# A Graph Theory Based Approach to MDAO Problem Formulation

David Pate, *Dr. Brian German †Justin Gray,‡

**blah blah blah**

## Nomenclature

| | |
|---|---|
| AAO | All-At- |
| MDAO | Multidisciplinary Design Analysis and Optimization |
| FPF | Fundamental Problem Formulation |

## I. Introduction

As the size and complexity of engineering systems grow, the time and expense for setting up analysis models grow with them. Multidisciplinary Design Analysis and Optimization (MDAO) frameworks such as OpenMDAO[1] and ModelCenter have enabled a new level of analysis tool integration and paved the way for models with more analyses and increasing numbers of interdisciplinary couplings. That new capability has created a new challenge since configuring a model with the larger nubmer of analyses, associated inputs and outputs can be very difficult. Even for models with 10's of analyses, there could be hundreds of variables that need to be managed. It is not hard to imagine that the task of combining all the analyses into a consistent system model capable of solving a relevant engineering design problem could easily become much more costly than createing the discipline analyses themselves. Thus, for a large system as the couplings between the disciplines begin to dominate the design space, the couplings between the analyses begin to dominate the job of setting up the model.

When building a model, coupling represents the reciprical flow of information between two analyses. We propose the use of graph-based syntax to describe the flow of information from inputs, through analyses, to outputs. If you allow for any free inputs to become design variables and some set of the outputs to become design objectives and constraints, then the graph represents the problem formulation for a given design effort. The graph-based syntax presented has application in all phases of the design process. It can be used early on, to help construct a valid problem formulation from the set of available analysis tools. It also provides a consistent structure to describe problem formulation that provides a foundation to apply algorithmic methods for selecting and then implementing effective solution strategies inside an MDAO framework.

We've already identifed that a problem formulation graph should include design variables, analysis tools and their associated inputs and outputs, and objectives and constraints. Notably absent from that list are any kind of solvers, optimizers, or other iterative solution finding tools. At its most fundamental state, a problem formulation includes only information about what is being sought after in a design or what the goals of a given problem are. It need not contain any information about solution paths or strategies to reach those goals. While solution information can be represented with the graph syntax presented here, a unique aspect of this syntax is that such information always remains seperable from the underlying problem formulation. In other words, a given a graph representing a specific solution strategy for a specific problem formulation, it is possible to remove the solution information from the graph and reclaim the more basic problem formulation.

*Georgia Tech
†Georgia Tech...
‡Aerospace Engineer, MDAO Branch, Mail Stop 5-11, AIAA Member

## II. Specific vs Fundamental Problem Formulation

The Fundamental Problem Formulation (FPF) for any given problem will be constant regardless of which MDAO framework, optimization algorithm, iterative solver, or solution strategy is used to solve the problem. For example, consider the following notional problem:

$$\begin{aligned} given \quad & A : x, y \rightarrow m, z \\ & B : x, z \rightarrow y^t \\ min. \quad & f(m) \\ w.r.t. \quad & x, y, z \\ s.t. \quad & g(y^t, y) = 0 \end{aligned} \tag{1}$$

$A$ and $B$ represent analysis tools, and $f$ and $g$ are the objective and constraint functions respectively. Equation 1 makes an inherent assumption about the solution strategy for the problem. Analysis $A$ outputs $z$, which is an input to $B$. Hence, $A$ should be run before $B$ with with $y$ being iterated on to convergence with $y^t$. However, a slightly different formulation is equally valid and still represents the exact same problem:

$$\begin{aligned} given \quad & B : x, z \rightarrow y \\ & A : x, y \rightarrow m, z^t \\ min. \quad & f(m) \\ w.r.t. \quad & x, y, z \\ s.t. \quad & g(z^t, z) = 0 \end{aligned} \tag{2}$$

Equation 2 differs only slightly from Eq. 1. $A$ is now dependent on the output of $B$, and $z$ will be iterated on to convergence with $z^t$. Now the problem can be solved by running $B$ first and then $A$. Since the formulations in Eqs. 1 and 2 both describe the same problem and neither can be the FPF. They are both specific versions of the same more fundamental description of the problem that is common between them. We present the FPF as follows:

$$\begin{aligned} given \quad & A : x, y \rightarrow m, z^t \\ & B : x, z \rightarrow y^t \\ min. \quad & f(m) \\ w.r.t. \quad & x, y, z \\ s.t. \quad & g1(z^t, z) = 0 \\ & g2(y^t, y) = 0 \end{aligned} \tag{3}$$

The FPF in Eq. 3 differs from both Eq. 1 and Eq. 2 because it has two constraints which both must be met. The presence of both of these constraints fully decouples the problem so that either $A$ or $B$ could be run first or both could be run simultaneously. By removing either constraint and replacing it with a direct dependence between the two analyses you could regain the earlier two problem formulations. Alexandrov and Lewis demonstrated the value of a more modular approach to problem formulation because it enables one to transition between different MDAO solution strategies depending on the specifics of the problem.[2]

## III. Existing Graph–Based Problem Formulation Syntax

As shown above, the mathematical language for specifying problem formulations is very general and can be used both for fundamental and specific problem formulations. Tedford and Martins used the above mathematical syntax to specify the FPF for a set of test problems and also to describe specific formulations for solving them with a number of optimization architectures.[3] Their work demonstrates clearly how multiple specific problem formulations can all relate back to a common FPF. The challenge with using this traditional

mathematical syntax is that it is not easily manipulated or analyzed. A number of graph-based methods have been used successfully to translate the mathematical syntax into a more useful computational form.

Steward's Design Structure Matrix (DSM) is a square adjacency matrix which captures the relationship between analysis tools where off diagonal elements of the matrix indicate coupling.[4] Since a DSM describes a square adjacency matrix, it can be represented in an equivalent directed graph where nodes represent analysis tools and edges represent information dependence between those tools. The ordering of elements in a DSM can be used to indicate execution order. For more complex problems, choosing the proper order to run analysis tools is a non-trival task. Rogers et. al developed DeMAID to manipulate a DSM to find an ordering for analysis tools that reduces the cost of solving highly coupled systems.[5] This re-ordering is done through row operations on the DSM matrix and yields multiple specific problem formulations which all solve the same FPF. In other words, manipulation of the DSM does not fundamentally alter the problem formulation, which makes DSM an excellent foundation for specifying the FPF itself.

Despite it's attractive properties, a DSM by itself is insufficient to describe complete problem formulations. Traditional DSM only captures information about data dependency between analyses. Objective and constraint information is missing from the description of the problem. An alternative matrix-based syntax, called a Functional Dependency Table (FDT), was proposed by Wagner and Papalambros. FDT represents the relationship between functions, including objectives and constraints, and specific variables that affect them.[6] Similar to DSM, FDT also describes an adjacency matrix of a graph. Unlike the DSM graph, however, the graph is undirected and nodes can represent analysis tools, objectives, or constraints. Edges between nodes represent a dependence on the same variable. Michelena and Papalambros made use of the FDT to solve a graph partitioning problem that yielded more efficient optimization problem decompositions.[7] While FDT succeeds at capturing the information about objectives and constraints, it can not capture the coupled data dependency that DSM captures. For instance, we know from the FPF in Eq. 3 that the objective, $f$, is dependent on the output, $m$, of analysis $A$. You could not determine that from the FDT in Fig. 1 alone. This missing information means that, while FDT is very useful for partitioning problems, it is also not sufficient to contain a complete problem formulation.

| | $x$ | $y$ | $y^t$ | $z$ | $z^t$ | m |
|---|---|---|---|---|---|---|
| $A$ | 1 | 1 | | | | |
| $B$ | 1 | | | 1 | | |
| $f$ | | | | | | 1 |
| $g1$ | | | | 1 | 1 | |
| $g2$ | | 1 | 1 | | | |

**Figure 1: Functional Dependency Table (FDT) for Eq. 3**

Lamb and Martins included objectives and constraint functions as nodes in an Extended DSM (XDSM) in order to capture a more complete description of solution strategies for MDAO problems.[8] XDSM retains the square adjacency matrix form from DSM, but by adding in the new elements they partially combined a traditional DSM with an FDT. This allows XDSM to represent data dependency between multiple analysis tools as well as between analysis tools and objective/constraint functions. With the additional information included in an XDSM, Lu and Martins applied both ordering and partitioning algorithms on an MDAO test problem named the Scalable Problem.[9]

Although XDSM captures part of the functional aspects of FDT, it leaves out the variables themselves. As a result, all variable information is aggregated so that for the problem from Eq. 3 you can say that $A$ depends on $B$ or vice versa, but you can't identify which individual variables are interacting in the dependency cycle. Without the detailed variable information, you can't construct the compatibility constraints necessary to implement the problem. Additionally, XDSM requires the use of solver and optimizer blocks to represent the relationship between design variables and objectives/constraints. By introducing solver or optimizer blocks XDSM automatically provides some kind of solution strategy. The XDSM for Eq. 1 is given in Figure 2. This diagram is shown with an assumed gauss-siedel iteration scheme and a MDF solution architecture. Hence XDSM is too specific for use with a fundamental problem formulation.

Of the three methods, XDSM comes the closest to fully describing a problem formulation. Thought it does not contain specific variable information, this could be easily added back in simply by putting the

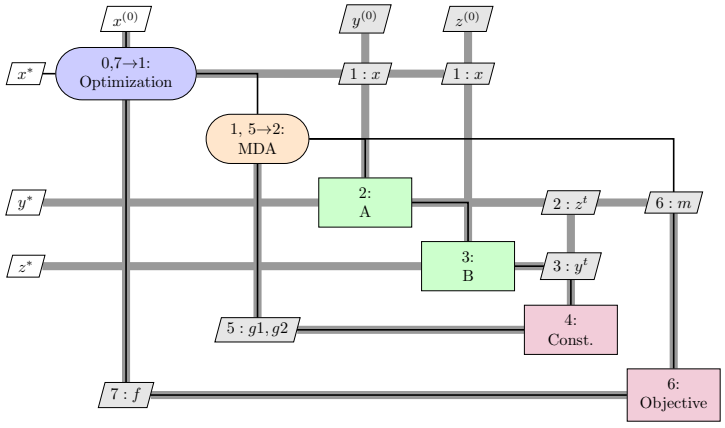



**Figure 2: XDSM for Eq. 1, with a gauss-siedel iteration and MDF solution architecture.**

variables back into the graph and drawing the apprpriat connections between them and the relevant analysis nodes. However, it's dependence on solution strategy specifics to represent couplings can not be removed without stripping it of the useful information contained in an FDT. In the following sections we describe a new graph syntax that retains the features of XDSM, without the dependence on solution strategy (although it does allow for that information to be present). Thus the new syntax allows for the compelete specification of an FPF.

## IV. New Graph–based Syntax

The ultimate goal of the new graph–based syntax presented here is to be able to fully describe the fundamental problem formulation for a complex design task, while still being able to accomodate the more specific case when a solution strategy is applied to the problem formulation. In order to achieve that goal the graph needs to accomodate a number of constructs from MDAO problem:

- Analysis tools
- Local and global variables
- Coupling between analyses
- Design variables
- Objective or objectives
- Local and global Constraints
- Iterative loops from solvers and optimizers

The following section provides an overview of graph theory basics, then describes in detail the different node and edge types, as well as rules governing allowable egdegs between nodes, used as the foundation of the new graph syntax. These basic graph primatives are them mapped to the MDAO constructs stated above, and used to build three distinct graph forms:

- Maximal Connectivity
- Fundamental Problem Formulation
- Specific Solution

Each of these forms serves a different purpose in the design process, and we present some transformations that can be applied to transition between any two of the three graph forms.

## A. Graph Theory Basics

The notation used in this work is adapted from Diestel.[10] A *graph* is a pair $G = (V, E)$ of sets such that $E \subseteq V \times V$, which means that the elements of $E$ are 2–element subsets of $V$. The set $V$ contains the *vertices* or *nodes* and the set $E$ contains the *edges*. For a *directed graph** we construct $E$ as a set of ordered pairs instead of a set of sets. Each ordered pair represents an edge starting at the node indicated by the first entry and directed to the node indicated by the second entry. Edge $e = (x, y)$ may be referred to simply as $xy$ and we say $y = E(x)$.

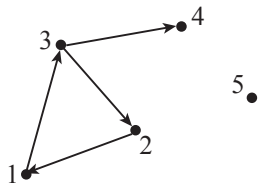



**Figure 3: Example directed graph.**

As an example, for the directed graph shown in Fig. 3 we have

$$V = \{1, 2, 3, 4, 5\},$$
$$E = \big\{(1, 2), (3, 2), (1, 3), (3, 4)\big\}.$$

## B. MDAO Graph Primatives

There are three node types:

**variable node** represents passing of data.

**model node** represents the handling of data.

**driver node** represents constrol structures capable of managing iteration

There are two edge types:

**fixed edge** A fixed edge may not be removed.

**free edge** A free edge may be removed.

A number of restrictions are places on the model node with respect to the number and type of edges that can be connected to it:

1. A model node can only have one edge directed to or from another model node.
2. A model node can only have fixed edges directed in or out.
3. A model node must have at least one edge directed in and at least one edge directed out.

The MDO problem formulation concepts represented by nodes are given in Table 1, and the concepts represented by edges are given in Table 2.

**Table 1: Problem formulation concept represented by a node**

| edges directed in | edges directed out | node representation |
|---|---|---|
| only free edges | none | objective or constraint |
| none | at least two free edges | global input |
| none | one fixed edge | design variable |

Finally, an analysis code is represented by an *analysis block*, which is a collection of model nodes, variable nodes, and fixed edges with a specific structure. This structure is derived from the need to represent each input and output individually while representing the actual analysis with a single node or edge. The variable nodes each represent a single input or output of the analysis block. Each of these nodes is connected to a

**Table 2: Problem formulation concept represented by an edge**

| from node | to node | edge representation |
|---|---|---|
| variable | variable | connection/passing of a variable |
| variable | model | local input |
| model | variable | local output |
| model | model | function call |

single model node via a fixed edge, which represents the gathering of the inputs to begin the analysis or the disseminating of outputs after the analysis. Finally, free edges are used to connect the variable nodes of the analysis block to other variables nodes outside of the analysis block. In this way, the analysis block graph is a fundamental building block of the MDAO problem data flow.

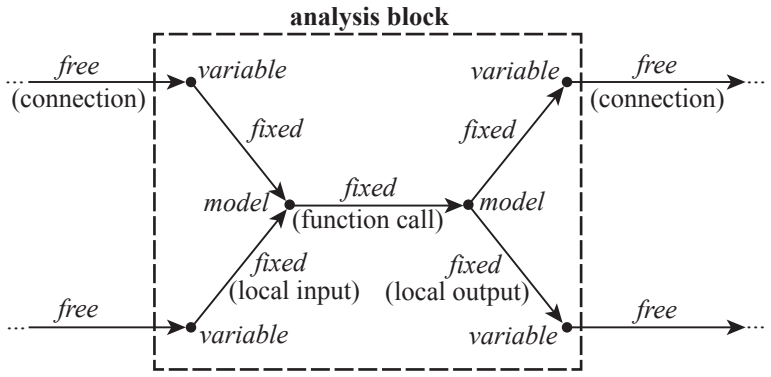



**Figure 4: Example analysis block. The each node type and edge type is labeled in italics and annotated parenthetically.**

## V. Graph–based Representation of the Fundamental Problem Formulation

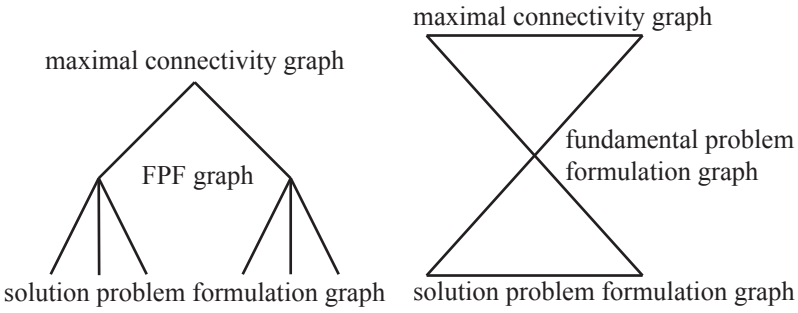
Next we provide the meaningful graphs that can be created from the suggested syntax. The first graph is the *maximal connectivity graph* which represents the full potential of all the analysis codes being considered, i.e. each analysis code is represented by an analysis block, and each potential connections between variables is represented by a free edge. The second graph that may be represented is *fundamental problem formulation graph*, which is the graph with the fewest number of edges and nodes needed to provide a valid problem formulation, as disscussed subsequently. Finally, a *solution problem formulation* may be represented by including additional edges and node times to represent the optimization architecture, though this is beyond the scope of this paper.

The relationship between these three graphs is depicted in Figs. 5(a) and 5(b). The tree diagram demonstrates the fact that it is generally possible to obtain multiple FPFs from a single maximal connectivity graph. This may correspond to different down–selections of analysis codes, different connections between them, or both. Then, for each FPF, different SPFs may be obtained by implementing different optimization architectures. The hourglass shape indicates that the FPF graph is the smallest graph that may represent a specific problem formulation. The FPF is obtained from the MCG by removing nodes and edges, and the SPF is obtained from the FPF by adding nodes and edges.

In this section we introduce the remaining necessary notion and give graph–theory based definitions of the MCG and FPF, and then process to demonstrate how the FPF is obtained from the MCG.

### A. Additional Notation

For node $v \in V$ the edges directed out are given by $E(v)$ and the edges directed into $v$ are given by $E^{-1}(v)$; $E(E(v))$ is denoted as $E^2(v)$, and likewise for additional levels. The *indegree* of a node is the number of edges directed in and is denoted as $\deg^-(v)$, and the *outdegree* is the number of edges directed out and it is




(a) number of possible graphs (b) graph size

**Figure 5: The relationship between the MCG, FPF, and SPF.**

denoted as $\deg^+(v)$. We also define the *upper indegree limit*

$$\deg_u^-(v) : V \to \mathbb{N} \tag{4}$$

and the *lower indegree limit*

$$\deg_l^-(v) : V \to \mathbb{N}. \tag{5}$$

These user-specified limits set the number of edges that may be directed into a node for a problem formulation to be valid. For example, a variable node $v$ will have $\deg_u^-(v) = \deg_l^-(v) = 1$, unless it is a multi–fidelity variable, in which case the user may specify some upper limit higher than one.

To keep track of which nodes and edges are which type, let $T_{\text{node}}$ and $T_{\text{edge}}$ be sets containing the possible node types and edge types, respectively, and then define mappings $t_{\text{node}} : V \to T_{\text{node}}$ and $t_{\text{edge}} : E \to T_{\text{edge}}$ to assign a type to each node and edge. Then, for example, for a variable node $v$ we have $t_{\text{node}}(v) =$ 'variable'.

### B. Maximal Connectivity Graph

To construct the maximal connectivity graph, we assume that a set of codes, global inputs, and objectives and constraints (collectively called global outputs). The codes are represented by analysis blocks $A_i = (V_{A_i}, E_{A_i}),\ i = 1, \ldots, m$, the global inputs are represented a set of variable nodes $I$, and the global outputs are represented by a set of variable nodes $O$. We assume that $O$, $I$, and $A_i$ are given, and that any potential connection between variables is given in the form of the free edges in the set $C_M$. Then we may construct the maximal connectivity graph $M = (V_M, E_M)$ as

$$V_M = I \cup O \cup \left( \bigcup_{i=1}^{m} V_{A_i} \right),$$
$$E_M = C_M \cup \left( \bigcup_{i=1}^{m} E_{A_i} \right),$$

The MCG $M$ is uniquely determined by the given set of analysis blocks, the required outputs, and the given global inputs. In the cases where the set of global inputs $I$ is not known a priori, the process of obtaining the FPF will reveal the required inputs, as discussed subsequently.

### C. Fundamental Problem Formulation Graph

We now define the fundamental problem formulation graph, $F = (V_F, E_F)$, as a directed graph meeting the following conditions

(1) $V_F = I_F \cup O \cup \left( \bigcup_{i \in \mathcal{A}} V_{A_i} \right),\ I_F \subset I$

(2) $E_F = C_F \cup \left( \bigcup_{i \in \mathcal{A}} E_{A_i} \right)$

(3) $\forall v \in V_F$ with $t_{\text{node}}(v) =$ 'variable,' $\deg_l^-(v) < \deg^-(v) \leq \deg_u^-(v)$

The set $\mathcal{A}$ is an index set containing the indices of the analysis blocks in $F$; for the case where all of the analysis blocks are used, we would have $\mathcal{A} = \{1, 2, \ldots, m\}$. The first requirement for $F$ is that $V_F$ be composed of all of the global outputs (the required objectives and constraints), the nodes from each analysis block in $F$, and any global input nodes that are needed, $I_F$. The second requirement suggests that the edges in $F$ comprise the edges for each analysis block and the edges between them, the global inputs, and the global outputs. The final requirement is specific to the number of edges directed into a variable. If there are no edges directed inward, the node is called a *hole*, and if more connections are directed in than are allowed, the node is called a *collision*; a hole is allowed if $\deg_l^-(v) = 0$. The final requirement is therefore a requirement on $C_F$.

### D. Obtaining the Fundamental Problem Formulation Graph

In general, there may be multiple different graphs that satisfy the FPF conditions, though there may be none at all. Here, we describe a process for obtaining an FPF by starting with the MCG and disconnecting free edges until the FPF conditions are met. Then the problem is reduced to deciding which free edges to remove.

Then, mathematically, the FPF starts with $C_{F,0} = C_M$. First address the nodes where $\deg^-(v) < \deg_l^-(v)$ then the ones where $\deg^-(v) > \deg_u^-(v)$

1. The first step is to detect holes and disconnect the free edges following them. These free edges are removed because they represent variables which cannot be determined because the analysis function does not have adequate inputs. The set of variable nodes which are holes is created as

$$H = \{v \in V | \deg^-(v) < \deg_l^-(v)\} \tag{6}$$

   Then the updated set of edges is

$$C_{F,1} = C_{F,0} \setminus \{\ e \in E,\ e = xy | x = E^3(v) \text{ for } v \in H\} \tag{7}$$

   This step is demonstrated by Fig. 6.

2. (())

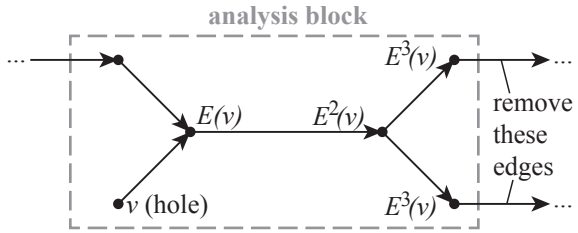



**Figure 6: Example variable node indicating a hole.**

### E. Solution Graph Syntax

What is the difference between a problem formulation and a problem solution method? Convert from a cyclic graph, to an acyclic graph

- Cycles indicate convergence loops or design variable loops
- Problem can't be solved until all loops are *removed* by adding solvers/optimizers
- *Special* nodes for solvers and optimizers that *break* loops (from an algorithmic point of view)
- FPF represents the minimal amount of information necessary to define a problem
- Any solution path grows the graph complexity by adding edges and nodes (or possibly have an empty solution graph, which you build up as you remove edges from problem formulation graph?)

## VI. Example Problem

## VII. Applications

## VIII. Conclusions

## References


[1]Gray, J., Moore, K. T., Hearn, T. A., and Naylor, B. A., "A Standard Platform for Testing and Comparison of MDAO Architectures," *8th AIAA Multidisciplinary Design Optimization Specialist Conference (MDO)*, Honolulu, Hawaii, 2012, pp. 1–26.

[2]Alexandrov, N. and Lewis, M., "Algorithmic Perspectives on Problem Formulations in MDO," *8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, No. September, Long Beach, CA, 2000.

[3]Tedford, N. P. and Martins, J. R. R. a., "Benchmarking multidisciplinary design optimization algorithms," *Optimization and Engineering*, Vol. 11, No. 1, March 2009, pp. 159–183.

[4]Steward, D. V., "The Design Structure System: A Method for Managing the Design of Complex Systems," *IEEE Transactions on Engineering Management*, Vol. EM-28, No. 3, 1981, pp. 71–74.

[5]Rogers, J., McCulley, C., and Bloebaum, C., *Integrating a genetic algorithm into a knowledge-based system for ordering complex design processes*, NASA Technical Memorandum, Hampton Virginia, 1996.

[6]Wagner, T. C. and Papalambros, P. Y., "A general framework for decomposition analysis in optimal design," *Advances in Design Automation*, Vol. 2, 1993, pp. 315–325.

[7]Michelena, N. F. and Papalambros, P. Y., "A Hypergraph Framework for Optimal Model-Based Decomposition of Design Problems," *Mechanical Engineering*, Vol. 8, No. 2, 1997, pp. 173–196.

[8]Lambe, A. B. and Martins, J. R. R. A., "Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes," *Structural and Multidisciplinary Optimization*, 2012.

[9]Lu, Z. and Martins, J., "Graph Partitioning-Based Coordination Methods for Large-Scale Multidisciplinary Design Optimization Problems," *ISSMO Multidisciplinary Analysis Optimization Conference*, No. September, Indianapolis, Indiana, 2012, pp. 1–13.

[10]Diestel, R., *Graph Theory*, Springer, 2010.