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

The number of analysis tools required in multidisciplinary design optimization (MDO) studies is growing in parallel with the increasing scope of typical MDO problems. An example of this growth in scope can be observed in the historical evolution of the disciplines involved in MDO problems in aircraft design. Multidisciplinary optimization emerged as a separate field from structural optimization through the need to introduce formal techniques for managing the coupling of aerodynamic loads and structural deformations, through the linking of aerodynamic vortex lattice or panel methods with structural finite element models [[REF]]. Subsequently, flight performance and life cycle economics tools were integrated into MDO analysis workflows for conceptual and preliminary design studies [[REF]]. Currently, MDO problems for aircraft design also often include tools for aircraft noise and emissions. In sum, it is now commonplace for 5-10 analysis tools to be employed for typical aircraft design optimization studies [[REF]].

The number of analysis tools is expected to grow in the future as the scope of MDO problems continues to evolve, and as computing is increasingly commoditized. This expansion of scope will be driven, in part, by consideration of additional disciplines. Current trends on the horizon for aircraft MDO studies include incorporation of manufacturing analyses [[REF]], subsystem performance [[REF]], and fleet-level aggregate models of emissions, noise, or economics [[REF]]. A recent survey by Shan and Wang covered a range of different techniques proposed to solve High Dimentional, Expensive, Black-box (HEB) problems. The large amount of relevant research they identified demonstrates the growing interest large scale MDO problems that arise from multiple disciplines.[1]

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[2] 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. Even for models with 10's of analyses there could be hundreds or thousands of variables that need to be linked together. Commonly different analyses will provide competing values for the same physical quantity, and these conflicts need to be resolved somehow. To make matters worse, the competeing values may not even be in the same format. These occurrences are particularly acute when analysis tools have differing fidelities. For example, an abstracted aerodynamic analysis such as an empirical drag buildup model may return only integrated drag, whereas a CFD tool may return pressure and shear stress distributions across the entire surface grid. If an analysis downstream of the aerodynamics tool needs only integrated drag as an input, then the designer has a free choice of which of the two possible aerodynamics

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

analysis tools to select to provide the drag estimate (presuming that drag can be computed from surface distributions by a simple integration algorithm). On the other hand, if a downstream analysis needs a pressure distribution in order to compute pitching moment, for example, then any feasible MDO problem formulation must include CFD or similar analysis in the data flow, regardless of whether the empirical drag buildup model is also included.

With all the added complexity from larger models 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. Just as for a large system 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. It is this problem of determining sets of analysis tools and their interconnectivities to form realistic multidisciplinary problems that is the subject of this paper. We are motivated by the following notional but real problem of organizing an MDO study for a complex system:

A new system is being designed for which there is little or no historical precedent. The system is complex, as measured by the number of disciplines and/or components involved in the analysis. A general optimization problem statement has been formulated based on system- level objectives and constraints; however, it is unclear which engineering analysis tools should be interconnected in order to solve the optimization problem. A team of disciplinary and/ or component design engineers has been formed in which each engineer has expertise in a particular analysis tool or component model. The engineers meet to discuss the approach to interconnecting their tools to achieve the required system-level MDO model.

Our goal is to develop formalism for expressing analysis interconnectivity and for determining feasible data flows to assist an engineering team conducting this task. Because the problem deals with interconnectivity, we base our approach on the representations and techniques of graph theory. The proposed graph syntax is closely related to the REMS system proposed by Alexandrov and Lewis.[3] The approach taken here should be viewed as an extention of the concepts proposed in REMS with the goals of providing greater insight into design problem structure. The approach begins by fconstructing the *maximal connectivity graph (MCG)* describing all possible interconnections between the analysis tools proposed by the engineers. Graph operations are then conducted reduce the MCG down to a *fundamental problem graphs (FPG)* that describes the minimum set of analysis tools needed to solve the specified system-level design problem. The FPG does not predispose any particular solution procedure for the problem; any relevant MDO solution architecture, e.g. MDF, IDF, CO, BLISS, could be selected *post facto* to implement the system-level optimization by constructing a *problem solution graph (PSG)* and the corresponding problem solution approach. The concept of the FPG and the identification of feasible FPGs are the main contributions of the paper. For completeness we include node and edge types for building a PSG, but the details of how a given problem can be decomposed are beyond the scope of this work.

The paper is organized as follows. First, we describe the differences between a fundamental problem formulation based only on the design problem that the engineers desire to solve and on the available analysis tools, and a specific problem formulation, which additionally presumes a specific solution to the problem. Next, we survey the literature in applying graph theoretic and formal language approaches to multidisciplinary design problem formulation. We then discuss our graph syntax and representation of MDO problems and describe how different structures within a graph map to important features of an MDAO problem. Next we present a procedure for determining possible FPG from a given MCG. Finally, we present the graph based formulation of commercial aircraft design problem and discuss additional applications enabled by our approach.

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 then neither can be the FPF. They are both specific versions of a more fundamental description of the problem that is common between them. We present the fundamental formulation 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 formulation 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. Alexandrov and Lewis, in their work with REMS, assert that fully decoupling the analyses is key to achieving reconfigurability in the problem formulation.[3]

III. Existing Graph-Based 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.[4] Their work demonstrates clearly how multiple specific problem formulations can all relate back to a common FPG. 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.[5] 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.[6] 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.[7] 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.[8] 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

Alexandrov and Lewis introduced a graph based syntax called REMS which incorporates objectives and constraints into a graph, effectively combining FDT and DSM.[3] REMS retains the square adjacency matrix form from DSM, but by adding in the new elements it partially combined a traditional DSM with an FDT. This allows REMS to represent data dependency between multiple analysis tools as well as between analysis tools and objective/constraint functions. One significant conclusion from REMS is the need for variables that are local to discipline analyses. This allows all information exchange to be stated as equality constraints, which become edges in the graph. Additionaly, REMS address the need to be able to transition between multiple solution strategies while keeping a constant graph representation of the fundamental problem formulation. REMS only provides syntax for variables and functions, but does not provide for inclusion of solvers or optimizers in the graph.

Lamb and Martins also 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.[9] Unlike REMS, XDSM also includes nodes for solvers and optimziers to enable complete definition of MDAO architectures. Martins uses XDSM to describe 13 different optimziation architectures in a survey paper that provides a novel classification of the different techniques.[10] 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.[11]

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.

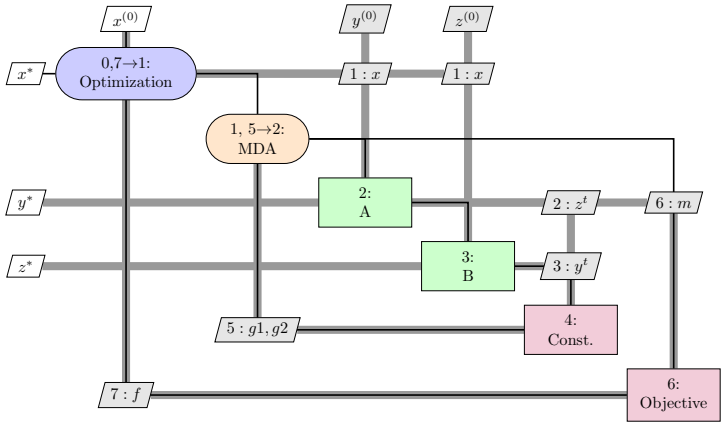


Figure 2: XDSM for Eq. 1, with a gauss-siedel iteration and MDF solution architecture.

Hence XDSM is too specific for use with a fundamental problem formulation.

Of all the methods, REMS comes the closest to fully describing a problem formulation, but lacks the ability to represent optimziers and solvers in more specific problem formulations. While XDSM does include optimizers and constraints, it relies on their presence and can't be used for a more fundamental problem statement. We propose a new graph syntax that combines the key features of REMS and XDSM and provides new language to enable more complete description of MDAO problems.

IV. Requirements for New Graph Syntax

The ultimate goal of the new graph–based syntax presented here is to be able to fully describe the general structure of an MDAO problem indpendently of any solution information, while still being able to accomodate the more specific case when a solution strategy is applied. In order to achieve that goal the graph needs to accomodate a number of constructs of MDAO problems:

- Analysis tools and connections between them
- Design variables, objectives, and constraints
- Local and global properties
- Coupling between analyses
- Multi-fidelity analyses

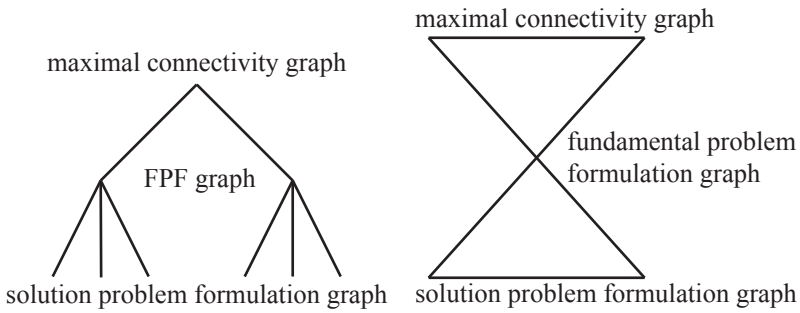
Beyond those basic constructs, there are also three phases of a design process that all need to be representable with the new syntax. Firstly there is the initial problem definition phase where the specific analysis tools and design goals are identified. At the end of this phase, a single formal problem formulation is selected specifying design variables, constraints, objectives, analysis tools, etc. Lastly some specifc procedure for solving the problem is selected, for example picking an MDAO optimization architecture. Using the proposed graph syntax, each of these phases can be represented with a related graph.

- Maximal Connectivity Graph (MCG)
- Fundamental Problem (FPG)
- Problem Solution Graph (SPG)

The *maximal connectivity graph* represents the first phase with all analysis codes being considered and all possible connections between them also present. The second graph is *fundamental problem formulation graph*, which is the smallest possible graph that still fully defines a given problem formulation. Finally, a

specific problem formulation may be represented by including additional edges and nodes to represent the solution strategy being employed to solve the problem.

The relationship between these three graphs is depicted in Figs. 3(a) and 3(b). The tree diagram demonstrates the fact that it is generally possible to obtain multiple FPGs from a single maximal connectivity graph. This may correspond to different down–selections of analysis codes, different connections between them, or both. Each down–selection shrinks the number of possible FPGs that could be reached untill only one is remaining. Then, from a single FPG, different PSGs may be obtained by implementing different solution strategies. If you consider the size of a graph to be the sum of all its edges and nodes then the hourglass shape in Fig. 3(b) illustrates how the MCG gets reduced to a single FPG, then multiple possible PSGs exist to solve the problem. In other words the FPG is obtained from the MCG by removing nodes and edges, and the PSG is obtained from the FPG by adding nodes and edges.



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

Figure 3: The relationship between the MCG, FPG, and PSG.

V. Syntax Definition

In the rest of this section we present the necessary graph theory fundamentals to construct any of the different graphs discussed above. The notation used in this work is adapted from Diestel.[12] 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. 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.

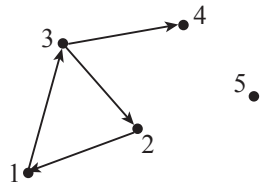


Figure 4: Example directed graph.

As an example, for the directed graph shown in Fig. 4 we have

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

Let I be a nonempty set of counting numbers such that for each $i \in I$ there is a corresponding set A_i. The set of sets $\mathcal{A} = \{A_i \mid i \in I\}$ is called an indexed family of sets with index i and indexing set I.[13] The union over this family of sets can be described in a few different ways:

$$\bigcup_{i \in I} A_i = \bigcup_{A \in \mathcal{A}} A = \{x \mid x \in A \text{ for some } A \in \mathcal{A}\}. \tag{4}$$

Lastly, the cardinality of a set B is the number of elements in B and is denoted as $|B|$.

A. Node and Edge Types

There are four node types:

variable: represents scalar or array data

model: responsible for mapping inputs to outputs

expression: simple algebreic combinations of variables (such as for constraint of objective functions)

driver: control logic capable of managing iteration

In addition to the four node types, there are two edge types:

connection edge: Exchange of information between two nodes. These edges can either be fixed (can not be removed from graph) or free (can be removed).

implicit edge: Implicit passing of information from a driver node to a variable node. A single variable node can have many incomming implicit edges. These edges are free and can be added or removed from the graph.

[[Insert legend for node and edge types]]

In Alexandrov and Lewis's REMS syntax only two node types were present, variable and function, and only one edge type.[3] We've chosen to rename the "function" node type to "model" to be more consistent with modern MDAO frameworks. This work adds two more node types and one more edge type to the syntax to allow descriptive graphs for all three phases of the design process.

B. Rules for Nodes and Edges

The driver node and the implicit edge are only allowed to be pressent in PSGs. All other node and edge types can be present in any of the three graphs, subject to a few restrictions:

1. A model node can only have one edge directed to or from another model node.
2. A model node can only have fixed connection edges directed in or out.
3. A model node must have at least one edge directed in and at least one edge directed out.
4. If a variable node has an outgoing edge to a model node then it may not have any other outgoing edges.

C. In-degree and Out-degree

The *in-degree* of a node is the number of connection edges directed in and is denoted as $\deg^-(v)$, and the *out-degree* is the number of connection edges directed out and it is denoted as $\deg^+(v)$. The degree of a given node is only a function of the connection edges attached to it. The number of implicit edges is not relevant since any number of drivers could be involved, at different parts of an solution process. For example in a sequential optimization where a global optimizer is first applied and a gradient based one is applied second, design variables would have incomming implicit edges from both optimizers but this would not affect the in-degree of those variable nodes.

We also define the *upper in-degree limit*

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

and the *lower in-degree limit*

$$\deg_l^-(v) : V \to \mathbb{N}. \tag{6}$$

These user-specified limits govern the number of fixed and connection edges that may be directed into a node for a valid graph. Consider a variable node v with $\deg_u^-(v) = \deg_l^-(v) = 1$. In this case, that node must have exactly one incomming edge or the graph is invalid. Two possiblities exist for voilating the limit conditions:

hole: The number of incomming edges is less than $\deg_l^-(v)$.

$$\deg^-(v) < \deg_l^-(v) \tag{7}$$

collision: The number of incomming edges is greater than $\deg_u^-(v)$.

$$\deg^-(v) > \deg_u^-(v) \tag{8}$$

VI. Graph Representation of MDAO Constructs

A. Analysis Blocks and Connections

Analysis tools take in a number input variables, then perform some work to calculate the values for their respective outputs. In an MDAO graph, this process is represented by an group of nodes and edges called an *analysis block*, shown in Fig. 5. Within an analysis block each variable node represents a single input or output and is connected to a single model node via a fixed edge. Note that in Fig. 5, the analysis block contains two model nodes, with a single edge connecting them. This edge represents the necessary calculations to map given inputs to proper output values. If constructing a weighted graph, the weight calculation edge provides the opportunity to computational cost. However, it is also allowed to skip this edge and have both incomming and outgoing edges to variables through a single model node for a given analysis block. Although analysis blocks are comprised of multiple nodes and edges, since all the edges within them are fixed, they become fixed sub-graphs within a graph.

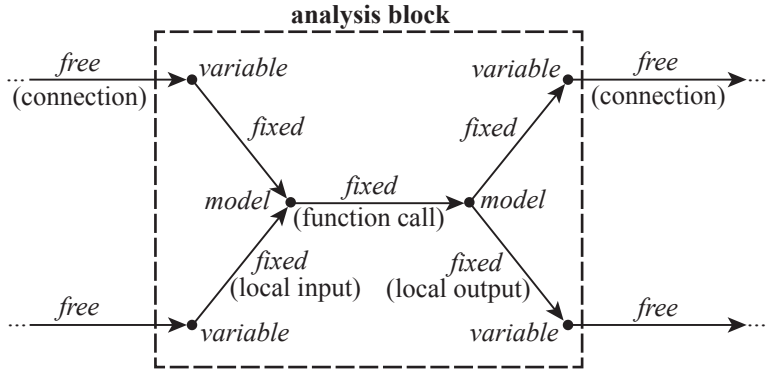


Figure 5: Example analysis block. The each node type and edge type is labeled in italics and annotated parenthetically.

Via the analysis block structure, we can also distinguish between input and ouptut variable nodes. Inputs are any variable nodes that have an outgoing edge into a model node. Conversely, outputs are any variable nodes that have an incoming edge from a model node. Note that by this definition, inputs and outputs can only exist when variable nodes poses an edge joining them to a model node. In Fig. 5 these are labeled out as local inputs and local outputs. The variables are local because they are explicitely tied to the analysis block and have no relevance without it.

MDAO problems require that information be passed between sets of analyses. When information from the output of one analysis block is passed, or connected, to the input of another analysis block, a new connection edge is added connecting the two variable nodes involved in the exchange. These connection edges are free, unlike the edges within an analysis block, they could be added or removed depending on the needs of a given design problem. In other words, free connection edges are created by engineers linking the output of one tool to the input of another one. Note that it is allowed for a single output to have outgoing connection edges to multiple downstream inputs.

B. Design Variables, Objectives, and Constraints

Design variables, objectives, and constraints all require external decisions to be made about a given system to identify them. Consider a notional problem given by the graph in Fig. 6 There are three input variables

which, could potentially become design variables: w, x, and y. Because the output, x, of A is connected to the input, x, of B x can't actually be a design variable. The inputs w and y have no incomming edges and could each be selected as design variables.

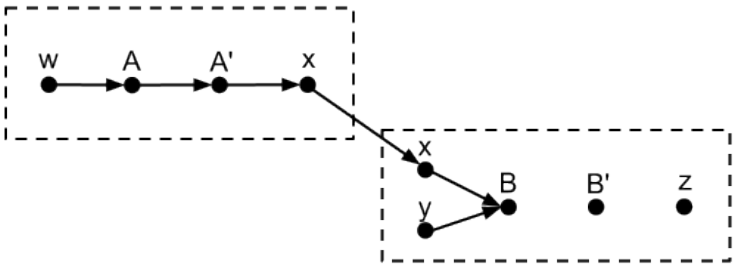


Figure 6: Notional graph with two potential design variables

More formally, any input variable that can be defined as a hole by Eqn. 7 would be considered a design variable in the graph. When working with a MCG or FPG, only connection edges are allowed, in which case a design variable is simply a variable node with no incomming edges. For a valid PSG any design variable node would have at least one, but possibly multiple, incomming implicit edges from one or more driver nodes.

Objective and constraints are also constructs that need to be identified by engineers as part of the design process. In the case of objective functions single output values could be selected, but commonly multiple values are assembeled together via simple athrimetic opperations to form the final objective function. For example, the problem from Fig. 6 might have an objective function of z^2 or $2y + z$. Constraints are always given in the form of either an inequality of an equality, e.g. $\frac{2y}{10} - 1 < 0$ or $3x + 2y - 5 = 0$. In both cases, an engineer identifies variables or groups of variables and assigns a simple relationship between them.

For the proposed syntax, objectives and constraints are represented through the use of expression nodes. These nodes contain the actual mathematical expression used to define the objective or constraint value. Objective or constraint nodes are distinguished via the presence of a simple expression (objective) vs an inequality/equality (constraint). Fig. 7 shows the notional problem graph augmented with objective and constraint values.

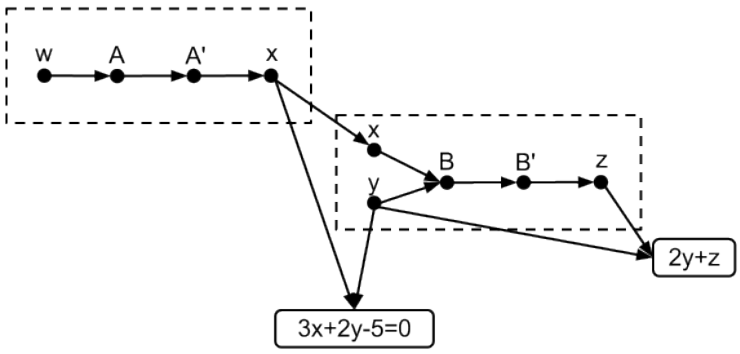


Figure 7: Notional graph with and objective and constraint nodes

C. Local vs Global Nodes

A simple definition of a global node is any node that involves information from more than one analysis block. Likewise, a local node is any node that involves information from only a single analysis block. As stated before, variable nodes with fixed edges directed to model nodes (as part of an analysis block) are inherently local and similarly model nodes are also local. Expression and driver nodes are not inherently local or global though. For these, their locality is determined by the incomming and outgoing edges they possess. Consider the graph in Fig. 8, where the expression node g_1 represents some kind of constraint for the optimziation

problem. Since g_1 only has edges comming from analysis block B, it would be considered a local constraint on block B. g_2 has incomming edges from both A and B analysis blocks, so it would be a global constraint.

[[insert graph here]]

Figure 8: A simple graph with local and global nodes

Although we've already established that variable nodes in an analysis block are local to that block, global variables are a part of many MDAO problems. A global variable would be any variable node with multiple outgoing explicit edges. For example, in Fig. 8 the global node w^* has outgoing edges to the w variable nodes feeding both A and B analysis blocks.

While this simple definition is valid, it glosses over a more subtle aspect of some MDAO problems. Fig. 8, has three analysis blocks A, B, and C. Above we identified node g_2 as global, but it has no incomming edges from block C. So you could say that g_2 is local to the the set of blocks A, B. In this sense, locality becomes a relative term. A given node can be local to a specific sub-graph of a large problem. This subtle structure can become significant when determining how to effectively partition an MDAO problem into sub-problems.[14,15,16]

D. Coupling Between Analyses

Coupling exists in a design problem when a set of two or more analyses each depend on the others outputs. Consider the notional problem from section II where two analyses, A and B are coupled through their respective variables. The graph of this is shown in Fig. 9. In the graph $A.z$ has an outgoing explicit edge to $B.z$, indicating that there is a connection between the two. Similarly a connection exists between $B.y$ and $A.y$. These two connections form a cycle which represent the coupling between them. This cycle means that blocks A and B need to be iterated on untill they converge.

[INSERT GRAPH HERE]

Figure 9: Graph of a notional problem with simple coupling

It is important to note that the cycle in graph in Fig. 9 does not contain any driver nodes. That means that no solver, optimizer, or other iterative construct is invovled in the coupling. This absence is intentional, as the coupling is an inherant part of the problem and would exist regardless of . This corresponds directly To Eqn. 3, where each equality constraint is one edge in the graph. Transitionining to either Eqn. 1 or Eqn. 2 means selecting one of the nodes as the starting point for the iteration, indicating the associated analysis block should be executed first.

E. Multi-fidelity Problems

A multi-fidelity problem can be characterized by two or more different analyses each calculating the same data. One simple example of this might be a two CFD meshes of the same geometry, one course and one refined, each predicting lift and drag of a wing. Another example might involve an empirical weight estimation method and a structural FEM analysis combined with a structural layout model. Mapping this type of multi-fidelity situation onto a graph will yeild two analysis blocks, each with thier own output variable nodes, having outgoing edges to a third analysis block that needs the data as input. Figure 10 shows a notional problem that exhibits this structure, where the key feature is that variable node y in analysis block C has two incomming edges.

[INSERT GRAPH HERE]

Figure 10: Graph of a notional multi-fidelity problem

When an input variable node has more than one incomming edge, designers must make some kind of decision about how to manage the flow of information in their problem. In section C we defined a collision as the situation where there are more incomming explicit edges than the $deg_u^-(y)$. So desginers can indicate

how to handle conflicting edges in a problem graph by setting the $deg_u^-(y)$ for a given node. In more concrete terms, setting this limit equates to defining the part of the graph in question as single or multi-fidelity.

When $deg_u^-(y) = 1$, then any conflicting edges in the graph will cause a collision and a choice between the colliding analysis blocks must be made. This results in a single fidelity design problem. On the other hand if $deg_u^-(y) = 2$, then two different analysis blocks can co-exist with a given graph without causing a collision. The design problem is then a multi-fidelity problem.

Multi-fidelity problems are always charaterized by graphs with variable nodes that have an upper-degree-limit greater than one. These problems require special techniques for resolving the conflicting edges by introducing some mechanism to manage when each of the different fidelity analyses are run.[17,18,19]

VII. Building MCG and FPG

A. 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) are provided. The codes are represented by analysis block graphs ((ABGs?)) $A_i = (V_{A_i}, E_{A_i})$, $i \in I$, where m is the number of codes and $I = \{1, 2, \ldots, m\}$. The global inputs are represented as a set of variable nodes V_{in}, and the global outputs are represented by a set of variable nodes V_{out}. We assume that V_{in}, V_{out}, and each A_i are given, and that any potential connection between variables is given in the form of explicit edges in the set C_M. Then we may construct the maximal connectivity graph $M = (V_M, E_M)$ as

$$V_M = V_{\text{in}} \cup V_{\text{out}} \cup \left(\bigcup_{i \in I} V_{A_i} \right),$$

$$E_M = C_M \cup \left(\bigcup_{i \in I} E_{A_i} \right),$$

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 is not known a priori, the process of obtaining the FPG will reveal the required inputs, as discussed subsequently.

B. 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, which are explained subsequently,

(1) $I_F \subset I$ such that the remaining conditions hold

(2) $\forall i \in I_F$, $\exists v \in V_{A_i}$ with $t_{\text{node}}(E^{-1}(v)) =$ 'model' and $\deg^+(v) > 0$

(3) $V_{\text{in},F} = \{v \in V_{\text{in}} \mid \deg^+(v) > 0\}$ ((maybe this should be 1 instead of 0 to match the previous definition of global input))

(4) $V_F = V_{\text{in},F} \cup V_{\text{out}} \cup \left(\bigcup_{i \in I_F} V_{A_i} \right)$

(5) $E_F = C_F \cup \left(\bigcup_{i \in I_F} E_{A_i} \right)$, $C_F \subset C_M$

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

The set I_F is an index set containing the indices of the analysis blocks in F and it may only include the analysis blocks A_i that meet requirements (2) and (6). Requirement (2) stipulates that each analysis blocks in F must have at least one local output that is being used. Requirement (3) stipulates that only the global inputs that are being used should be included. Requirements (4) and (5) provide the construction of the sets of nodes and edges, respectively. Lastly, requirement (6) stipulates that the number of edges directed

into the variable nodes must be within the lower and upper in-degree limits; if $\deg^-(v) < \deg_l^-(v)$ the node is called a *hole*, and if $\deg^-(v) > \deg_u^-(v)$ the node is called a *collision*.

The set C_F is a set containing explicit edges representing connections. Since C_M is the set of all potential connections, we must have $C_F \subset C_M$. While no other conditions are explicitly stated for C_F, requirement (6) is actually a requirement on both I_F and C_F.

C. Obtaining the Fundamental Problem Graph

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

Then, mathematically, the FPG starts with $C_{F,0} = C_M$, $E_{F,0} = E_M$, $V_{F,0} = V_M$. The nodes where $\deg^-(v) < \deg_l^-(v)$ are considered first.

1. The first step is to detect holes and disconnect the explicit edges following them. These explicit 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

$$H = \{v \in V_M \mid t_{\text{node}}(v) = \text{'variable' and } \deg^-(v) < \deg_l^-(v)\}, \tag{9}$$

   which is the set of variable nodes with fewer incoming edges than are allowed by the lower in-degree limit. Then the updated set of edges is created by removing the first set of explicit edges following the hole:

$$C_{F,1} = C_{F,0} \setminus \{(x,y) \in E_M \mid x = E_M^3(v) \text{ for } v \in H\}, \tag{10}$$

   and this step is demonstrated by Fig. 11. Because removing these edges can create new holes, this step must be repeated until no more holes are found.

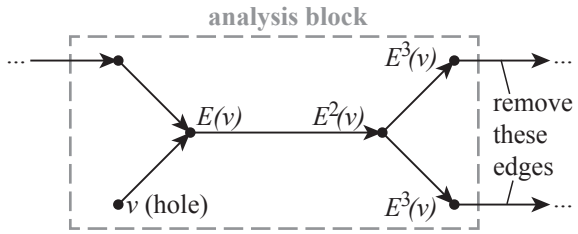


Figure 11: Example variable node indicating a hole.

2. The second step is to detect collisions and to then disconnect enough explicit edges so that the collisions are resolved. The set of variable nodes which contain collisions is created as

$$S_{\text{nodes}} = \{v \in V_M \mid t_{\text{node}}(v) = \text{'variable' and } \deg^-(v) > \deg_u^-(v)\}. \tag{11}$$

   For each collision node we can construct a set contained the edges directed in. The set containing all of these sets is constructed as

$$S_{\text{edges}} = \big\{\{(x,y) \in E_M\} \mid y \in S_{\text{nodes}}\big\} \tag{12}$$

   Let $J = \{1, 2, \ldots, |S_{\text{edges}}|\}$ be an indexing set for S_{edges} such that each $S_{\text{edges},j}$ corresponds to a set in S_{edges} for $j \in J$. An example collision is shown in Fig. 12 to indicate the definition of $S_{\text{edges},j}$. We can assume that J also indexes S_{nodes} because there is a one–to–one correspondence between the elements in S_{nodes} and the elements in S_{edges} (which are sets). Then we may construct sets of edges as

$$B_j = \big\{e_k \in S_{\text{edges},j} \mid k \in \{1, 2, \ldots, K\} \text{ with } K \leq \deg_u^-(v_j)\big\},\ j \in J, \tag{13}$$

   which means that each set B_j is constructed from the set $S_{\text{edges},j}$ by taking only as many edges as are allowed by the upper in–degree limit of v_j. The construction of each B_j corresponds to making a decision on which edges to include and which edges not to include. Then let

$$C_{F,2} = \{e \in C_{F,1} \mid e \in B_j \text{ for some } j \in J\}, \tag{14}$$

$$E_{F,2} = E_M \setminus (C_M \setminus C_{F,2}) \tag{15}$$

and

$$F_2 = (V_M, E_{F,2}) \tag{16}$$

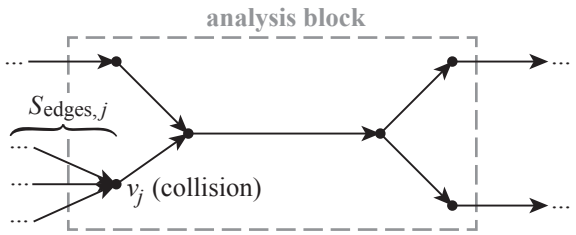


Figure 12: Example variable node indicating a collision.

3. The third and final step is to remove any unused nodes and edges. The indexing set for only the analysis blocks with at least one used local output may be constructed as

$$I_F = \{i \in I \mid \exists v \in V_{A_i} \text{ such that } t_{\text{node}}(E_{F,2}^{-1}(v)) = \text{'model' and } \deg^+(v) > 0\}, \tag{17}$$

where the degree is calculated with respect to F_2. This set excludes both the analysis blocks that were unused in the original MCG and those that became unused in step 1 or step 2.

The set of used global inputs is constructed as

$$V_{\text{in},F} = \left\{ v \in V_{\text{in}} \;\middle|\; |\{(x,y) \in E_{F,2}(v) \mid y \in V_{A_i} \text{ for } i \in I_F\}| > 1 \right\}, \tag{18}$$

which requires that at least two edges directed out of the nodes in $V_{\text{in},F}$ must be to the analysis blocks indexed by I_F.

Finally, the set of nodes and edges describing a fundamental problem formulation is given by

$$V_F = V_{\text{in},F} \cup V_{\text{out}} \cup \left(\bigcup_{i \in I_F} V_{A_i} \right), \tag{19}$$

$$C_F = \{(x,y) \in C_{F,2} \mid x, y \in V_F\}, \tag{20}$$

$$E_F = C_F \cup \left(\bigcup_{i \in I_F} E_{A_i} \right), \tag{21}$$

$$F = (V_F, E_F). \tag{22}$$

VIII. Example Problem

This section presents an example problem to demonstrate the process of obtaining the FPG and the advantages of doing so. The task is the conceptual sizing of a single–aisle subsonic transport using an MDAO framework with objectives for performance and total weight for a required mission. The full set of analysis codes available for use is given in Table 1. Each of these analysis codes contribute individual disciplinary analysis but they are not mutually exclusive. For example, VORLAX and PMARC are both aerodynamics codes that predict inviscid drag but with different levels of fidelity. The Flight Optimization System (FLOPS) is included twice to represent two different uses. FLOPSa denotes FLOPS being executed for both mission performance and engine analysis, while FLOPSb indicates that FLOPS is executed for only mission performance analysis.

In this example, the MCG is formed using a consistant naming convention and then connecting every variables with the same name, though this is not the only way to form an MCG. Table 2 presents the full list of variables in the leftmost column and then indicates whether the variable is an input or an output of each analysis code. Some of the variables, such as geometry and performance, represent many different variables

Table 1: Analysis codes for subsonic transport sizing

analysis code	description
VSP	parametric geometry
PDCYL	wing and fuselage weight estimation
NPSS	engine sizing and performance
VORLAX	aerodynamics using the vortex lattice method
PMARC	aerodynamics using a low-order panel method
WATE	engine weight estimation
FLOPSa	mission performance and engine sizing and weight estimation
FLOPSb	mission performance only

Table 2: Analysis code input and output description

variable	analysis code							
	VSP	PDCYL	NPSS	VORLAX	PMARC	WATE	FLOPSa	FLOPSb
geometry	in	in		in	in	in	in	in
number of engines			in				in	in
mission							in	in
fuselage weight		out					in	in
wing weight		out					in	in
engine weight						out	out	in
wetted area	out						in	in
inviscid drag				out	out			in
drag			in				out	out
engine performance			out			in	out	in
performance							out	out
total weight		in					out	out

which have been bundled together due to there similarity. This bundling is not necessary and does not limit the generality of this example, it is done to simplify the presentation.

The MCG M is formed in four steps:

1. An analysis block is created for each analysis code using the information in each column of Table 2. Each analysis block is formed by creating variable nodes for each input and directing them into a single model node. This model node is then directed into another model node, which is then directed into a variable node for each output. A sample analysis block is shown for analysis code FLOPSb in Fig. 13.

2. Variable nodes are created to represent the objectives for performance and total weight.

3. Variable nodes are created to represent the geometry variable and mission variable global inputs.

4. Explicit edges are created from each variable node to every other variable node representing variables with the same name.

The resulting MCG is shown in Fig. 16. Emerging cycles are shown in yellow and indicate potential cycles that may exist when an FPG is formed.

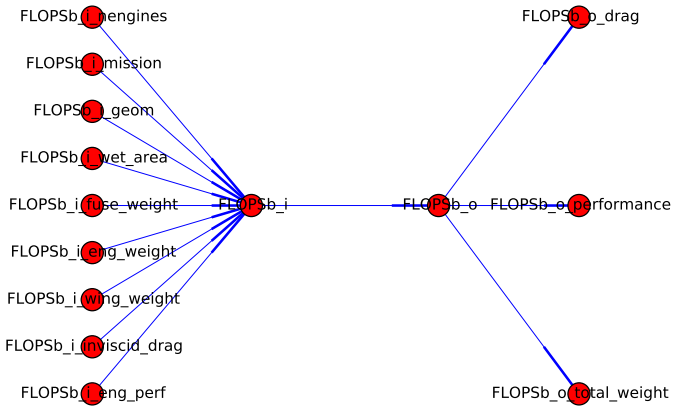


Figure 13: Sample analysis block for analysis code FLOPSb using Table 2

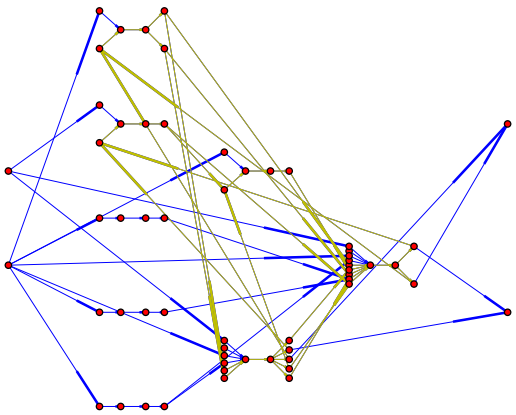


Figure 14: Maximal connectivity graph for the subsonic transport example problem

A. Obtaining an FPG

As presented in Sec.(()), the first step to obtaining an FPG is to detect holes and, to start, we set the lower indegree limit for every node to be one. In this case, it turns out that the variable nodes representing the number of engines is a hole for NPSS, FLOPSa, and FLOPSb. This implies than an FPG cannot be obtained from the given MCG. Therefore, we may now create another variable node to represent the number of engines as a global input and also create explicit edges to supply this new input. Now this step is repeated with the new MCG and no holes are found.

While Sec.(()) indicated the process for obtaining an FPG, it did not specify how to decide which edges to keep when resolving a collision, that freedom is left to the specific implementation. The benefit of this approach is that standard graph theory algorithms can be used to make these decisions. The first example of this is to obtain an FPG with the fewest possible number of cycles, which are found using the implementation

of Johnson's algorithm[?] in the Python package NetworkX. The resulting FPG is shown in Fig. 15 and the only cycle is shown in yellow. A similar alternative would be to minimize the number of analysis blocks involved in cycles, counting multiplicity. While resulting graph has only one cycle, it may be more desirable to include the higher fidelity codes, such as NPSS.

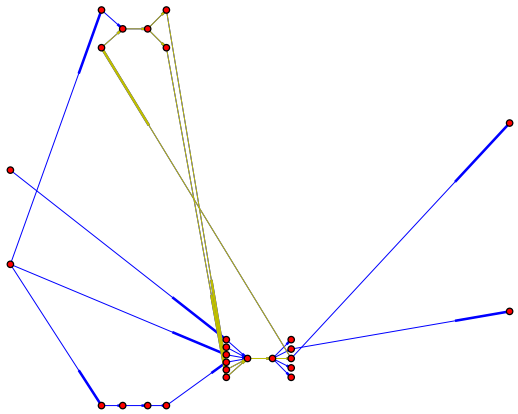


Figure 15: FPG with the fewest number of cycles.

One method to ensure that the edges directed from certain analysis blocks are the ones chosen to resolve the conflicts is to use a ranking system. For this case, consider the rankings for each analysis code given in Table 3. Step two in Sec.(()) (the resolution of collisions) is now conducted by selecting the edges directed

Table 3: Example ranking of importance

analysis code	rank
VSP	5
PDCYL	5
NPSS	4
PMARC	4
FLOPSb	4
VORLAX	3
WATE	3
FLOPSa	2

from the analysis blocks with the highest rank for each collision. The resulting FPG is shown in Fig. 16 and this graph has four cycles, which are enumerated in Table 4.

Finally, now consider the case where it is desired to have the inviscid drag input into FLOPSb be a multi–fidelity input. This is done simply by setting the upper indegree limit for that node to two and repreating the (automated) process. In this case, both VORLAX and PMARC are retained, resulting in an FPG that omits only FLOPSa.

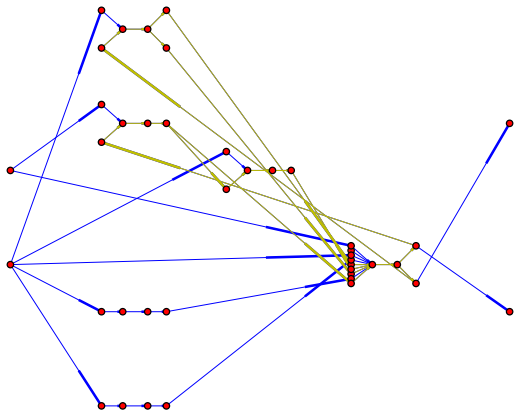


Figure 16: Simple notional problem with two potential design variables

Table 4: Cycles for the FPG obtained using ranking

output/input	analysis code	output/input	analysis code	output/input	analysis code	output/input
engine weight	FLOPSb	drag	NPSS	engine performance	WATE	engine weight
engine performance	FLOPSb	drag	NPSS	engine performance		
fuselage weight	FLOPSb	total weight	PDCYL	fuselage weight		
wing weight	FLOBSb	total weight	PDCYL	wing weight		

IX. Applications

X. Conclusions

References

[1]Shan, S. and Wang, G., "Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions," *Structural and Multidisciplinary Optimization*, Vol. 41, No. 2, 2010, pp. 219–241.

[2]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.

[3]Alexandrov, N. and Lewis, R., "Reconfigurability in MDO Problem Synthesis, Part 1 and Part 2," Tech. rep., Papers AIAA-2004-4307 and AIAA-2004-4308, 2004.

[4]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.

[5]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.

[6]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.

[7]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.

[8]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.

[9]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.

[10]Martins, J. R. R. A. and Lambe, A. B., "Multidisciplinary Design Optimization: Survey of Architectures," *AIAA Journal*, 2012, pp. 1–46.

[11]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.

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

[13]Smith, D., Eggen, M., and Andre, R. S., *A Transition to Advanced Mathematics*, Brooks/Cole Pub Co, 2006.

[14]Krishnamachari, R. S. and Papalambros, P. Y., "Optimal hierarchical decomposition synthesis using integer programming," *J MECH DES, TRANS ASME*, Vol. 119, No. 4, 1997, pp. 440–447.

[15]Allison, J. T., Kokkolaras, M., and Papalambros, P. Y., "Optimal partitioning and coordination decisions in decomposition-based design optimization," *Journal of Mechanical Design*, Vol. 131, 2009, pp. 081008.

[16]Michelena, N. F. and Papalambros, P. Y., "A hypergraph framework for optimal model-based decomposition of design problems," *Computational Optimization and Applications*, Vol. 8, No. 2, 1997, pp. 173–196.

[17]March, A. and Willcox, K., "Provably Convergent Multifidelity Optimization Algorithm Not Requiring High-Fidelity Derivatives," *AIAA journal*, Vol. 50, No. 5, 2012, pp. 1079–1089.

[18]Alexandrov, N. M., Lewis, R. M., Gumbert, C. R., Green, L. L., and Newman, P. A., "Approximation and model management in aerodynamic optimization with variable-fidelity models," *Journal of Aircraft*, Vol. 38, No. 6, 2001, pp. 1093–1101.

[19]Huang, D., Allen, T. T., Notz, W. I., and Miller, R. A., "Sequential kriging optimization using multiple-fidelity evaluations," *Structural and Multidisciplinary Optimization*, Vol. 32, No. 5, 2006, pp. 369–382.