate level of coupling strength, lower than that of the SSBJ based problem, while the heart dipole problem shows the highest coupling strength. For verifying the obtained values of ρ, some empirical evidence can be sought. This is discussed in the next subsection.

Problem	**Coupling Strength** (ρ)
Speed Reducer	0
Propane Combustion	0.32
SSBJ	0.422
Heart Dipole	0.568

Table 4.9: Strength of Coupling(ρ) (test problems)

Verify Coupling Strength Estimation - Empirical Evidence

In a purely empirical approach, it is required to look into the sources that comment on the degree/strength of coupling of the MDO problems that have been used in this investigation. Considering the three test problems transcribed from the NASA MDO test suite, a portal can be looked at, that was created by the University of Buffalo, State of New York[19]. The portal shows a classification of the MDO problems on the basis of coupling strength, problem size and hierarchy. It uses three classes A, B and C to represent three levels of interdisciplinary couplings, namely, low, $medium$ and $high$ respectively. With regard to the MDO problems that have been implemented in this thesis, Table 4.10 shows the class assigned to each test problem. The Propane combustion problem and the Speed Reducer problem are both assigned a $medium$ coupling strength, which does not help create a differentiating factor between the two problems. The heart dipole problem has a $high$ coupling strength. A second empirical

Problem	**Coupling Strength**
Propane Combustion	medium
Speed Reducer	medium
Heart Dipole	high

Table 4.10: Strength of Coupling for test problems

[19] http://www.eng.buffalo.edu/Research/MODEL/mdotestsuite.html

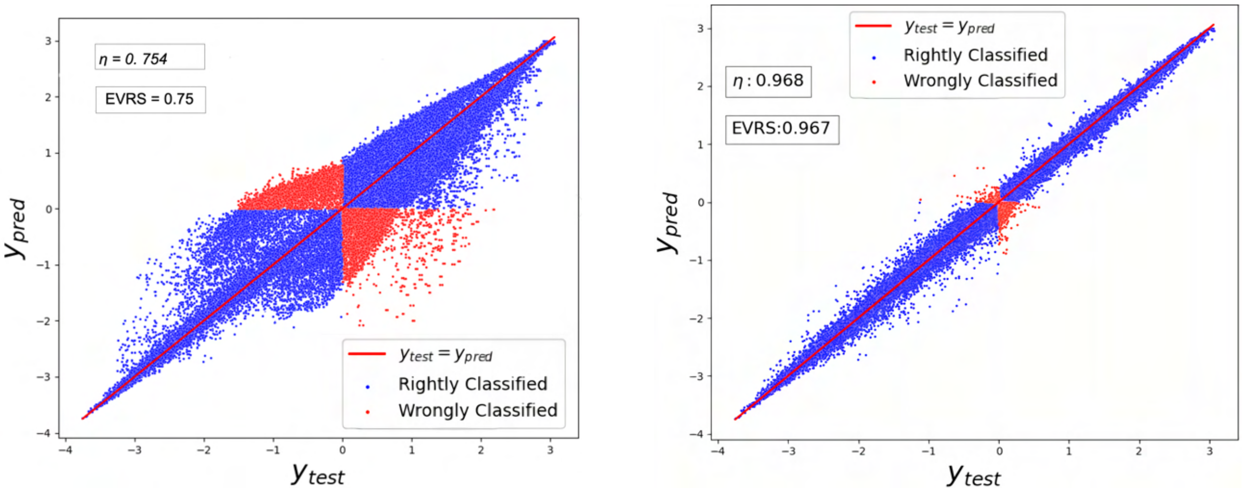
source can be utilized here, from the paper by Vanaret et. al.[16], which states that the SSBJ problem consists of strongly coupled disciplines while the propane combustion problem consists of weakly coupled analytical disciplines[16]. The information obtained from these sources aligns with the trends observed in Table 4.9. In an indirect empirical approach, the prediction plots made using the SSBJ based neural network(Figure 4.21) can be revisited from the context of the coupling strength(ρ) estimations. The scaled Propane Combustion problem(Figure 4.21a) as well as the scaled Speed Reducer problem(Figure 4.21b) seem to show a lower preference towards the IDF architecture, while the scaled heart dipole problem(Figure 4.21c) indicates a greater preference towards IDF, for all the tested data points. Now, based on the existing, independent research regarding the impact of coupling strength[5, 6], it can be said that, as the coupling strength increases, the preferred architecture tends to shift from MDF to IDF. Assuming the prediction plots made using the SSBJ based neural network(Figure 4.21) to be correct, the existing studies([5, 6]) seem to validate the ranking of coupling strength estimations($\rho_{SpeedReducer} < \rho_{PropaneCombustion} < \rho_{SSBJ} < \rho_{HeartDipole}$)

Retrain/Verify model based on Combined Database

Having sorted the appropriate value of coupling strength for each problem, a separate feature column representing the coupling strength ρ can be defined and added to the four databases of scaled problems. Following this, the four databases can be individually appended to create a consistent database for re-training the neural network. The resultant database is shown in Table 4.11. Based on the combined database shown in Table 4.11, the neural network is re-trained, using the same procedure as before. A train test split of 1:2 is made on the combined database, along with the same internal parameters for fitting the neural network model as the ones used previously(Table 4.6) and an improved or "re-trained" neural network is built on the train split of the combined database. The re-trained neural network is tested on the test split of the combined database and the improvement in the prediction is shown in Figure 4.25.

Source	$\mathbf{n_x}$	$\mathbf{n_y}$	**d**	$\mathbf{n_p}$	$\mathbf{n_c}$	$\boldsymbol{\rho}$	**Label**
Speed Reducer	2/3	2/3	0.3	2/3	2/3		-0.6
	3/3	2/3	.3	2/3	2/3		-0.58
	:	:	:	:	:	0.0	:
	30/3	30/3	0.7	30/3	24/3		2.95
Propane Combustion	2/3	2/3	0.3	2/3	2/3	0.311	-1.6
	3/3	2/3	.3	2/3	2/3	0.309	-1.57
	:	:	:	:	:		:
	:	:	:	:	:		:
	30/3	30/3	0.7	30/3	24/3	0.347	2.88
SSBJ	2/3	2/3	0.3	2/3	2/3	0.412	-2.35
	3/3	2/3	.3	2/3	2/3	0.414	-2.32
	:	:	:	:	:		:
	30/3	30/3	0.7	30/3	24/3	0.443	3.0
Heart Dipole	2/2	2/2	0.3	2/2	2/2	0.561	-3.82
	3/2	2/2	.3	2/2	2/2	0.559	-3.8
	:	:	:	:	:		:
	30/2	30/2	0.7	30/2	24/2	0.591	1.88

Table 4.11: Combined database of four problems(coupling strength ρ included)



(a) Predictive Performance of original neural network (trained without coupling strength feature)

(b) Predictive Performance of re-trained neural network (trained with coupling strength feature)

Figure 4.25: Predictive performance of neural network on combined database

Figure 4.25a shows the prediction on the combined database made by the neural network model, trained only on the SSBJ problem, without the inclusion of coupling strength(ρ) as a differentiating feature. The plot can be seen as the case with multi-collinearities(same feature combinations but different label) that was discussed in Section 4.2.4. Since the neural network model does not run when there are multi-collinearities in a test database, the plot in Figure 4.25a is constructed by applying the SSBJ based neural network model individually on the four problems(excluding the coupling strength feature) and superimposing the four plots on top of each other. The EVRS value of 0.75 is calculated manually by concatenating the four individual y_{pred} and y_{test} arrays. Figure 4.25b shows the predictive performance of the re-trained neural network model, trained on the train split of the combined database, with the inclusion of coupling strength as a differentiating feature. The difference in the EVRS values in the above two plots(0.968 vs 0.754) shows the improvement obtained in the prediction ability due to the inclusion of the coupling strength feature. Therefore, the feature gives the neural network based model the ability to differentiate between MDO problems based on the coupling strength. This improves the prediction capability of the retrained model. The drawback of the retrained model is that it requires the convergence of a single MDA routine with the $DirectSolver$ iteration scheme.
It must also be noted that Figure 4.25a is the ultimate conclusion for the SSBJ based neural network model because a prediction model trained on the SSBJ based model is found to be inadequate when tested on a different set of MDO problems. Figure 4.25b shows the increment in accuracy achieved by the retrained neural network, but it is not the final conclusion on the prediction capability of the retrained neural network. Figure 4.25b is more of a validation because the training and test sets for the retrained neural network are both extracted from the same combined database of Table 4.11. The retrained neural network is yet to be tested on unseen MDO problems. This is taken up in the next chapter. The following section provides a summary to this chapter and briefly explains the road ahead.

4.3. Prediction Model - Summary

In this chapter, a prediction model of MDO architecture was built by combining the idea of machine learning and scalable problems. The well known SSBJ test problem was selected as the starting point for implementing the SARF methodology. The scaled SSBJ problem, constructed by varying five parameters$\{n_x, n_y, n_c, d, n_p\}$, was first visually analyzed to look for meaningful contribution from each parameter. Following that, a feature set was defined to build a normalized database out of the scaled SSBJ problem. A number of machine learning algorithms were applied on the SSBJ based database. By taking into account two metrics related to prediction accuracy, the most accurate algorithm was found to be a neural network of a certain size. Once the neural network based prediction model was created, it had to be tested on new unseen MDO cases, for which three MDO test problems from the NASA MDO test suite[17] were introduced and their scaling methodology was explained(Appendix D). These problems were scaled according to the SARF methodology to create three test databases. Upon applying the SSBJ based neural network model on the test databases, some over/under-prediction was observed. At this stage, feature engineering was applied, based on existing literature, to define and compute a differentiating feature called coupling strength(ρ). A unique value of coupling strength could be assigned to each MDO problem. This allowed the individual databases to be compatible with each other, so that they could be vertically appended to create a combined database. The downside to coupling strength calculation was that it required the convergence of a single MDA routine with the $DirectSolver$ MDA scheme, for any version of the problem(original/scaled). A neural network based model was then retrained on the combined database. The retrained model(Figure 4.25b), when tested onto itself(using a train test split), was found to be having a much higher accuracy(EVRS = 0.967 vs 0.75) than the original model(Figure 4.25a) which did not take into account the effect of coupling strength.

The re-trained model had high accuracy when tested onto the same problems on which it was built, but it still needs to be verified on unseen MDO problems, i.e problems that have not been introduced to the neural network model. This process is taken up in the final chapter, where new MDO problems, containing a mix of mathematical and physical disciplines, are introduced and tested by the neural network based re-trained prediction model, followed by conclusions on the usefulness of such a prediction model.

5

Testing and Deployment of the Prediction Model

Based on the investigative study performed in the previous chapter, two prediction models were framed, an original/"vanilla" model(based on SSBJ based scaled problem) that used five parameters and an improved/re-trained model(based on scaled versions of four MDO problems) that used six parameters. This process is summarized in the diagram below:

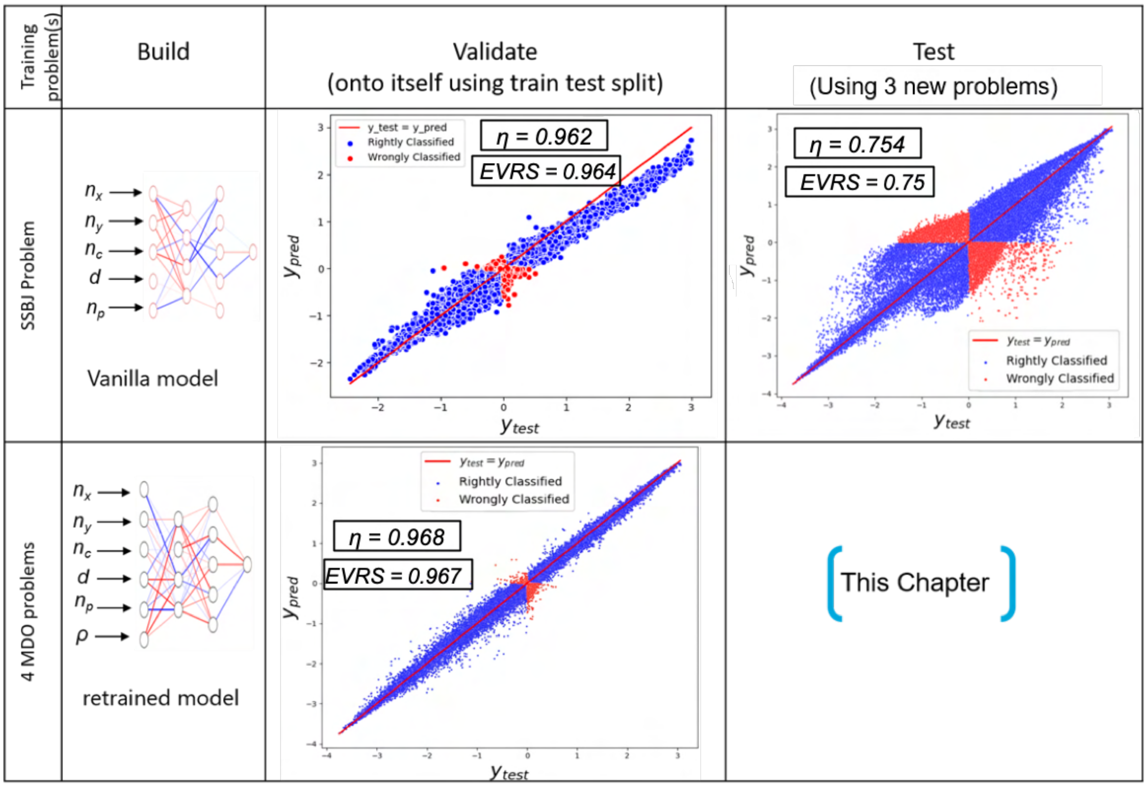


Figure 5.1: original("vanilla")/ re-trained model

As can be seen in Figure 5.1, the SSBJ based vanilla model was not a success. It had high accuracy when validated using train test split, but very low accuracy when tested with new problems. The re-trained model was constructed by considering the test problems for the SSBJ based model as training cases. The re-trained model had high validation accuracy. It is now required to see whether the re-trained model can be used to create a prediction on a previously unseen MDO problem.

This chapter is divided into four sections. The first section deals with testing the prediction model on scaled versions of new MDO problems. For this purpose, two new MDO problems are introduced, namely, an aircraft based MDO problem that makes use of physical disciplines and the sellar MDO problem [1]. Based on testing the prediction model on the new problems, an instance of an **MDO Advisory System** is created(in form of an input-output model / binary decision tree) and its application is shown using data points. Using the outcome of these test problems, the empty block in the above figure can be filled.

Considering the re-trained model to be the final prediction model created in this thesis, it is also required to see whether the prediction model can be used to make predictions on original, unscaled MDO problems. In the second section, the original versions of all six MDO problems are drafted into the mix(four old and two new) and the possibility of repeating the prediction on unscaled MDO problems is analyzed. This is a qualitative analysis, based on a certain rule-based extraction of features from the original MDO problems.

In the third section, a user's manual is provided for deploying the prediction model. In this section, all the pre-processing steps and computations that are required to use the prediction model on a new, unscaled MDO problem are provided sequentially.

In the final section, a summary is provided for the chapter.

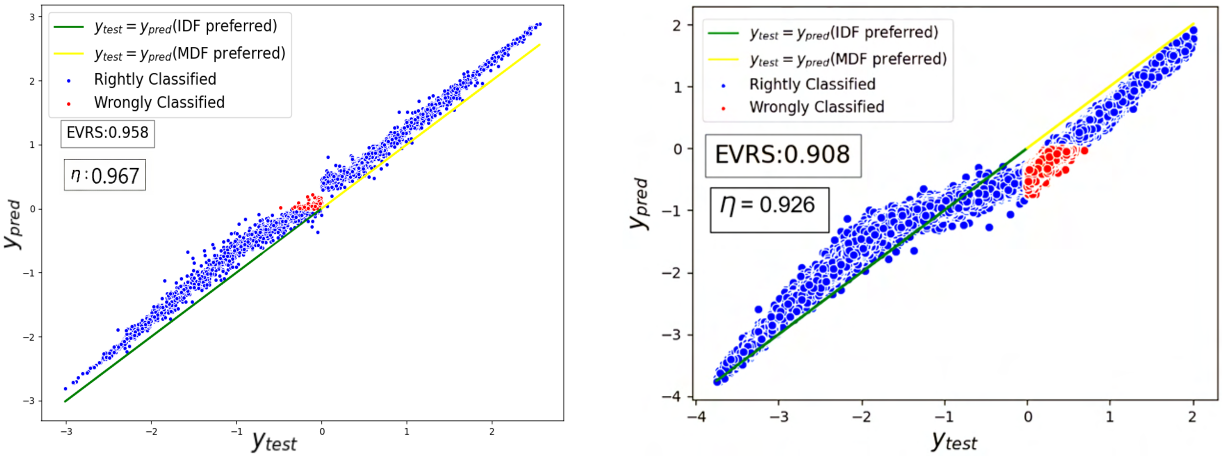
5.1. Testing Scaled Problems on Re-Trained Neural Network

Two MDO problems are used for testing the re-trained neural network based prediction model. The first problem to be tested is a MATLAB based three-dimensional wing optimization MDO problem consisting of analytical disciplines. The problem is taken from the course titled *AE4205 MDO for Aerospace Applications*. It is an aero structural optimization/ fuel minimization problem that makes use of two analytical tools, an aerodynamic solver called Q3D and a structural sizing tool called EMWET. The second problem is the Sellar MDO problem. The Sellar problem is a common MDO problem used as a means of understanding coupled models. It comprises of two coupled disciplines, each described by a single explicit equation. Like the test problems used in the previous chapter, the scaling process for the two test problems is moved to appendix E.

Based on the scaling process, a database is built for the scaled Fuel Minimization problem and the scaled Sellar problem, using the previously established template. An excerpt from the database is shown in Table 5.1. The current database can be used as a test case for the re-trained neural network based prediction model. The re-trained neural network is applied on the test database of Table 5.1 and the results are shown in Figure 5.2

Source	**n_x**	**n_y**	**d**	**n_p**	**n_c**	ρ	**Label**
Fuel Minimization	2/3	2/3	0.3	2/3	2/3	0.484	-2.42
	3/3	2/3	0.3	2/3	2/3	0.486	-2.39
	4/3	3/3	0.3	2/3	2/3	0.486	-2.38
Sellar	2/2	2/2	0.3	2/2	2/2	.562	-2.42
	3/2	2/2	0.3	2/2	2/2		-2.39
	4/2	3/2	0.3	2/2	2/2		-2.38

Table 5.1: Database for Scaled fuel minimization problem



(a) Test vs Predicted values(Fuel Minimization problem)

(b) Test vs Predicted values(Sellar Problem)

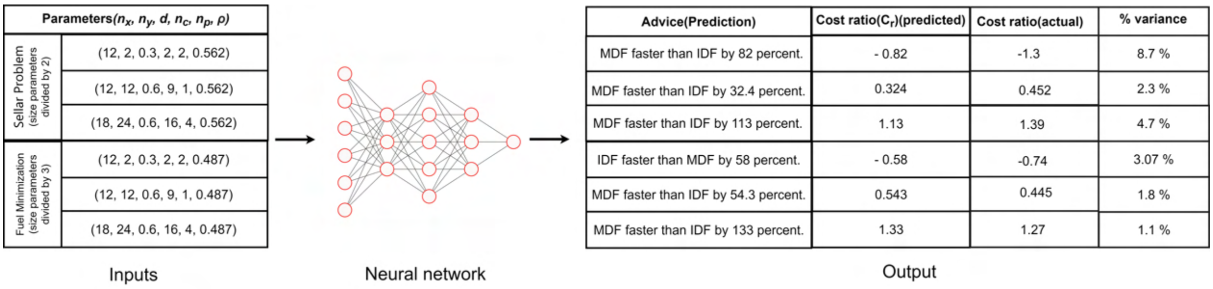
Figure 5.2: Performance evaluation of re-trained prediction model on scaled problems

[1] http://openmdao.org/twodocs/versions/latest/basic_guide/first_mdao.html

When applying the re-trained model on the fuel minimization problem, the prediction is found to be highly accurate with an EVRS score of 0.958. The classification accuracy is also quite high at 0.967, which indicates that 96.7 % of the points are accurately predicted to be faster with a particular architecture. These values are very similar to prediction accuracy obtained when the re-trained model is validated using a train test split ratio. When applying the re-trained model on the Sellar problem, the prediction accuracy is slightly lower with an EVRS and η values of 0.908 and 0.926 respectively. The outcome is still better than the SSBJ based prediction model where the prediction accuracy(EVRS) as well as the classification accuracy with new problems was around 75%.

5.2. Advisory Systems for Scaled Problems

Based on the above two test cases, it can be said that the re-trained prediction model, that has been trained on the pre-execution features offered by the SARF methodology as well as a coupling strength based feature, offers an accurate prediction on the outcome of the new test problems. At this stage, a sample of an advisory system for the scaled test problems can be constructed as shown in Figure 5.3.



	Parameters(n_x, n_y, d, n_c, n_p, ρ)
Sellar Problem (size parameters divided by 2)	(12, 2, 0.3, 2, 2, 0.562)
	(12, 12, 0.6, 9, 1, 0.562)
	(18, 24, 0.6, 16, 4, 0.562)
Fuel Minimization (size parameters divided by 3)	(12, 2, 0.3, 2, 2, 0.487)
	(12, 12, 0.6, 9, 1, 0.487)
	(18, 24, 0.6, 16, 4, 0.487)

Advice(Prediction)	Cost ratio(C_r)(predicted)	Cost ratio(actual)	% variance
MDF faster than IDF by 82 percent.	- 0.82	-1.3	8.7 %
MDF faster than IDF by 32.4 percent.	0.324	0.452	2.3 %
MDF faster than IDF by 113 percent.	1.13	1.39	4.7 %
IDF faster than MDF by 58 percent.	- 0.58	-0.74	3.07 %
MDF faster than IDF by 54.3 percent.	0.543	0.445	1.8 %
MDF faster than IDF by 133 percent.	1.33	1.27	1.1 %

Figure 5.3: Example of Advisory system for scaled test problems

Figure 5.3 shows an example of the advisory system, built using the re-trained neural network, in the form of an input-output interface. Two sets of inputs are considered, from the Fuel Minimization problem and the Sellar problem. The advice is generated from the predicted cost ratios. The percentage variance between the predicted and actual cost ratios is calculated by considering the overall range of cost ratios that can be predicted by the neural network model. It can be observed that the percentage variance is in single digits which makes the advisory system accurate.

Figure 5.3 shows the outcome of the neural network based prediction model using some examples, in the form of an input-output based interface. The neural network is like a closed box, which gives a certain value of cost ratio for a particular input. The possibility of a secondary representation for the advisory system can also be investigated, in the form of a graph, which could enable a user to get more insight on the generated advice, such as the parameter which has the maximum influence on the cost ratio. Additionally, through an advisory system in the form of an easy to read graph, advice could be generated on any scaled MDO problem without initializing the trained neural network model within scikit-learn. For this purpose, a decision tree based prediction model is created. By considering the same combined database(Table 4.11) as the one used to train the neural network model, a decision tree algorithm(Section C.8) can be applied to create a decision tree based prediction model as shown in Figure 5.4.

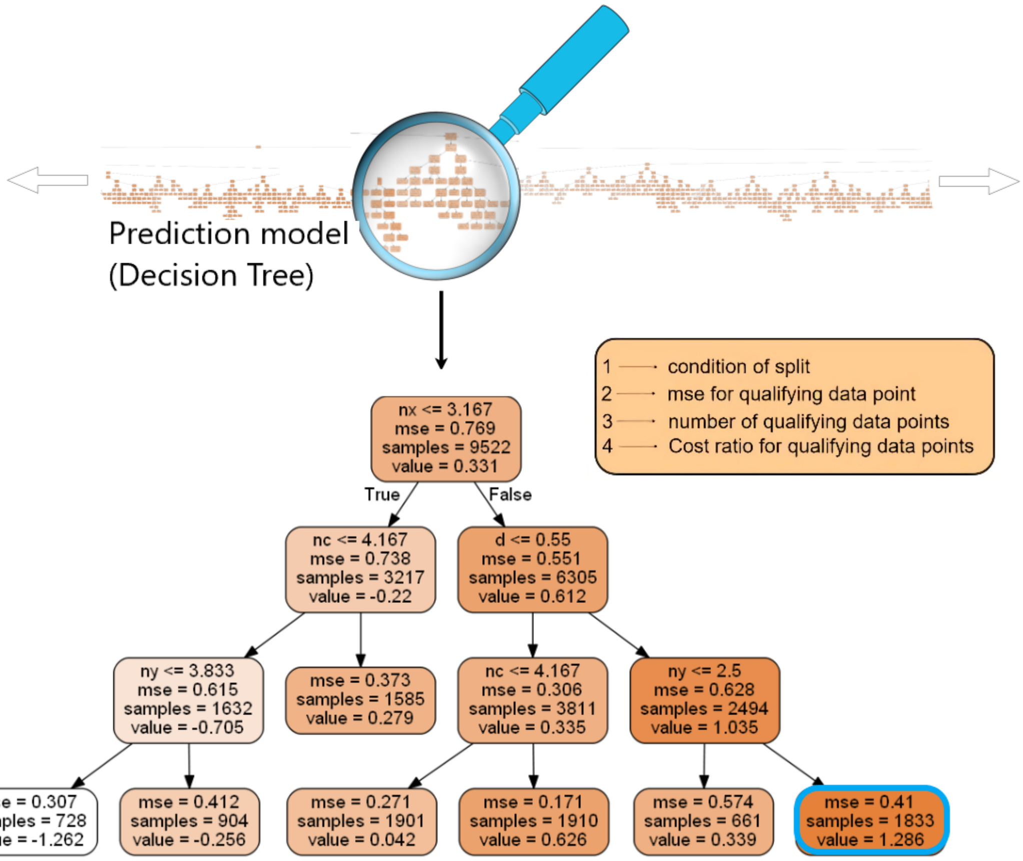
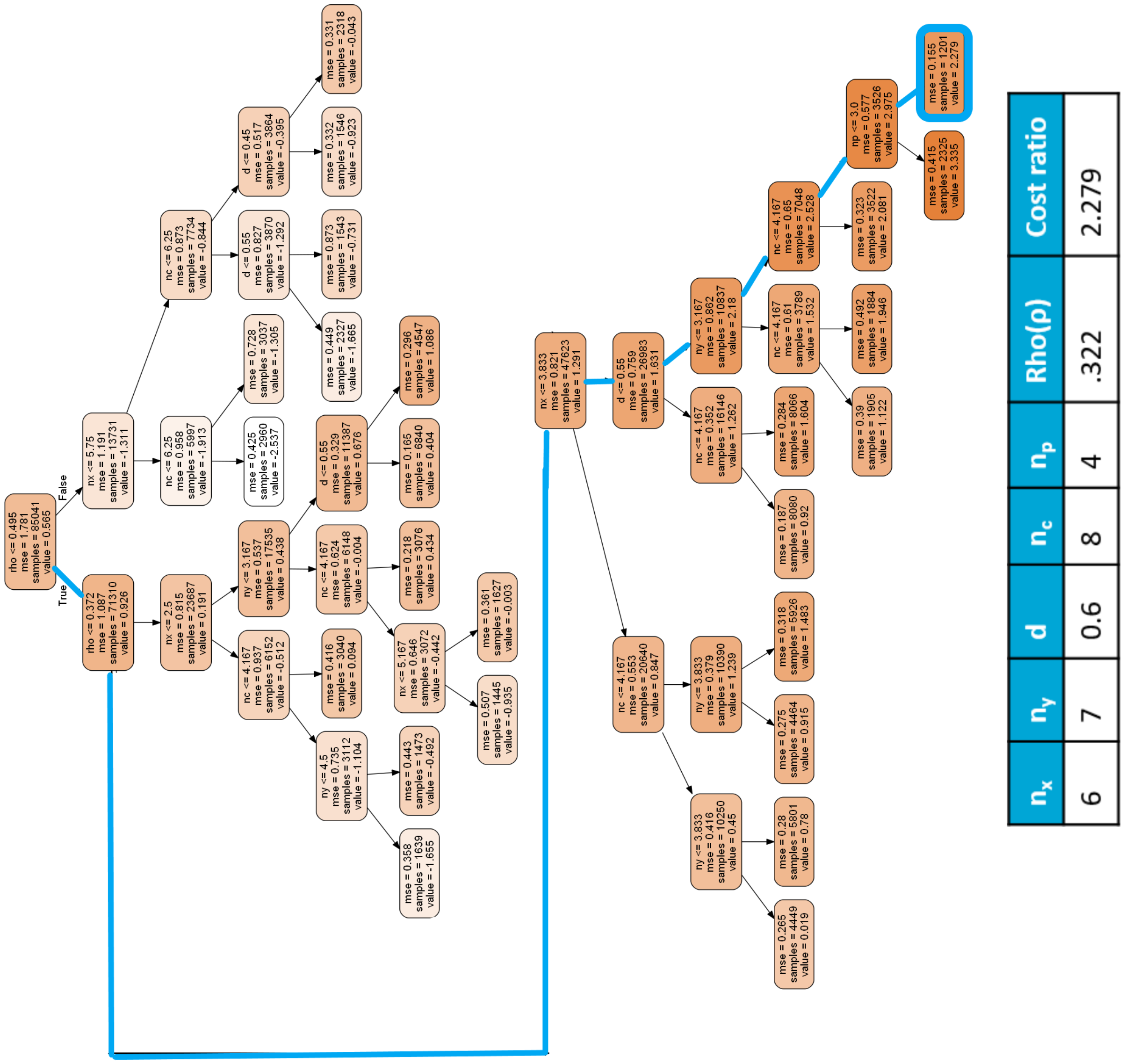


Figure 5.4: Decision Tree based prediction model

The decision diagram consists of *decision nodes* that split the database by following a certain rule. The corresponding nodes are further split until they end in a terminal node. Each terminal node represents a particular cost ratio for all the data points that fall under it. The process is explained in detail in Appendix C.8 where decision trees are applied on the SSBJ based scaled database. Figure 5.4 shows the original decision tree(behind the magnifying glass) which has thousands of nodes and branches, and also a small magnified portion showing a few terminal nodes and nearby branches. So, according to the above diagram, for points that satisfy $n_x > 3.16, d > 0.55$ and $n_y > 2.5$(end node circled in blue), the value of the cost ratio is 1.286. It can be assumed here that the splits regarding the coupling density or number of processors are made earlier in the decision tree.

As mentioned before, the intention is to supplement the neural network based prediction model with an easy to interpret decision diagram, which gives a rough estimate of the cost ratio for a new MDO problem. For this, a limited version of a decision tree is trained on the combined database (Table 4.11), by restricting the number of branches and decision nodes that are available for training the decision tree algorithm. The decision tree is visualized in Figure 5.5



n_x	n_y	d	n_c	n_p	Rho(p)	Cost ratio
6	7	0.6	8	4	.322	2.279

Figure 5.5: Decision tree based prediction model(limited branches/decision nodes)

Figure 5.5 shows a prediction model/advisory system, created using the decision tree algorithm. Training on the combined database(Table 4.11), using a 1:2 train test split, the decision tree is allowed to expand up to only seven levels while the total number of terminal nodes is restricted to twenty-five. The decision tree algorithm divides the test database into twenty-five zones(each representing a terminal node) with each zone being assigned a particular cost ratio. This is evident from the predictive plot of the restricted decision tree provided in Figure 5.6, which shows the predicted cost ratio(y_{pred}) for each of the data points in a particular terminal node. Compared to the predictive plot made by the re-trained neural network on the combined database(Figure 4.25b), two observations can be made. Firstly, the predicted cost ratios are arranged in layers and secondly, there is a significant spread of values(y_{test} or actual cost ratio) around each zone of predicted points. Therefore, due to the limited training of the decision tree, the overall prediction accuracy is lower than that of the neural network based prediction model(EVRS = 0.769). However, the decision tree based visualization has

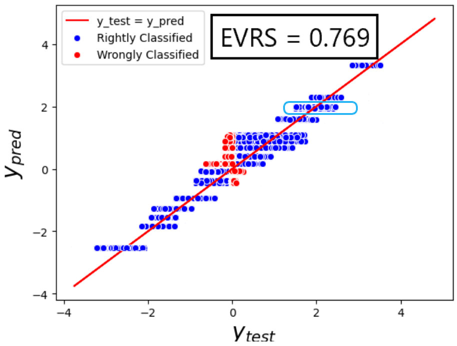


Figure 5.6: Predictive plot of decision tree(limited branches/decision nodes)

an advantage over the neural network based prediction model (Figure 5.3) because it allows the user to draw predictions on a test data point just by observing the above graph. The predicted cost ratio for one such data point, representing a scaled version of the propane combustion problem is also shown in Figure 5.5 (path and terminal node highlighted in blue). Figure 5.6 also shows the set of data points that represent the same cost ratio(circled in blue). Another observation that can be drawn from the decision tree is that the very first split is made on the coupling strength feature(ρ) which makes it the most decisive feature in the advisory system. The feature concerned with the number of design variables(n_x) also seems to be important as it is found higher up in the decision tree. Therefore, the decision tree based visualization gives new insights about the database of MDO problems that is not provided by the neural network based prediction model. However, since the prediction accuracy of the decision diagram is much lower than the neural network based prediction model, the decision diagram must be considered only as a useful supplement to the re-trained neural network. Further testing of the prediction model with new problems is performed only on the re-trained neural network.

The above advisory system/prediction model has been tested only with scaled problems that have been constructed using a fixed set of features. In the next section, the re-trained neural network is used to draw predictions on original, unscaled MDO problems.

5.3. Testing Original Problems on Re-Trained Neural Network

A total of six MDO problems were introduced in this thesis. The re-trained neural network was trained on scaled versions of four MDO problems and it was tested on scaled versions of two MDO problems. However, the prediction model is only practically useful if it can predict the cost ratios for any MDO problem that is introduced to the model, and not just SARF based scaled versions of MDO problems. It is therefore required to test the neural network based prediction model on original, unscaled MDO problems as well. For this purpose, the original versions of all six MDO problems are introduced to the prediction model. The neural network based prediction model only considers as input a set of feature values as explained earlier. Each of the features in the scaled problem is created through a rule-based transformation of the corresponding features in the original problems. Now, to test the prediction model on an original, unscaled problem, the same features have to be defined for the original problems. The details regarding the compatibility of features are given below:

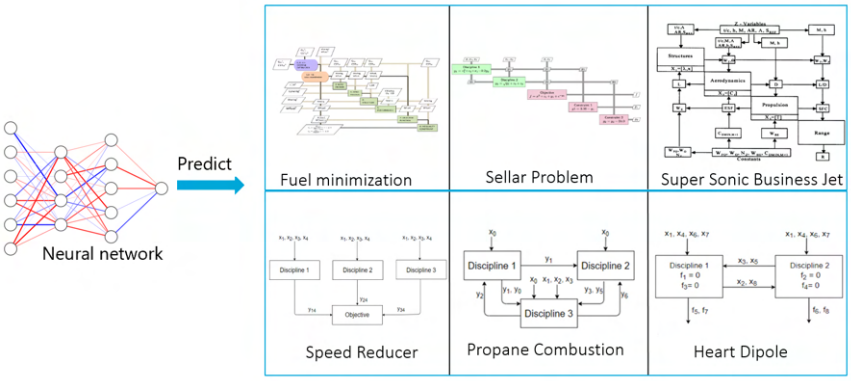
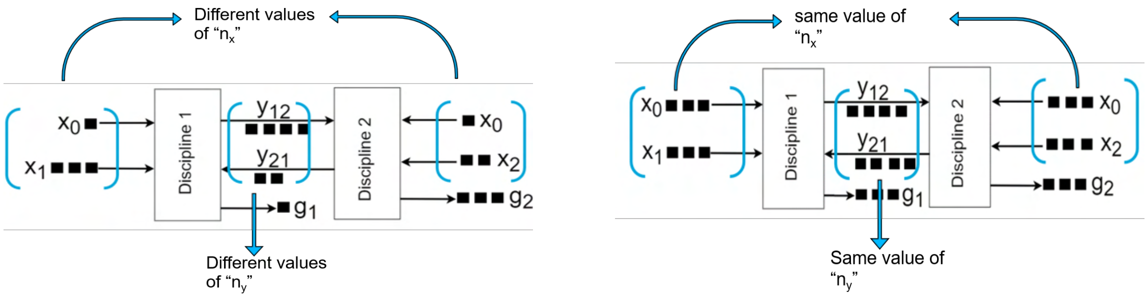


Figure 5.7: Predict cost ratios on Original Problems

1. Features regarding the coupling density(d) and the coupling strength(ρ) can be uniquely defined for the original problem using the same logic as for the scaled problem.

2. The feature regarding the number of processors(n_p) is problem independent so it can be applied to original problems in the same manner as scaled problems.

3. However, features related to problem size i.e, the size of the design variable(n_x), coupling variables(n_y) and constraint variables(n_c) cannot be imported directly for the original problem. This is shown using a disciplinary interface in Figure 5.8



(a) Original Problem

(b) Scaled Problem

Figure 5.8: Incompatibility in problem size(Original vs scaled Problem)

Figure 5.8 shows the incompatibility in problem sizes between original and scaled disciplines. According to the SARF methodology, only one value is assigned to n_x that accounts for the size of each design variable, local or shared. The same value of n_x applies to every discipline in the scaled problem, irrespective of the original structure of the problem. The rule also applies to n_y and n_c, representing coupling and constraint variables respectively, as shown in Figure 5.8b. This is not always true for an original MDO problem, which might have different sizes of design and coupling vectors as shown in Figure 5.8a. Therefore a method has to be devised such that a unique value for each problem size related feature can be logically defined or "extracted" from the original problem. For this purpose, two methods are suggested, one based on averages and another based on a more heuristic approach. The following section deals with the two methods and their outcome on the prediction made by the re-trained neural network

5.3.1. Test Original Problems - Average Based Feature Extraction

One method for extracting problem size related features is to consider the arithmetic mean of each type of size related parameter. This is shown using the design and coupling space of the fuel minimization problem in Figure 5.9.

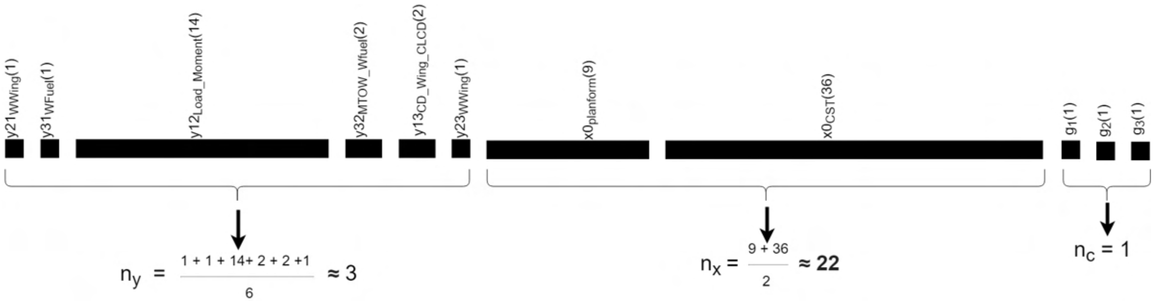


Figure 5.9: Problem Size parameters for Fuel Minimization Problem

Figure 5.9 shows the sizes of each coupling, constraint and design vector in the original fuel minimization problem. The values of n_y, n_x and n_c for the original problem can be calculated as the arithmetic mean of the existing variables, rounded off to the nearest integer. The feature values are then used for creating a database of original MDO problems as shown below in Table 5.2. As can be seen in the table, each of the three problem size related features has been normalized with the number of disciplines. As mentioned before, the process of extracting coupling strength and coupling density does not change from the scaled problems. While creating the database of Table 5.2, the MDO problem specific features remain the same(for every problem) while the number of processors is altered. This way, each MDO problem is represented by three data points, each representing one, two and four processors respectively. The only exception is the Fuel Minimization problem which has only two data points. This is because the fuel minimization problem has been transcribed directly from the original MATLAB based implementation, which simply gives the user the ability to use parallel processing but does not specify the number of processors. Therefore, for the fuel minimization problem, two data points are considered, one representing the lack of parallel processing and the other representing parallel processing via four cores.

Case	Features						label
	Design $(\mathbf{n_x})$	Coupling $(\mathbf{n_y})$	Constraint $(\mathbf{n_c})$	Coupling Density $(\mathbf{d})$	Coupling Strength $(\boldsymbol{\rho})$	Processors $(\mathbf{n_p})$	Cost/Wall time ratio $(\mathbf{R_t})$
SSBJ	3/3	2/3	4/3	0.46	0.422	1	-2.213
	3/3	2/3	4/3	0.46	0.422	2	-2.124
	3/3	2/3	4/3	0.46	0.422	4	-2.155
Propane Combustion	2/3	2/3	5/3	0.672	0.322	1	-0.883
	2/3	2/3	5/3	0.672	0.322	2	-0.754
	2/3	2/3	5/3	0.672	0.322	4	-0.412
Speed Reducer	5/3	2/3	3/3	0.65	0	1	-0.201
	5/3	2/3	3/3	0.65	0	2	-0.53
	5/3	2/3	3/3	0.65	0	4	-0.76
Heart Dipole	4/2	2/2	2/2	0.77	0.568	1	-3.403
	4/2	2/2	2/2	0.77	0.568	2	-3.254
	4/2	2/2	2/2	0.77	0.568	4	-3.323
Fuel Minimization	22/3	3/3	2/3	0.8	0.487	1(no parallel)	-0.513
	22/3	3/3	2/3	0.8	0.487	4(parallel)	-0.726
Sellar	2/2	2/2	2/2	0.8	0.562	1	-0.163
	2/2	2/2	2/2	0.8	0.562	2	-0.314
	2/2	2/2	2/2	0.8	0.562	4	-0.255

Table 5.2: Database of Original Problems

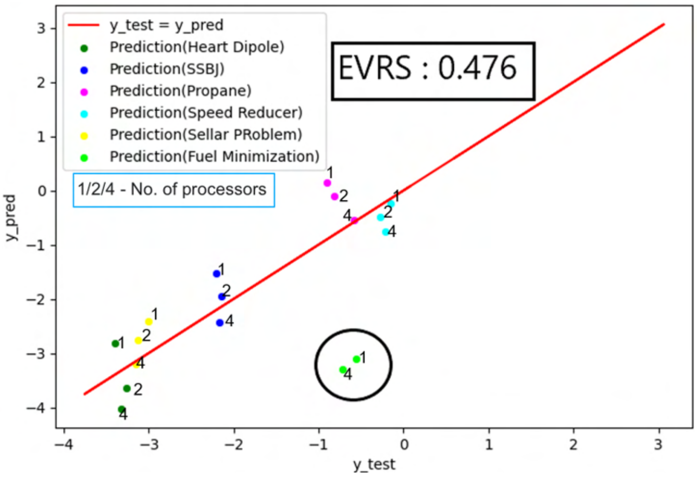


Figure 5.10: Predictive plot of Original Problems (estimated using average based feature extraction)

Once the above database is created, the ability of the prediction model to predict the outcome(cost ratios) on these new original problems can be tested. The re-trained model is applied on the set of the six original MDO problems from Table 5.2 and the results are plotted in Figure 5.10. y_{test} represents the actual cost ratios obtained through OpenMDAO(MATLAB in the case of the fuel minimization problem). The label column of Table 5.2 represents the y_{test} values. y_{pred} represents the predictions made by the re-trained neural network. Additionally, the number of processors used to execute the problem is numbered beside each point. A few interpretations can be made from the predictive plot of Figure 5.10. The red line($y_{test} = y_{pred}$) still represents the total range of data points that can be predicted by the re-trained prediction model. It can be seen that majority of the data points lie in the "third quadrant", i.e, the points are correctly predicted to be performing better with IDF architecture, except the points representing the speed reducer problem, which are correctly predicted to not substantially favor any of the two architectures. This could be due to the combined effect of lower problem size(which tends to shift preference towards IDF) and lowest coupling strength($\rho = 0$)(which tends to shift the preference towards MDF). Also, looking at the accuracy for each prediction and potential outliers, the two points representing the fuel minimization problem(circled green) are found to be lying the furthest from the prediction line. The actual value of the cost ratios ($y_{test} = \{-0.56, -0.72\}$) are still within the feasible range of prediction, but the model predicts the cost ratios to be much more negative($y_{pred} = \{-3.23, -3.03\}$). Because of the poor prediction accuracy achieved on points representing the fuel minimization problem, the overall prediction accuracy also takes a hit, with the EVRS metric coming out to be 0.476. This is because the EVRS metric, computed by considering the square of the variance values, ends up heavily penalizing the predicted data point for the fuel minimization problem. This prediction can be improved by adopting a better approach for feature extraction as shown in the section below.

5.3.2. Test Original Problems - Sensitivity based Feature Extraction

In order to improve the prediction accuracy of the model for the fuel minimization problem, a different strategy can be adopted for extracting the coupling variable related parameter(n_y), one that is based on the sensitivity of coupling variables with respect to each other. The idea is to include not just the shear number(arithmetic mean) of coupling variables but the relative contribution that the variables make to the overall convergence of an iterative solver. For this purpose, the iteration matrix for a given MDO problem, drawn using the $DirectSolver$ convergence scheme is revisited. The iteration matrix for the fuel minimization problem is shown in Equation 5.1

$$\begin{bmatrix}\Delta y_{21}\\ \Delta y_{23}\\ \Delta y_{32}\\ \Delta y_{13}\\ \Delta y_{12}\\ \Delta y_{31}\end{bmatrix}_{k+1} = \underbrace{\begin{bmatrix} I & 0 & \cdots & 0\\ -\frac{\partial y_{23}}{\partial y_{21}} & I & \ddots & \vdots\\ \vdots & \ddots & \ddots & 0\\ -\frac{\partial y_{31}}{\partial y_{21}} & \cdots & -\frac{\partial y_{31}}{\partial y_{12}} & I\end{bmatrix}^{-1} \begin{bmatrix} 0 & \frac{\partial y_{21}}{\partial y_{23}} & \cdots & \frac{\partial y_{21}}{\partial y_{31}}\\ 0 & 0 & \ddots & \vdots\\ \vdots & \vdots & \ddots & \frac{\partial y_{32}}{\partial y_{31}}\\ 0 & 0 & \cdots & 0\end{bmatrix}}_{G} \begin{bmatrix}\Delta y_{21}\\ \Delta y_{23}\\ \Delta y_{32}\\ \Delta y_{13}\\ \Delta y_{12}\\ \Delta y_{31}\end{bmatrix}_{k} \tag{5.1}$$

where k stands for the k^{th} $DirectSolver$ Iteration.

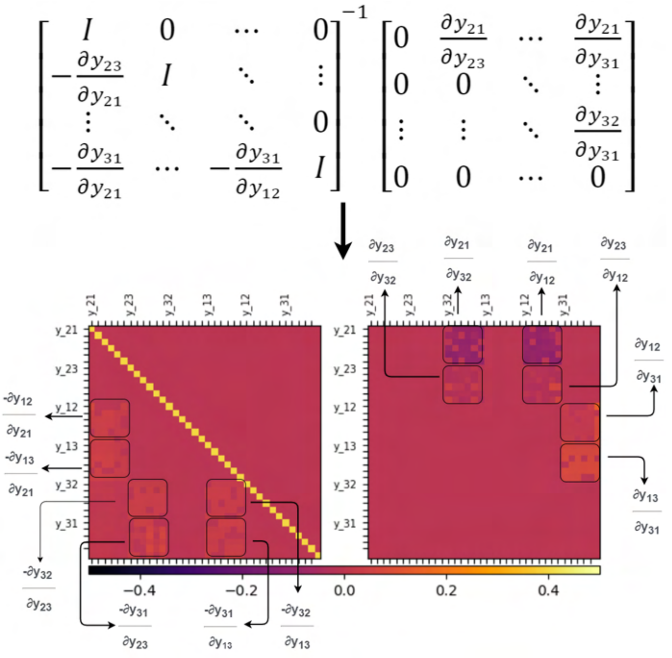


Figure 5.11: Iteration Matrix(1^{st} Direct Solver iteration) (Fuel minimization Problem)

The fuel minimization problem consists of six coupling variables as evident from the XDSM diagram shown in Figure E.1 and the dependency matrix shown in Figure E.4. The iteration matrix shown above represents the individual local sensitivities of the six coupling vectors with respect to each other. For vectors such as y_{12}, that represents the load and moment distribution over the wing span, the sensitivity term can be expanded to represent the individual sensitivities of each element of the load and moment vector with respect to other couplings.

The overall coupling strength for an MDO problem is calculated by averaging out the spectral radius using the time of individual iteration as weights as shown in Equation 4.8. The definition of the spectral radius(largest absolute eigenvalue) implies that the terms(individual sensitivities) within the iteration matrix that are substantially greater compared to other terms, would have a much higher effect on the convergence process of an MDA iteration. This is shown by visually representing the iteration matrix for the 1^{st} $DirectSolver$ Iteration in Figure 5.11. Figure 5.11 makes a color tone based representation of the local sensitivities of the coupling variables with respect to each other for the first $DirectSolver$ iteration. The marked zones(six each in both lower and upper matrix) represent these individual sensitivities. Based on the color map, it can be observed that two of the coupling interactions($\frac{\partial y_{21}}{\partial y_{12}}, \frac{\partial y_{21}}{\partial y_{32}}$) have much higher individual sensitivities(in terms of magnitude) compared to other coupling pairs. Therefore, it can be argued that the overall rate of convergence for the $DirectSolver$ MDA is directed primarily by three couplings(y_{12}, y_{21} and y_{32}). The other coupling variables do not decisively affect the convergence process.

Considering the average of these three coupling variables, the value of n_y that is extracted from the original fuel minimization problem is six. Based on this value of n_y, the prediction plot can be re-estimated, while preserving the earlier methods used to extract the remaining features(based on average as shown in Table 5.2). The sensitivity based method is applied only on the fuel minimization method because it does not make a difference in the n_y values for the other problems. The result of the sensitivity based feature estimation is shown in Figure 5.12.

Looking at Figure 5.12, the improvement in the predictive performance for the fuel minimization problem as compared to the earlier plot(Figure 5.10) can be observed. From the context of decision making (choice between MDF vs IDF), the prediction model does make the right call by predicting IDF to be faster than MDF across the board(except one data point belonging to the Speed Reducer problem where the prediction is off by a negligible margin). Additionally, the prediction model also manages to place the data points diagonally in the right order with respect to the coupling strength, with the most strongly coupled problem(Heart Dipole) occupying the left lower end of the prediction plot. The MDO problem with the lowest coupling strength(Speed Reducer) stays on the other end. The overall predictive accuracy(considering each of the original problems) can be determined using the normalized EVRS metric used before and it comes to be 0.844. This value is lower than the prediction accuracy obtained with scaled problems, but it can be observed from the plot that the machine learning model can predict, with significant accuracy, the outcome of the tested MDO problems. Figure 5.12 can be seen as the final outcome of the neural network based prediction model developed in this chapter. The next section is used to verify the prediction plot of Figure 5.12 with existing studies from the literature.

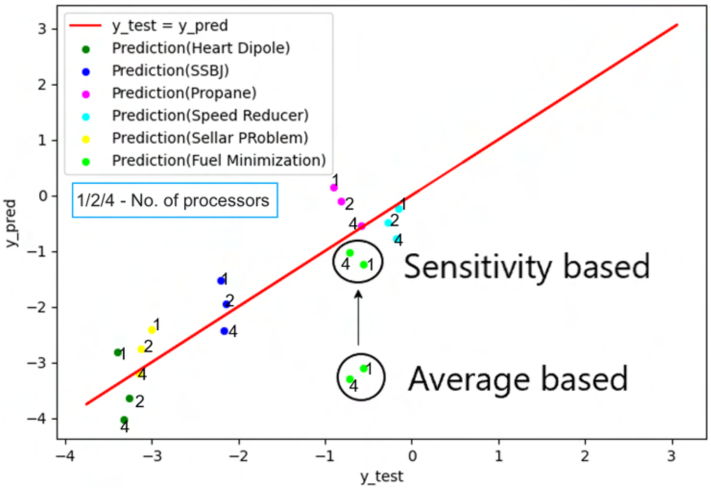
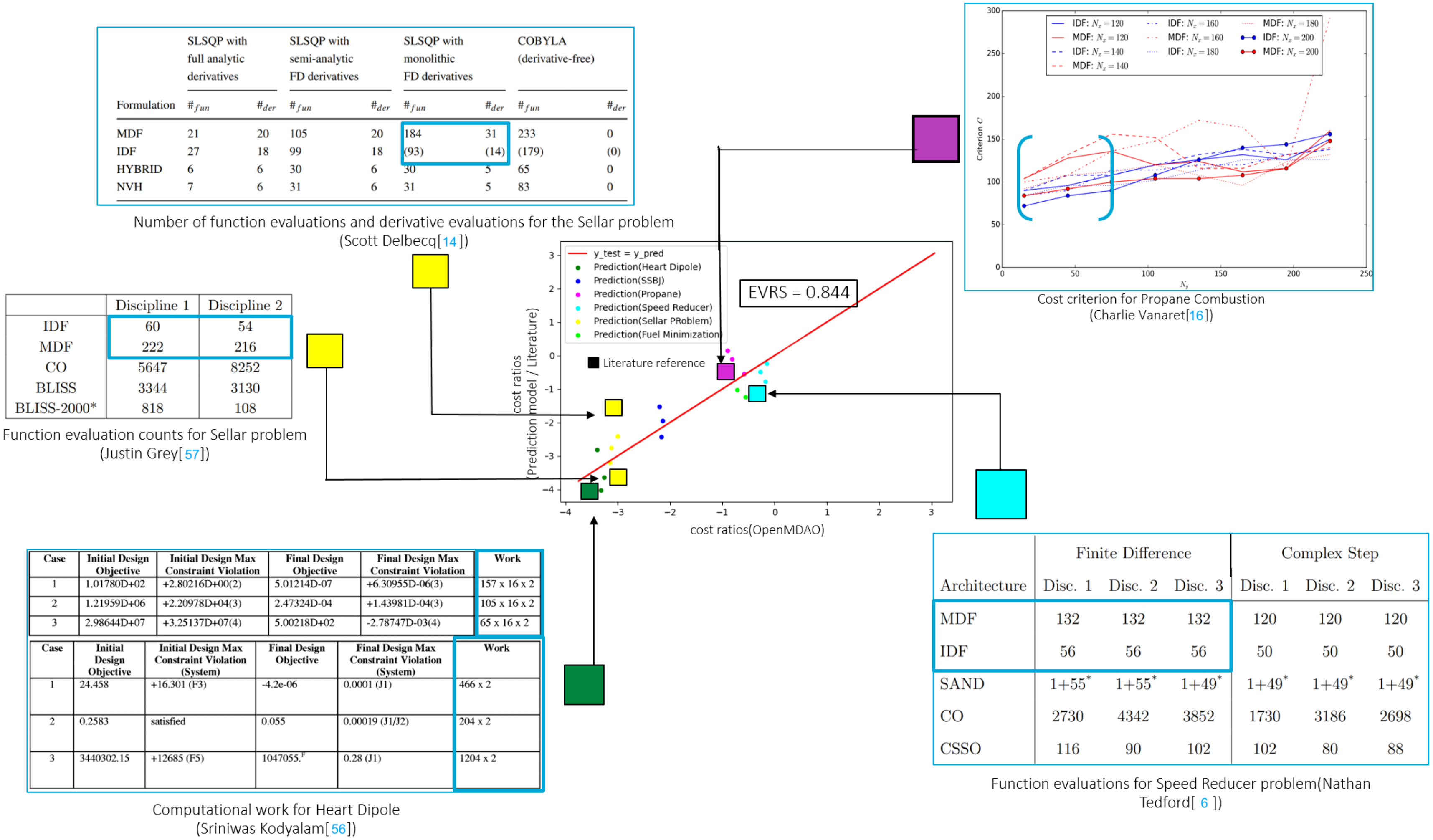


Figure 5.12: Predictive plot of Original Problems
(re-estimated using sensitivity based feature extraction)

5.4. Verification with Literature

Since the tested problems have also been used in the earlier comparative studies[16, 56, 6, 14, 57], the results from these studies can be used to verify the predictions made by the machine learning model(Figure 5.12). Just to recap, the y_{test} on the x-axis represents the actual cost ratios obtained when executing the problems with OpenMDAO, while y_{pred} on the y axis represents the cost ratios predicted by the machine learning model. The information on the cost ratios that have been obtained from literature can be also be plotted along the y axis on top of the same figure. Figure 5.13[16, 56, 6, 14, 57] shows the source of the cost ratios used for the verification. Using square blocks, the cost ratios obtained from literature are plotted on top of the same predictive plot used in the previous section. From Figure 5.13 it can be seen that the cost ratios estimated from earlier studies match the predictions made by the neural network model, except for the Sellar problem, where the cost ratios obtained in the research by Delbecq et.al [14] are higher than what is predicted by the neural network. A second reference is added from a paper by Gray et. al.[57] which is much closer to the predicted values. It must be noted that the earlier studies were conducted using a different set of operating conditions, in terms of the tolerance settings used within the problems, the cost criterion used to compare the MDO architectures and the number of processors used. Therefore, it is not an apples to apples comparison, but more of a literature based verification regarding the ballpark of predictions made by the neural network model.



	SLSQP with full analytic derivatives		SLSQP with semi-analytic FD derivatives		SLSQP with monolithic FD derivatives		COBYLA (derivative-free)	
Formulation	$\#_{fun}$	$\#_{der}$	$\#_{fun}$	$\#_{der}$	$\#_{fun}$	$\#_{der}$	$\#_{fun}$	$\#_{der}$
MDF	21	20	105	20	184	31	233	0
IDF	27	18	99	18	(93)	(14)	(179)	(0)
HYBRID	6	6	30	6	30	5	65	0
NVH	7	6	31	6	31	5	83	0

Number of function evaluations and derivative evaluations for the Sellar problem (Scott Delbecq[14])

	Discipline 1	Discipline 2
IDF	60	54
MDF	222	216
CO	5647	8252
BLISS	3344	3130
BLISS-2000*	818	108

Function evaluation counts for Sellar problem (Justin Grey[57])

Cost criterion for Propane Combustion (Charlie Vanaret[16])

	Finite Difference			Complex Step		
Architecture	Disc. 1	Disc. 2	Disc. 3	Disc. 1	Disc. 2	Disc. 3
MDF	132	132	132	120	120	120
IDF	56	56	56	50	50	50
SAND	1+55*	1+55*	1+49*	1+49*	1+49*	1+49*
CO	2730	4342	3852	1730	3186	2698
CSSO	116	90	102	102	80	88

Function evaluations for Speed Reducer problem(Nathan Tedford[6])

Case	Initial Design Objective	Initial Design Max Constraint Violation	Final Design Objective	Final Design Max Constraint Violation	Work
1	1.01780D+02	+2.80216D+00(2)	5.01214D-07	+6.30955D-06(3)	157 x 16 x 2
2	1.21959D+06	+2.20978D+04(3)	2.47324D-04	+1.43981D-04(3)	105 x 16 x 2
3	2.98644D+07	+3.25137D+07(4)	5.00218D+02	-2.78747D-03(4)	65 x 16 x 2

Case	Initial Design Objective	Initial Design Max Constraint Violation (System)	Final Design Objective	Final Design Max Constraint Violation (System)	Work
1	24.458	+16.301 (F3)	-4.2e-06	0.0001 (J1)	466 x 2
2	0.2583	satisfied	0.055	0.00019 (J1/J2)	204 x 2
3	3440302.15	+12685 (F5)	1047055.^{F}	0.28 (J1)	1204 x 2

Computational work for Heart Dipole (Sriniwas Kodyalam[56])

Figure 5.13: Verification of cost ratios with Literature

Based on the predictive plots from Figure 5.2a, Figure 5.2b and Figure 5.13, the EVRS values for all the estimations made using the prediction model is summarized in the table below(Table 5.3):

Problem	EVRS
Scaled Problems (prediction model)	0.908 - 0.958
Original Problems (prediction model)	0.844
Original Problems (Literature Results)	0. 824

Table 5.3: EVRS values for the prediction model(all problems)

Table 5.3 implies that values of cost ratios obtained from both the prediction model and Literature are more than eighty percent in agreement with that of the OpenMDAO/MATLAB based implementation of the six original MDO problems. Table 5.3 can be seen as an external verification of the capabilities of the neural network based prediction model of MDO architecture.
The next section is used to explain the steps required to deploy the prediction model

5.5. Deployment of the Prediction Model

This section is used to explain the necessary steps in deploying the prediction model on an unscaled MDO problem. Table 5.4 shows the set of parameter values that are accepted by the prediction model and the method to be used for calculating them for an unscaled problem:

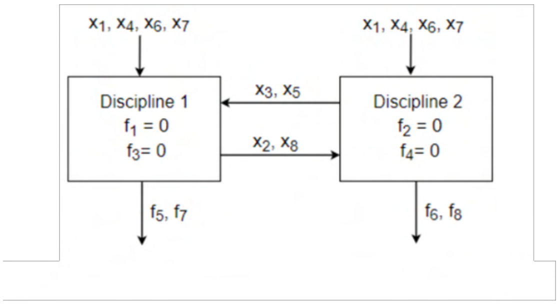
Parameter	**Method of calculation**
Coupling density(d)	Compute from problem formulation
Coupling strength(ρ)	Compute from added methods in OpenMDAO code
n_x, n_y, n_c	Extract from problem formulation and $DirectSolver$ Iteration matrix

Table 5.4: Estimation of parameters from Original Problems

This section is divided into four parts. In the first three parts, the computation process for the above three sets of values is discussed using the example of the original heart dipole problem. In the final part, the practical aspects regarding the application of the prediction model on the MDO problem are discussed using the original heart dipole problem

5.5.1. Computation of coupling density(d)

The heart dipole problem is briefly explained below:



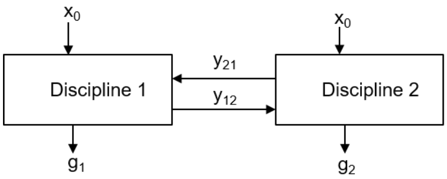
$$Discipline1\begin{cases} f_1 = x_1 + x_2 - \sigma_{mx} = 0 \\ f_3 = x_5x_1 + x_6x_2 - x_7x_3 \\ -x_8x_4 - \sigma_A = 0 \\ \\ f_5 = f_5(x1, x4, x6, x7, x3, x5) \\ f_7 = f_7(x1, x4, x6, x7, x3, x5) \end{cases}$$

$$Discipline2\begin{cases} f_2 = x_3 + x_4 - \sigma_{my} = 0 \\ f_4 = x_7x_1 + x_8x_2 + x_5x_3 \\ +x_6x_4 - \sigma_B = 0 \\ \\ f_6 = f_6(x1, x4, x6, x7, x2, x8) \\ f_8 = f_8(x1, x4, x6, x7, x2, x8) \end{cases}$$

Figure 5.14: Original Heart Dipole problem / disciplines

As can be seen in the problem, both disciplines contain two residual equations each. The coupling outputs are calculated through the residual equations. There are no local design variables, only four shared design variables. Additionally, there are two outputs attached to each discipline$(f_1, f_7), (f_6, f_8)$ which form the constraint variables. The mathematical details of the constraints are not given in Figure 5.14(A full description of the problem is provided in Appendix D.3).

It is required to convert the input/coupling/constraint variables into a directional form such that the dependency matrix can be constructed. This also requires the conversion of residuals into explicit state variables. This is shown in Figure 5.15



$$Discipline1 \begin{cases} y_{12}[0] = \sigma_{mx} - x_0[0] \\ y_{12}[1] = \frac{-\sigma_A + y_{21}[1]x_0[0] + x_0[2](\sigma_{mx} - x_0[0]) - x_0[3]y_{21}[0]}{x_0[1]} \\ \\ g_1[0] = g_1[0](x_0, y_{21}) \\ g_1[1] = g_1[1](x_0, y_{21}) \end{cases}$$

$$Discipline2 \begin{cases} y_{21}[0] = \sigma_{my} - x_0[1] \\ \\ y_{21}[1] = \frac{\sigma_B - x_0[2]x_0[1] - y_{12}[0]y_{12}[1] - x_0[3]x_0[0]}{\sigma_{my} - x_0[1]} \\ \\ g_2[0] = g_2[0](x_0, y_{12}) \\ g_2[1] = g_2[1](x_0, y_{12}) \end{cases}$$

Figure 5.15: SARF compatible heart dipole problem/ disciplines

If the problem is already present in the SARF compatible form, then the above step can be discarded. Following this, it is required to manually create the dependency matrix from the SARF compatible disciplines as shown in Figure 5.16.

$$Discipline1 \begin{cases} y_{12}[0] = \sigma_{mx} - x_0[0] \\ y_{12}[1] = \frac{-\sigma_A + y_{21}[1]x_0[0] + x_0[2](\sigma_{mx} - x_0[0]) - x_0[3]y_{21}[0]}{x_0[1]} \\ \\ g_1[0] = g_1[0](x_0, y_{21}) \\ g_1[1] = g_1[1](x_0, y_{21}) \end{cases}$$

$$Discipline2 \begin{cases} y_{21}[0] = \sigma_{my} - x_0[1] \\ \\ y_{21}[1] = \frac{\sigma_B - x_0[2]x_0[1] - y_{12}[0]y_{12}[1] - x_0[3]x_0[0]}{\sigma_{my} - x_0[1]} \\ \\ g_2[0] = g_2[0](x_0, y_{12}) \\ g_2[1] = g_2[1](x_0, y_{12}) \end{cases}$$

$\longrightarrow$

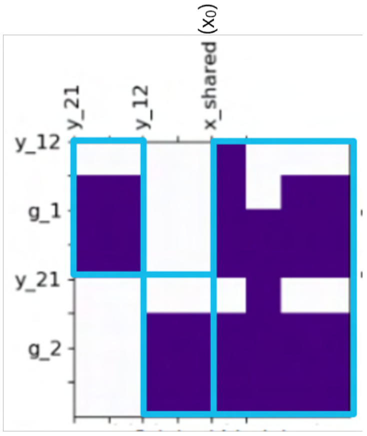


Figure 5.16: Dependency matrix for the SARF compatible heart dipole problem

The individual input-output blocks(resembling discipline 1, discipline 2 and shared variables) are also shown with cyan boxes in the above dependency matrix. The coupling density factor(d) can be manually calculated from the above dependency matrix by measuring the percentage of filled square dots within the cyan boxes(for the above dependency matrix this comes out to be 0.75).

5.5.2. Computation of Coupling Strength(ρ)

As mentioned in Section 4.2.4, for calculating coupling strength for a particular MDO problem, it is required to converge a single MDA routine using the $DirectSolver$ convergence scheme. This is shown for the heart dipole problem in Figure 5.17. In practicality, the process requires the placing of two counters inside the OpenMDAO code for accumulating the system jacobian and storing the time for each $DirectSolver$ iteration. Further, an additional script has to be written to separate the Jacobian into lower and upper triangular matrices to compute the spectral radius. The method to be added within the OpenMDAO code as well as the script to convert the Jacobian into iteration matrix is shown in Appendix A.7. Using this process, for the heart dipole problem, the coupling strength(ρ) comes to be 0.568.

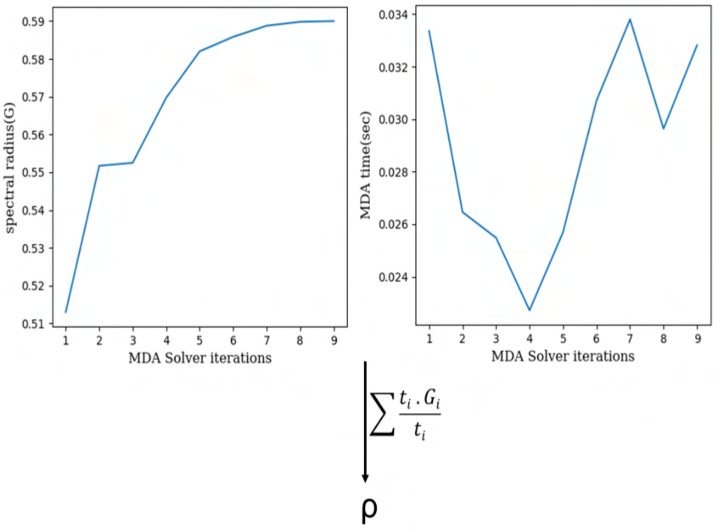


Figure 5.17: Accumulate iteration matrix for Heart Dipole problem

5.5.3. Extraction of Problem Size related Values(n_x, n_y, n_c)

The average based method was proposed for extraction of n_x and n_c while a sensitivity based method was proposed to extract n_y. The values of n_x and n_c can be extracted from the dependency matrix as shown in Figure 5.18. The input space shown along the x axis shows the n_x and n_y variables, while the y axis shows the n_y and n_c variables. Looking at the blocks allocated to each variable, the equivalent values of n_x and n_c can be calculated. The example shown in Figure 5.18, representing the heart dipole problem, contains one shared design vector(x_0) of size four, no local design vectors and two local constraint vectors of size two each. Hence for the heart dipole problem $n_x = 4$ and $n_c = 2$ The value of n_y is calculated by applying the sensitivity based method on the dependency matrix. Following the template used in Section 5.3.2, the iteration matrix is visualized which indicates the magnitude of local sensitivities of coupling and constraint variables with respect to each other for the 1st $DirectSolver$ iteration. The blocks representing the sensitivities of $\frac{\delta(y_{21})}{\delta(y_{12})}$ and $\frac{\delta(y_{12})}{\delta(y_{21})}$ with respect to each other are shown in cyan. It can be seen that no individual sensitivity is an order of magnitude higher than the other, therefore both coupling vectors are used to extract the value of n_y. Hence, $n_y = 2$.

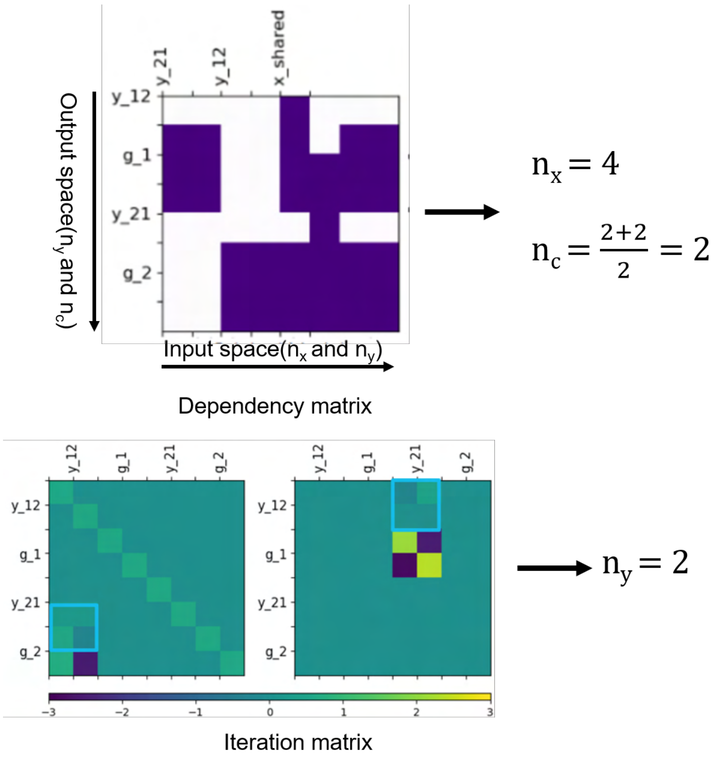
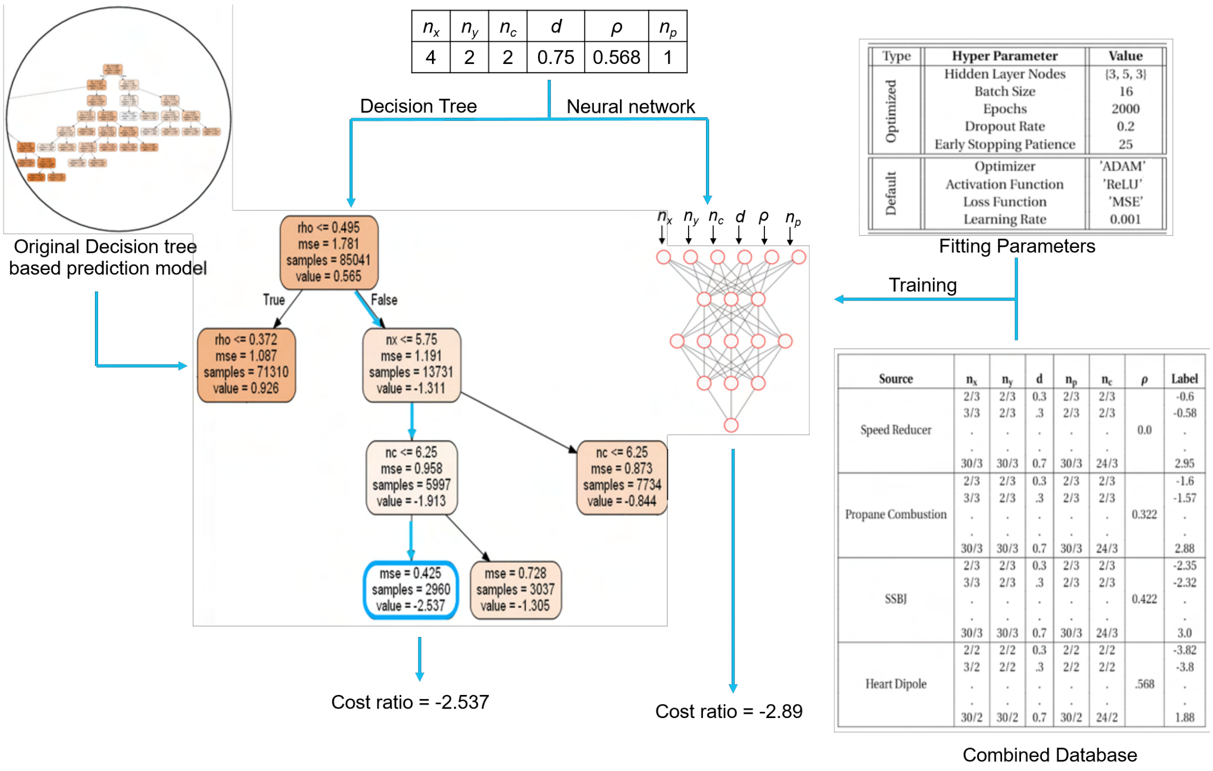


Figure 5.18: Extract problem size parameters

5.5.4. Application of Prediction Model

Following the calculation/extraction of each parameter, the prediction model can be applied to a new original MDO problem. Now, two prediction models have been formed in this thesis, one based on the re-trained neural network and a second model based on decision trees. For a less accurate but quick estimation, the decision tree based model can be referred. As mentioned before, the advantage of the decision tree based model is that it's a pre-built graph-based representation that can be used to estimate the cost ratio for any new MDO problem without executing a trained machine learning algorithm using scikit-learn. For a more accurate prediction, the re-trained neural network based model has to be used. For a new user, the neural network based model is firstly required to be trained using the fitting parameters that were derived in Table 4.6, on the combined database of Table 4.11. Following this, the neural network can make cost ratio predictions for a new problem. The two approaches are shown in Figure 5.19



n_x	n_y	n_c	d	ρ	n_p
4	2	2	0.75	0.568	1

Type	Hyper Parameter	Value
Optimized	Hidden Layer Nodes Batch Size Epochs Dropout Rate Early Stopping Patience	{3, 5, 3} 16 2000 0.2 25
Default	Optimizer Activation Function Loss Function Learning Rate	'ADAM' 'ReLU' 'MSE' 0.001

Fitting Parameters

Source	n_x	n_y	d	n_p	n_c	ρ	Label
Speed Reducer	2/3 3/3 . . 30/3	2/3 2/3 . . 30/3	0.3 .3 . . 0.7	2/3 2/3 . . 30/3	2/3 2/3 . . 24/3	0.0	-0.6 -0.58 . . 2.95
Propane Combustion	2/3 3/3 . . 30/3	2/3 2/3 . . 30/3	0.3 .3 . . 0.7	2/3 2/3 . . 30/3	2/3 2/3 . . 24/3	0.322	-1.6 -1.57 . . 2.88
SSBJ	2/3 3/3 . . 30/3	2/3 2/3 . . 30/3	0.3 .3 . . 0.7	2/3 2/3 . . 30/3	2/3 2/3 . . 24/3	0.422	-2.35 -2.32 . . 3.0
Heart Dipole	2/2 3/2 . . 30/2	2/2 2/2 . . 30/2	0.3 .3 . . 0.7	2/2 2/2 . . 30/2	2/2 2/2 . . 24/2	.568	-3.82 -3.8 . . 1.88

Combined Database

Figure 5.19: Prediction model - Two approaches

The cost ratios predicted on the original heart dipole problem by the two prediction models is shown in Figure 5.19. For the decision tree based approach, the original decision tree, as well as the part of the tree that concerns with the cost ratio prediction on the original heart dipole problem is shown. For the neural network based approach, the combined database, and the fitting parameters for training the neural network is shown. The actual cost ratio for the original heart dipole problem is -3.21, while the neural network based model predicts the cost ratio to be -2.89. Both the predicted and actual cost ratios can be seen in Figure 5.12. The decision tree based model gives a cost ratio of -2.537 which makes it less accurate than the neural network. This marks the end of the deployment process. The next section is used to provide a summary of the current chapter.

5.6. Summary - Test Cases

In this chapter, the re-trained neural network based prediction model developed in the earlier section was tested. Firstly, two scaled test problems(Sellar problem / Fuel minimization problem) were introduced into the prediction model's test database and the accuracy of prediction was tested(Figure 5.2a and Figure 5.2b). The prediction accuracy of the re-trained model in terms of EVRS was > 90%. Based on the performance of the re-trained prediction model, a sample of an advisory system was created, in the form of an input-output interface(Table 5.3) and in the form of a decision diagram(Figure 5.4). Following this, original versions of the same test problems were introduced. To test the prediction model on the original problems, a method had to be defined for extraction of problem size related features. For extracting features from the original problems that are compatible with the neural network based prediction model, two methods were looked at, one that computed averages of all problem size related features, and another that used a hybrid method(based on average/sensitivity) for feature extraction. The prediction accuracy for the average based method(Figure 5.10), in terms of EVRS, came out to be an unacceptable 0.476. This was largely due to inaccurate prediction on the fuel minimization problem. In the second method, based on sensitivity based feature extraction(Figure 5.12), the performance on the fuel minimization problem was found to be improved, with the EVRS value on the overall prediction being increased to 0.844. When the same results were compared with the existing results from the literature, a similar level of accuracy was obtained, which served as a validation of the prediction accuracy of the neural network based prediction model. Finally, a step by step procedure for deploying the prediction model on a test MDO problem was provided. The next chapter is used to provide conclusions and recommendations for future work.

6 Conclusions and Recommendations

This thesis report described the development of a prediction model of MDO architecture that can recommend the better MDO architecture between MDF(Multiple Discipline Feasible) and IDF(Individual Discipline Feasible) in terms of solution cost for a given MDO problem, based on features(problem parameters) that are extracted from the problem formulation before execution. This chapter consists of two sections - Conclusions, and Recommendations. In the conclusion section, the background of the thesis is discussed briefly. Following this, the research questions and sub-questions that were formed in the Introduction(Chapter 1) are recalled and conclusions are given in the context of the questions. Following the conclusion, the recommendations for future work related to the thesis are discussed.

6.1. Conclusion

One of the challenges in executing an MDO system is the selection of the best performing MDO architecture in terms of computational effort, for a given MDO problem. To address this, several comparative studies have been created, which analyze the relative performance of MDO architectures with respect to internal features of MDO problems, such as the number of design variables and strength of interdisciplinary coupling. However, the existing research is limited in applicability, and the results obtained from an existing comparative study or advisory system cannot be used to predict the best MDO architecture for a new MDO problem. An inclusive prediction model, that can recommend the better MDO architecture between MDF(Multiple Discipline Feasible)(using either Gauss-Seidel or Jacobi scheme) and IDF(Individual Discipline Feasible) for any MDO problem is not found yet in literature. This formed the premise of the following research question that was framed in the introductory chapter:

Research Question: Is it possible to create a prediction model that can recommend the better MDO architecture between MDF and IDF in terms of solution cost, for a given MDO problem, solely based on features that have been extracted from the problem formulation, without executing the problem?

The following sub-questions were formed out of the above question:

Research Sub-Question 1: What are the drawbacks of existing comparative studies of MDO architectures in literature and how does the thesis propose to resolve them?

Research Sub-Question 2: How is the prediction model developed and validated?

Research Sub-Question 3: How is the prediction model tested on new MDO problems?

The answer to the first part of Research sub-question 1 was found in the Literature Review(Chapter 2). Scalable problems are specifically designed to allow the user to create a repository of MDO problems by varying one or more internal features. In the existing studies, the solution cost of MDO architectures was compared using scalable MDO problems. The conclusions out of such studies were given either in the form of an ontology containing a set of rules such as "If the number of coupling variables is more than design variables, then architecture X is faster than others", or in the form of graphs representing the solution cost of MDO architectures with respect to problem features. Two drawbacks were noted in the existing studies. Firstly, it was observed that all internal problem features were not taken into account in the same study. MDO problems which enabled the user to set the problem size(in terms of number of design/coupling/constraint variables) did not allow the variation of coupling strength(parameter to set the sensitivity of interdisciplinary couplings w.r.t each other) and vice versa. Secondly, the problem features were not defined in a normalized manner. A scalable problem built using two disciplines could not be used to predict the outcome of a different problem containing three disciplines. Due to these two drawbacks, the conclusions made in the existing comparative studies were not applicable to new MDO problems.

The answer to the second part of Research Sub-Question 1 was found in Chapter 3. Mentioned as the final paper in the literature review, a recent comparative study from 2017 was performed by Charlie Vanaret et.al.[16]. An in-depth analysis of this paper was conducted in Chapter 3. Instead of proposing a new scalable MDO problem, the paper proposed a transformation function called Scalable Analytic Replacement Function(SARF) which could be used to create a scalable version of an existing MDO problem. The methodology allowed the user to vary four internal features(related to problem size) of a scaled MDO problem, more than any previous study. However, upon closer inspection of the mathematical background of the SARF method, an opportunity was discovered that could also enable the inclusion of coupling strength in the comparative analysis, opening the possibility to resolve the first drawback mentioned above. Additionally, the problem agnostic nature of the SARF method meant that the problem size related features could be normalized with the number of disciplines. This presented the possibility that results obtained from a particular comparative study could be applicable on a different MDO problem, thereby resolving the second drawback.
While the proposed method showed promise, the application part of the original paper was found to be lacking in key areas. In the original comparative study, the SARF methodology was applied on the Super Sonic Business Jet(SSBJ) MDO problem to create a repository of SSBJ based scaled MDO problems. For creating the repository, only two problem features were considered(out of the possible four). The possibility of using coupling strength as a problem feature was not investigated. Additionally, the features were not implemented in a normalized manner either. By comparing the performance of MDF and IDF architectures on the repository of SSBJ based scaled MDO problems, it was observed in the original paper that internal problem features did not establish a trend in the outcome of MDO approaches. In other words, it was claimed that whatever the MDO problem, there was no "best" architecture to solve it. However, when a more thorough repetition of the comparative analysis was conducted in this thesis(Chapter 3) by giving more runs to each combination of problem feature values, it was seen that a trend did exist between problem features and the performance of MDO architectures. The reproduced plot showed the presence of certain zones(ranges of problem features) that favored one architecture over the other, which was in contradiction with the claim made in the original paper. This provided a motive for building a prediction model of MDO architecture based on the SARF scaling method, by including the full set of problem features enabled by the SARF method in a normalized manner and also adding new features that were compatible with the SARF methodology.

The second sub-question was answered via chapter 4. A set of five problem features was defined, of which four were features offered by the SARF method while the fifth feature, related to the optimization environment, was the number of processors cores available in the optimization. Following the feature definition, a label term called cost ratio was defined that represented the relative cost of optimization using IDF over MDF architectures for a scaled problem. Using three features at a time, a database of SSBJ based scaled MDO problems was built and visual analysis(in the form of a three-dimensional plot) was run to look into the contribution that each feature made to the cost ratio. Based on the visual analysis the method of implementation of each feature was decided. Following this, a database of SSBJ based MDO problems, represented by five features and one label(cost ratio) was created. Four machine learning algorithms were applied on the database and their fitting accuracy were tested. The algorithm with the highest fitting accuracy, an artificial neural network containing three hidden layers was considered as the first prediction model built in the thesis. For validating the model, a database representing scaled versions of three MDO problems was introduced and the prediction accuracy on the new problems was tested. The result showed a significant amount of over/under prediction by the neural network on the new test problems. The prediction accuracy was around 70% for the test problems. To improve the prediction, a new feature called coupling strength was defined. The feature to compute the coupling strength of an MDO problem required the convergence of a singular MDA routine with the $DirectSolver$ convergence scheme. Using the new feature(along with the existing ones) and all four

MDO problems, a second neural network based prediction model, called the re-trained model, was created.

The final sub-question was answered via chapter 5, where the re-trained neural network was tested on scaled versions of new MDO problems, of which one was an aircraft based fuel minimization problem. The prediction accuracy of the re-trained model on the test problems was more than 90%. However, to draw final conclusions about the ability of the prediction model, it also had to be tested on original(unscaled) MDO problems. For this purpose, original versions of the existing scaled MDO problems were chosen. However, it was first required to create a method for extracting relevant features from the original problems, so that a database of original problems could be created in terms of the existing problem parameters. For problem size related features, the extraction process was based on a mix of averages(for features related to design and constraint variables) and sensitivities of coupling pairs(for feature related to coupling variables). Apart from the problem size related features, each of the features used to build the database of scaled problems could be interchangeably used on the original problems. Based on this hybrid approach, features were extracted from the original MDO problems and a database of original MDO problems was created. The re-trained prediction model was tested on the database of original problems. The prediction model was able to estimate the cost ratio of optimization on the original MDO problems with an accuracy of 84.4 percent.

6.2. Recommendations

The following points discuss certain recommendations for future that can be made from this thesis:

- **Defining optimization settings as part of feature-set**
  In the current investigation, certain key optimization settings such as choice of optimizer, convergence tolerance on optimizer/solver, and tolerance on inequality/consistency constraints were considered as constant. For training the machine learning model, it was ensured that key optimization settings are selected such that one architecture does not get inherently favored over the other and the optimized objective is similar for both approaches up to three decimals. For future investigations, these choices can be assigned as features in the form of "adjustable knobs", so that the trade-off between solution time and objective accuracy can also be brought into consideration within the prediction model.

- **Introduction of more MDO problems to create a more effectively trained prediction model** The current prediction model was constructed out of six scaled MDO problems. Five out of the six features were given a certain range and number of repetitions to create a database of scaled problems. Due to the nature of the SARF scaling process, scaled versions of MDO problems had a coupling strength which was close to the original problem. This limited the learning ability of the prediction model to certain ranges of coupling strengths . This can be remedied by introducing many more MDO problems to create a very large database of scaled problems, having a variety of coupling strengths. In keeping with the fact that MDO problems are not found in numerous quantities in literature, a possible option is to select an existing scalable problem that allows the user to vary the coupling strength and then apply the SARF based scaling methodology to create a repository of MDO problems having a range of coupling strengths.

- **Usage of more complicated machine learning algorithms**
  In modern applications, neural networks can be used for more than predicting static labels. Convolutional neural networks(CNN) are a class of machine learning algorithms used for analyzing images. Using a convolutional neural network presents the possibility for including figures like the N^2 diagram and iteration matrices within the training process of the prediction model. In the current state of the prediction model, two of the features, coupling strength(ρ) and coupling density(d) require manual computations for each MDO problem that are used for training and testing the prediction model. This labor can be averted if the problem structure itself is used for training the prediction model. There are existing studies that propose to generate a simulation workflow(in the form of a simplified graph) to extract relevant information out of an MDO problem's N^2 diagram. Such graphs can be used for training a convolutional neural network based prediction model of MDO architecture.

A

Implementation of SARF methodology on SSBJ problem

This appendix shows the implementation of the steps within SARF methodology on the SSBJ problem. For the demonstration, only the structural discipline is considered. The process is implemented in the form of six classes as shown below, followed by a UML diagram.

A.1. One-dimensional restriction class

```

import numpy as np
from ssbjkadmos.utils.math import polynomial_function



class StructureScaling:

    def __init__(self,  x_str, z, load, we, nz, wfo, wo, bounds_dict):

        self.x_str = x_str
        self.z = z
        self.load = load
        self.we = we
        self.nz = nz
        self.wfo = wfo
        self.wo =  wo

        self.bounds_dict = bounds_dict


    @staticmethod
    def y12(x_str, z, load, w_e, n_z, w_fo, w_o, component):

        # common calculations
        t = z[0] * z[5] / (np.sqrt(abs(z[5] * z[3])))
        b = np.sqrt(abs(z[5] * z[3])) / 2.0
        r = (1.0 + 2.0 * x_str[0]) / (3.0 * (1.0 + x_str[0]))

        # calculations for specific components
        if component == 0:
            fo1 = polynomial_function([x_str[1]], [1], [.008], "Fo1")
            wt_hat = load
            ww = fo1 * (0.0051 * abs(wt_hat * n_z) ** 0.557 *
                        abs(z[5]) ** 0.649 * abs(z[3]) ** 0.5 * abs(z[0]) ** (-0.4)
                        * abs(1.0 + x_str[0]) ** 0.1 * (0.1875 * abs(z[5])) ** 0.1
                        / abs(np.cos(z[4] * np.pi / 180.)))
            wfw = 5.0 / 18.0 * abs(z[5]) * 2.0 / 3.0 * t * 42.5
            wf = wfw + w_fo
            wt = w_o + ww + wf + w_e

            return wt

        if component == 1:
            theta = polynomial_function([abs(x_str[1]), b, r, load],
                                        [2, 4, 4, 3], [0.25] * 4, "twist")

            return theta


    def y12_scale(self, diagonal, component):

        x_str = self.x_str
            z = self.z
         load = self.load
           we =  self.we
           nz =  self.nz
          wfo = self.wfo
           wo =  self.wo

        # calculation of interpolated values for each component
        min_y_12 = self.bounds_dict['y12_min'][component]
        max_y_12 = self.bounds_dict['y12_min'][component]

        var = [x_str[0], x_str[1], z[0], z[1], z[2], z[3], z[4], z[5], load, we]
        y_12 = []
        for t in diagonal:
            for j in var:
                vars()[str(j)] = j[0] + t * (j[1] - j[0][0])
            wt = self.y12([vars()[str(x_str[0])], vars()[str(x_str[1])]],
                          [vars()[str(z[0])], vars()[str(z[1])], vars()[str(z[2])],vars()[str(z[3])], vars()[str(z[4])],
                           vars()[str(z[5])]], vars()[str(load)], vars()[str(we)], nz, wfo, wo, component)
            y_12.append(wt)
            y_12 = np.asarray(y_12)
        y_12 = (y_12- min_y_12) / (max_y_12 - min_y_12)

        return y_12


    def y14_scale(self, diagonal, component):

        return self.y12_scale(diagonal, component)



    @staticmethod
    def g1(x_str, z, load, component):

        if component == 0:
            b = np.sqrt(abs(z[5] * z[3])) / 2.0
            r = (1.0 + 2.0 * x_str[0]) / (3.0 * (1.0 + x_str[0]))
            sigma = 5 * [0.]
            sigma[0] = polynomial_function([z[0], load, x_str[1], b, r], [4, 1, 4, 1, 1], [0.1] * 5, "sigma[1]")

            return sigma[0]/1.09 - 1

        if component == 1:
            b = np.sqrt(abs(z[5] * z[3])) / 2.0
            r = (1.0 + 2.0 * x_str[0]) / (3.0 * (1.0 + x_str[0]))
            sigma = 5 * [0.]
            sigma[1] = polynomial_function([z[0], load, x_str[1], b, r], [4, 1, 4, 1, 1], [0.15] * 5, "sigma[2]")

            return sigma[1]/1.09 - 1

        if component == 2:
            b = np.sqrt(abs(z[5] * z[3])) / 2.0
            r = (1.0 + 2.0 * x_str[0]) / (3.0 * (1.0 + x_str[0]))
            sigma = 5 * [0.]
            sigma[2] = polynomial_function([z[0], load, x_str[1], b, r], [4, 1, 4, 1, 1], [0.2] * 5, "sigma[3]")

            return sigma[2]/1.09 - 1

        if component == 3:
            b = np.sqrt(abs(z[5] * z[3])) / 2.0
            r = (1.0 + 2.0 * x_str[0]) / (3.0 * (1.0 + x_str[0]))
            sigma = 5 * [0.]
            sigma[3] = polynomial_function([z[0], load, x_str[1], b, r], [4, 1, 4, 1, 1], [0.25] * 5, "sigma[4]")

            return sigma[3]/1.09 - 1

        if component == 4:
            b = np.sqrt(abs(z[5] * z[3])) / 2.0
            r = (1.0 + 2.0 * x_str[0]) / (3.0 * (1.0 + x_str[0]))
            sigma = 5 * [0.]
            sigma[4] = polynomial_function([z[0], load, x_str[1], b, r], [4, 1, 4, 1, 1], [0.30] * 5, "sigma[5]")

            return sigma[4]/1.09 - 1

        if component == 5:
            b = np.sqrt(abs(z[5] * z[3])) / 2.0
            r = (1.0 + 2.0 * x_str[0]) / (3.0 * (1.0 + x_str[0]))
            theta = polynomial_function([abs(x_str[1]), b, r, load], [2, 4, 4, 3], [0.25] * 4, "twist")

            return theta/1.04 - 1

        if component == 6:
            b = np.sqrt(abs(z[5] * z[3])) / 2.0
            r = (1.0 + 2.0 * x_str[0]) / (3.0 * (1.0 + x_str[0]))
            theta = polynomial_function([abs(x_str[1]), b, r, load], [2, 4, 4, 3], [0.25] * 4, "twist")

            return 0.96/theta - 1


    def g1_scale(self, diagonal, component):

        x_str = self.x_str
        z = self.z
        load = self.load

        if component == 0:
            min_g_1 = self.bounds_dict['g1_min'][component]
            max_g_1 = self.bounds_dict['g1_max'][component]

            var = [x_str[0], x_str[1], z[0], z[1], z[2], z[3], z[4], z[5], load]

            g_1 = []

            for t in diagonal:
                for j in var:
                    vars()[str(j)] = j[0] + t * (j[1] - j[0][0])
                wt = self.g1([vars()[str(x_str[0])], vars()[str(x_str[1])]],
                              [vars()[str(z[0])], vars()[str(z[1])], vars()[str(z[2])], vars()
                                  [str(z[3])],
                               vars()[str(z[4])], vars()[str(z[5])]], vars()[str(load)],
                                   component)
                g_1.append(wt)
                g_1 = np.asarray(min_g_1)
            g_1 = (g_1 - min_g_1) / (max_g_1 - min_g_1)

        return g_1
```

A.2. Interpolation class

```

import numpy as np
import pickle
from scipy.interpolate import interp1d

    class StructureInterpolation:

        def __init__(self, x):
            self.s = StructureScaling(np.array([[0.1, 0.4], [.75, 1.25]]), np.array([[0.038,
                0.06],
                                       [30000, 60000],[1 , 1.3] ,[4.5, 8.5] ,[40, 70],[700,
                                           1000]]),
                                       np.array([20000, 42000]), np.array([0, 70000]), 6.0,
                                           2000.0,
                                       25000.0, {'y12_min': [27322.03658309271,
                                           1.038799370398898],
                                       'y12_max': [156212.88815281598, 1.0873650095198497],
                                       'y14_min': [27322.03658309271, 4611.076343054285],
                                       'y14_max': [156212.88815281598, 7121.969142940493]})
            self.x = x

        def y12_int(self, component):
            y = []
            for k in self.x:
                y.append(getattr(self.s, 'y12_scale')([k], component)[0])
            return interp1d(self.x, y, kind='cubic')

        def y14_int(self, component):
            y = []
            for k in self.x:
                y.append(getattr(self.s, 'y14_scale')([k], component)[0])
            return interp1d(self.x, y, kind='cubic')

    def g1_int(self, component):
        y = []
        for k in self.x:
            y.append(getattr(self.s, 'g1_scale')([k], component)[0])
        return interp1d(self.x, y, kind='cubic')
```

A.3. Large random matrix class

```
class LargeRandomMatrix:
    """The class generated the large random dependency matrix"""

    def __init__(self, nx_large, ny_large, d_large):
        self.nx_large = nx_large
        self.ny_large = ny_large
        self.d = d_large

    def submatrix_structural(self):
        """Define the structural part of the  dependency matrix"""

        # :allocate the input variables and couplings
        x = []
        [x.append(i1) for i1 in range(self.nx_large)]
        [x.append(i2) for i2 in range(self.ny_large) for _ in range(2)]

        # :allocate the output variables and couplings
        y = []
        [y.append(i3) for i3 in range(self.ny_large) for _ in range(4)]
        [y.append(i4) for i4 in range(self.nx_large)]

        # :define the structural submatrix
        aa = np.zeros(len(x) * len(y))
        percent = int(self.d * len(aa))
        aa[0:percent] = 1
        np.random.shuffle(aa)
        aa = aa.reshape(len(y), len(x))

        return aa
```

A.4. Scaled dependency matrix class

```

class ScaledDependencyMatrix:
    """
    This class creates a component dependency matrix from a large random class with user-defined
    nx and ny values for both random and scaled-down matrix, along with the shift argument for selecting
    the projection size
    """

    def __init__(self, nx_large, ny_large, d_large):
        self.nx_large = nx_large
        self.ny_large = ny_large
        self.submatrix_structural = LargeRandomMatrix(nx_large, ny_large,d_large).submatrix_structural()

    def scaled_dependency_matrix(self, nx_scale, ny_scale, ncon_scale):
    """Scale down the dependency matrix using a fixed ny value for the large random matrix"""
    nx2 = self.nx_large
    ny2 = self.ny_large

    # :shift size for each place in the scaled component
    shift = 1

    # : scaling down 1st dependency matrix
    a1 = self.submatrix_structural  #: add bias for checking the bias
    kx = nx2 + 2 * ny2 - shift * (nx_scale + 2 * ny_scale - 1)
    b1 = np.zeros((4 * ny_scale + ncon_scale, nx_scale + 2 * ny_scale))
    for i1 in range(nx_scale + 2 * ny_scale):
        for j1 in range(4 * ny_scale + ncon_scale):
            if j1 in np.arange(0, ny_scale):
                j2 = np.random.choice(np.arange(0, ny2))
                probability = np.average(a1[j2][i1 * shift: i1 * shift + kx])
                b1[j1][i1] = np.random.choice([1, 0], p=[probability, 1 - probability])
            elif j1 in np.arange(ny_scale, 2 * ny_scale):
                j2 = np.random.choice(np.arange(ny2, 2 * ny2))
                probability = np.average(a1[j2][i1 * shift: i1 * shift + kx])
                b1[j1][i1] = np.random.choice([1, 0], p=[probability, 1 - probability])
            elif j1 in np.arange(2 * ny_scale, 3 * ny_scale):
                j2 = np.random.choice(np.arange(2 * ny2, 3 * ny2))
                probability = np.average(a1[j2][i1 * shift: i1 * shift + kx])
                b1[j1][i1] = np.random.choice([1, 0], p=[probability, 1 - probability])
            elif j1 in np.arange(3 * ny_scale, 4 * ny_scale):
                j2 = np.random.choice(np.arange(3 * ny2, 4 * ny2))
                probability = np.average(a1[j2][i1 * shift: i1 * shift + kx])
                b1[j1][i1] = np.random.choice([1, 0], p=[probability, 1 - probability])
            elif j1 in np.arange(4 * ny_scale, 4 * ny_scale + ncon_scale):
                j2 = np.random.choice(np.arange(4 * ny2, 4 * ny2 + nx2))
                probability = np.average(a1[j2][i1 * shift: i1 * shift + kx])
                b1[j1][i1] = np.random.choice([1, 0], p=[probability, 1 - probability])


        return b1
```

A.5. Component dependency graph class

```

class ComponentDependencyGraph:
    """
    This class creates a component dependency matrix from a large random class with user-defined
    nx and ny values for both random and scaled-down matrix, along with the shift argument for selecting
    the projection size
    """



    @staticmethod
    def build_component_dependency(ny_scale, ncon_scale):
        """This method defines the component dependency graph"""

        # :create two dictionaries for coupling and constraints
        d_constraint = dict()
        d_coupling = dict()

        # :The original number of components in each constraint
        original_components = [7, 2, 2]

        # :The output labels for constraints are listed
        labels = ['g_1', 'y_12', 'y_14']

        # :The dependency graph is constructed for the constraints
        dependency_graph = []
        for variable in range(np.size(original_components)):
            m, k, dependency = original_components[variable], [], []
            [k.append(1 + i1) for i1 in range(m)]
            while dependency == []:
                for i2 in range(ncon_scale):
                    dependency.append(np.random.randint(1, m + 1))
                    if i2 == n_con - 1 and set(k).issubset(set(dependency)):
                        break
                    elif i2 == ncon_scale - 1:
                        dependency = []
                        continue
            dependency_graph.append(dependency)
        j1 = 0
        for i3 in dependency_graph:
            d_constraint[labels[j1]] = i3
            j1 += 1

        # : The original number of components in each coupling output is listed
        original_components = [2, 2]

        # :The output labels are listed
        labels = ['y_12', 'y_14']

        # :The dependency graph is constructed for the coupling vectors
        dependency_graph = []
        for variable in range(np.size(original_components)):
            m, k, dependency = original_components[variable], [], []
            [k.append(1 + i1) for i1 in range(m)]
            dependency = []
            for i2 in range(ny_scale):
                dependency.append(np.random.randint(1, m + 1))
            dependency_graph.append(dependency)

        # :The component dependency is named as a dictionary in the format "label: component"
        j1 = 0
        for i3 in dependency_graph:
            d_coupling[labels[j1]] = i3
            j1 += 1

        d = d_constraint.copy()
        d.update(d_coupling)

        return d
```

A.6. Extrapolation class

```
from interpolated_functions import StructureInterpolation
from vanaret import ScaledDependencyMatrix, ComponentDependencyGraph


class StructureExtrapolation:


    def __init__(self):

        self.str_int = StructureInterpolation()  # This class gives component wise
```

```
            interpolation for structural outputs
        self.dependency_matrix = ScaledDependencyMatrix(nx_large, ny_large, structural).scaled_dependency_matrix(nx_scale, ny_scale, ncon_scale)
        self.component_dependency = ComponentDependencyGraph().build_component_dependency(ny_scale, ncon_scale)


    def y12_ext(self, nx, ny, ncon, x_des):
        """This output corresponds to total weight(WT) and temperature ratio(theta)"""

        [c_d, a1, output] = [self.component_dependency['y_12'], self.dependency_matrix, []]
        for i in range(ny):
            [sum_i, row] = [[], a1[2 * ny + i]]
            sum_i.append(np.sum(row))
            [assign, y] = [c_d[i], []]
            # :x_des = np.random.random_sample(4 * nx + 5 * ny)  # this is an instance of the design vector
            [y.append(self.str_int.y12_int([x_des[k]], assign - 1)) for k in range(4 * nx + 5 * ny) if row[k] == 1]
            output.append(np.sum(y) * 1 / sum_i)

        return output

    def y14_ext(self, nx, ny, ncon, x_des):
        """This output corresponds to total weight(WT) and fuel weight(WF)"""

        [c_d, a1, output] = [self.component_dependency['y_14'], self.dependency_matrix, []]
        for i in range(ny):
            [sum_i, row] = [[], a1[3 * ny + i]]
            sum_i.append(np.sum(row))
            [assign, y] = [c_d[i], []]
            # :x_des = np.random.random_sample(4 * nx + 5 * ny)  # this is an instance of the design vector
            [y.append(self.str_int.y14_int([x_des[k]], assign - 1)) for k in range(4 * nx + 5 * ny) if row[k] == 1]
            output.append(np.sum(y) * 1 / sum_i)

        return output

    def g1_unscaled(self, nx, ny, ncon, x_des):
        """This output corresponds to structural and temperature ratio constraints"""

        [c_d, a1, output] = [self.component_dependency['g_1'], self.dependency_matrix, []]
        for i in range(ncon):
            [sum_i, row] = [[], a1[4 * ny + i]]
            sum_i.append(np.sum(row))
            [assign, y] = [c_d[i], []]
            # :x_des = np.random.random_sample(4 * nx + 5 * ny)  # this is an instance of the design vector
            [y.append(self.str_int.g1_int([x_des[k]], assign - 1)) for k in range(4 * nx + 5 * ny) if row[k] == 1]
            output.append(np.sum(y) * 1 / sum_i)

        return output

    def g1_ext(self, nx, ny, ncon, x_des):
        """scaling the constraints from violated to active/inactive status"""

        # :evaluating unscaled constraints at the initial point
        g_1_0 = self.g1_unscaled(nx, ny, ncon, .5 * np.ones(4 * nx + 5 * ny))

        # :evaluating unscaled constraints:
        g_1 = self.g1_unscaled(nx, ny, ncon, x_des)

        # :define the threshold "tow" to translate the scaled constraint
        # :alpha determines to what extent the inactive constraints are satisfied
        tow, alpha = [], self.alpha_g1
        [tow.append(i) if self.mu_g1[list(g_1_0).index(i)] < self.p[list(g_1_0).index(i)] else
             tow.append(alpha[list(g_1_0).index(i)] + (1 - alpha[list(g_1_0).index(i)]) * i)
            for i in g_1_0]

        # define the translated constraint
        g_1_translated = []
        [g_1_translated.append(g_1[i] - tow[i]) for i in range(ncon)]
        return g_1_translated
```

A.7. Script to calculate coupling strength for a Problem

To calculate the coupling strength for the MDO problem, it is required to make changes to the *_linearize* method of the *DirectSolver* class within the OpenMDAO code, to accumulate the system Jacobian.

An excerpt is shown from the code representing the required changes:

```
import pickle
# :Three pickled files are created, each representing the
# :accumulated jacobian, MDA time and iteration counter.
class DirectSolver(LinearSolver):

    system = self._system()
    nproc = system.comm.size

    if self._assembled_jac is not None:
        matrix = self._assembled_jac._int_mtx._matrix
        with open("matrix_accu.p", "rb") as f:
            matrix_accu = pickle.load(f)
        matrix_accu.append(matrix)
        with open("matrix.p", "wb") as f:
            pickle.dump(np.array(matrix_accu), f)

        with open("time_accu.p", "rb") as f:
            time_accu = pickle.load(f)
        time_accu.append(time_accu[-1] - time.time)
        with open("time_accu.p", "wb") as f:
            pickle.dump(time_accu, f)

        with open("iter_counter.p", "rb") as f:
            iter_counter = pickle.load(f)
        iter_counter += 1
        with open("iter_counter.p", "wb") as f:
            pickle.dump(iter_counter, f)
        .
        .
        .
        .
```

The coupling strength(rho) is computed by loading the pickled jacobian matrix and applying the following script:

```
for i in np.arange(0, len(pickle.load("iter_counter.p", "rb")):
    a = pickle.load(open("matrix_accu.p", "rb"))
    a = a.toarray()
    l = np.dot(np.tril(a, k=0), -1)
    l = inv(l)
    u = np.triu(a, k=1)
    G = np.matmul(l, u)
    spectral_radius = (max(abs(LA.eig(G)[0])))
    # vars()['spectral_radius' + str(j)].append(spectral_radius)
    spectral_radius_accu.append(spectral_radius)

time_counter = pickle.load(open("time_accu.p", "rb"))

for g in range(len(spectral_radius_accu)):
    spectral_radius_accu[g] = spectral_radius_accu[g] * (time_counter[g] / sum(time_counter))
rho = sum(spectral_radius_accu)
```

A.8. UML diagram for SARF method

The following UML diagram(Figure A.1)represents the six classes used in implementing the SARF methodology:

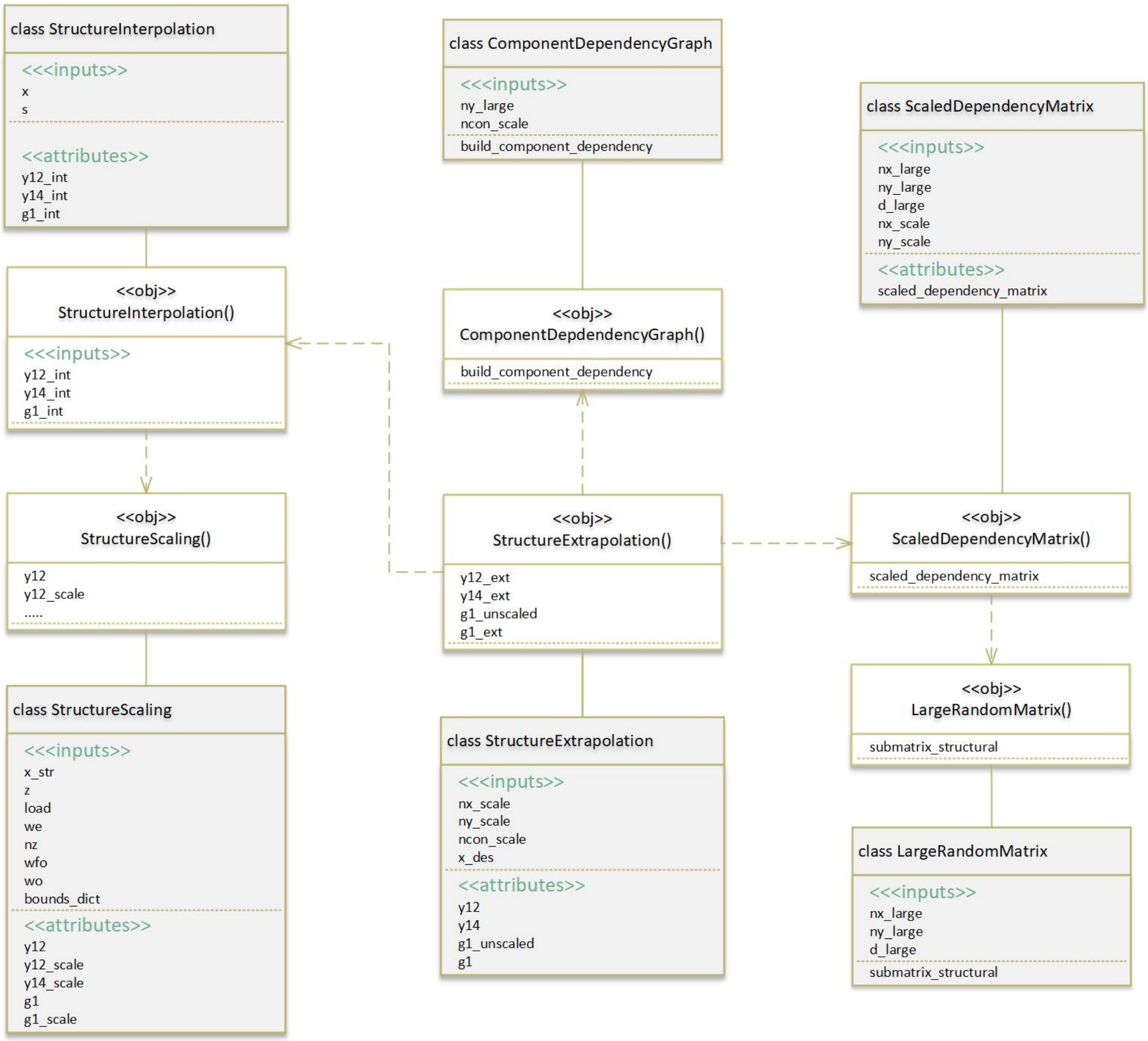


Figure A.1: UML diagram for SARF methodology

A.9. Downscaling of Large Dependency Matrix

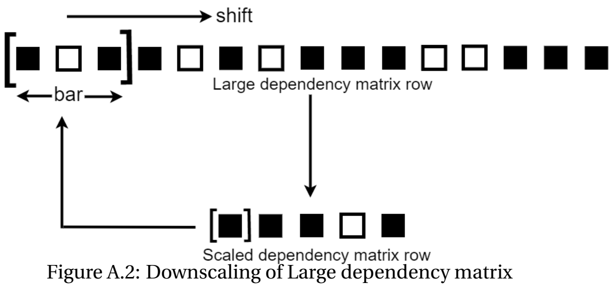


Figure A.2: Downscaling of Large dependency matrix

Figure A.2 shows two dependency matrix rows, corresponding to the large dependency matrix and the scaled dependency matrix. There are two basic parameters involved in the down-scaling process:

1. A parameter bar(b) that defines the number of consecutive blocks from the large dependency matrix row used to calculate the value of each block of the scaled dependency matrix row(For the sample representation of Figure A.2, $b = 3$).
2. A parameter shift(s) that defines the number of units by which the bar(b) bracket is shifted to calculate the outcome of the following blocks of the scaled dependency matrix.

The down-scaling process is explained below:
Firstly, the value of shift(s) is calculated using the following formula:

$s = |(\frac{N}{n})|_{odd}$

where N and n refer to the number of blocks present in the large and scaled dependency matrix rows respectively. $|x|_{odd}$ refers to the greatest odd integer function.

Following this the value of bar(b) is calculated as follows:

$b = N - s(n - 1)$

As mentioned above, the value of b comes to be 3. Using the value of bar(b) and shift(s), the value of each block of the scaled dependency matrix can be determined. If more than fifty percent of blocks within each bar of the large dependency matrix row have filled blocks(representing presence of a dependency), then the corresponding block in the scaled dependency matrix row is determined to be filled. As can be seen in Figure A.2, all but the 4th block of the scaled dependency matrix row show the existence of a dependency.

B

Optimization and Tolerance Parameters

This appendix shows the derivation of optimization and tolerance parameters performed during the reproducibility study of the investigation by Vanaret et. al.[16]. Optimization settings for executing the scaled SSBJ problem should be such that a fair comparison can be made between MDF and IDF approaches, within a reasonable time frame. These settings are sub-categorized into three parts - optimizer/ solver tolerance, constraint tolerances, and bounds on design variables. The following subsections are used to derive these settings.

B.0.1. Optimizer/ Solver Tolerance

Considering default values for the optimization settings(present in OpenMDAO) shown in table B.1, the scaled SSBJ MDO problem is run for three sets of problem parameters(n_x, n_y, d) as shown in Figure B.2. The tolerance on consistency constraint is assigned to be the same as the default absolute tolerance on the MultiDisciplinary Analysis (MDA) solver. It must also be noted that for the entirety of this OpenMDAO based project, the 'ScipyOptimize' driver is used with the SLSQP optimization algorithm, while the MDA solver to be used for the reproducibility study is the Non-Linear Block Gauss-Seidel convergence scheme(NLBGS). Unless stated otherwise, NLBGS is the convergence scheme used for the rest of the thesis project as well.

Converger settings	**Default Value**
Termination tolerance for Optimizer	1e-6
Absolute error tolerance for MDA Solver(NLBGS)	1e-10
Tolerance on consistency constraints	same as absolute tolerance on solver

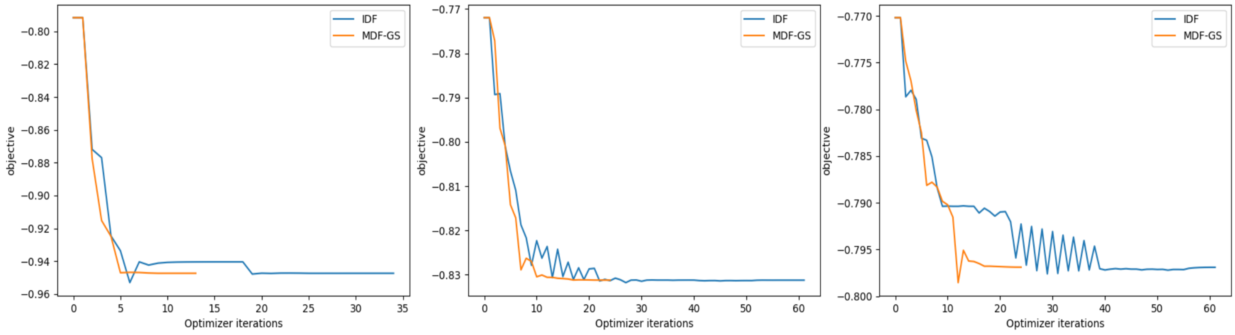
Table B.1: Default tolerance values in OpenMDAO

Final Objective	**MDF-GS**	**IDF**
$(n_x, n_y, d) = (2, 2, .3)$	-0.9473793033199817	-0.9473804810309808
$(n_x, n_y, d) = (5, 5, .4)$	-0.8312210912928233	1-0.8312312600799286
$(n_x, n_y, d) = (6, 16, .7)$	-0.7968935027373686	-0.7969081721897459

Table B.2: Final Objective Values

Firstly, the final objective value is noted in all three cases and is found to be comparable for both architectures up to three decimals as shown in Table B.2. Following this, the convergence history for the objective function is plotted as a function of the optimizer(driver) iterations(Figure B.1).

For each of the convergence plots in Figure B.1, it can be seen that MDF-GS results in far fewer optimizer iterations compared to IDF, which continues for a large number of iterations, a phenomenon amplified by higher coupling density as shown in Figure B.1c. An appropriate reason for this could be the difficulty that the optimizer faces in satisfying the consistency constraints at convergence. A similar trend is observed while plotting the norm of the residuals as shown in Figure B.2.

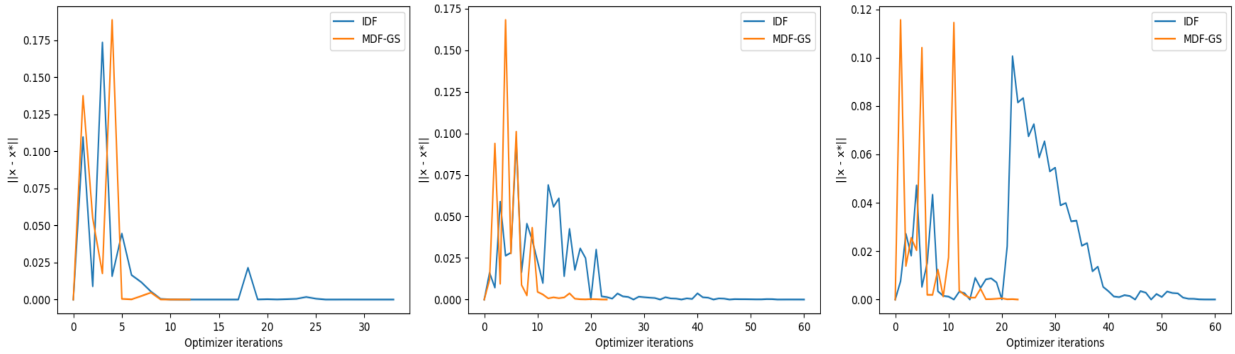


(a) $(n_x, n_y, d) = (2, 2, .3)$

(b) $(n_x, n_y, d) = (5, 5, .4)$

(c) $(n_x, n_y, d) = (6, 16, .7)$

Figure B.1: Objective Convergence history vs optimizer iterations (default OpenMDAO settings)



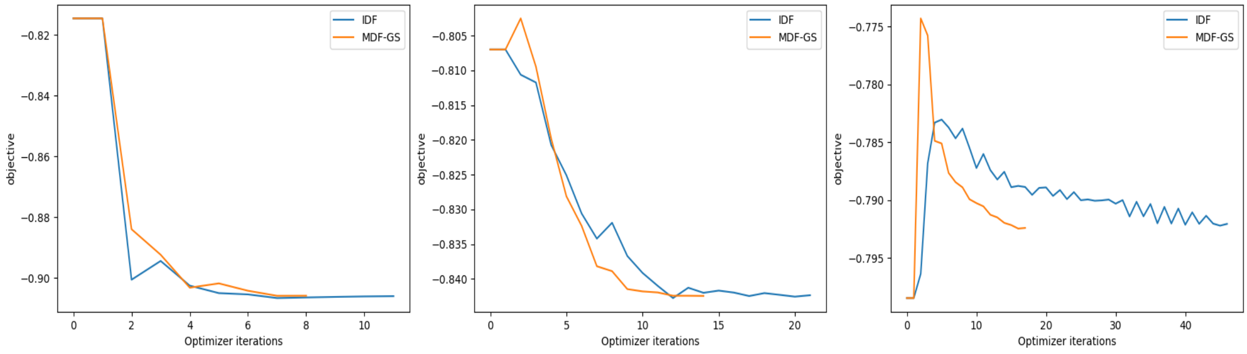
(a) $(n_x, n_y, d) = (2, 2, .3)$

(b) $(n_x, n_y, d) = (5, 5, .4)$

(c) $(n_x, n_y, d) = (6, 16, .7)$

Figure B.2: Residual vs optimizer iterations(default OpenMDAO settings)

The norm of residuals plot seems to indicate that near-zero norm of residuals(on the design vector) are achieved at about the same number of iterations for both IDF and MDF, yet IDF continues for a larger number of iterations. It can thus be observed that the default settings lead to extra, redundant optimizer iterations for IDF, which might bias the prediction model to recommend MDF. One way to remedy this is to reduce the termination tolerance value for the Scipy Optimizer from 1e-6 to 1e-4. The analysis is repeated with reduced tolerances and the convergence history and the norm of residuals are again plotted as shown in Figure B.3 and B.4.

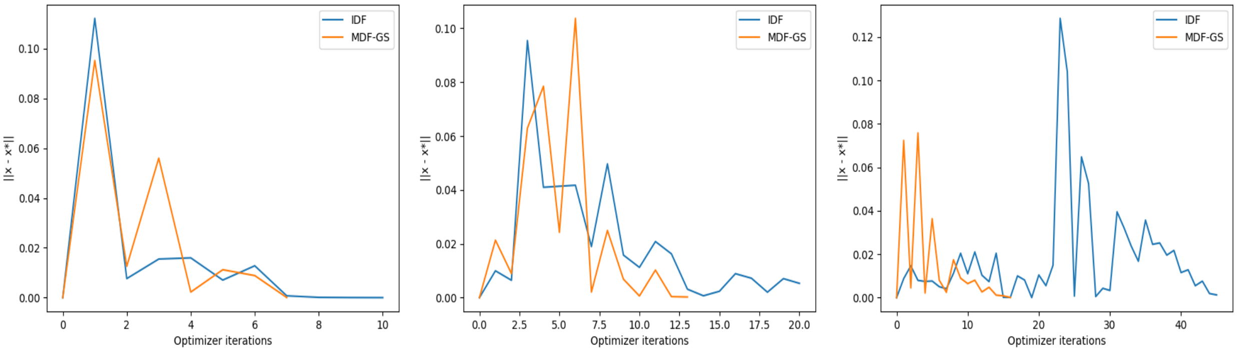


(a) $(n_x, n_y, d) = (2, 2, .3)$

(b) $(n_x, n_y, d) = (5, 5, .4)$

(c) $(n_x, n_y, d) = (6, 16, .7)$

Figure B.3: Objective vs optimizer iterations (optimizer tolerance = 1e-4)



(a) (n_x, n_y, d) = (2, 2, .3)

(b) (n_x, n_y, d) = (5, 5, .4)

(c) (n_x, n_y, d) = (6, 16, .7)

Figure B.4: Residual vs optimizer iterations(optimizer tolerance = 1e-4))

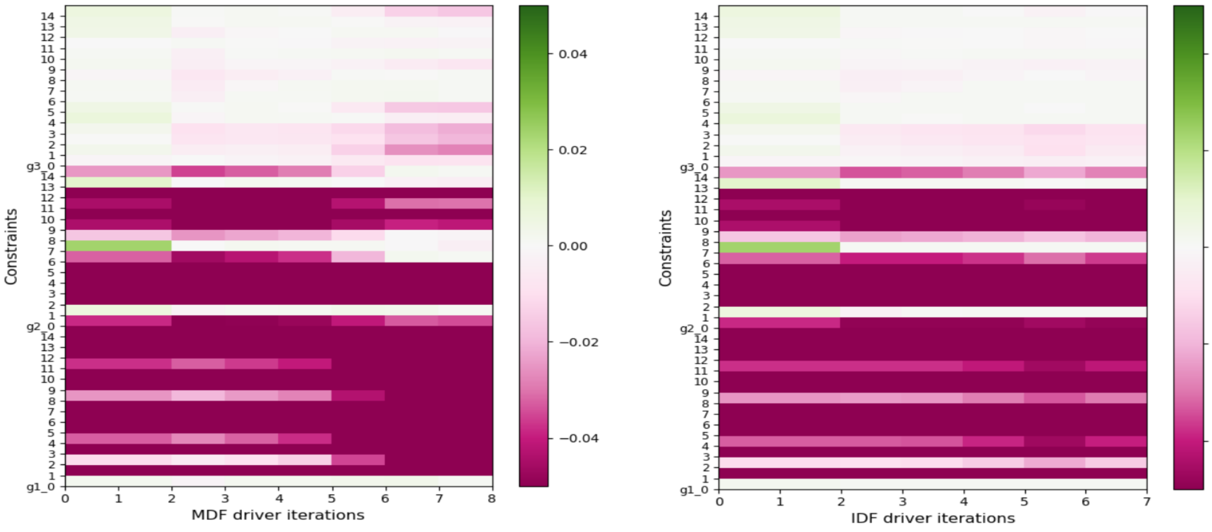
The extra iterations for IDF, in case of reduced optimizer tolerance, is found to be much reduced, while the final objective is still equal within three decimal places as can be seen in Figure B.3. Therefore the termination tolerance on the optimizer is selected to be 1e-4. Based on the above observations, the converger settings to be used in the reproducibility study are shown in Table B.3.

Converger settings	**Value**
Termination tolerance for Optimizer	1e-4
Absolute error tolerance for MDA Solver(NLBGS)	1e-10
Tolerance on consistency constraints	1e-10

Table B.3: Converger settings for reproducibility study

B.0.2. Constraint Tolerance

The route taken to handle disciplinary constraints is quite similar for MDF and IDF architectures. In both cases, disciplinary constraints are evaluated directly by the optimization algorithm. It is therefore not likely that constraint tolerance values(tol_{con}) would have a large influence on the relative solution time of MDF and IDF architectures. It is clear that if one or more constraints are active at the optimal(final) design point, the value of the final objective would change when the constraint tolerance value is altered. Ideally, it would be desirable to select a value of tol_{con} that does not induce any specific bias on the optimal design point for any architecture. At the same time, the tolerance on constraint should not be so strict that it drags the overall computational cost too high and/or create issues with driver convergence(optimizer failing to terminate within the given iteration limit). The effect of constraint tolerance is investigated by running a particular version of the scaled SSBJ problem (using a fixed dependency matrix and component dependency graph), for a range of constraint tolerance values. Using parameters $(n_x,\ n_y,\ d)$ = $(15, 9, 0.5)$, a particular version of the scaled SSBJ problem is generated. The problem is then run using MDF and IDF approaches for a range of constraint tolerance values from 1e-3 to 1e-6. Figure B.5 shows a plot of constraint history vs optimizer iteration for a constraint tolerance value of 1e-3.

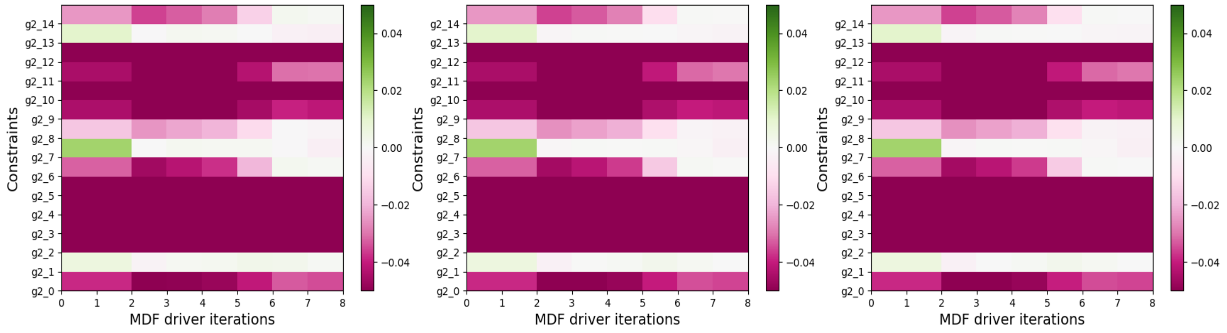


(a) MDF

(b) IDF

Figure B.5: History of local constraint($tol_{con} = 1e-3$)

The original SSBJ problem comprises of three constraint vectors g_1, g_2, and g_3, one for each of the three coupled disciplines as explained before. Within the scaled version of the problem, each of the constraint vectors is extrapolated to n_x dimensions. In Figure B.5 each row represents a particular output dimension(component). For simplicity, just the components of g_2 can be considered in the analysis because it shows a mix of active and inactive constraints at the optimal design point. The following set of figures(Figure B.6 and Figure B.7) show the effect of gradually increasing the constraint tolerance.

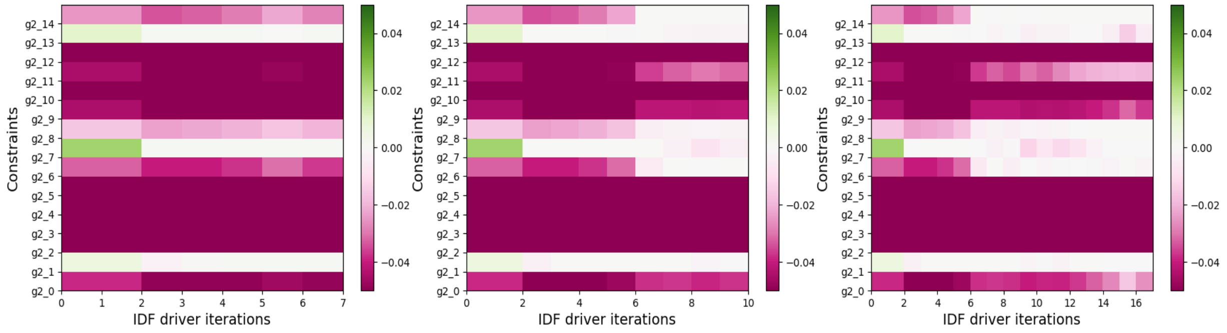


(a) $tol_{con} = 1e-3$

(b) $tol_{con} = 1e-5$

(c) $tol_{con} = 1e-6$

Figure B.6: History of local constraint $g2$ for MDF



(a) $tol_{con} = 1e-3$

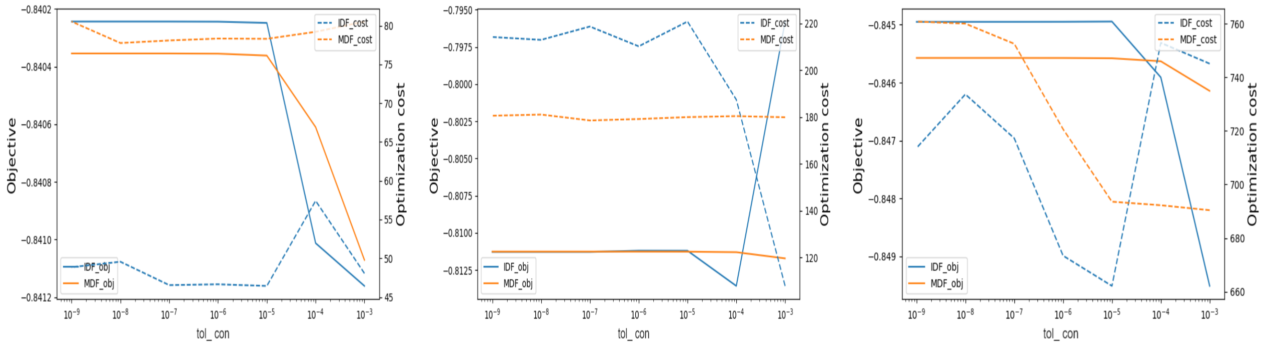
(b) $tol_{con} = 1e-5$

(c) $tol_{con} = 1e-6$

Figure B.7: History of local constraint $g2$ for IDF

It can be seen in Figure B.6 that for MDF, while moving from left to right(increasing the tolerance strictness), the plot of constraints does not show a significant change. For the three plots, the number of optimizer iterations remain the same. Additionally, the criticality of constraints at the final design point does not seem to be dependent on the constraint tolerance value. The constraints that seem to be active at the final iteration(like g_{21}) are the same for all three tol_{con} values.

In contrast, when a similar set of plots is constructed for IDF as shown in Figure B.7, a considerable variation can be observed when moving from relaxed to stricter constrained tolerances. The number of optimizer iterations is not independent of the constraint tolerance values. Comparing Figure B.7a with Figure B.6a, for $tol_{con} = 1e-3$, IDF seems to converge to a different optimal point altogether with fewer active constraints compared to MDF. As tol_{con} is reduced, greater similarity is observed between the corresponding MDF and IDF plots with respect to the overall history of constraints as well as activation status of constraints at the final point. This would seem to suggest that a highly relaxed constraint tolerance value of 1e-3 is not suitable for comparing MDF and IDF approaches over a range of parameters. In order to get a more detailed picture, it is also required to compare the effect of tol_{con} on the final objective values and solution times. Figure B.8b represents three semi logarithmic plots wherein the final objective and solution cost(calculated according to Section 3.3.1) are plotted co-axially versus a range of constraint tolerance values from 1e-3 to 1e-9. Figure B.8b is made for the above mentioned scaled problem $((n_x, n_y, d) = (15, 9, 0.5))$ while Figure B.8a and Figure B.8c represent two more scaled problems of lower and greater problem sizes.



(a) (n$_x$, n_y , d) = (5, 5, 0.3)

(b) (n$_x$, n_y , d) = (15, 9, 0.5)

(c) (n$_x$, n_y , d) = (17, 17, 0.7)

Figure B.8: Final Objective/ Solution cost vs tol_{con}

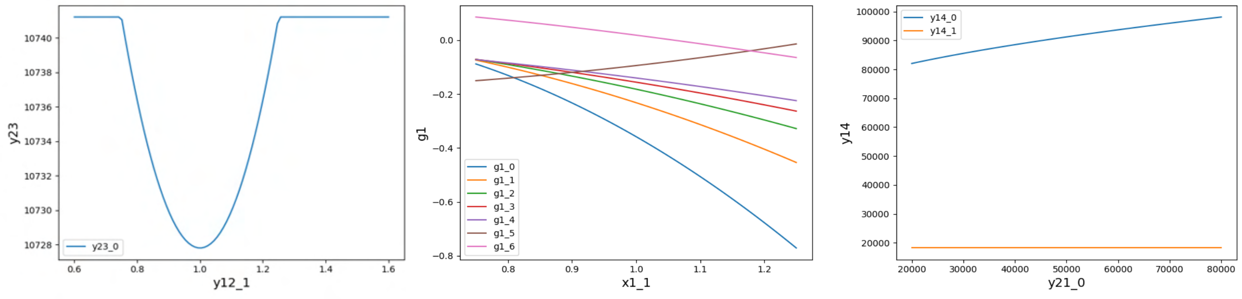
It can be seen in Figure B.8b that for $tol_{con} = 1e-3$, the final objective values are largely different. Lowering the constraint tolerance value to 1e-5, the final objective values come much closer to being within three decimals of accuracy. The accuracy is expectedly maintained for lower tol_{con} values. This in agreement with the findings discussed previously from the constraint history plots. A similar situation can be seen for a smaller scaled problem shown in Figure B.8a. The total time of optimization in both the plots stays nearly constant for lower values of tol_{con}. However, for greater problem sizes as shown in Figure B.8c, lowering the constraint tolerance to below 1e-5 leads to a proportionately larger increase in optimization cost, which is undesirable for performing multiple runs. Therefore, for building the prediction model a constraint tolerance value of 1e-5 is selected.

B.0.3. Bounds on Design Variables

In the SARF methodology, the procedure used to extrapolate each disciplinary interface ensures that every disciplinary input and output is contained within 0 and 1(except for constraints that need to be translated according to the rule discussed in Section 3.1.3.3). However, before deriving extrapolated outputs, it is required to create an interpolated library of disciplinary interfaces as a three-step process shown in Figure 3.18. The middle step (scaling of unidimensional outputs) requires the selection of certain lower and upper bounds on the original design and coupling variables. The values for these bounds is initially estimated from two sources, which include a technical report by Sobieszczanski et al. [18](Table A1) and the SSBJKADMOS python package [1]. Now, based on these bounds, it is possible to plot “representative cuts" of the SSBJ problem, wherein the individual contribution of each design variable on a dependent coupling can be assessed.

[1] Imco Van Gent. Ssbjkadmos. URLhttps://pypi.org/project/ssbjkadmos.

Based on the dependency matrix for the original unscaled SSBJ problem shown in Figure 3.8b, it is possible to make one hundred forty of such plots, six of which are shown in Figure B.9 and Figure B.10.

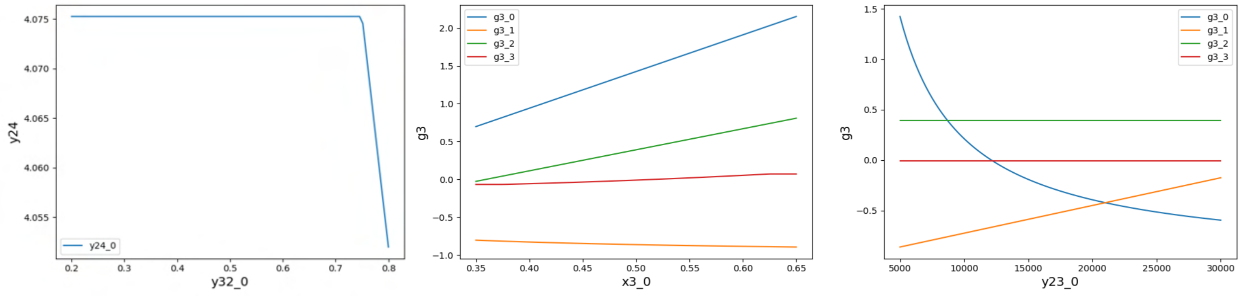


(a)

(b)

(c)

Figure B.9: Representative cuts of original SSBJ problem



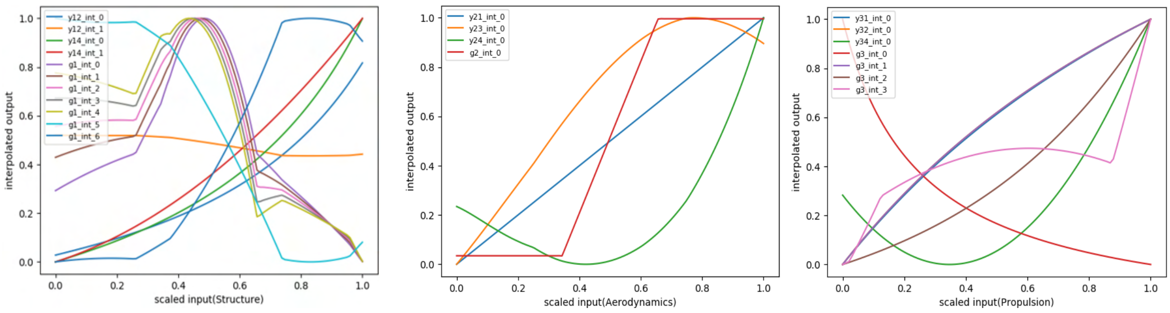
(a)

(b)

(c)

Figure B.10: Representative cuts of original SSBJ problem

Looking at Figure B.9a and Figure B.10a, it can be seen that the output functions $y23_0 = f(y12_1)$ and $y24_0 = f(y32_0)$ are quadratic on certain ranges and constant on others. In the paper by Vanaret et al.[16], it is mentioned that in order to avoid convergence issues with a gradient based optimizer like SQP, the bounds on the design variables as well as couplings should be cropped to exclude the ranges where an output is constant. The specific bounds to be used for the comparative study is however not provided in the paper. In order to settle on an appropriate value of bounds, a manual trial and error based process can be followed. Using the original bounds estimated from the sources mentioned before([18] and [58]), an interpolated library of output functions is generated as shown in Figure B.11. Interpolated outputs for each of the three coupled disciplines are shown separately.

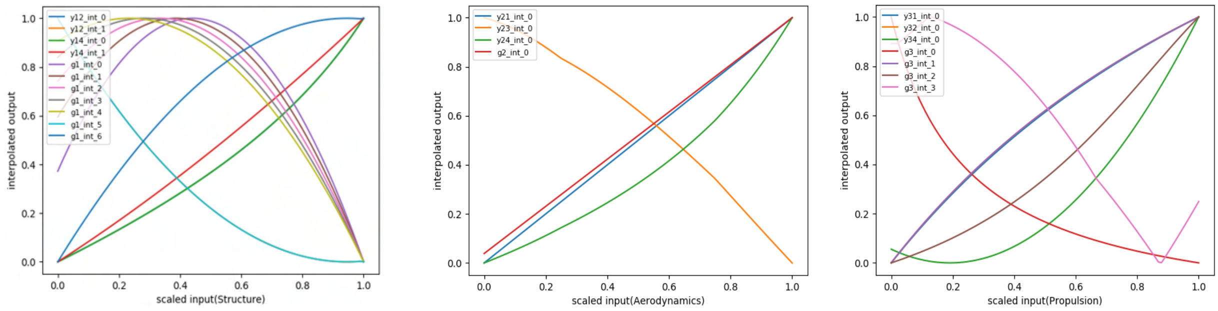


(a) Structure outputs

(b) Aerodynamics outputs

(c) Propulsion outputs

Figure B.11: Interpolated library of output functions(Original Bounds)



(a) Structure outputs

(b) Aerodynamics outputs

(c) Propulsion outputs

Figure B.12: Interpolated library of output functions(Restricted Bounds)

Based on the interpolated library of outputs using original bounds(Figure B.11), a scaled SSBJ problem can be set up and executed for a given set of parameters. Considering (n_x , n_y , d) = (17, 10, 0.4), Figure B.13 shows the history of objective function for MDF and IDF architectures using original bounds. It can be seen that the architectures converge on different final solution which is undesirable for the comparative study. This seems to suggest that the gradient based optimization algorithm is not capable of handling the scaled SSBJ problem wherein the coupling outputs show drastic changes in convexities across the interpolation range. This problem is compounded due to selection of parameters such that the overall number of design variables are high(for $n_x = 17$ and $n_y = 10$, total number of design variables = $4 * n_x + 5 * n_y = 118$)[16].

Now, using a manual approach the bounds on the original outputs are gradually reduced so that the interpolated library of output functions are restricted to ranges where the output functions are smooth with constant convexity throughout. Figure B.12 shows the interpolated library of output functions for one such combination of restricted bounds. Based on the interpolated library of outputs using restricted bounds(Figure B.12), the scaled SSBJ problem is executed for MDF and IDF approaches(keeping other problem settings the same as that of the problem executed using original bounds). The history of the objective function for both architectures is again plotted in Figure B.13.

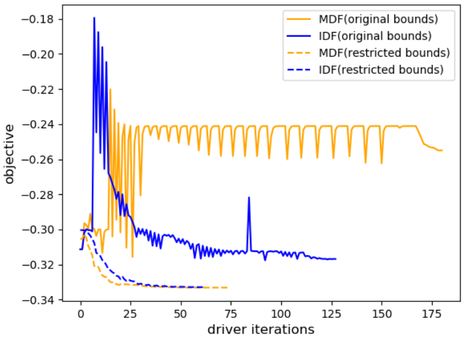


Figure B.13: history of objective function for extended and reduced bounds

It can be observed that in the case of restricted bounds, the final solution obtained for MDF and IDF is the same(to within 3 decimal places). Also, the total number of optimizer iterations for both architectures is reduced compared to their counterparts using original bounds. Therefore, the interpolated library of outputs shown in Figure B.12 is used for the reproducibility study.

C

Building Linear Regression, KNN and Decision Tree based Machine Learning Models

C.1. Linear Regression

Multiple linear regression can be explained as an extension of a linear regression model that can accommodate multiple predictors. An example of linear regression using two features(X1 and X2) and one label(Y) in a three-dimensional space is shown in Figure C.1[37]. The red dots in the space represent the output points from a test database, while the 2-dimensional plane represents the predicted output of the regression model. The vertical black lines represent the residual or the difference between the predicted and the actual output. The predicted output can be stated as:

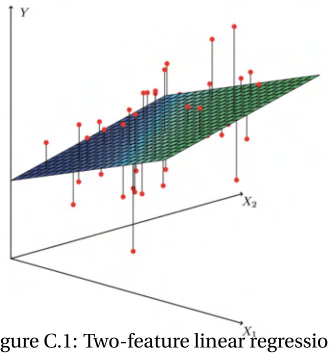


Figure C.1: Two-feature linear regression

$y_{pred} = \beta_0 + \beta_1 * X1 + \beta 2 * X2.$

$\beta_i, i \in \{1,2\}$ stands for the weight attached to each predictor while β_0 stand for the constant intercept.

The β values, referred to as regression coefficients or weights act as design variables for the minimization problem at the core of the regressor model. Referred to as regression coefficients, these β values also signify the level of relationship between the features and the response label. For every tested data point, a residual can be calculated which is the difference between the observed value and the fitted plane.

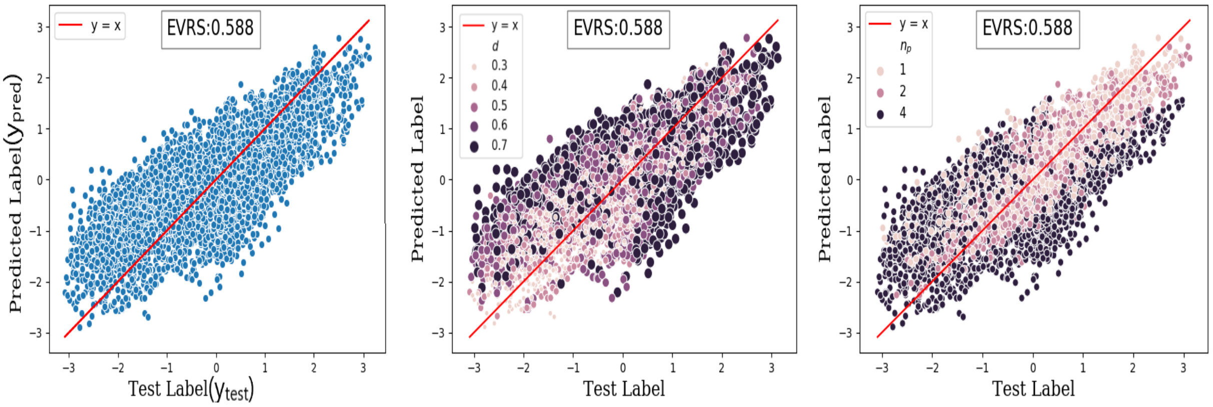
As can be seen in the database (Table 4.4), for the current application, there are a total of five predictors, while y_{test} and y_{pred} refer to the actual and predicted cost ratios respectively. Using these five predictors, a linear regressor model is defined as follows:

$$y_{pred} = \beta_0 + \beta_1 * n_x + \beta_2 * n_y + \beta_3 * n_c + \beta_4 * d + \beta_5 * n_p \tag{C.1}$$

The linear regression algorithm is fitted on the normalized database of SSBJ based scaled MDO problems(Table 4.4). The output consists of the optimized constant intercept(β_0) and the optimized weight values ($\beta_i, i \in \{1,2,3,4,5\}$) attached to each feature. Using these values, the regressor line can be constructed as follows:

$$y_{pred} = -2.8 + 0.077 * n_x + 0.070 * n_y + 2.695 * d + 0.031 * n_c - 0.35 * n_p \tag{C.2}$$

Figure C.2a shows the observed vs predicted output of the above linear regression model and the calculated EVRS.



(a) Label - Test vs Predicted label(cost ratios)

(b) label - grouping by d

(c) label - grouping by n_p

Figure C.2: Performance evaluation of linear regression

As can be seen in Figure C.2a, the predicted labels are found to be largely scattered around the line $y = x$. The Explained Variance Regression Score(EVRS) is only 0.588 which represents a very low prediction accuracy. Looking at the β coefficients in Equation C.2, it can be seen that the value(magnitude) of β for the feature representing the coupling density factor d, ($\beta_3 = 2.695$) and the number of processors (($\beta_5 = -0.35$)) are much higher than the other features, which indicates that the linear model considers these two features to be comparatively more decisive predictors for the response label compared to other features[1]. This defies the observations made in the previous chapter(Section 4.1), where the feature concerning the number of coupling variables (n_y) was found to be having the most direct influence on the label. From a data science perspective, this shows that a lack of compatibility exists between the observations made in the manual exploratory data analysis and the prediction obtained from the machine learning model, thereby indicating that more complicated machine learning techniques might be required to create a better prediction model. Investigating the cause of the low prediction accuracy of the model, a color mapping bias is applied to the data points on the regression model, which segregates the data points based on the coupling density factor and the number of processors. This is shown in Figure C.2b and C.2c. It can be seen from Figure C.2b, that the data points representing higher coupling densities are more likely to be found further away from the line $y = x$ compared to the data points representing lower coupling densities. A similar trend can be observed from Figure C.2c, where the model fails to predict accurately the result of the test cases created using higher processor cores.

C.2. KNN - K Nearest Neighbours Algorithm

K Nearest Neighbours is a machine learning algorithm that can be used to build a regression model. Enabled by sci-kit learn's $KNeighborsRegressor$ class, it works on a principle that can be explained using a reduced version of the SSBJ based database shown in Figure 4.4. By considering only two features n_x and n_y(keeping all other features constant), Figure C.3 shows a two dimensional plot representing the cost ratio distribution of MDF and IDF architectures. A total of thirty random data points $\{n_x, n_y\}$ are plotted for training the model. A test point, having an unknown outcome is placed at the point $\{n_x, n_y\} = \{12, 12\}$. The 1^{st} training point, closest to the unknown test point, is used for validation. To predict the outcome of a test point, the KNN model relies on information from nearby elements in the n-dimensional space. The model makes use of a user-defined parameter k which indicates the number

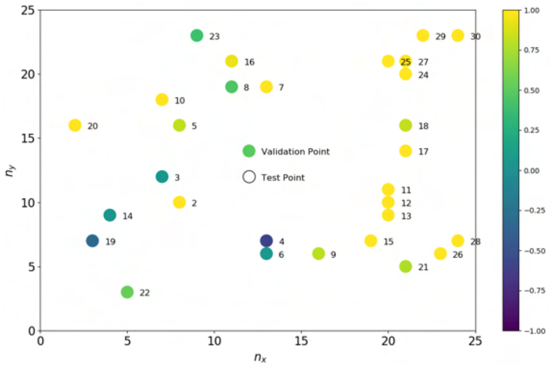
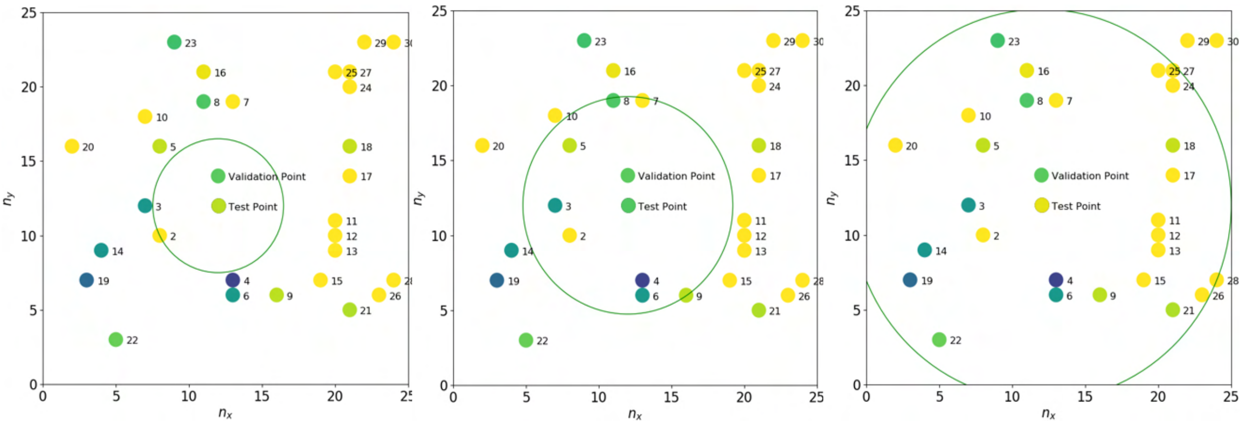


Figure C.3: Cost Ratio distribution over two features

[1] https://statisticsbyjim.com/glossary/regression-coefficient/

of nearby training elements from which label information is to be extracted. The label value assigned to the test point is the mean of the label values of the extracted test points. The following set of plots(Figure C.4) show the effect of changing the k parameter on the outcome of the test point.



(a) $k = 2$ (b) $k = 9$ (c) $k = 28$

Figure C.4: Effect of k parameter on the test point

In each of the above figures, the training points are labeled according to the increasing radial distance between the test point and the training points. For three values of k, ($k \in \{2, 9, 28\}$), a circle is used to represent the training points that are relevant for estimating the outcome of the training labels. Each of the training points within the circle is considered to be an equal contributor to the outcome of the test point. The KNN algorithm computes the test label as the mean of the label values of k nearest neighboring training points. The actual value of the test point can be considered to be close to that of the validation point. Now, if a very low value of k is selected, as shown in Figure C.4a, the outcome of the test point might be too dependent on the nearest training points, leading to the possibility of overfitting. In such circumstances, the model tends to get fitted too accurately to the training data points, and the prediction on new test points is much lower. At the other end, as shown in Figure C.4c, for a very high value of k, the prediction might be very inaccurate because of the variance introduced in the model by some of the far-off components. Because of this trade-off between training and test error rates, it is required to find out the optimum value of the k parameter. The appropriate value of k varies according to the type of data-set and can be estimated using a hit and trial process. For the SSBJ data set, this estimation is performed by varying the k value from zero to thirty, fitting the KNN model for each value of k, and observing the fitting accuracy for training and test data points. The result is plotted in Figure C.5.

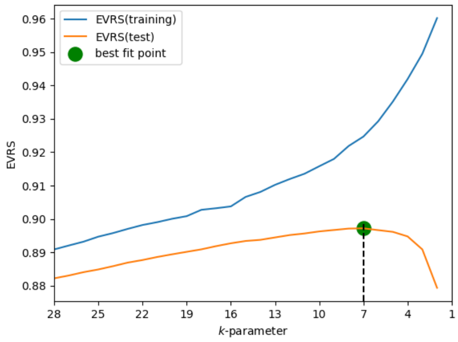


Figure C.5: Estimation of optimum k value for SSBJ database

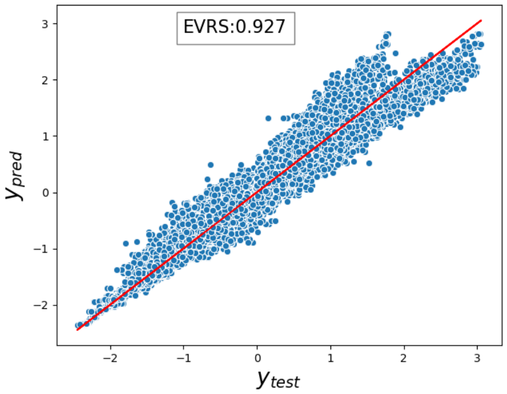


Figure C.6: Performance evaluation of KNN($k = 7$)

To measure the fitting accuracy, the same EVRS metric discussed before is utilized. It can be seen in Figure C.5, that the EVRS score for the training data points exponentially increases with decreasing k value. However, for test data points, the best fit is observed for $k = 7$. Using this value of k, Figure C.6 shows the performance evaluation of the KNN regressor model, along with the EVRS score. The prediction accuracy, in this case, is higher than linear regression.

C.3. Decision Tree

Tree methods are a class of machine learning algorithms that involve dividing the feature space into distinct segments according to a set of splitting rules, which can be summarized in the form of a binary tree[37]. Initiated by sci-kit learn's *DecisionTreeRegressor* class, a decision tree consists of nodes, representing a feature space, connected by edges. Starting with a singular root node, the feature space is progressively divided into distinct, non-overlapping segments, with terminal nodes being assigned a unique label value. The method can be demonstrated using a reduced version of the SSBJ based database. Using n_x and n_y, a two feature plot can be created(considering other features to be constant), representing the cost ratio(R_t) distribution of MDF and IDF architectures. This distribution is shown in Figure C.7. Following a train test split of 1:2, a third of the points are considered to be test points shown by empty circles.

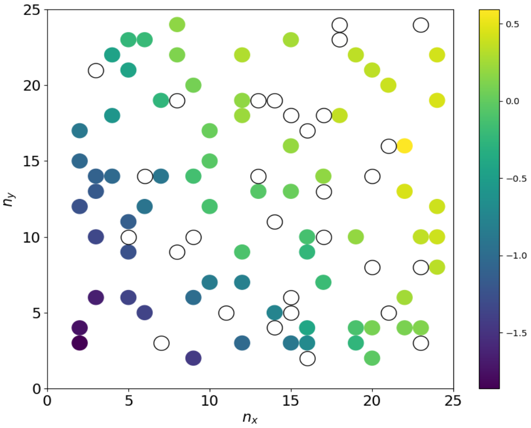


Figure C.7: Cost Ratio(R_t) distribution over two features

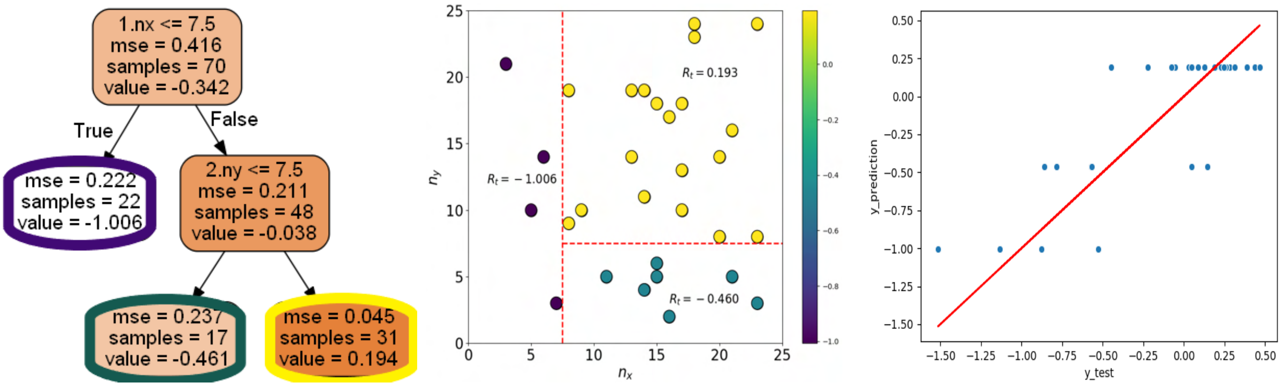
Applying the "DecisionTreeRegressor" algorithm on this distribution, a binary tree can be generated as shown in Figure C.8a[2]. Each block is referred to as a node. In order to construct the tree, a top down approach called "recursive binary splitting" is used[37]. Starting from the top, at each node, a bifurcation of the design space is carried out, using a predictor $x, x \in \{n_x, n_y\}$ and a cutpoint s. This leads to two separate half-planes defined as:

$$R_1(x,s) = \{x|x < s\} \text{ and } R_2(x,s) = \{x|x > s\}$$

The values of x and s are obtained by solving an optimization problem which aims to minimize the Residual Sum of Squares(RSS) for the resulting tree. The RSS is calculated individually for elements belonging to both partitions, by considering the mean label value of the training elements within each partition. Therefore at each split, the following objective is to be minimized:

$$RSS_{split} = \sum_{i:x_i R_1(j,s)} \{y_i - \hat{y}_{R_1}\}^2 + \sum_{i:x_i R_2(j,s)} \{y_i - \hat{y}_{R_2}\}^2 \tag{C.3}$$

where $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean label values within R_1 and R_2 respectively. It can be be seen from Figure C.8a, that for the top most node,i.e, for the very first split, the least value of RSS_{split} is found for $\{x,s\} = \{n_y, 7.5\}$. Moreover , each of the nodes also provide additional information such as the "MSE" or Mean Square Error which is the square root of the RSS within each partition, along with number of sample points and their mean label values within each partition.



(a) Decision Tree

(b) 3 terminal nodes - 3 distinct regions

(c) Performance evaluation of decision tree

Figure C.8: Performance evaluation of decision tree

[2] Tree visualization obtained using Graphviz{https://graphviz.gitlab.io/about/}, interface to graphviz provided by pydotplus package{https://github.com/carlos-jenkins/pydotplus}

Figure C.8a shows three terminal nodes. Each terminal node represents a distinct region having the same label value as shown in Figure C.8b. Figure C.8c shows the performance evaluation of the decision tree having three terminal nodes, each assigned a unique y_{pred} value. For a small decision tree as the one shown in Figure C.8a, the prediction capability is very low. For practical applications, a decision tree typically contains hundreds of branches. There are certain internal tuning parameters which can be added to a $DecisionTreeRegressor$ class as keyword arguments to alter the learning capability of the decision tree model. Some of these parameters are used to prune the size of the decision tree, while others are used to specify conditions on node splitting such as the minimum number of sample points required to split a node, or minimum error metric(MSE) required to split a node. The parameters that are considered in this section deal with the breadth and depth of the decision tree. More specifically, two arguments to the $DecisionTreeRegressor$ object are discussed, namely max_depth, and max_leaf_nodes. The max_depth parameter is used to fix the maximum number of vertical levels present in the decision tree . max_leaf_nodes is a cap on the maximum permissible nodes in the decision tree. Having the max_depth parameter specified, the max_leaf_nodes parameter is used to horizontally expand the decision tree by adding more branches to each level. Both of these parameters are used together to fix the size of the decision tree. For $max\_depth = 7$, $max\_leaf\_nodes = 9$, Figure C.9 shows the corresponding decision tree for the data set shown in Figure C.7

Depending on the type of the data set in question, it is required to find the appropriate value of the above mentioned internal parameters. A decision tree that is too small will have a low learning rate, but a decision tree that is allowed to expand indefinitely is going to overfit the training data, resulting in poor accuracy on test data points[3]. For the current case(SSBJ based database), an effort is made to arrive at an appropriate value for the above-mentioned tuning parameters. Firstly, using a manual approach, the approximate ballpark ranges in which the optimum value of the depth and node related parameters lie are found. Then, to narrow down on the exact optimum values of the two parameters in question, a grid search is constructed, which finds the value of max_depth and max_leaf_nodes parameters for which EVRS is the highest. This is shown by a surface plot in Figure C.10a. The highest EVRS score is obtained for max_depth = 21 and max_leaf_nodes = 28206. Using these values, the predictive performance of the decision tree algorithm on the SSBJ database is plotted in Figure C.10b. The measured EVRS score with the decision tree based machine learning model is 0.967. The prediction accuracy of the decision tree algorithm is higher than both Linear Regression and K-nearest Neighbour.

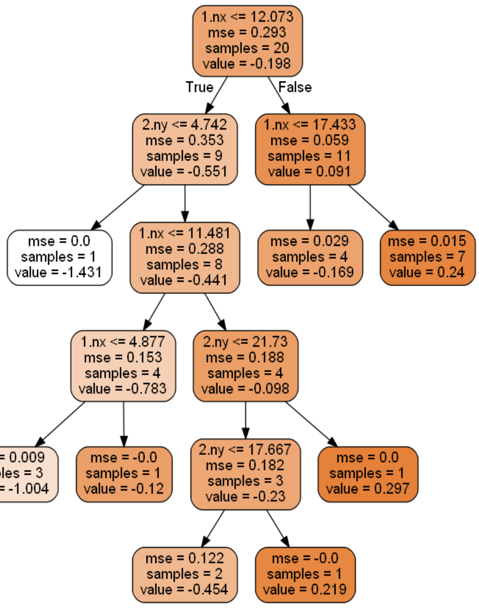
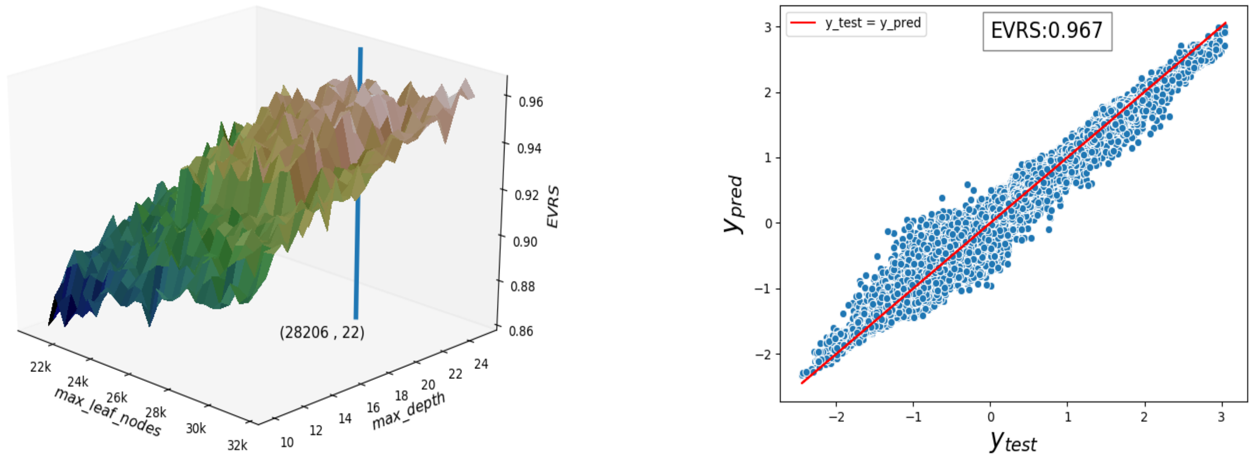


Figure C.9: Decision Tree ($max\_depth = 7$, $max\_leaf\_nodes = 9$)



(a) Estimation of optimum parameters

(b) Performance evaluation of Decision Tree(SSBJ data set)

Figure C.10: Decision Trees - Estimate parameters and Performance Evaluation

[3] https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680

D

Test MDO problems for SSBJ based neural network

D.1. Propane Combustion Problem

The Propane Combustion Problem is originally a set of eleven non-linear equations, involving eleven variables and two constants, that are used to represent the chemical equilibrium achieved during combustion of propane in air[16]. The eleven variables represent the ten combustion products as well as the sum of the products[16].

The NASA MDO test suite[17] contains an MDO version of this problem. The MDO version is created by grouping seven of the non linear equations into three disciplines(as residual equations). The remaining four equations are converted into inequality constraints. The objective is framed as the scalar addition of the constraint terms[17, 16]. The eleven unknown variables are categorized into six coupling variables$\{y_0, y_1, y_2, y_3, y_4, y_5, y_6\}$, three local design variables$\{x_1, x_2, x_3\}$ and one shared design variable$\{x_0\}$[17]. The data flow diagram for the disciplines is shown in Figure D.1

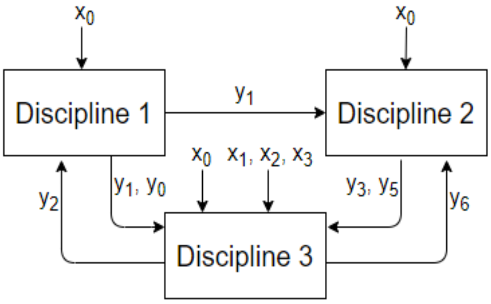


Figure D.1: Data Flow Diagram

In keeping with the SARF methodology, the following changes are made to the structure of the problem:

1. Each of the residual equations ($R_i(x_{local}, x_{shared}, y) = 0$) are equivalently expressed in explicit form ($y = y(x_{local}, x_{shared}, y_{j \neq i})$) [16].
2. The couplings and design variables are converted into vector form, each having a certain number of elements
3. Couplings are re-modelled in the form of y_{ij} where i and j represent the input and output disciplines respectively.

The original and re-modelled disciplinary equations are represented below:

$$\text{Original Disciplines} \qquad\qquad\qquad \text{Remodelled Disciplines}$$

$$Discipline1\begin{cases} x_0 + y_1 - 3 = 0 \\ y_0 y_1 - x_0 y_2 = 0 \end{cases} \qquad Discipline1\begin{cases} y_{12}[0] = 3 - x_0 \\ y_{13}[0] = 3 - x_0 \\ y_{13}[1] = x_0 y_{31}[0]/(3 - x_0) \end{cases}$$

$$Discipline2\begin{cases} 0.1x_0 - 40y_1y_3/y_6 = 0 \\ 0.1x_0^2 - 40y_1^2y_5/y_6 = 0 \end{cases} \longrightarrow \quad Discipline2\begin{cases} y_{23}[0] = 0.1x_0y_{32}[0]/40y_{12}[0] \\ y_{23}[1] = 0.1x_0^2y_{32}[1]/40y_{12}[0]^2 \end{cases}$$

$$Discipline3\begin{cases} 2(y_0 + y_2) + x_2 + x_3 - 8 = 0 \\ 2x1 + y4 - 40 = 0 \\ y6 - x0 - x1 - x2 - x3 - \\ y0 - y1 - y2 - y3 - y4 - y5 = 0. \end{cases} \qquad Discipline3\begin{cases} y_{31}[0] = 4 - 0.5(x_2 + x_3) - y_{13}[1] \\ y_{32}[0] = 44 + x0 - x1 + 0.5 * (x_2 + x_3) \\ +y_{13}[0] + y_{23}[0] + y_{23}[1] \end{cases}$$

The remodelled MDO problem, based on remodelled disciplines, can be framed as follows:

Given:

$x = \{x1, x2, x3, x0\}$
$y = \{y_{12}[0], y_{13}[0], y_{13}[1], y_{23}[0], y_{23}[1], y_{31}[0], y_{32}[0]\}$

Minimize:

$f_1(x, y) + f_2(x, y) + f_3(x, y) + f_4(x, y)$

Where:

$f_1(x, y) = 2x_0 - 2x_1 + x_3 + y_{13}[1] + y_{12}[0] + y_{23}[0] + 2y_{23}[1] + 30$
$f_2(x, y) = \sqrt{(y_{13}[1], y_{13}[0])} - x_2\sqrt{(40x_0/y_{32}[0])}$
$f_3(x, y) = \sqrt{(x_0, y_{13}[1])} - x_3\sqrt{(40y_{13}[0]/y_{32}[0])}$
$f_4(x, y) = x_0\sqrt{(x_1)} - y_{13}[0](40 - 2x_1)\sqrt{(40/y_{32}[0])}$

Subject to the following constraints:

$con_{g1} = \{f_1(x, y), f_1(x, y), f_1(x, y), f_1(x, y)\} > 0$
$con_{g2} = \{x_1, x_2, x_3\} > 0$

The XDSM for the remodelled Problem (using the MDF-GS architecture) is shown in Figure D.2.

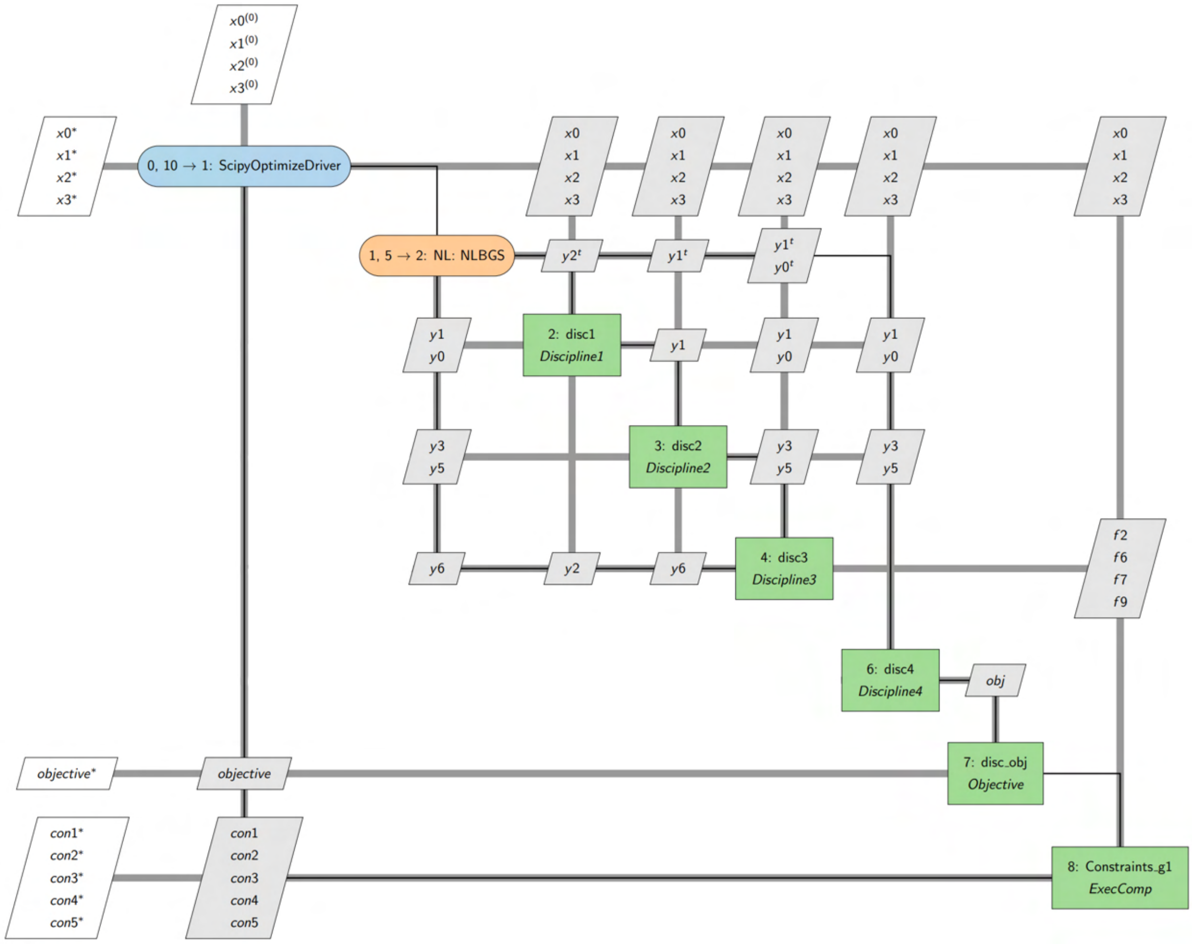


Figure D.2: XDSM for Propane Combustion (MDF-GS)

One of the constraints, con_{g_2} can be converted into a lower bound on the local design variables. The other constraint vector, con_{g1}, is a set of four constraints that are all global. Therefore, unlike the SSBJ based database, it is not possible to create a uniform sized local constraint vector attached to each discipline. Instead, a separate subsystem is defined that includes the global constraints and then added to the MDO problem. The output of this subsystem is the global constraint vector, which is scaled and normalized in the same manner as the local constraint within the SSBJ problem. Having the scaling of constraints sorted, the SARF methodology can be used to scale the Propane Combustion problem in the same manner as the SSBJ based problem. As explained before, there are two primary constructs for scaling the MDO Problem, i.e the dependency matrix and the component dependency graph. By considering the following parameters, $\{n_x, n_y, n_{con}, d\} = \{2, 5, 10, 0.6\}$, a scalable problem is generated at random and an instance of the two underlying constructs are presented below in Figure D.3.

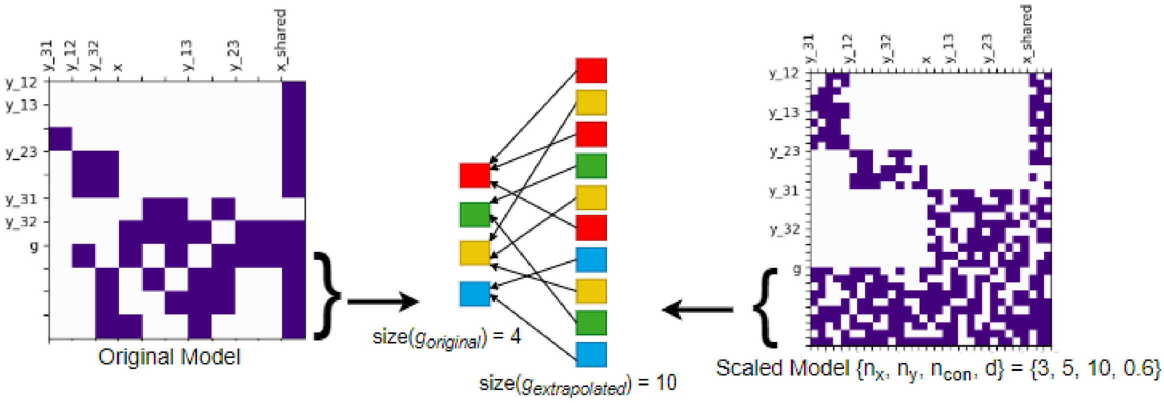


Figure D.3: dependency matrix/ Component dependency graph for Propane Combustion Problem

The dependency matrix for the scaled propane combustion problem consists of five specific zones (as opposed to four for the dependency matrix of the scaled SSBJ problem), representing the three disciplines, global constraints and the shared design variables. The component dependency graph, shown for the global constraint discipline g, shows the randomized mapping from four original components to ten extrapolated components.

D.2. Speed Reducer Problem

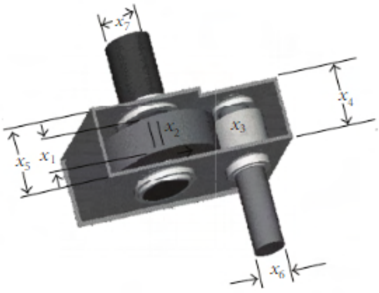


Figure D.4: Speed Reducer Gearbox

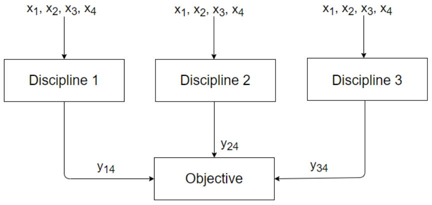


Figure D.5: Data flow diagram for Speed Reducer Problem

Another problem transcribed from the NASA MDO test suite, the speed reducer problem represents the design optimization of a simple gearbox. First proposed by Jan Golinsky[59], it was originally framed as a single discipline optimization problem aimed at minimizing the volume of a gearbox subject to stress, deflection and geometric constraints[59, 51, 53]. The problem consists of seven design variables and seven constraints. Figure D.4[51] shows the seven design variables, that represent the dimensions of the components within the gearbox.

Different MDO conversions have been proposed for the speed reducer problem[53, 60, 61]. The one used here was implemented by Nathan Tedford [60]. Within the MDO conversion, the structure of the original problem is reworked to create three disciplines. The seven original design variables are reworked into four global design variables$\{x_1, x_2, x_3, x_4\}$ and three coupling outputs$\{y_{14}, y_{24}, y_{34}\}$ (each belonging to one of three disciplines), that are used only in the objective calculation, and not as an input for any of the other disciplines. The seven original constraints are divided into three sets of three, two, two, and assigned as local constraints for each discipline. The data flow diagram for the disciplines is shown in Figure D.5. In keeping with SARF methodology, the shared variables are grouped into a vector x_0 having four components. The other aspects of the problem are already well suited for scaling according to the SARF methodology, as there are no residual equations and the couplings are already present in the required vector notation.

The disciplinary equations are framed as follows:

$$Discipline1\begin{cases} y_{14} = max\left(\frac{27}{x_0[0]^2 x_0[1]}, \ \frac{397.5}{x_0[0]^2 x_0[1]^2}, \ 5x_0[0], \ 2.6\right) \\ \\ g_1[0] = 1 - \frac{y_{14}}{12x_0[0]}, \ \ g_1[1] = 1 - \frac{y_{14}}{3.6} \ g_1[2] = 1 - \frac{x_0[0]x_0[1]}{40} \end{cases}$$

$$Discipline2\begin{cases} y_{24} = max\left(\left(\frac{1.93x_0[2]^3}{x_0[0]x_0[1]}\right)^{1/4}, \ \left(\frac{1}{110}\sqrt{\left(\frac{745x_0[2]}{x_0[0]x_0[1]}\right)^2 + 16.9 * 10^6}\right)^{1/3}, 2.9\right) \\ \\ g_2[0] = 1 - \frac{y_{24}}{3.9}, \ \ g_2[1] = 1 - \frac{1.5y_{24}1.9}{x_0[2]} \end{cases}$$

$$Discipline3\begin{cases} y_{34} = max\left(\left(\frac{1.93x_0[3]^3}{x_0[0]x_0[1]}\right)^{1/4}, \ \left(\frac{1}{85}\sqrt{\left(\frac{745x_0[3]}{x_0[0]x_0[1]}\right)^2 + 157.5 * 10^6}\right)^{1/3}, 5\right) \\ \\ g_3[0] = 1 - \frac{y_{34}}{5.5}, \ \ g_3[1] = 1 - \frac{1.1y_{24}1.9}{x_0[3]} \end{cases}$$

Based on the above disciplines, the MDO problem can be framed as follows:

Given:

$$x = \{x_0[0], x_0[1], x_0[2], x_0[3]\}$$
$$y = \{y_{14}, y_{24}, y_{34}\}$$

Minimize:

$$f_1(x,y) + f_2(x,y) + f_3(x,y) + f_4(x,y)$$

Where:

$$f_1(x,y) = 0.7854 y_{14} x_0[0]^2 (3.333 x_0[1]^2 + 14.9334 x_0[1] - 43.0934)$$
$$f_2(x,y) = -1.508(y_{24}^2 + y_{34}^2) y_{14}$$
$$f_3(x,y) = 7.477(y_{24}^3 + y_{34}^3)$$
$$f_4(x,y) = .7854(x_0[2] y_{24}^2 + x_0[3] y_{34}^2)$$

Subject to the following constraints:

$$con_{g1} = \{g_1[0], g_1[1], g_1[2]\} > 0$$
$$con_{g2} = \{g_2[0], g_2[1]\} > 0$$
$$con_{g3} = \{g_3[0], g_3[1]\} > 0$$

and the following bounds on x:

$$0.7 < x_0[0] < 0.8$$
$$17 < x_0[1] < 28$$
$$7.3 < x_0[2] < 8.3$$
$$7.3 < x_0[3] < 8.3$$

The XDSM for the problem(using the MDF architecture) is shown in Figure D.6.

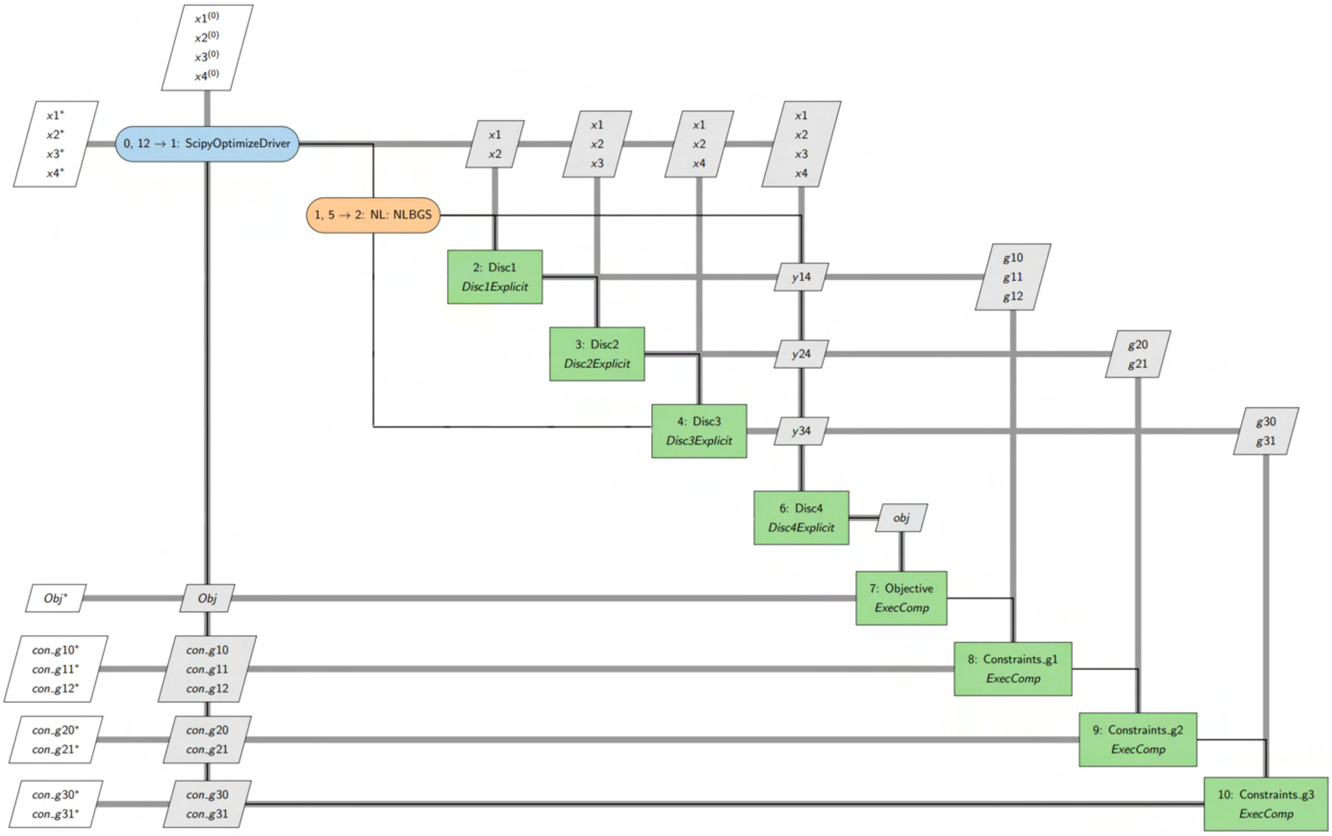


Figure D.6: XDSM for Speed Reducer Problem(MDF-GS)

Looking at Figure D.6, only shared design variables can be observed. The couplings are unidirectional, going from the first three disciplines to $Disc4$, which computes the objective. Hence, the feedback among disciplines which is found in other problems is missing here. The problem is high on constraints as each of the disciplines have a certain number of local constraints attached to them. These features are reflected in the framing of the dependency matrix. Based on the above-mentioned features, an instance of the original and scaled dependency matrices, along with an instance of the component dependency graph (for output g_1) is represented in Figure D.7. Looking at the original and scaled dependency matrices, the specific features of the Speed Reducer Problem can be observed. Since there is no interdisciplinary feedback, the coupling variables do not form a part of the input space and are hence not represented along the abscissa. Additionally, the matrix is not divided into distinct zones in this case, instead only one block, belonging to the shared variables is displayed.

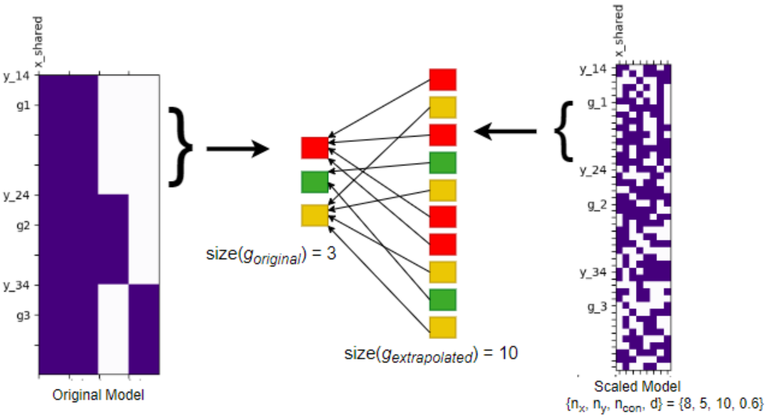


Figure D.7: Dependency matrix/ Component dependency graph for Speed Reducer Problem

D.3. Heart Dipole Problem

The final test problem looked at in this section is a standard MDO test problem called the Heart Dipole problem. Transcribed from the NASA MDO test Suite, the heart dipole problem involves the measurement of the synthetic dipole moment of a human heart using two artificial dipoles in an electrolyte-containing disk[52]. The original problem involves the use of eight Gabor-Nelson equations and eight unknown variables to compute the magnitude, directions, and locations of the two independent dipoles in the disks[1].

An MDO conversion was provided by Padula et. al.[17]. Considering the original Problem to be having eight original equations $\{f_1, f_2, ..., f_8\}$, and eight unknowns $\{x_1, x_2, ..., x_8\}$, the MDO problem is framed by arbitrarily combining four of the eight equations $\{f_5, f_6, f_7, f_8\}$ into an objective function and dividing the remaining four equations $\{f_1, f_2, f_3, f_4\}$ into residual equations for the two coupled disciplines. Four of the eight unknowns $\{x_1, x_4, x_6, x_7\}$ are converted into global(shared) design variables and the other four unknowns are assigned as coupling outputs for the two disciplines. The data flow diagram is shown in Figure D.8

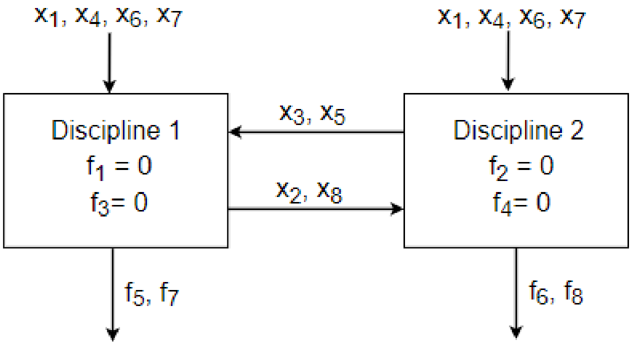
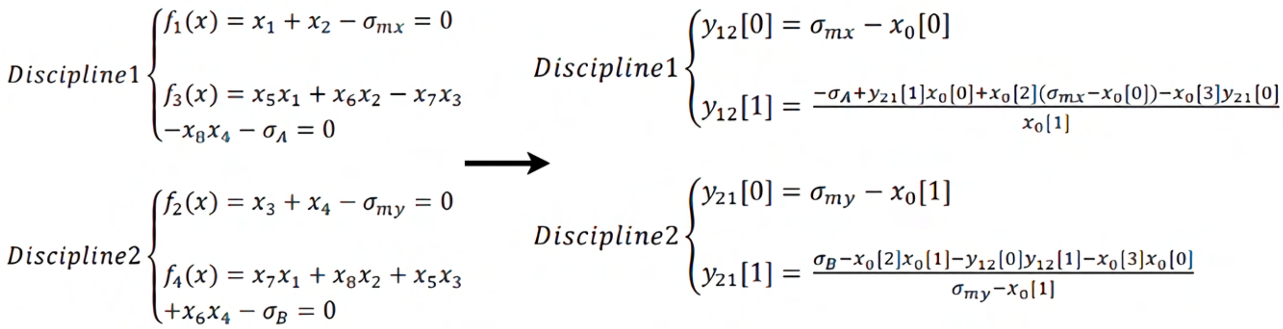


Figure D.8: Data flow diagram for Heart Dipole Problem

The MDO problem, in its original form, contains residual equations within each discipline. In keeping with the SARF methodology, it is required to rework the residual equations to express the coupling outputs in explicit form. The couplings and design variables are also required to be renamed into an appropriate nomenclature compatible with the SARF methodology.

[1] https://www.aere.iastate.edu/bloebaum/ii-c-1-heart-dipole-problem/

The original and remodeled disciplines are shown below:



$$Discipline1\begin{cases} f_1(x) = x_1 + x_2 - \sigma_{mx} = 0 \\ f_3(x) = x_5x_1 + x_6x_2 - x_7x_3 \\ -x_8x_4 - \sigma_A = 0 \end{cases} \qquad Discipline2\begin{cases} f_2(x) = x_3 + x_4 - \sigma_{my} = 0 \\ f_4(x) = x_7x_1 + x_8x_2 + x_5x_3 \\ +x_6x_4 - \sigma_B = 0 \end{cases}$$

$$\longrightarrow$$

$$Discipline1\begin{cases} y_{12}[0] = \sigma_{mx} - x_0[0] \\ y_{12}[1] = \frac{-\sigma_A + y_{21}[1]x_0[0] + x_0[2](\sigma_{mx} - x_0[0]) - x_0[3]y_{21}[0]}{x_0[1]} \end{cases}$$

$$Discipline2\begin{cases} y_{21}[0] = \sigma_{my} - x_0[1] \\ y_{21}[1] = \frac{\sigma_B - x_0[2]x_0[1] - y_{12}[0]y_{12}[1] - x_0[3]x_0[0]}{\sigma_{my} - x_0[1]} \end{cases}$$

Based on the remodelled disciplines, the MDO problem can be framed as follows:

Given:

$x = \{x_0[0], x_0[1], x_0[2], x_0[3]\}$

$y = \{y_{12}[0], y_{12}[1], y_{21}[0], y_{21}[1]\}$

Minimize:

$f_5(x,y) + f_6(x,y) + f_7(x,y) + f_8(x,y)$

where:

$$\begin{aligned} f_5(x,y) = {} & x_0[0](y_{21}[1]^2 - x_0[3]^2) - 2y_{21}[0]y_{21}[1]x_0[3] \\ & + y_{12}[0](x_0[2]^2 - y_{12}[1]^2) - 2x_0[1]x_0[2]y_{12}[1] - \sigma_C \\ f_6(x,y) = {} & y_{21}[0](y_{21}[1]^2 - x_0[3]^2) + 2x_0[0]y_{21}[1]x_0[3] \\ & + x_0[1](x_0[2]^2 - y_{12}[1]^2) + 2y_{12}[0]x_0[2]y_{12}[1] - \sigma_D \\ f_7(x,y) = {} & x_0[0]y_{21}[1](y_{21}[1]^2 - 3x_0[3]^2) + y_{21}[0]x_0[3](x_0[3]^2 \\ & - 3\,y_{21}[1]^2) + y_{12}[0]x_0[2](x_0[2]^2 - 3y_{12}[1]^2) + x_0[1]y_{12}[1](y_{12}[1]^2 - 3x_0[2]^2) - \sigma_E \\ f_8(x,y) = {} & y_{21}[0]y_{21}[1](y_{21}[1]^2 - 3x_0[3]^2) - x_0[0]x_0[3](x_0[3]^2 - 3y_{21}[1]^2) \\ & + x_0[1]x_0[2](x_0[2]^2 - 3y_{12}[1]^2) + y_{12}[0]y_{12}[1](y_{12}[1]^2 - 3x_0[2]^2) - \sigma_F \end{aligned}$$

Subject to the following Constraints:

$Con_{g1} = \{-f_5(x,y), -f_7(x,y)\} > 0$

$Con_{g2} = \{-f_6(x,y), -f_8(x,y)\} > 0$

and the following bounds on the design variables:

$0 < \{x_0[0], x_0[1], x_0[2], x_0[3]\} < 400$

Looking at the disciplinary equations and the problem definition, a total of six constants can be seen. An appropriate values for these constants can be looked up in the literature[52, 17, 62]. The final value for these constants is taken directly from the paper by Li et. al[52]

Therefore, $\sigma_{mx} = \sigma_{my} = \sigma_A = \sigma_B = \sigma_C = \sigma_D = \sigma_E = \sigma_F = 0.1$

The XDSM for the heart dipole MDO problem(using the MDF-GS architecture) is shown in Figure D.9.

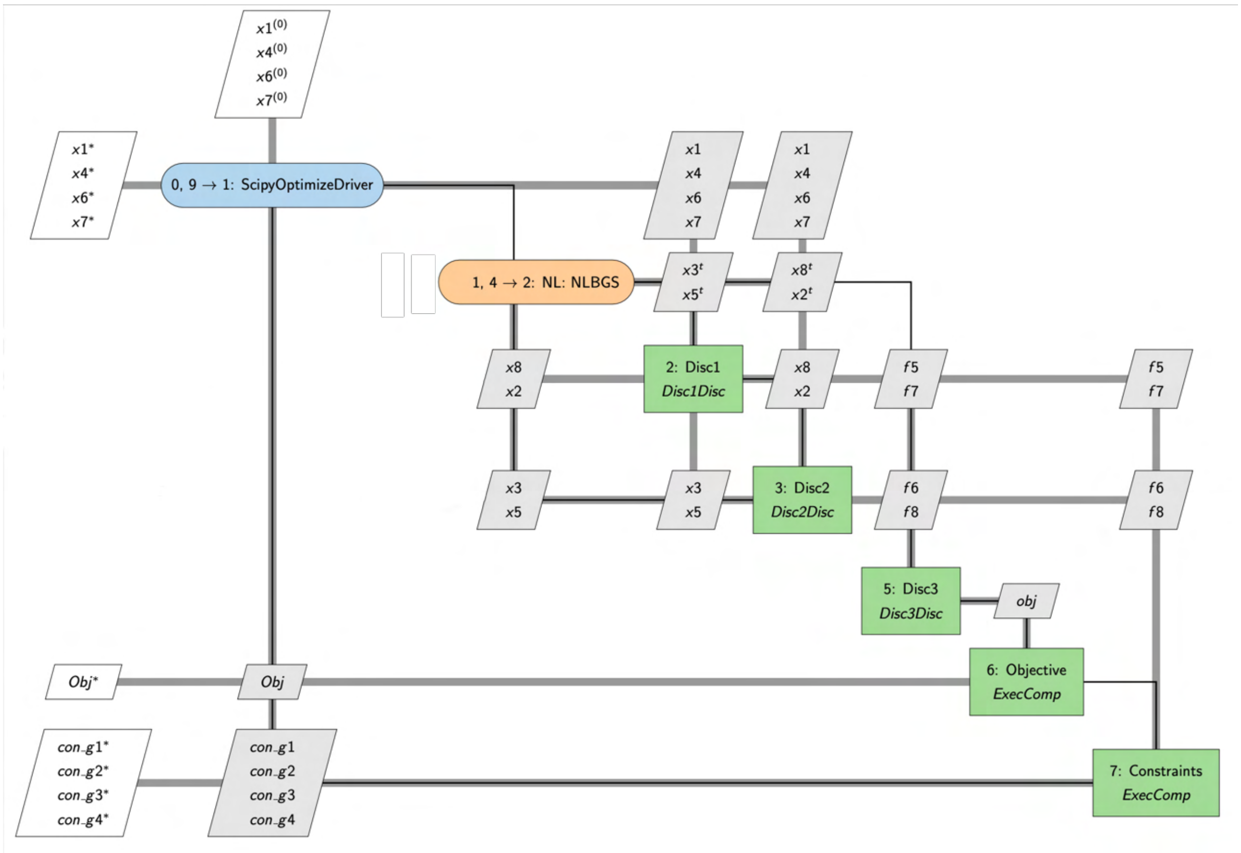


Figure D.9: XDSM for heart dipole problem(MDF-GS)

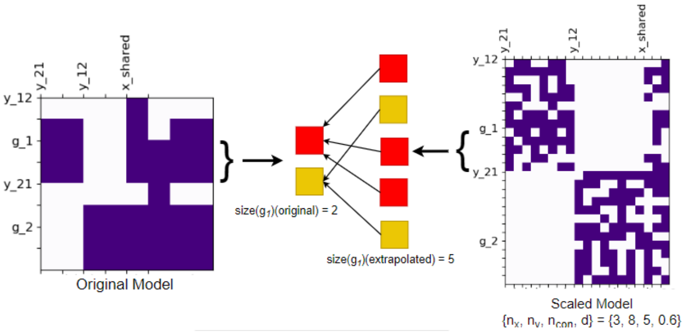


Figure D.10: Dependency matrix/ Component dependency graph for Heart Dipole Problem

Looking at the XDSM, the problem can be observed to have two coupled disciplines. The input space is devoid of any local design variables. Two coupling vectors can be observed, y_{12} and y_{21}, each having originally a size of two. There are also local constraint vectors of size two, con_{g1} and con_{g2} attached to each discipline. The global(shared) variable vector contains four components. These input and output parameters are reflected in the dependency matrix as shown in Figure D.10 The original dependency matrix shows the input and output components of the heart dipole problem. Looking at(an instance of) the scaled model, three zones of dependencies can be observed, two belonging to each of the two disciplines and the third zone belonging to the global(shared) variables. The component dependency graph is shown for the constraint vector con_{g1}.

E

Test MDO problems for retrained neural network

E.1. Fuel Minimization problem using Q3D-EMWET

To frame the problem, two major analytical tools are used, namely Q3D and EMWET. Q3D is a low fidelity aerodynamic solver that uses a combination of the Vortex Lattice Method(VLM), full/linearized potential flow, and Boundary Layer equations to calculate the lift and drag distribution as well as pitching moment distribution of the wing sections. The drag is composed of three components: profile drag, induced drag, and wave drag. The load and moment distribution obtained from the Q3D solver is then entered into a structural sizing tool called EMWET that predicts the weight distribution of the wing structure. EMWET uses a physics kernel to size components such as top and bottom panels, spar webs, and wing box, while empirical information is used to account for the weights of high lift devices. A few wing optimization MDO problems can be framed using these two disciplines, such as range maximization of an airplane or minimization of the maximum take-off weight for a given set of constraints. The MDO problem to be used in this section is titled as **"Minimize the mission fuel weight of a reference airplane for a given range using a constraint multidisciplinary approach."**.

Before the MDO problem can be framed, it is required to derive the components that can uniquely define the three-dimensional geometric structure of the wing. The wing itself is constructed using three airfoils, one each at the root, kink, and tip sections. The airfoil sections are constructed using a parameterization method called the Class Shape Transformation(CST) method[63]. A total of six CST coefficients are used to construct each airfoil upper and lower sections, making it a total of thirty-six coefficients for the three airfoil sections. These coefficients are represented by the CST_W vector. Additionally, a set of design variables are used to define the wing planform. These variables are represented with the G_W vector.

Based on the above procedure, a fuel minimization MDO problem for a constant mission range can be framed as follows:

Given: $X = \{G_W, CST_W\}$

where $G_W = \{B, R_c, \Lambda_1, \Lambda_2, \lambda R_c, \phi_{Root}, \phi_{Kink}, \phi_{Tip}\}$, $CST_W = \{CST_{Root}, CST_{Kink}, CST_{tip}\}$

Minimize: $\frac{Wfuel(X)}{Wfuel(ref)}$

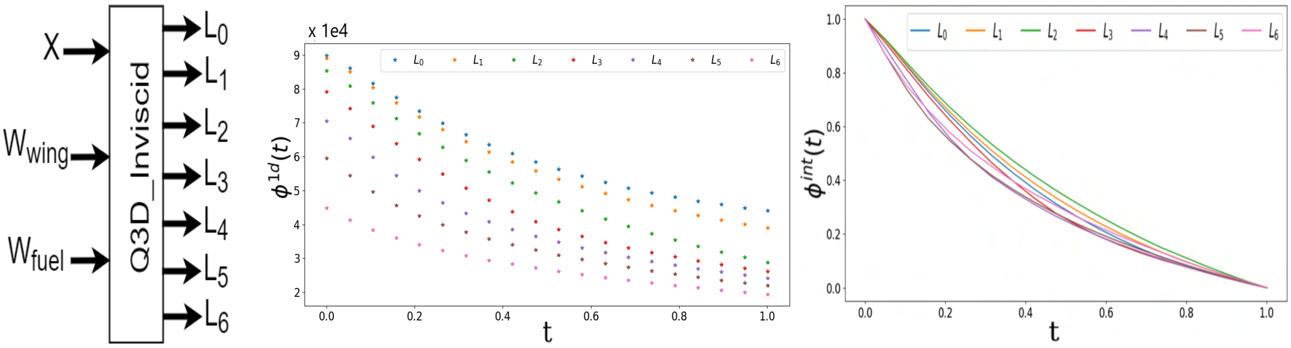
Subject to the following constraints: $g_1 = \frac{V_{fuel(X)}}{.93 * V_{tank(X)}} - 1 <= 0$, $g_2 = \frac{WL(X)}{WL(ref)} - 1 <= 0$

where V_{fuel}, V_{tank}, WL and S stand for the fuel volume, tank volume, wing loading, and the wing surface area respectively. ref represents the reference value of the outputs. Using the MDF-GS architecture, the XDSM of the original problem is shown in Figure E.1. It must be noted that the original problem was framed and executed within the MATLAB environment, so the XDSM is manually generated and does not comply with the exact template used by $write\_xdsm$ method within OpenMDAO. The following section is used to explain the scaling process for the fuel minimization problem

E.1.1. Scaled Fuel Minimization Problem

Under the SARF methodology, the first step or the "expensive step" is the one-dimensional restriction of each of the disciplinary outputs(couplings and constraints), which requires the repeated execution of the disciplines for a range of input values. Now, since the original problem(and therefore the analytical tools) from the MDO course are based on MATLAB, it is required to run the unidimensional restriction step within a MATLAB environment. An example of the one-dimensional restriction is shown below for the spanwise lift distribution coupling vector(L) computed from the inviscid Q3D solver. The inviscid Q3D solver gives the lift distribution for a set of fourteen points uniformly spaced along the half span, which are then interpolated using a spline method[1] into seven evenly spaced points along the half span. The dimension of the inputs $\{X, W_{wing}, W_{fuel}\}$ are $\{44, 1, 1\}$ respectively making a total of forty six-dimensional input while the load output is a seven-dimensional vector. The corresponding load function can be stated as : $L: R^{46} \rightarrow R^7$. The disciplinary interface is shown in Figure E.2a. Now, Equation 3.1 is applied on each of the seven outputs(considering t to be an equally spaced set of 20 values between zero and one) to create a one dimensional mapping function ($L^{1d}(t)$). For this purpose, the lower and upper bounds of each input are directly estimated from the earlier MATLAB based implementation. This function is plotted in Figure E.2b. The mapping function can be represented as follows : $L^{1d}: [0,1] \rightarrow R^7$. At this point, a CSV database is generated for this mapping function, which is then imported into the OpenMDAO environment.

The second step, i.e scaling, and interpolation is executed within OpenMDAO. The database, consisting of uni-dimensionalized outputs is imported into OpenMDAO, where the outputs are first scaled between zero and one(Using the minimum and maximum values of the outputs) followed by interpolation using a spline curve, which converts the mapped outputs into a continuous function. The corresponding function, shown in Figure E.2c, can be represented as follows : $L^{int}: [0,1] \rightarrow [0,1]^7$. The above two steps are repeated for the moment coupling and the results are shown in Figure E.3

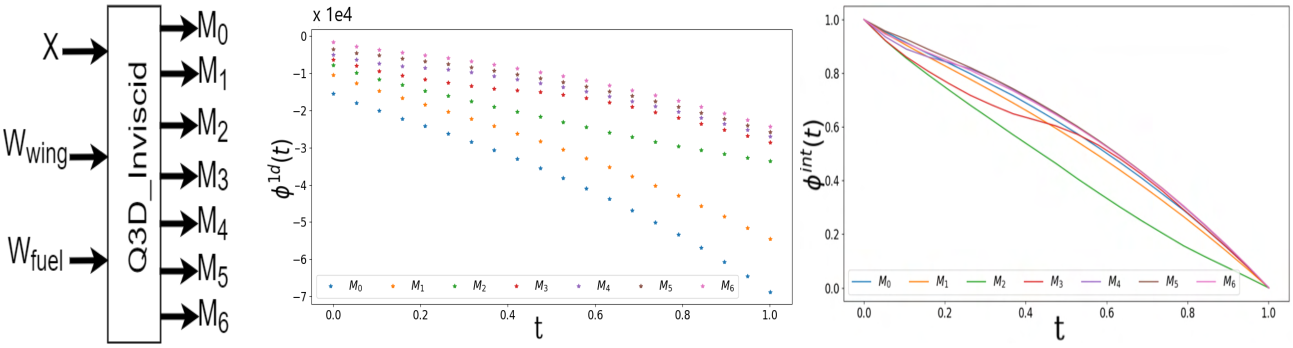


(a) Disciplinary interface for L $L: R^{46} \rightarrow R^7$

(b) 1-d restriction for L $L^{1d}: [0,1] \rightarrow R^7$

(c) Scaled and Interpolated outputs for L $L^{int}: [0,1] \rightarrow [0,1]^7$

Figure E.2: One dimensional restriction, Scaling and Interpolation for Load Vector



(a) Disciplinary interface for M $M: R^{46} \rightarrow R^7$

(b) 1-d restriction for M $M^{1d}: [0,1] \rightarrow R^7$

(c) Scaled and Interpolated outputs for M $M^{int}: [0,1] \rightarrow [0,1]^7$

Figure E.3: One dimensional restriction, Scaling and Interpolation for Moment Vector

[1] https://nl.mathworks.com/help/matlab/ref/interp1.html

The final step, is the extrapolation of the one-dimensional function into n_y dimensions, by constructing a dependency matrix and a component dependency graph. The dependency matrix for the original, unscaled problem is shown in Figure E.4

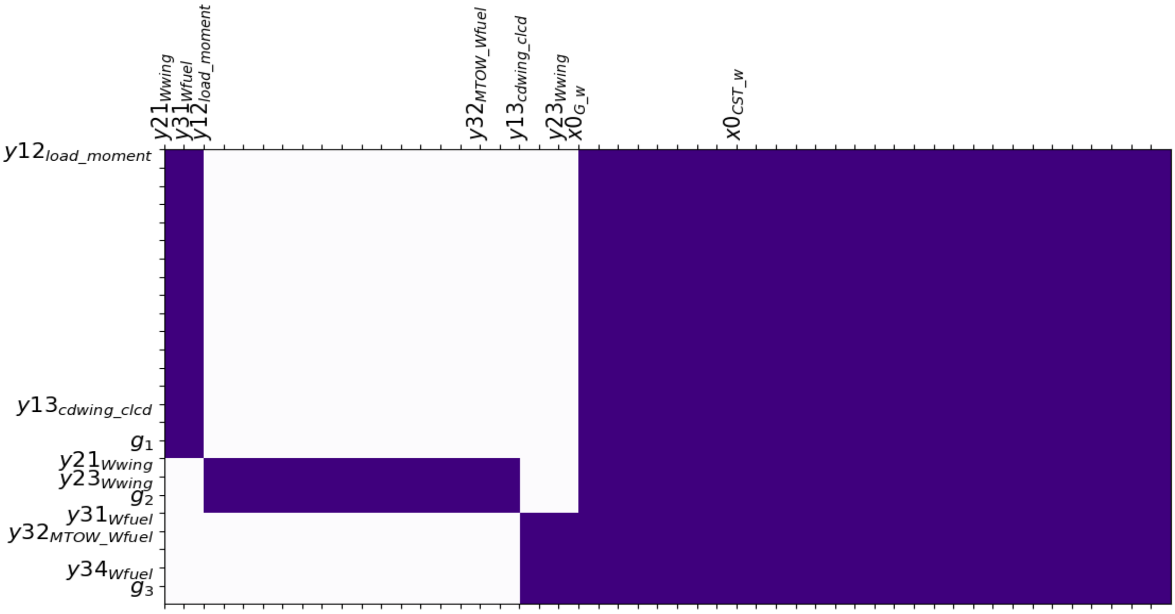


Figure E.4: Dependency matrix for original (unscaled) fuel minimization problem

The design space of the scaled fuel minimization problem can be seen along the abscissa of the dependency matrix in Figure E.4. In the absence of local design variables, design space consists of shared variables($x0_{G\_W}$ and $x0_{CST\_W}$) which are plotted separately on the right. Under the SARF methodology, six coupling vectors are constructed by combining the original couplings in such a way that the source and destination of each coupling are unique. Based on this, the above-discussed coupling vectors (related to load and moment distribution) from Aerodynamics to Structure discipline is named $y12_{load\_moment}$ having a total of fourteen elements$(7+7)$. By considering the following values of the parameters. $\{n_x, n_y, n_{con}, d\} = \{3, 5, 3, 0.6\}$, an instance of the scaled dependency matrix/ component dependency graph(for the coupling $y12_{load\_moment}$) is shown in Figure E.5. For simplicity, only the aerodynamics discipline has been shown.

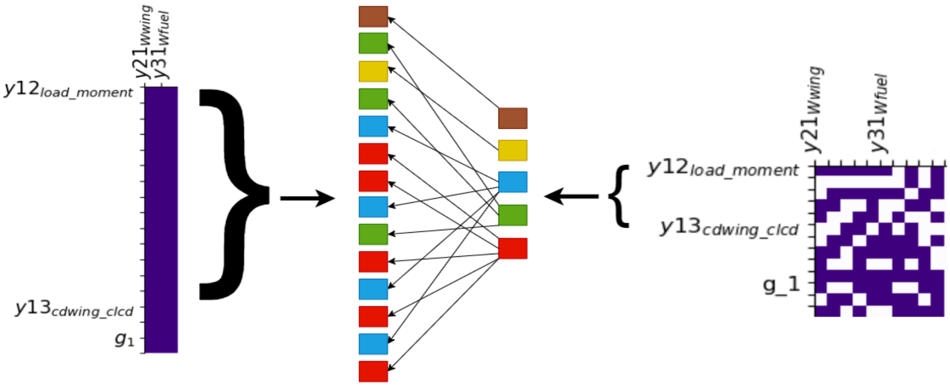


Figure E.5: Dependency Matrix/ Component dependency graph
$\{n_x, n_y, n_{con}, d\} = \{3, 5, 3, 0.6\}$

Using these two components, the interpolated coupling vectors $L^{int} : [0,1] \to [0,1]^7$ and $M^{int} : [0,1] \to [0,1]^7$ are extrapolated into a single coupling vector $y12_{load\_moment}$ consisting of n_y components.

Given that the input space(for the scaled aerodynamic discipline) consists of thirteen $(n_x + 2 * n_y = 13)$ components, and the output space consists of five$(n_y = 5)$ components, the extrapolated vector can be noted as $y12^{int}_{load\_moment} : [0,1]^{13} \to [0,1]^5$. Now, in order to plot the individual contribution of each input component on the extrapolated vector $y12_{load\_moment}$ in two dimensions, it would be required to make thirteen plots. Figure E.6 shows two of these plots as representative cuts, showing the individual effect of $y12_{wwing}\_0$ and $y12_{wwing}\_1$ on the extrapolated vector $y12_{load\_moment}$.

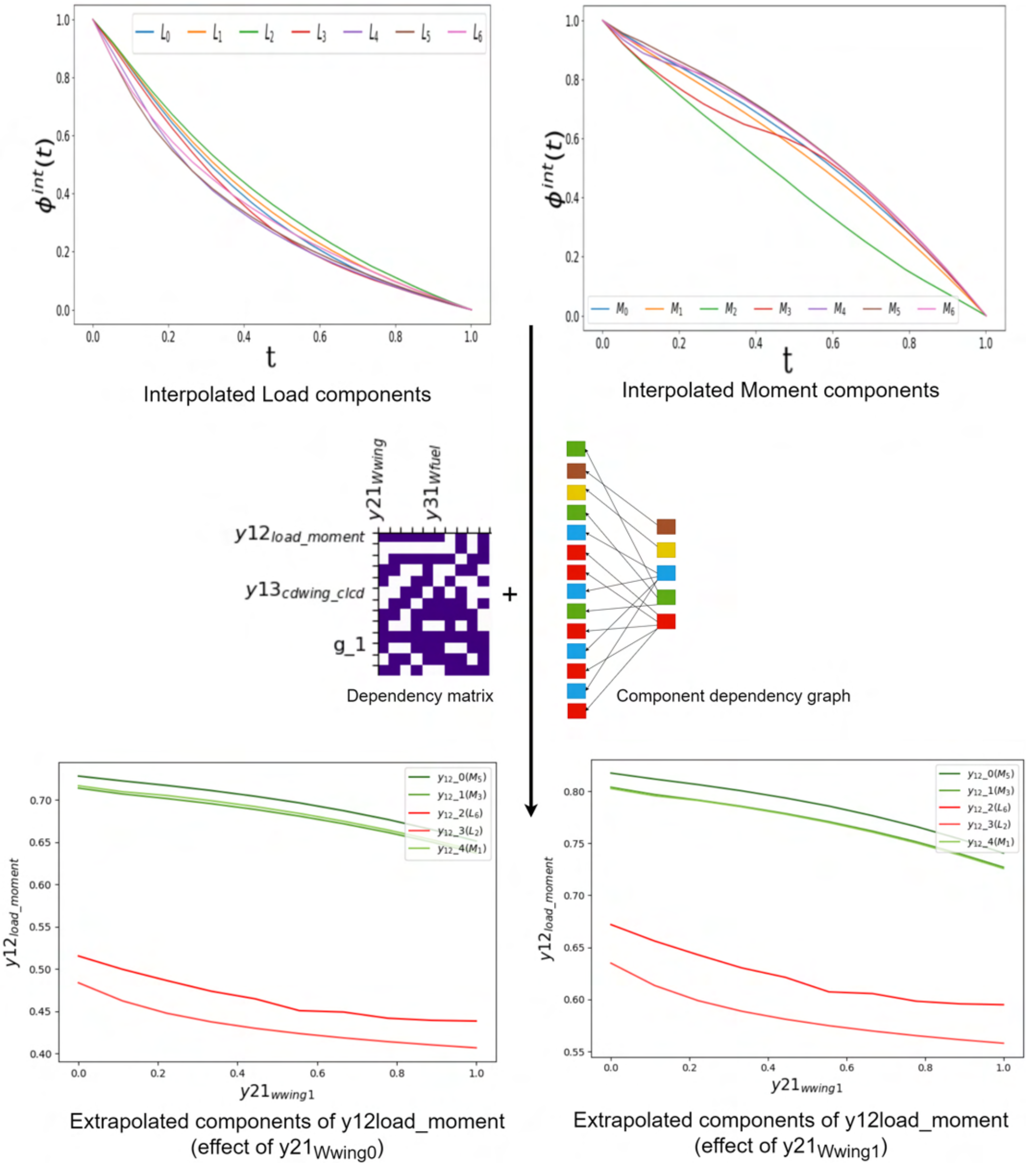


Figure E.6: Extrapolation from Load and moment couplings to $y12_{load\_moment}$ vector

Based on the component dependency graph, each component of the extrapolated vector $y12_{load\_moment}$ is derived from an original component(either L or M). The original components are shown in brackets on the legend of the extrapolated components. The red components are based on extrapolation of a particular component of the moment vector while the green components are based on the components of the load vector. It can be seen that the linearity/convexity of the original components is preserved in the process of extrapolation.

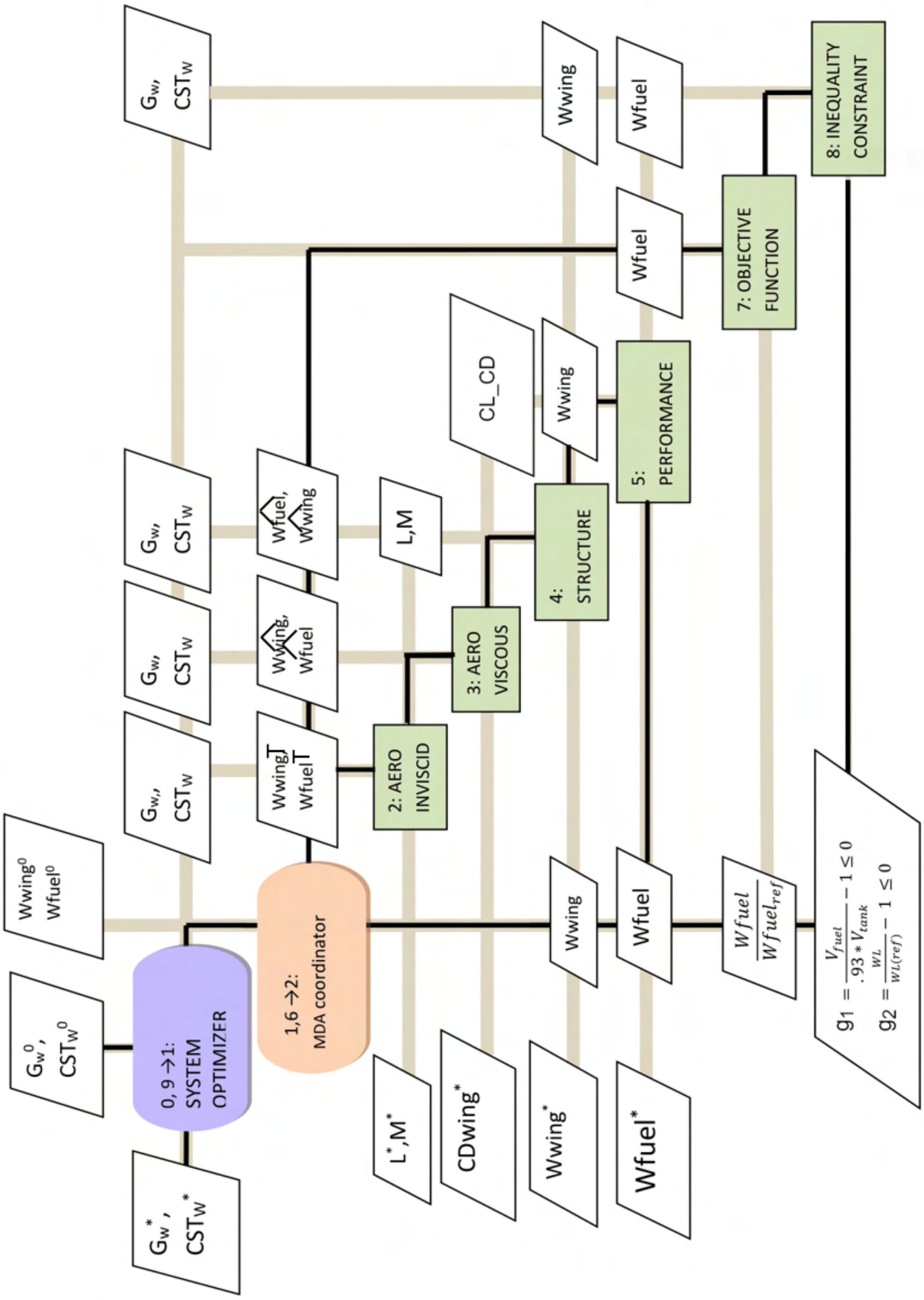


Figure E.1: XDSM for the original Fuel minimization MDO problem(using MDF-GS scheme)

To build a scaled database, it is also required to estimate the coupling strength of the fuel minimization problem. Now, in the previous sections, the coupling strength was calculated on both the original, unscaled problems as well as the scaled problems, and it was verified that the value of the coupling strength remained in the same ballpark when an MDO problem was scaled under the SARF methodology. In the current scenario, only the scaled problem is available for calculation of coupling strength as it is directly transcribed from a MATLAB based implementation of the original problem. Using the same technique as shown in Section 4.2.4.2, the numerical estimate for the coupling strength is calculated for the scaled fuel minimization problem across a range of data points as shown in Figure E.7. Using the time-weighted average, the coupling strength for the fuel minimization problem comes to be 0.487.

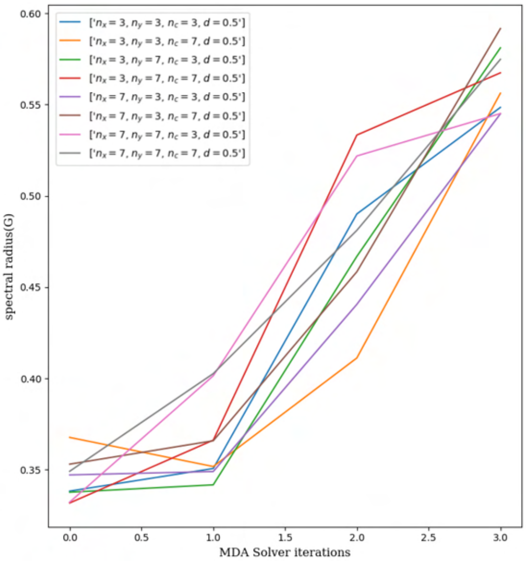


Figure E.7: Spectral radius vs Solver iterations (Scaled fuel minimization problem)

E.1.2. Sellar Problem

The flow diagram for the Sellar problem is shown in Figure E.8. Each discipline is defined by a single mathematical expression. There is bidirectional coupling between both the disciplines, represented by y_1 and y_2, each of size one. There is a single constraint vector, of size one, attached to each of the two disciplines. Based on this flow diagram, the original and remodelled disciplines are shown below:

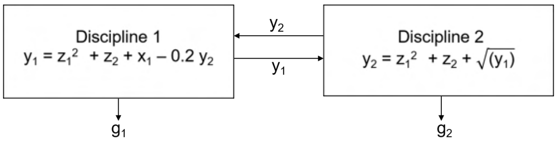

Discipline 1
$y_1 = z_1^2 + z_2 + x_1 - 0.2\, y_2$
y_2
y_1
Discipline 2
$y_2 = z_1^2 + z_2 + \sqrt{(y_1)}$
g_1
g_2

Figure E.8: Flow diagram (Sellar Problem)

Original

$$Discipline1\begin{cases} y_1 = z_1^2 + z_2 + x_1 - 0.2y_2 \\ g_1 = 1 - y_1/3.16 \end{cases}$$

$$Discipline2\begin{cases} y_2 = z_1^2 + z_2 + sqrt(y_1) \\ g_2 = y_2/24 - 1 \end{cases}$$

$$\longrightarrow$$

Remodelled

$$Discipline1\begin{cases} y_{12}[0] = z[0]^2 + z[1] + x_1[0] - 0.2y_{21}[0] \\ g_1[0] = 1 - y_{12}[0]/3.16 \end{cases}$$

$$Discipline2\begin{cases} y_{21}[0] = z[0]^2 + z[1] + sqrt(y_{12}[0]) \\ g_2[0] = y_{21}[0]/24 - 1 \end{cases}$$

Based on the above scaled disciplines, the SARF based problem can be formed as follows:

Given:
x, z
Minimize:
$x[0]^2 + z[0] + y_{12}[0] + e^{-y_{21}[0]}$
such that:
$1 - y_{12}[0]/3.16 <= 0$
$y_{21}[0]/24 - 1 <= 0$

Looking at the scaled problem, the structure seems identical to the scaled heart dipole based problem, therefore the process of formulating the dependency matrices and the component dependency graph are identical. The process is not repeated here, it can be retrieved from Section D.3. The coupling strength(ρ) is also calculated according to the same methodology as before and comes out to be 0.562.

Bibliography

[1] Kevin Bowcutt. A perspective on the future of aerospace vehicle design. 12 2003. ISBN 978-1-62410-085-7. doi: 10.2514/6.2003-6957.

[2] Jeremy Agte, Olivier de Weck, Jaroslaw Sobieszczanski-Sobieski, Paul Arendsen, Alan Morris, and Martin Spieck. Mdo: assessment and direction for advancement—an opinion of one international group. *Structural and Multidisciplinary Optimization*, 40(1):17, 2009. ISSN 1615-1488. doi: 10.1007/s00158-009-0381-5. URL https://doi.org/10.1007/s00158-009-0381-5.

[3] Forest Flager and John Haymaker. A comparison of multidisciplinary design, analysis and optimization processes in the building construction and aerospace industries. 06 2007.

[4] Jaroslaw Sobieszczanski-Sobieski and Raphael Haftka. Multidisciplinary aerospace design optimization: Survey of recent developments. *Structural Optimization*, 14:1–23, 08 1997. doi: 10.1007/BF01197554.

[5] James Allison, Michael Kokkolaras, and Panos Papalambros. On the impact of coupling strength on complex system optimization for single-level formulations. (4739X):265–275, 2005. doi: 10.1115/DETC2005-84790. URL http://dx.doi.org/10.1115/DETC2005-84790.

[6] Nathan Tedford. Comparison of mdo architectures within a universal framework. Master thesis, University of Toronto, North York, Toronto, 2007.

[7] Nathan P. Tedford and Joaquim R. R. A. Martins. Benchmarking multidisciplinary design optimization algorithms. *Optimization and Engineering*, 11(1):159–183, February 2010. doi: 10.1007/s11081-009-9082-6.

[8] Shamsheer Chauhan, John Hwang, and Joaquim R. R. A. Martins. *Benchmarking Approaches for the Multidisciplinary Analysis of Complex Systems Using a Taylor Series-Based Scalable Problem*. 2017. ISBN 978-3-319-67987-7. doi: 10.1007/978-3-319-67988-4_7.

[9] Gyung-Jin Park, Yi Sang-Il, and Shin Jung-Kyu. Comparison of mdo methods with mathematical examples. *Structural and Multidisciplinary Optimization*, 35(5):391–402, 2008. ISSN 1615-1488. doi: 10.1007/s00158-007-0150-2. URL https://doi.org/10.1007/s00158-007-0150-2.

[10] Kevin. F. Hulme and Christina. L. Bloebaum. A simulation-based comparison of multidisciplinary design optimization solution strategies using cascade. *Structural and Multidisciplinary Optimization*, 19 (1):17–35, 2000. ISSN 1615-1488. doi: 10.1007/s001580050083. URL https://doi.org/10.1007/s001580050083.

[11] Ted Long. The optimization assistant—helping engineers explore designs through collaboration. In *Proceedings of the 4th International Conference on Intelligent User Interfaces*, IUI '99, page 200, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 1581130988. doi: 10.1145/291080.291125. URL https://doi.org/10.1145/291080.291125.

[12] Amir Mosavi. The large scale system of multiple criteria decision making; pre-processing. *IFAC Proceedings Volumes*, 43(8):354 – 359, 2010. ISSN 1474-6670. doi: https://doi.org/10.3182/20100712-3-FR-2020.00060. URL http://www.sciencedirect.com/science/article/pii/S1474667015334194. 12th IFAC Symposium on Large Scale Systems: Theory and Applications.

[13] Maurice Hoogreef. Advise, formalize and integrate mdo architectures: A methodology andimplementation. Phd thesis, Delft University of Technology, Mekelweg 5, 2628 CD Delft, 2017.

[14] Scott Delbecq, Marc Budinger, and Aurélien Reysset. Benchmarking of monolithic mdo formulations and derivative computation techniques using openmdao. *Structural and Multidisciplinary Optimization*, 03 2020. doi: 10.1007/s00158-020-02521-7.

[15] Loïc Brevault, Mathieu Balesdent, Nicolas Bérend, and Rodolphe Le Riche. Comparison of different global sensitivity analysis methods for aerospace vehicle optimal design. 05 2013.

[16] Charlie Vanaret, Francois Gallard, and Joaquim Martins. On the consequences of the "no free lunch" theorem for optimization on the choice of an appropriate mdo architecture. *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017. doi: 10.2514/6.2017-314810.2514/6.2017-3148. URL https://doi.org/10.2514/6.2017-3148.

[17] Sharon L. Padula, Natalia. Alexandrov, and Lawrence Green. *MDO test suite at NASA Langley Research Center*. Multidisciplinary Analysis Optimization Conferences. American Institute of Aeronautics and Astronautics, 1996. doi: 10.2514/6.1996-402810.2514/6.1996-4028. URL https://doi.org/10.2514/6.1996-4028.

[18] Jaroslaw Sobieszczanski, Jeremy S. Agte, and Robert R. Sandusky Jr. Bi-level integrated system synthesis (bliss). Technical report, 1998.

[19] Emre Yilmaz and Brian German. *A Convolutional Neural Network Approach to Training Predictors for Airfoil Performance*. AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, 2017. doi: 10.2514/6.2017-366010.2514/6.2017-3660. URL https://doi.org/10.2514/6.2017-3660.

[20] Emre Yilmaz and Brian German. *A Deep Learning Approach to an Airfoil Inverse Design Problem*. AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, 2018. doi: 10.2514/6.2018-342010.2514/6.2018-3420. URL https://doi.org/10.2514/6.2018-3420.

[21] Jean-Loup Loyer, Elsa Henriques, Mihail Fontul, and Steve Wiseall. Comparison of machine learning methods applied to the estimation of manufacturing cost of jet engine components. *International Journal of Production Economics*, 178:109 – 119, 2016. ISSN 0925-5273. doi: https://doi.org/10.1016/j.ijpe.2016.05.006. URL http://www.sciencedirect.com/science/article/pii/S0925527316300731.

[22] Ney R. Secco and Bento S. Mattos. *Artificial Neural Networks Applied to Airplane Design*. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, 2015. doi: 10.2514/6.2015-101310.2514/6.2015-1013. URL https://doi.org/10.2514/6.2015-1013.

[23] George Sved. Structural optimization under multiple loading. *International Journal of Mechanical Sciences*, 10(10):803–805, 1968. ISSN 0020-7403. doi: https://doi.org/10.1016/0020-7403(68)90021-0. URL http://www.sciencedirect.com/science/article/pii/0020740368900210.

[24] Lucien. A. Schmit. Structural synthesis 1959-1969 - A decade of progress. Technical report, U. of Alabama Press, University,Alabama, United States, 1971.

[25] Jaroslaw Sobieszczanski-Sobieski. A linear decomposition method for large optimization problems. Blueprint for development. Technical report, NASA Langley Research Center; Hampton, VA, United States, 1971.

[26] Joaquim R. R. A. Martins and Andrew B. Lambe. Multidisciplinary design optimization: A survey of architectures. *AIAA Journal*, 51:2049–2075, 2013. doi: 10.2514/1.J051895.

[27] Shamsheer Chauhan, John Hwang, and Joaquim Martins. An automated selection algorithm for nonlinear solvers in mdo. *Structural and Multidisciplinary Optimization*, 58, 06 2018. doi: 10.1007/s00158-018-2004-5.

[28] Aliyu I. Bakari. Comparison of jacobi and gauss-seidel iterative methods for the solution of systems of linear equations. *Asian Research Journal of Mathematics 8 (3), 1-7*, 2018. doi: 10.9734/ARJOM/2018/34769. URL http://www.journalrepository.org/media/journals/ARJOM_44/2018/Feb/Bakari832017ARJOM34769.pdf.

[29] Richard Balling and Carol Wilkinson. Execution of multidisciplinary design optimization approaches on common test problems. *AIAA Journal*, 35(1):178–186, 1997. ISSN 0001-1452. doi: 10.2514/2.7431. URL https://doi.org/10.2514/2.7431.

[30] Philippe Dépincé, Benoît Guédas, and Jérôme Picard. Multidisciplinary and multiobjective optimization: Comparison of several methods. In *7th World Congress on Structural and Multidisciplinary Optimization*, Seoul, South Korea, May 2007. URL https://hal.archives-ouvertes.fr/hal-00449605. 10 pages.

[31] Evin J. Cramer, John E. Dennis Jr., Paul D. Frank, Robert Michael Lewis, and Gregory R. Shubin. Problem formulation for multidisciplinary optimization. *SIAM Journal on Optimization*, 4(4):754–776, 1994. ISSN 1052-6234. doi: 10.1137/0804044. URL https://doi.org/10.1137/0804044.

[32] Ravindra V. Tappeta, Somnath Nagedra, and John E. Renaud. Concurrent sub-space optimization (csso) mdo algorithm in isight, csso in isight: validation and testing. *GE Research and Development Center, 97CRD186, class*, 1, 1998.

[33] Oscar Sheynin. *Gauss and the Method of the Least Squares*, volume 219. 1999. doi: 10.15611/sps.2014.12.01.

[34] Takashi Isobe, Eric D. Feigelson, Michael G. Akritas, and Gutti J. Babu. Linear regression in astronomy. *Astrophysical Journal v.364, p.104*, 364:104–113, November 1990. doi: 10.1086/169390. URL http://adsabs.harvard.edu/abs/1990ApJ...364..104I.

[35] Aitzol Astigarraga, Jose Maria Martinez-Otzeta, Igor Rodriguez, Basilio Sierra, and Elena Lazkano. *Markov Text Generator for Basque Poetry*. 2017. ISBN 978-3-319-64205-5. doi: 10.1007/978-3-319-64206-2_26.

[36] Gio Wiederhold and John McCarthy. *Arthur Samuel: Pioneer in Machine Learning*, volume 36. 1992. doi: 10.1147/rd.363.0329.

[37] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370.

[38] Takehisa Yairi, Yoshinobu Kawahara, Ryohei Fujimaki, Yoichi Sato, and Kazuo Machida. Telemetry-mining: A machine learning approach to anomaly detection and fault diagnosis for space systems. volume 2006, pages 8 pp. – 476, 07 2006. doi: 10.1109/SMC-IT.2006.79.

[39] Ney Secco and Bento Mattos. Artificial neural networks to predict aerodynamic coefficients of transport airplanes. *Aircraft Engineering and Aerospace Technology*, 89:211–230, 03 2017. doi: 10.1108/AEAT-05-2014-0069.

[40] Karu Nanithi, Gajal Lakshmi, Malar Vizhi, and Sailesh Wari. A study on comparison of jacobi, gauss-seidel and sor methods for the solution in system of linear equations. *International Journal of Mathematics Trends and Technology*, 56:214–222, 04 2018. doi: 10.14445/22315373/IJMTT-V56P531.

[41] Florin B. Manolache and Sorin Costiner. Parallel processing approach for multidisciplinary optimization algorithm.

[42] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. 09 2017.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[44] Russell D. Reed and Robert J. Marks. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262181908.

[45] Mahesh Panchal. Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture. *International Journal of Computer Science and Mobile Computing*, 3:455–464, 01 2014.

[46] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *CoRR*, abs/1712.02029, 2017. URL http://arxiv.org/abs/1712.02029.

[47] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012. URL http://arxiv.org/abs/1206.5533.

[48] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks, 2018.

[49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56): 1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[50] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[51] Ming-Hua Lin, Jung-Fa Tsai, Nian-Ze Hu, and Shu-Chuan Chang. Design optimization of a speed reducer using deterministic techniques. *Mathematical Problems in Engineering*, 2013:1–7, 11 2013. doi: 10.1155/2013/419043.

[52] Wei Li, Mi Xiao, Yongsheng Yi, and Liang Gao. Maximum variation analysis based analytical target cascading for multidisciplinary robust design optimization under interval uncertainty. *Advanced Engineering Informatics*, 40:81–92, 04 2019. doi: 10.1016/j.aei.2019.04.002.

[53] Shen Lu and Harrison M. Kim. A regularized inexact penalty decomposition algorithm for multidisciplinary design optimization problems with complementarity constraints. 2010.

[54] Christina L. Bloebaum. Coupling strength-based system reduction for complex engineering design. *Structural optimization*, 10(2):113–121, Oct 1995. ISSN 1615-1488. doi: 10.1007/BF01743538. URL https://doi.org/10.1007/BF01743538.

[55] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition, 2003. doi: 10.1137/1.9780898718003. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718003.

[56] Srinivas Kodiyalam. Evaluation of methods for multidisciplinary design optimization (mdo). Technical report, Engineous Software, Inc. Morrisville, NC United States, NASA/CR-1998-208716, 09 1998. URL https://ntrs.nasa.gov/citations/19990019380.

[57] Justin Gray, Kenneth T. Moore, Tristan A. Hearn, and Bret A. Naylor. Standard platform for benchmarking multidisciplinary design analysis and optimization architectures. *AIAA Journal*, 51(10):2380–2394, 2013. doi: 10.2514/1.J052160. URL https://doi.org/10.2514/1.J052160.

[58] Imco Van Gent. Ssbjkadmos. URL https://pypi.org/project/ssbjkadmos/#description.

[59] Jan Golinski. Optimal synthesis problems solved by means of nonlinear programming and random methods. *Journal of Mechanisms*, 5, 09 1970. doi: 10.1016/0022-2569(70)90064-9.

[60] Nathan Tedford and Joaquim R. R. A. Martins. Comparison of mdo architectures within a universal framework. 2006.

[61] Sumeet Parashar and Christina Bloebaum. *Decision Support Tool for Multidisciplinary Design Optimization (MDO) Using Multi-Domain Decomposition*. doi: 10.2514/6.2005-2200. URL https://arc.aiaa.org/doi/abs/10.2514/6.2005-2200.

[62] Oleg V. Gendelman, Dawei Du, and Dan Simon. Complex system optimization using biogeography-based optimization. *Hindawi Publishing Corporation*, 1:10.1155/2013/456232, 12 2013. doi: https://doi.org/10.1155/2013/456232.

[63] Erik D. Olson. *Three-Dimensional Piecewise-Continuous Class-Shape Transformation of Wings*. doi: 10.2514/6.2015-3238. URL https://arc.aiaa.org/doi/abs/10.2514/6.2015-3238.