Structrual and Multidisciplinary Optimization

A Graph Theoretic Approach to Problem Formulation for Multidisciplinary Design Optimization

David J. Pate, Justin Gray, Brian J. German

Abstract The formulation of multidisciplinary design, analysis, and optimization (MDAO) problems has become increasingly complex as the number of analysis tools and design variables included in typical studies has grown. This growth in the scale and scope of MDAO problems has been motivated by the need to incorporate additional design disciplines and to expand the parametric design space to enable the exploration of unconventional design concepts. In this context, given a large set of disciplinary analysis tools, the problem of determining a feasible data flow between tools to produce a specified set of system-level outputs is combinatorically challenging. The difficulty is compounded in multi-fidelity problems, which are of increasing interest to the MDAO community. In this paper, we propose an approach for addressing this problem based on the formalism of graph theory. The approach begins by constructing the maximal connectivity graph (MCG) describing all possible interconnections between a set of analysis tools. Graph operations are then conducted to reduce the MCG to a fundamental problem graph (FPG) that describes the connectivity of analysis tools needed to solve a specified system-level design problem. The FPG does not predispose a particular solution procedure; any relevant MDO solution architecture could be selected to implement the optimization. The approach is applied to an example problem to formulate an FPG for a commercial aircraft MDAO study.

Nomenclature

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

Justin Gray
NASA Glenn Research Center, Mail Stop 5-11, 21000 Brookpark Rd Clevland OH 44135

David J. Pate
Graduate Research Assistant, Georgia Institute of Technology, 270 Ferst Drive, Atlanta, GA, 30332, U.S.A.

Brian German
Assistant Professor, Georgia Institute of Technology, 270 Ferst Drive, Atlanta, GA, 30332, U.S.A.

1 Introduction

The number of analysis tools required in multidisciplinary design optimization (MDO) studies is growing in parallel with the increasing scope of typical problems. An example 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 [1]. Subsequently, flight performance and life cycle economics tools were integrated into MDO analysis workflows for conceptual and preliminary design studies [2]. 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.

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 [3], subsystem performance [4,5], and models of emissions, noise, or economics [6,7,8].

The time and expense for setting up analysis models is growing in conjunction with the increasing scope of MDO problems. Multidisciplinary Design Analysis and Optimization (MDAO) frameworks such as OpenMDAO[9] and ModelCenter®have enabled a new level of analysis tool integration and have paved the way for models with more analyses and increasing numbers of interdisciplinary couplings. This new capability has created a new challenge. Even models with tens of analyses could include hundreds or thousands of variables that are interdependent and must be linked in the framework. A common situation is that different analyses provide differing values for the same physical quantity, and these conflicts need to be resolved. These differing 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 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 the added complexity from larger models, it is plausible that the task of combining all the analyses into a consistent system model capable of solving a relevant engineering design problem could approach the cost and time requirements of creating any of the discipline analyses themselves. For these large scale MDO problems, the couplings between the analyses begin to dominate the effort required in 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 realistic 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 coupled 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[10]. The approach taken here should be viewed as an extention of the concepts proposed in REMS with the goal of allowing the graphs to more closely interact with existing design processes and MDAO frameworks.

The approach begins by constructing the *maximal connectivity graph (MCG)* describing all possible interconnections between the analysis tools proposed by the engineers. Graph operations are then conducted to reduce the MCG down to a *fundamental problem graph (FPG)* that describes the set of analysis tools needed to solve the specified system-level design problem. 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.

The paper is organized as follows. First, we describe the differences between a fundamental problem formulation, which is based only on the system-level optimization problem statement that the engineers desire to solve and on the available analysis tools, and a specific problem formulation, which additionally presumes a specific MDO solution approach 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 the procedures for determining the MCG and FPG. Finally, we present an example problem based on an MDO analysis of a commercial aircraft and discuss additional applications enabled by our approach.

2 Motivation

2.1 Specific vs Fundamental Problem Formulation

The Fundamental Problem Formulation (FPF) is our terminology for a statement of the overall system-level optimization problem that does not depend on the choice of

solution-specifc elements such as the MDAO framework, optimization algorithm, iterative solver, or execution sequence used to solve the problem. The FPF contrasts with problem statements that are written without reference to a specific solution strategy, but that still imply a specific execution sequence. For example, consider the Sellar test problem [11], with an FPF given as follows:

$$\begin{aligned} \text{given} \quad & y_1 = D_1(x_1, y_2, z_1, z_2) \\ & y_2 = D_2(y_1, z_1, z_2) \\ \text{min.} \quad & F(x_1, y_1, y_2, z_2) \\ \text{w.r.t.} \quad & x_1, y_1, y_2, z_1, z_2 \\ \text{s.t.} \quad & G_1(y_1) \geq 0 \\ & G_2(y_2) \geq 0 \end{aligned} \tag{1}$$

In Eq. 1, y_1 is an output of D_1 as well as an input to D_2. Similarly, y_2 is an output of D_2 as well as an input to D_1. This implies that D_1 and D_2 are coupled together by means of their reciprical dependence. However no information is given as to weather D_1 or D_2 should be run before the other. Given Eq 1, it would be equally valid to run D_1 first, D_2 first, or both simultaneously.

Eq. 2 gives a modified problem formulation where D_1 is run first. A new variable is introduced, $\hat{y_2}$, to break the dependence of D_1 on D_2. Also a new coupling constraint, H, to enforce the consistency that would otherwise be lost due to $\hat{y_2}$. Equation 2 is a specific alternative expression of the more fundamental problem description given in Eq. 1.

$$\begin{aligned} \text{given} \quad & y_1 = D_1(x_1, \hat{y_2}, z_1, z_2) \\ & y_2 = D_2(y_1, z_1, z_2) \\ \text{min.} \quad & F(x_1, y_1, y_2, z_2) \\ \text{w.r.t.} \quad & x_1, y_1, \hat{y_2}, z_1, z_2 \\ \text{s.t.} \quad & H(y_2, \hat{y_2}) = 0 \\ & G_1(y_1) \geq 0 \\ & G_2(y_2) \geq 0 \end{aligned} \tag{2}$$

2.2 Existing Graph-Based Syntax

As indicated in the examples in Sec. 2.1, 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 this 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[12]. Their work demonstrates how multiple specific problem formulations can relate to a common FPG. The challenge with 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. Off-diagonal elements of the matrix indicate coupling[13]. Since

a DSM describes a square adjacency matrix, it can be represented in an equivalent directed graph in which 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 challenging 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[14, 15]. This re-ordering is done through row operations on the DSM matrix and yields multiple specific problem formulations which all solve the same FPF.

A DSM by itself is insufficient to describe complete problem formulations because it captures only information about data dependency between analyses; objective and constraint information is missing from the DSM 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[16]. Similar to DSM, FDT also describes an adjacency matrix of a graph. Unlike the DSM graph, however, the FDT 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[17]. While FDT succeeds at capturing the information about objectives and constraints, its lack of directed edges implies that it cannot capture the coupled data dependency that a DSM captures. The FDT in Fig. 1 shows that D_1 is dependent on y_2 and that D_2 is dependent on y_1, but you can not infer their coupled dependence from that information alone. This missing information implies that, while FDT is very useful for partitioning problems, it is also not sufficient to completely describe a problem formulation.

	x_1	y_1	y_2	z_1	z_2
D_1	1		1	1	1
D_2		1		1	1
F	1	1	1		1
G_1		1			
G_2			1		

Fig. 1: Functional Dependency Table (FDT) for Eq. 1

Alexandrov and Lewis introduced a graph based syntax called REMS which incorporates objectives and constraints into a graph, effectively combining FDT and DSM[10]. REMS retains the square adjacency matrix form from DSM, but by adding the objectives and constraints, it partially combines a traditional DSM with an FDT. This formulation allows REMS to represent data dependency between multiple analysis tools as well as between analysis tools and objective/constraint functions. 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 provides syntax for variables and functions but does not facilitate 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[18]. Unlike REMS, XDSM also includes nodes for

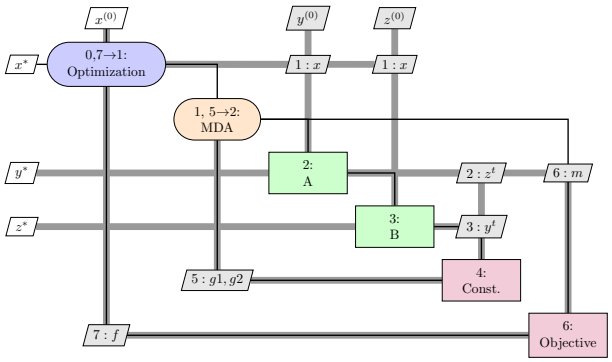


Fig. 2: XDSM for Eq. 2, with a Gauss-Seidel iteration and MDF solution architecture.

solvers and optimziers to enable complete definition of MDAO architectures. Martins uses XDSM to describe 13 different optimization architectures in a survey paper that provides a novel classification of the different techniques [19]. With the additional information included in an XDSM, Lu and Martins applied both ordering and partitioning algorithms on an MDAO test problem termed the Scalable Problem [20].

Although XDSM captures many of the functional aspects of FDT, it 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 implies a strategy. The XDSM for Eq. 2 is given in Figure 2. This diagram is shown with an assumed Gauss-Seidel iteration scheme and an MDF solution architecture.

Of these methods, REMS can describe most aspects of a fundamental problem formulation, but lacks the ability to represent optimizers and solvers in more specific problem formulations. XDSM relies on the provision of optimizers and constraints and is not formulated for a more fundamental problem statement. We propose a new graph syntax that combines the relevant features of REMS and XDSM while retaining the flexiblity to represent a fundamental problem formulation free of solution specific information.

2.3 Requirements for a New Graph Syntax

The goal of the new graph-based syntax presented here is to enable the general structure of an MDAO problem to be described independently of any solution information, while still being able to accomodate the more specific case when a solution strategy is applied. In order to achieve this goal, the graph syntax needs to accomodate a number of MDAO problem constructs:

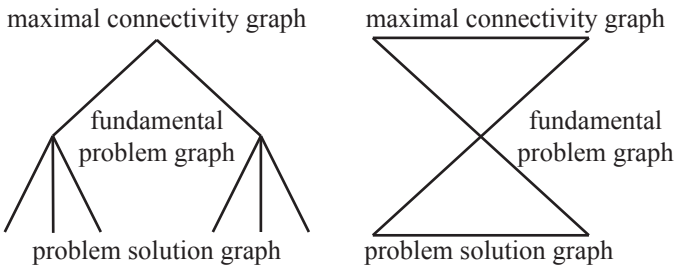
- Analysis tools and their interconnections
- Design variables, objectives, and constraints
- Local and global information
- Coupling between analyses
- Multi-fidelity analyses

The new syntax is intended to represent three phases of the design problem formulation process. In the initial problem definition phase, the specific analysis tools and design goals are identified. Next, a single formal problem formulation is identified that specifies design variables, constraints, objectives, analysis tools, and all other elements required to represent the overall MDAO problem. Lastly, a specific procedure for solving the problem is selected, e.g. selecting an MDAO optimization architecture. Using the proposed graph syntax, these phases can be represented with the following three graphs:

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

The *maximal connectivity graph* represents the first phase of the problem formulation with all analysis tools being considered and all possible connections between them also present. The second graph is the *fundamental problem graph*, which is comprises only the analyses participating to solve the problem. Finally, a *problem solution graph* may be represented by including additional edges and nodes to represent the solution strategy being employed to solve the problem. This paper focuses on the MCG and FPG and does not describe the PSG in detail.

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. The multiple FPGs may correspond to different down-selections of analysis tools, different connections between the tools, or both. Each down-selection shrinks the number of possible FPGs that could be reached until only one remainins. For a given FPG, different PSGs may be obtained by implementing different solution strategies. Considering the size of a graph to be the sum of all of its edges and nodes, the hourglass shape in Fig. 3(b) qualitatively illustrates how 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

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

3 Graph-Based Syntax Definition and Formalisms

In this section we present our graph-based syntax, beginning with the necessary graph theoretic fundamentals to construct the graphs discussed in Sec. 2.3. The notation used in this paper is adapted from Diestel [21]. 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* (or *digraph*) 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. An edge $e = (x, y)$ may be referred to simply as xy. The edges directed out from node v are denoted by $E(v)$ and the edges directed into v are given by $E^{-1}(v)$. If E is not a one-to-one mapping, $E(v)$ may be the empty set, a single node, or a set of nodes. As an example, for the directed

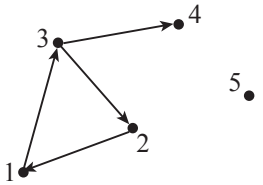


Fig. 4: Example directed graph.

graph shown in Fig. 4 we have

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

A *path* $P = (V, E)$ from x_0 to x_k in graph G is a subgraph of G with $V = \{x_0, x_1, \ldots, x_k\}$ and $E = \{(x_0, x_1), (x_1, x_2), \ldots, (x_{k-1}, x_k)\}$. Path P is a *cycle* if $x_0 = x_k$. A *reverse path* P_R in graph G is a path on R, where R is the reverse graph of G obtained by switching the orientation of every edge.

Let I be a nonempty set 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[22]. The union over this family of sets can be described in a few different ways:

$$\bigcup_{i \in I} A_i = \bigcup_{A \in \mathcal{A}} A = \{x \mid x \in A \text{ for some } A \in \mathcal{A}\}. \tag{3}$$

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

3.1 Node and Edge Types

Along with these fundamentals of graph theory, we categorize nodes and edges into distinct types in order to provide an intuitive approach to MDAO problem formulation. The three node types are

variable node: represents scalar or array data, inputs and outputs,

function node: represents the computation peformed by analysis tools,
driver node: control logic capable of managing iterations (PSG),
and the three edge types are
connection edge: exchange of information external to analysis tools,
function edge: exchange of information internal to analysis tools,
driven edge: passing of information from a driver node to a variable node (PSG).
Figure 5 demonstrates the usage of these node and edge types via the Sellar problem. function nodes are indicated by squares, variable nodes are indicated by circles, connection edges are indicated by dotted lines, and function edges are indicated by solid lines.

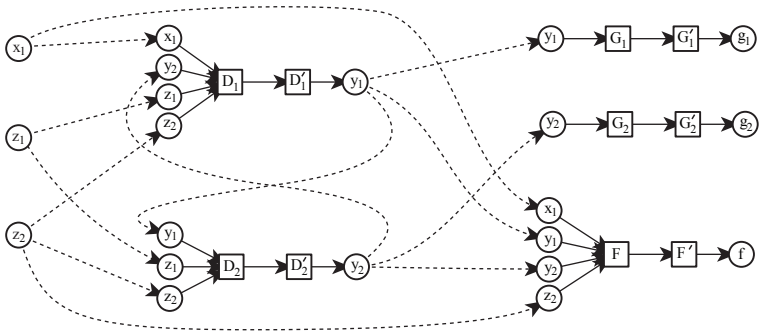


Fig. 5: Sellar problem represented as a graph.

A rule set is provided for the usage of these node and edge types to provide a structure to the graph-based representation. The driver node and the driven edge are allowed to be present only in PSGs, whereas the other node and edge types can be present in any of the three graph types, subject to the following restrictions:

1. A function node may have only one edge directed to or from another function node.
2. A function node may have only function edges directed in or out.
3. A function node must have at least one edge directed in and at least one edge directed out.
4. If a variable node has an outgoing function edge, then it may not have other outgoing edges.
5. If a variable node has an incoming function edge, then it may not have any additional incoming edges.

In Alexandrov and Lewis's REMS syntax only two node types (variable and function) and one edge type are included [10]. The present work adds one additional node type and two additional edge types to the syntax to allow descriptive graphs for all three phases of the design problem formulation process indicated in Figs. 3(a) and 3(b).

3.2 Analysis Blocks

Analysis tools take in a set of input variables and then calculate the values for their respective outputs. We represent this process by a digraph called an *analysis block*; a

notional analysis block is shown in Fig. 6. As indicated in this figure, analysis blocks comprise three sets of nodes representing the distinct local inputs, the analysis (computation), and the distinct local outputs. The local input and local outputs nodes are variable nodes, while the nodes representing the analysis are function nodes. All of the edges within the analysis block are function edges, and are considered fixed to the analysis block. This graph structure demonstrated by Fig. 6 satisfies the rules listed in Sec. 3.1. Reciprocally, the rules ensure that analysis tools are represented via this analysis block structure.

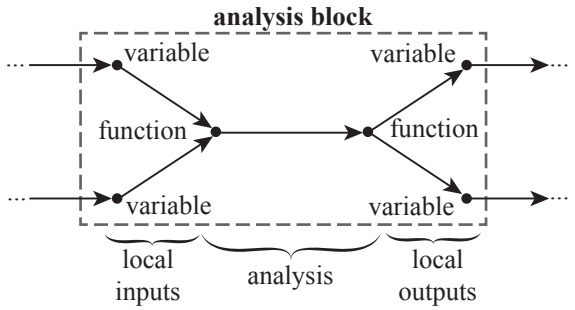


Fig. 6: Example analysis block with node types labeled.

The function edge connecting the two function nodes represents the necessary calculations to map the inputs to the outputs of the analysis. This edge also provides the opportunity to encode computational cost or other characteristics of the analysis code as a weight in a weighted graph formulation. Since all the edges within analysis blocks are fixed, the blocks themselves are fixed sub-graphs within the overall MDAO problem graph. The connectivity of nodes and edges in an analysis block cannot be altered for the purposes of MDAO problem formulation; however, analysis block sub-graphs can be added or removed from the MDAO problem graph as needed. Variable nodes in an analysis block can be distinguished as either an input or an output by the manner in which they are connected. As shown in Fig. 6, inputs are represented as variable nodes that have an outgoing edge into a function node, and outputs are represented as variable nodes that have an incoming edge from a function node.

3.3 Objectives, Constraints, and Inputs

Along with analysis tools, objectives, constraints, and inputs also need to be represented. In the case of objective functions, a single output value generated by an analysis block could be selected, but commonly multiple values are assembeled together to form a composite objective function. Generally, both objective and constraint functions accept a set of inputs and map them to an output value of significance to the overall design problem.

The operations of implementing composite objective and constraint functions, although typically simple, are effectively identical to the task performed by an analysis block. A composite constraint or objective function can therefore be represented as an analysis block within the graph with its own input and output variable nodes. Although

fundamentally no different than an analysis block, for clarity and convience, it is useful to distinguish between analysis blocks corresponding to analysis codes and those that arise from the addition of objectives or constraints to the graph. Therefore, we use an *expression block* to represent an objective or constraint. These expression block graphs follow the same structure as analysis blocks presented in Sec. 3.2.

Finally, inputs to the problem formulation are represented in the graph by individual variable nodes with connection edges directed out to corresponding local inputs of analysis blocks, and they are referred to simply as *inputs*. These nodes serve to indicate that the variable nodes they are connected to represent the same variable in the problem formulation. These nodes serve to unify the variable nodes they are connected to. That is, they indicate that these nodes refer to the same input variable.

Figure 7 demonstrates the use of inputs (black circles), analysis blocks (dotted boxes), and expression blocks (dashed boxes), for the Sellar problem graph.

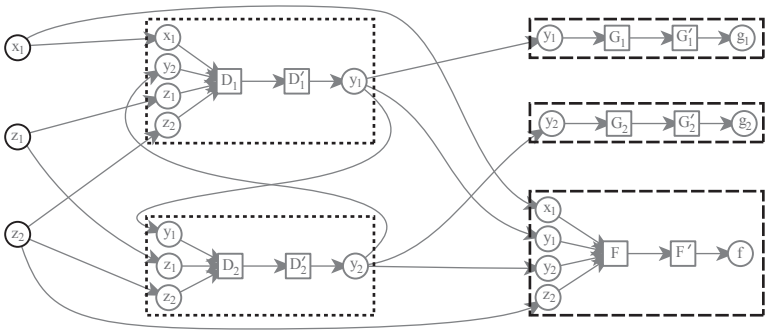


Fig. 7: Sellar problem graph indicating inputs (black circles), analysis blocks (dotted boxes), and expression blocks (dashed boxes).

3.4 Indegree and Outdegree Limits

To address the MDAO notion of design variables, we first introduce the concept of the degree of a node. In a digraph, *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 denoted as $\deg^+(v)$ [21]. In this paper, the indegree and outdegree of a given node refer only to the number of the connection edges directed into or out of the node, respectively.

We may now define the *upper indegree limit*

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

and the *lower indegree limit* as

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

These user-specified limits govern the number of connection edges that may be directed into a node for a valid graph; of course, $\deg_l^-(v) \leq \deg_u^-(v)$ must be satisfied. For

example, consider a variable node v with $\deg_u^-(v) = \deg_l^-(v) = 1$. In this case, v must have exactly one incoming connection edge or the graph is deemed an invalid problem formulation.

4 MDAO Constructs Derived from the Graph-Based Syntax

In Sec. 3 we presented the structure of the graph-based representation. We now discuss how these formalisms provide the remaining MDAO problem constructs identified in Sec. 2.3.

4.1 Design Variables, Holes, and Collisions

Design variables are input variables whose values are free to be changed by the designer or by an optimizer in an MDAO study. In the graph-based syntax, a node corresponds to a design variable when the indegree limit of the node is prescribed as zero.

If the indegree limits are violated by the number of incoming connection edges (the indegree of the node), the graph is regarded as an invalid problem formulation. There are two classifications for these violations:

hole: The number of incoming edges is less than the lower indegree limit:

$$\deg^-(v) < \deg_l^-(v). \tag{6}$$

A hole represents a lack of information being supplied to the variable node. This implies that the analysis tool being represented by the analysis block would not be capable of producing outputs.

collision: The number of incoming edges is greater than the upper indegree limit.

$$\deg^-(v) > \deg_u^-(v). \tag{7}$$

A collision represents redundant information being supplied to the variable node from two or more sources. This conflict implies an ambiguity as to which information is to be used as an input for the analysis tool.

The presence of holes and collisions in a graph represent conflicts that will give rise to an invalid problem formulation. It is expected that a maximal connectivity graph could indicate these conflicts, whereas a fundamental problem graph will not.

4.2 Coupling Between Analyses

In MDAO, coupling is the mutual dependence the outputs of two or more analysis codes. In this graph-based syntax, coupling is described by the presence of a cycle between two or more analysis blocks. In the Sellar problem from Eqn 1 a reciprical dependence between D_1 and D_2 exists through the variabled y_1 and y_2. The edges belonging to a cycle in the Sellar problem are highlighted in Fig. 8 where they form a closed path between analysis blocks D_1 and D_2.

Coupling cycles do not contain driver nodes in the MCG or FPG representation. Solvers, optimizers, and other iterative processes are not invovled in the coupling definition; however, one or more solvers will be necessary to build a valid PSG from a

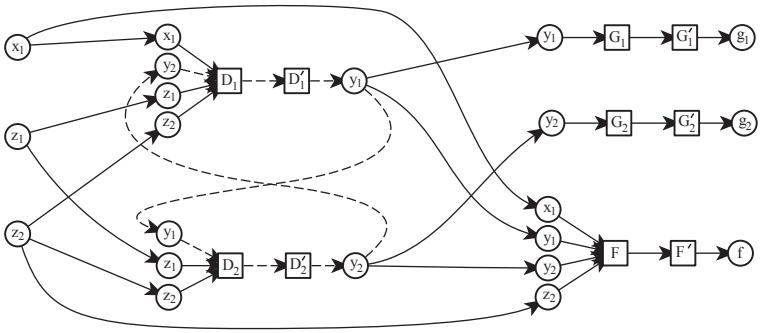


Fig. 8: Sellar problem graph with dashed edges indicating participation in a cycle.

given FPG. Additionally, a coupling cycle has no inherent start or end point. It would be acceptable to select node in the cycle as a starting point and proceed around the loop until the starting point is reached again. For the Sellar Problem, selecting D_1 as the starting point would yield a problem as given in Eq. 2. Larger problems can contain more complex cycles in their FPG, indicating more complex coupling between analyses. For example, a cycle can involve more than two analysis blocks. Multiple independent cycles could also exist, indicating independent coupling relationships. Cycles can also overlap, meaning that the same analysis blocks are involved in multiple different coupling cycles. All of these situations arise naturally as the size of problems grows, and managing this coupling may become difficult. In the present work, Sec. 6 demonstrates how building an FPG from an MCG provides an opportunity to identify and potentailly reduce the number of cycles in a graph.

If many couplings are present, convergence rates can be improved by searching for an effective ordering for the execution of analyses. Rogers' DEMAID tool uses a genetic algorithm to find an ordering that minimizes the overall coupling of the system by separating independent cycles in the graph [14, 15]. Rogers work focused on the matrix form of the DSM for ordering optimization. Lu and Martins more recently leveraged a weighted form of the DSM and used an iterative clustering approach to perform a similar task to DEMAID [20].

4.3 Multi-fidelity Problems

A multi-fidelity MDAO problem can be characterized by a formulation in which two or more different analyses each calculating the same data. Multi-fidelity is represented in a graph by a variable node having an indegree greater than one, which means that multiple connection edges are directed into it. When the upper indegree limit of a variable node is set above one, then the node is implied to allow multiple fidelity instances of the variable. When the lower indegree limit of a variable node is set above one, then multiple fidelity instances are required.

In a multi-fidelity problem, a given variable node may have multiple incoming edges without implying a collision as defined by by Eq. 7. An upper indegree limit above unity for any variable in a graph represents a decision to allow multiple fidelities to interact

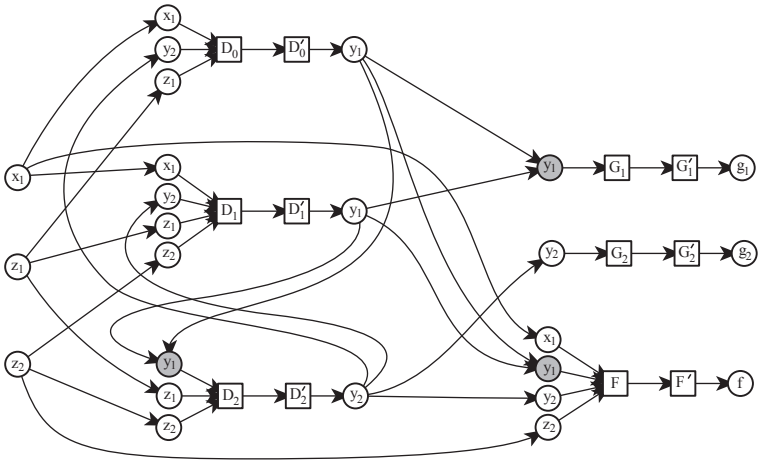


Fig. 9: Graph of the modified Sellar problem with multi-fidelity nodes highlighed in gray.

at that part of the graph in order to allieviate a conflict. Figure 9 shows a modified version of the Sellar problem with a new analysis, D_0, representing a low fidelity version of D_1. In this modified version, the variable nodes with an upper indegree limit of two are highlighted in gray, and changing these upper degree limits for the variable nodes representing y_1 in D_2, F, and G_1 avoids a collision. In section 5.3 we present an algorithm for finding conflicts within a given MCG so that a designer can make the necessary decisisons about each one in turn.

Multi-fidelity problems are always charaterized by graphs with variable nodes that have an $\deg_u^-(v) > 1$. 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[23, 24, 25]. The specifics of this mechanism are not given in any form withing an MCG or FPG. Instead the multi-fidelity mechanism specifics would be represented as a driver in a PSG derived from a multi-fidelity FPG.

Table 1 provides examples to summarize the classification of a variable node as a hole, design variable, single valid input (nominal), collision, or multi-fidelity.

Table 1: Variable Node Classification

$\deg_l^-(v)$	$\deg_u^-(v)$	$\deg^-(v)$	valid	classification
0	0	0	yes	design variable
0	0	1	yes	collision
0	1	0	yes	design variable
0	1	1	yes	single input
1	1	0	no	hole
1	1	1	yes	single input
1	1	> 1	no	collision
1	2	1	yes	supplied input
1	2	2	yes	multi-fidelity
2	2	< 2	no	hole
2	2	2	yes	multi-fidelity
2	2	> 2	no	collision

5 The MCG and FPG

We now discuss how the graph-based syntax defined in Sec. 3 is used to represent an MDAO problem in its entirety. This discussion is based on the MCG and the FPG.

5.1 Maximal Connectivity Graph

For a specified MDAO problem, the maximal connectivity graph represents every analysis tool, objective, constraint, and input being considered, as well as every possible interconnection among them. The definition of the MCG is given via construction. To construct the maximal connectivity graph, we presume that a set of analysis tools, inputs, objectives, and constraints are provided. Each of the m analysis codes correspond to an index $i \in I_A$, $I_A = \{1, 2, \ldots, m\}$, and are represented by an analysis block graph $G_{A_i} = (V_{A_i}, E_{A_i})$. Each of the n expression blocks correspond to an index $i \in I_E$, $I_E = \{1, 2, \ldots, n\}$, and are represented by an expression block graph $G_{E_i} = (V_{E_i}, E_{E_i})$. Finally, the inputs are represented as a set of variable nodes V_{in}. We presume that V_{in}, each $G_{A,i}$, and each $G_{E,i}$ are given, and that any potential connection between variables is specified in the form of connection edges in the set $E_{M,\text{con}}$. One method for defining these connections is to use a consistant variable naming convention. We may then construct the maximal connectivity graph $M = (V_M, E_M)$ as

$$V_M = V_{\text{in}} \cup \left(\bigcup_{i \in I_A} V_{A_i} \right) \cup \left(\bigcup_{i \in I_E} V_{E_i} \right),$$

$$E_M = E_{M,\text{con}} \cup \left(\bigcup_{i \in I_A} E_{A_i} \right) \cup \left(\bigcup_{i \in I_E} E_{E_i} \right).$$

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

The nodes and edges in M can be partioned in sets according to their type:

$$V_M = V_{M,\text{var}} \cup V_{M,\text{mod}}, \; V_{M,\text{var}} \cap V_{M,\text{mod}} = \emptyset, \tag{8}$$

where $V_{M,\text{var}}$ and $V_{M,\text{mod}}$ are the sets of variable nodes and function nodes, respecively;

$$E_M = E_{M,\text{con}} \cup E_{M,\text{mod}},\ E_{M,\text{con}} \cap E_{M,\text{mod}} = \emptyset, \tag{9}$$

where $E_{M,\text{con}}$ and $E_{M,\text{mod}}$ are the sets of connection edges and function edges, respectively. These sets will be referenced in the process for obtaining an FPG.

5.2 Fundamental Problem Graph

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

(1) $F \subset M$ and $G_{E_i} \subset F\ \forall i \in I_E$
(2) $\forall i \in I_A$, if $F \cap G_{A_i} \neq \emptyset$ then $G_{A_i} \subset F$
(3) $\forall v \in V_F$ with $v \in V_{M,\text{var}}$ and $\deg_l^-(v) \leq \deg^-(v) \leq \deg_u^-(v)$
(4) $\forall v \in V_F$ there exists a reverse path $P \subset R_F$ from x to v with $x \in V_{E_i}$

Condition (1) asserts that only the nodes and edges provided by the maximal connectivity graph can be used in the fundamental problem graph and that every expression block must be included. Condition (2) requires that analysis blocks must be included or excluded in their entirety. Condition (3) stipulates that the number of edges directed into each variable node must be within the lower and upper in-degree limits; if $\deg^-(v) < \deg_l^-(v)$ the node is the location of a *hole*, and if $\deg^-(v) > \deg_u^-(v)$ the node is the location of a *collision*. Lastly, condition (4) ensures that only the nodes that are being used are included in the FPG by requiring that for every node a reverse path exists from at least one expression block to that node. The reverse graph R_F is obtained from F by simply switching the orientation of every edge in E_F.

5.3 Algorithm for obtaining the Fundamental Problem Graph from the MCG

In general, there may be zero, one, or many graphs that satisfy the FPG conditions in Sec. 5.2. Here, we describe an algorithm for obtaining an FPG by starting with the MCG and disconnecting connection edges until the FPG conditions are met. With this approach, the problem is reduced to deciding which connection edges to remove.

Step 1: Holes The first step is to detect holes and disconnect the first set of connection edges downstream of them, as indicated in Fig. 10. These connection edges are removed because they represent output variables which cannot be determined because the analysis function does not have adequate inputs.
To begin the process, an initial graph is created as a copy of the MCG

$$F_0 = M, \tag{10}$$

with $F_0 = (V_{F_0}, E_{F_0})$, and $C_{F_0,\text{con}}$ is the set of connection edges. The set of variable nodes that are holes is identified as

$$H = \{v \in V_{F_0} \mid v \in V_{M,\text{var}} \text{ and } \deg^-(v) < \deg_l^-(v)\}, \tag{11}$$

which is the set of variable nodes with fewer incoming edges than are allowed by the lower indegree limit. The updated set of connection edges is then created by removing the edges preceding or succeeding the analysis block:

$$E_{F_1,\text{con}} = E_{F_0,\text{con}} \setminus \{(x,y) \in E_{F_0,\text{con}} \mid x \in V_{A_i} \text{ or } y \in V_{A_i}, \text{ and } H \cap V_{A_i} \neq \emptyset\}. \tag{12}$$

Because removing these edges can create new holes, this step must be repeated until no additional holes are found. If the hole identification step identifies a variable node in an expression block as hole, meaning $V_{E_i} \cap H \neq \emptyset$ for some $i \in I_E$, then the algorithm terminates because an FPG cannot be obtained.
If this step finishes without finding a hole in an expression block, it is guaranteed that an FPG can be obtained because F_1 now satisfies all four conditions from Sec. 5.2 expect for (3), which requires there to be no holes or collisions in the graph. Since no more holes remain, and collisions can always be resolved without producing a hole, (see Table 1), then it is guaranteed than an FPG can be obtained.

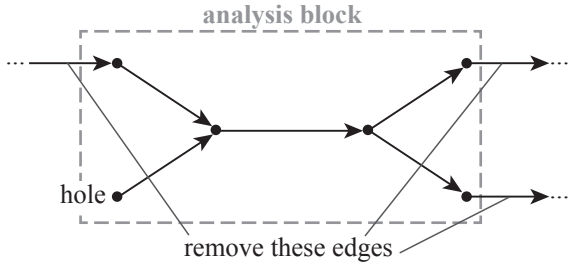


Fig. 10: Example variable node indicating a hole.

Step 2: Collisions The second step is to detect collisions and to disconnect precisely the number of connection edges required such that all collisions are resolved without introducting holes. The set of variable nodes at which collisions occur is

$$S_{\text{nodes}} = \{v \in V_{M,\text{var}} \mid \deg^-(v) > \deg_u^-(v)\}. \tag{13}$$

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

$$S_{\text{edges}} = \big\{\{(x,y) \in E_M\} \;\big|\; y \in S_{\text{nodes}}\big\} \tag{14}$$

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. 11 to exemplify 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). We may then construct sets of edges as

$$B_j = \big\{e_k \in S_{\text{edges},j} \mid k \in \{1,2,\ldots,K\} \text{ with } \deg_u^-(v_j) \leq K \leq \deg_u^-(v_j)\big\},\ j \in J, \tag{15}$$

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 and lower indegree limits of v_j. The

construction of each B_j corresponds to making a decision about which edges to include and which edges not to include. Let the new set of connection edges be

$$E_{F_2,\text{con}} = \{e \in E_{F_1,\text{con}} \mid e \in B_j \text{ for some } j \in J\}. \tag{16}$$

The set of all edges is created by removing the connection edges not in $E_{F_2,\text{con}}$:

$$E_{F,2} = E_{F_0} \setminus (E_{M,\text{con}} \setminus E_{F_2,\text{con}}), \tag{17}$$

which gives

$$F_2 = (V_{F_0}, E_{F_2}). \tag{18}$$

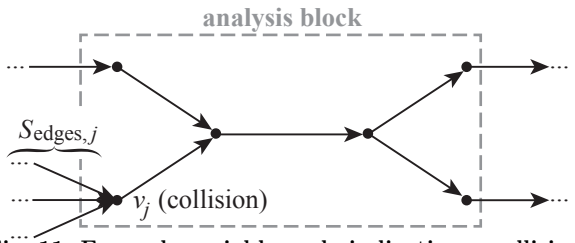


Fig. 11: Example variable node indicating a collision.

Step 3: Finalize The third and final step is to prune the graph to exclude any analysis blocks that became unneeded after the collisions were resolved. This is accomplished by first creating the reverse graph of F_2, $R = (V_R, E_R)$, where

$$E_R = \{(x, y) \mid (y, x) \in E_{F_2}\},\ V_R = V_{F_2} \tag{19}$$

Next, a new node b is added to V_R and edges directed from this node to each of the expression blocks are added:

$$\forall i \in I_E, \forall v \in V_{E_i},\ \text{if } \deg^+(v) = 0 \text{ then } (b, v) \in E_R \tag{20}$$

The set of nodes that may be reached from b is then constructed as

$$U = \{v \in V_R \mid \exists P \text{ a path from } b \text{ to } v,\ P \subset R\}. \tag{21}$$

Because b is directed into only the expression blocks, any path from b necessarily provides a path from at least one expression block, which means the node is being used in the problem formulation.
The list of analysis blocks with at least one node in U is constructed as

$$I_F = \{i \in I_A \mid V_{A_i} \cap U \neq \emptyset\} \tag{22}$$

Any analysis block not in I_F should be removed from the graph because none of its outputs contribute to the problem. Inputs that are not being used are also removed. The set of nodes to remove is then

$$N = (V_{\text{in}} \setminus U) \cup \left(\bigcup_{i \notin I_F} V_{A_i} \right), \tag{23}$$

and the new set of nodes is created as

$$V_F = V_{F_2} \setminus N. \tag{24}$$

Edges involving the removed nodes are also deleted as

$$E_F = E_{F_2} \setminus \{(x, y) \mid x \in N \text{ or } y \in N\}. \tag{25}$$

The fundamental problem graph is then

$$F = (V_F, E_F). \tag{26}$$

If desired, the indices of the analysis blocks implemented in the FPG can then be found as

$$I_F = \{i \in I_A \mid G_{A_i} \subset F\}. \tag{27}$$

Also, the set of connection edges can be extracted by considering only edges whos endpoints are not in the same analysis block:

$$E_{F,\text{con}} = \{(x, y) \in E_F \mid \sim (x \in V_{A_i} \text{ and } y \in V_{A_i} \text{ for some } i \in I_F)\}. \tag{28}$$

This algorithm will always provide an FPG if one exists. If an FPG does not exist, the limiting factors preventing a valid formulation are revealed.

5.4 Suggested Process for Using the FPG Algorithm

Section 5.3 provided an algorithm for obtaining an FPG from the given MCG. However, if an FPG cannot be obtained, this algorithm can be applied as part of a larger process in which the designer changes the problem or the supplied analyses to attempt to obtain an FPG. The following steps detail the suggested procedure for obtaining an FPG:

(A) Begin with a set of inputs, analysis codes, objectives, and constraints.
(B) Build the MCG as described in Sec. 5.1.
(C) Set indegree limits for variable nodes representing local inputs as described in Sec. 3.4 and Table 1
(D) Run the FPG algorithm described in Sec. 5.3. If a valid FPG is unattainable:
  (a) Change the MCG by adding analysis blocks and/or inputs, which means identifying new analyses to include in a potential MDAO workflow.
  (b) Change indegree limits (see Table 1).
(E) Repeat from step (A) until an FPG is obtained.

This process illustrates how the algorithm for obtaining a valid FPG is to be applied. However, this process also serves to illustrate how a designer may revist the MCG after an FPG was already produced. For example, if a designer wants to change an input to be multi-fidelity, new analysis blocks can be added to the original MCG and indegree limits can be modified accordingly, which corresponds to steps (A)–(C), and then the algorithm can be applied again.

6 Example Problem

This section presents an example problem to demonstrate the process of obtaining the FPG and the advantages of doing so. The example task is to create an FPG for the conceptual sizing of a single-aisle subsonic transport using a set of analysis tools with objectives including performance and gross weight for a required mission. The full set of analysis codes available for use is provided in Table 2. Each analysis code contributes

Table 2: Analysis codes for subsonic transport sizing

analysis code	description
VSP	parametric vehicle 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, engine sizing, and weight estimation
FLOPSb	mission performance only

individual disciplinary analysis capability, but the outputs of the codes 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 configurations corresponding to different uses. FLOPSa denotes FLOPS implemented to execute both mission performance and engine analysis, while FLOPSb indicates FLOPS implemented to execute for only mission performance analysis. This representation is useful for "supercodes" that are capable of being implemented in different ways and with different sets of inputs and outputs. The enumeration of these analysis tools and objectives concludes step (A) in Sec. 5.4.

Step (B) is the production of the MCG. In this example, the MCG is formed using a consistant variable naming convention and then connecting all variables with the same name with connection edges. Table 3 presents the full list of variables in the leftmost column and indicates whether the variable is an input or an output of each analysis code. Some variables, such as geometry and performance, represent groups of variables that are passed as arrays or other data structures due to their similarity. This bundling of variables is not fundamental and does not limit the generality of this example; rather, it simplifies the presentation.

In this example, 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 3. Each analysis block is formed by first creating variable nodes for each input and adding directed edges into a single function node. An edge is directed from this function node into a second function node, which is then directed into variable nodes corresponding to each output. A sample analysis block is shown for analysis code FLOPSa in Fig. 12.
2. Expression blocks are created to represent the objectives for performance and total weight.

Table 3: 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

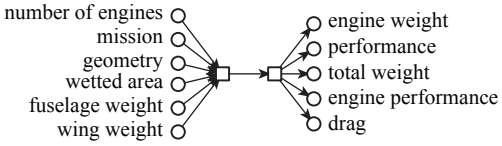


Fig. 12: Sample analysis block for analysis code FLOPSa using Table 3

3. A variable nodes is created to represent the geometry variable as a input. Any inputs incorrectly omitted will be identified in step (D).
4. Connection edges are created that connect each variable node to every other variable node representing variables with the same name. The direction is determined by whether the variable node has an edge directed into or out of a function node, i.e. whether it is a local input or a local output.

The resulting MCG is shown in Fig. 13. Cycles are shown as dashed lines. FPGs formed from this MCG may or may not retain these cycles.

To implement step (C) in Sec. 5.4, the indegree of every variable node is set to unity to reflect a single fidelity analysis for which the design variables have not yet been selected.

Step (D) now procededs by executing the FPG algorithm. An FPG cannot be obtained because the variable nodes representing the number of engines are holes for NPSS, FLOPSa, and FLOPSb, and the variable nodes representing the mission definition are holes for FLOPSa and FLOPSb. These holes propagated upstream and created holes in the objectives, thereby preventing a valid FPG.

Step (D)(a) begins to resolve this conflict by creating a input for the number of engines. In this example, the number of engines was not originally included as a input to demonstrate how the process responds to this error. Step (D)(b) then reduces the lower and upper indegree limits for the variable node representing the mission definition to zero, implying that these nodes must be specified as design variables (see Table 1). The new MCG is shown in Fig. 14.

Although an FPG may now be obtained, this process has not provided a method to decide which edges to retain when resolving a collision, and these decisions will likely determine which analysis tools are included in the FPG. These decisions are left to

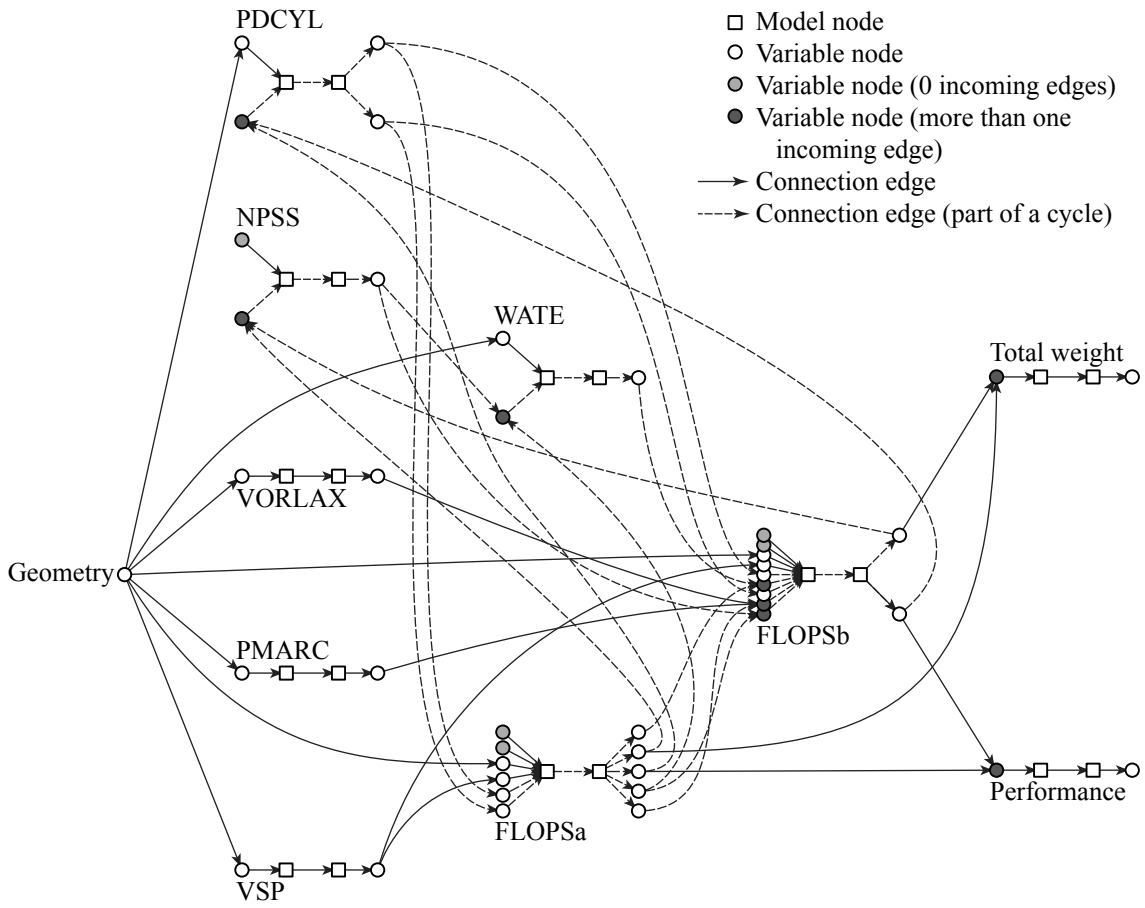


Fig. 13: Maximal connectivity graph for the subsonic transport example problem

the discretion of the designer in the context of the specific implementation. However, the graph-based approach presented in this paper enables standard graph algorithms to be employed to automate these decisions based on considerations of metrics related to the graph or other data related to the analysis. An example is a metric to obtain an FPG with the fewest possible number of cycles. In this example, the FPG with the fewest number cycles is shown in Fig. 15, in which the edges belonging to a cycle are inticated by dashed lines. For this example problem, the cycles were detected using the implementation of Johnson's algorithm [26] in the Python package NetworkX. There are two cycles which arise from each local output from PDCYL being directed into FLOPSa and then back into PDCYL.

Similar alternatives would be to minimize the number of analysis blocks involved in cycles, counting multiplicity, or to minimize the length of the longest cycle (called the *circumference* of the graph). It is beyond the scope of this work to explore efficient methods for finding the FPGs meeting these criteria. However, as discussed

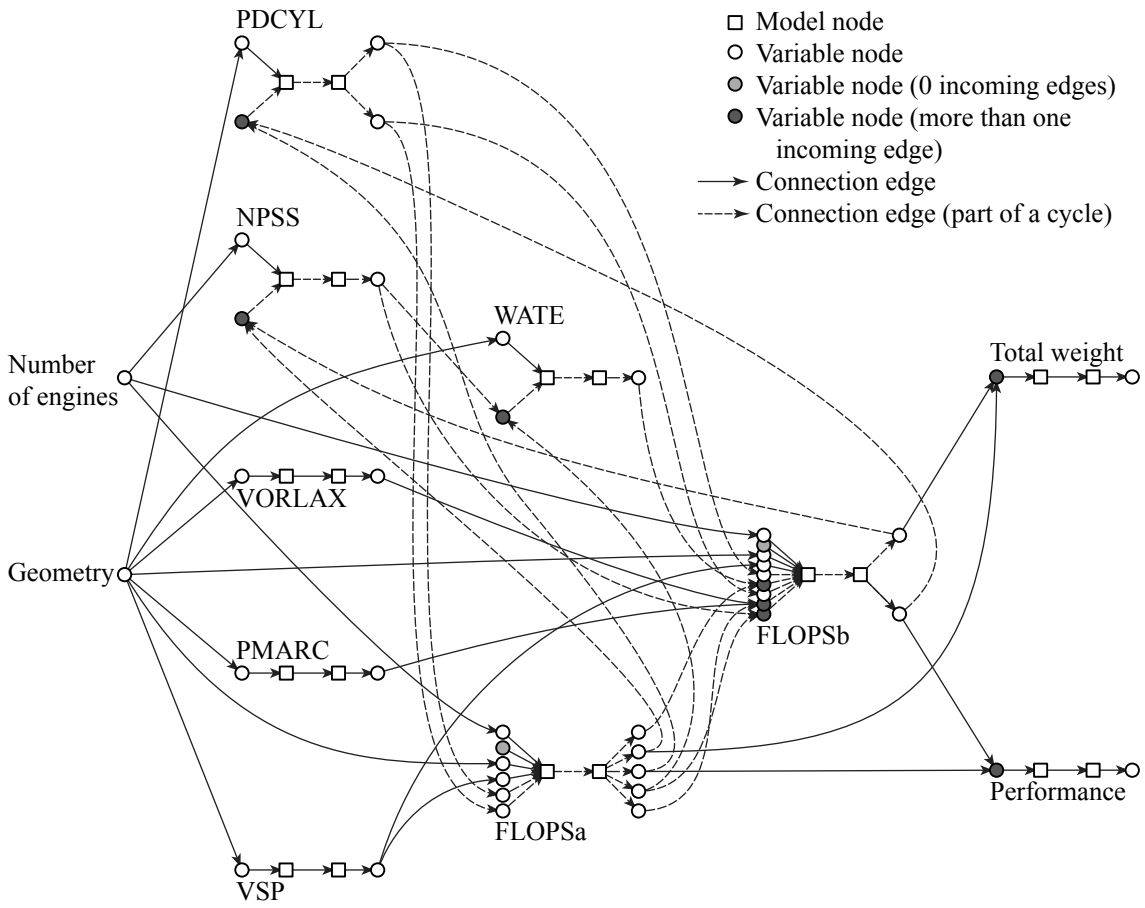


Fig. 14: The updated MCG with a new input.

by Gabow[27], graph algorithms typically scale well with the size of the graph. This suggests that the computation cost of finding an FPG should be small.

Alternatively, it may be desirable to resolve collisions by choosing the higher fidelity analysis tools. One method to ensure that the edges directed from certain analysis blocks are those chosen to resolve conflicts is to use a ranking system. Each analysis block is assigned a value by the designer describing how desireable it is for this block to be present in the FPG. The connection edges directed out of each analysis block are then given a weight equal to the value assigned to the analysis block. Finally, each collision is resolved by selected the edges with the highest weights. For the current example problem, consider the rankings for each analysis code given in Table 4. The resulting FPG has four cycles and is shown in Fig. 16.

Finally, consider a case in which the inviscid drag input into FLOPSb is a multi-fidelity input, meaning that multiple analysis codes calculate the same variable. This multi-fidelity formulation is implemented by setting the upper indegree limit for this

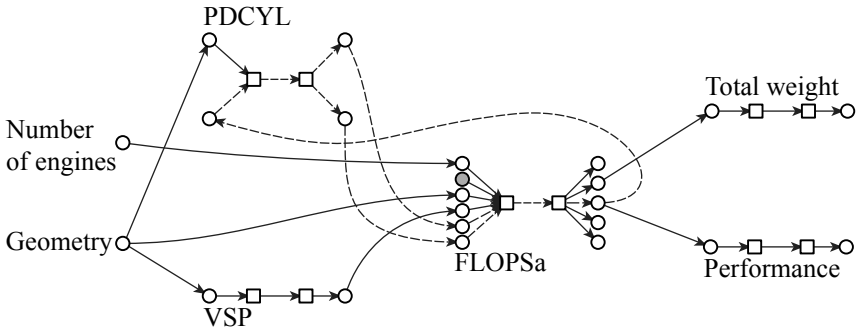


Fig. 15: FPG with the fewest number of cycles.

Table 4: Example ranking of importance

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

node as two and then repreating the (automated) process. In this case, both VORLAX and PMARC are retained, resulting in an FPG that omits only FLOPSa.

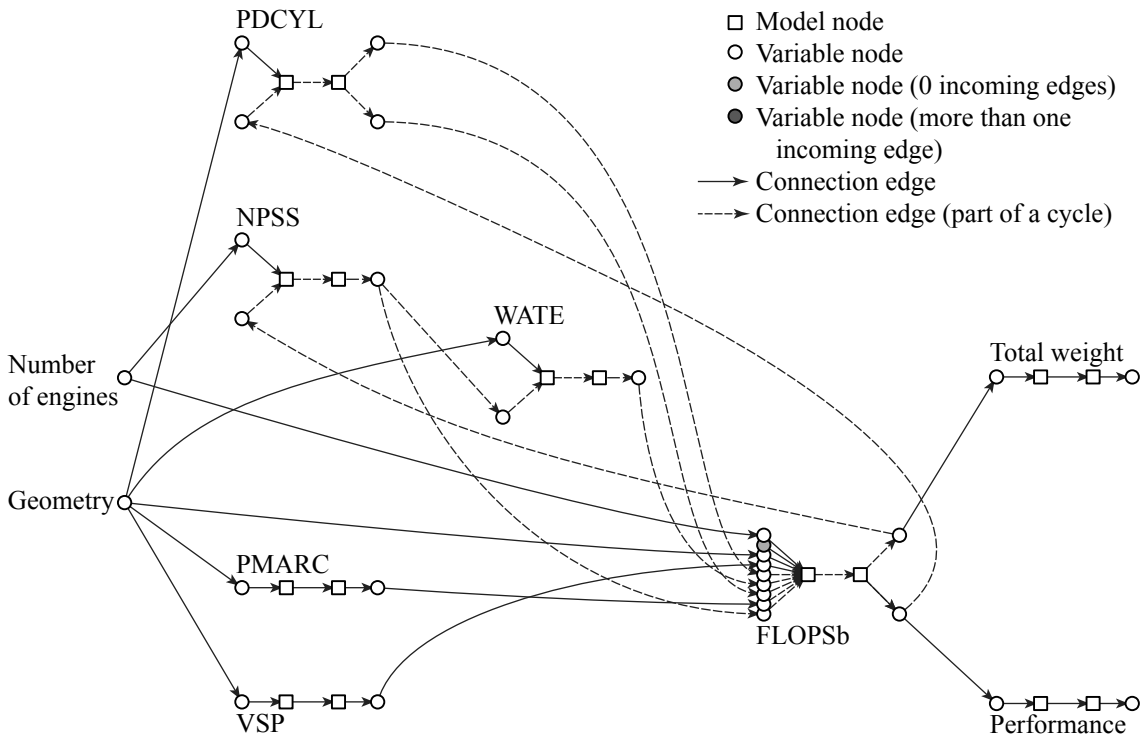


Fig. 16: FPG obtained using the ranking system.

Conclusions

In this paper, we have presented an approach for describing MDAO problems with constructs and syntax from graph theory. Our graph description shares similarities to other approaches such as REMS, Ψ, FDT, DSM, and XDSM, but it provides new constructs tailored to applications in early phases of MDAO problem formulation before specific problem solution approaches are envisioned. In particular, we introduce the concepts of the Maximally Connected Graph (MCG) and the Fundamental Problem Graph (FPG).

The MCG addresses the question, "Given a set of analysis tools, what are all of the variable interconnections between them that could be established?" The MCG provides a structured formalism to identify *holes* and *conflicts* in interconnections between input and output variables implied by the set of analysis tools. A hole corresponds to a variable that is not calculated in the MCG data flow but is needed by other tools. To create a valid problem formulation, all holes must be "filled" either by assigning the corresponding variable as a design variable to be set by the user or an optimizer, or by introducing additional tools to compute the variable.

A conflict corresponds to a situation in which two or more analysis tools compute separate instances of the same variable. To create a valid problem formulation, all conflicts must be resolved either by removing redundant analysis tools or by introducing

logic or solvers into the data flow to choose between or aggregate multiple instances during execution. The identification of conflicts in an MCG is especially important for problem formulations involving multiples analysis tools of different fidelities; in this context, a conflict represents an opportunity to formulate a multi-fidelity problem that can take best advantage of the efficiency/accuracy tradeoff between different analysis tools. Resolving conflicts and filling holes in an MCG requires choices by the user to specify the overall MDAO problem that he/she desires to solve, i.e. to produce the required set of system-level outputs given a specified set of design variables.

The FPG is a graph that describes a data connection structure corresponding to an MDAO problem formulation free of conflicts and holes. An FPG is the result of user choices to fill holes and resolve conflicts in the initial MCG. One benefit of using graph theory is the standard algorithms, such as cycle detection, which can be used to inform the user's decisions. Typically, many different FPGs could be attained from a given MCG, depending on the user choices. The number of possible FPGs that can be attained by selecting different design variables or introducing additional analysis tools to fill holes and/or resolve conflicts can be viewed as a measure of the freedom available to the user in implementing the available analysis tools to formulate valid MDAO problems. In Section 5, we present an algorithm for identifying holes and conflicts in an MCG that can be applied by the user in a process to arrive at valid FPGs. In Section 6, we provide an example application of formulating an FPG from an MCG for a commercial aircraft design problem.

Once a desired FPG is identified, the graph syntax described in the paper can be leveraged to formulate a Problem Solution Graph (PSG) describes particular MDAO solution architectures, e.g. MDF, IDF, BLISS, ATC, that can be used to solve the problem defined by the FPG. Many possible PSGs could be formulated to solve a problem corresponding to a specified FPG. The application of the syntax to PSGs has not been illustrated in the paper.

For simple problems with few analysis tools and variables, the formulation of a valid problem description is straightforward. However, MDAO problems continue to grow in scale and complexity. As the numbers of analysis tools and variables have increased, it has become increasingly challenging and time consuming for engineering teams to determine how the multiple analysis tools can be interconnected to produce valid problem formulations, to know when other tools must be introduced, and to determine the number of free variables in the problem that can or should be varied by the designer or an optimizer. The graph formalism presented in this paper is intended to offer value in this context of increasing problem complexity by providing a formal approach to identify and resolve missing and redundant data in order to create valid MDAO problem formulations.

References

1. Cramer, E. J., Dennis, Jr., J. E., Frank, P. D., Lewis, R. M., and Shubin, G. R., "Problem Formulation for Multidisciplinary Optimization," *SIAM Journal on Optimization*, Vol. 4, No. 4, November 1994, pp. 754–776.
2. Sobieszczanski-Sobieski, J. and Haftka, R. T., "Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments," *Structural Optimization*, Vol. 14, 1998, pp. 1–23.
3. de Weck, O., Agte, J., Sobieszczanski-Sobieski, J., Arendsen, P., Morris, A., and Spieck, M., "State-of-the-art and future trends in multidisciplinary design optimization," *48th*

AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 23 - 26 April 2007, Honolulu, Hawaii, 2007.
4. Dean, S. R. H., Doherty, J. J., and Wallace, T. R., "MDO-Based Concept Modelling and the Impact of Fuel Systems on Wing Design," *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition Orlando, Florida, 5 - 8 January*, 2009.
5. Gavel, H., Ölvander, J., and Krus, P., "Optimal Conceptual Design of Aircraft Fuel Transfer Systems," *Journal of Aircraft*, Vol. 43, No. 5, 2006, pp. 1334–1340.
6. Antoine, N. E. and Kroo, I. M., "Aircraft Optimization for Minimal Environmental Impact," *Journal of Aircraft*, Vol. 41, No. 4, 2004, pp. 790–797.
7. Rallabhandi, S. K. and Mavris, D. N., "Aircraft Geometry Design and Optimization for Sonic Boom Reduction," *Journal of Aircraft*, Vol. 44, No. 1, 2007, pp. 35–47.
8. Kirby, M. R. and Mavris, D. N., "The Environmental Design Space," *26th International Congress of the Aeronautical Sciences*, 2008.
9. 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.
10. 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.
11. Sellar, R., Batill, S., and Renaud, J., "Response surface based, concurrent subspace optimization for multidisciplinary system design," *34th AIAA Aerospace Sciences Meeting and Exhibit*, Citeseer, Reno, NV, Jan. 1996.
12. 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.
13. 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.
14. 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.
15. Rogers, J., "DeMAID/GA-an enhanced design managers aid for intelligent decomposition (genetic algorithms)," *the Proceedings of the 6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, AIAA-96-4157-CP*, 1996, pp. 1497–1504.
16. 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.
17. 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.
18. 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.
19. Martins, J. R. R. A. and Lambe, A. B., "Multidisciplinary Design Optimization: Survey of Architectures," *AIAA Journal*, 2012, pp. 1–46.
20. 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.
21. Diestel, R., *Graph Theory*, Springer, 2010.
22. Smith, D., Eggen, M., and Andre, R. S., *A Transition to Advanced Mathematics*, Brooks/Cole Pub Co, 2006.
23. 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.
24. 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.
25. 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.
26. Johnson, D. B., "Finding all the elementary circuits of a directed graph," *SIAM Journal on Computing*, Vol. 4, No. 1, March 1975, pp. 77–84.
27. Gabow, H. N., "Scaling algorithms for network problems," *Journal of Computer and System Sciences*, Vol. 31, No. 2, 1985, pp. 148–168.