Structrual and Multidisciplinary Optimization

A Graph Theory Based Approach to MDAO Problem Formulation

David Pate, Justin Gray,Dr. Brian German

Abstract blah blah blah

Nomenclature

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

1 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.

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

David Pate
Gatech

Dr. Brian German
Gatech

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]].

As the size and complexity of engineering systems grow, the time and expense for setting up analysis models grow with them. Multidisciplinary Design Analysis and Optimization (MDAO) frameworks such as OpenMDAO[1] and ModelCenter have enabled a new level of analysis tool integration and paved the way for models with more analyses and increasing numbers of interdisciplinary couplings. That new capability has created a new challenge. 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 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 creating 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[2]. 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 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.

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 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 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 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 execution sequence is used to solve the problem. However, it is very easy to formulate a problem without any reference to a specific solution strategy, but still impose a a specific execution sequence. For example, consider the Sellar test problem [3]:

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

D_1 and D_2 represent analysis tools, F, G_1, and G_2 are the objective and constraint functions respectively, and H is the coupling constraint. Equation 1 makes an inherent assumption about the execution order for the problem. Analysis D_1 outputs y_1, which is an input to D_2. Hence, D_1 should be run before D_2 with with y_2 being iterated on to convergence with $\hat{y_2}$. However, a slightly different formulation is equally valid and still represents the exact same problem:

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

Equation 2 differs only slightly from Eq. 1. Notice that the formulation from Eq. 1 includes $\hat{y_2}$, while Eq. 2 includes $\hat{y_2}$. The result is that D_1 is now dependent on the output of D_2, and y_1 will be iterated on to convergence with $\hat{y_1}$. Now the problem has to be solved by running D_2 first and then D_1. 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:

$$
\begin{aligned}
given \quad & y_1 = D_1(x_1, \hat{y_2}, z_1, z_2) \\
& y_2 = D_2(\hat{y_1}, z_1, z_2) \\
min. \quad & F(x_1, y_1, y_2, z_2) \\
w.r.t. \quad & x_1, \hat{y_1}, \hat{y_2}, z_1, z_2 \\
s.t. \quad & H_1(y_1, \hat{y_1}) = 0 \\
& H_2(y_2, \hat{y_2}) = 0 \\
& G_1(y_1) \geq 0 \\
& G_2(y_2) \geq 0
\end{aligned} \tag{3}
$$

The formulation in Eq. 3 differs from both Eq. 1 and Eq. 2 because it has two coupling constraints which both must be met. The result is that Eq. 3 required the presence of both $\hat{y_1}$ and $\hat{y_2}$, which means that D_1 and D_2 each now operate with their own local copies of the coupling variables. This fully decouples the problem so that either D_1 or D_2 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[2]. The key feature they identify to allow this decoupling is the creation of independent variables for each analysis. Tosserams et. al. in their development of the ψ language for heirarchical problem composition make the same requirement that all analyses get their own local copies of variables in order to use them as the fundamental building blocks for problems[4]. Martins and Hwang derive the same requirement, from a different perspective, in their work on computing analytic derivatives for MDAO problems[5]. They determined that local copies of all variables allows for derivation of all relevant methods for calculating analytic derivatives from the basic chain rule.

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

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

3 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[6]. 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[7]. 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[8,9]. This re-ordering is done through row operations on the DSM matrix and yields multiple specific problem formulations which all solve the same FPF.

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[10]. 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[11]. 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. The dependence of m on A is now captured in 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.

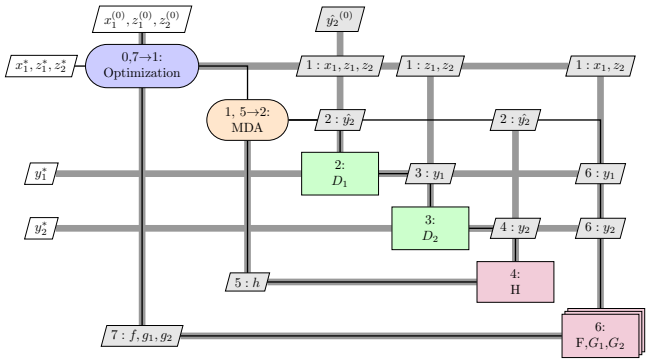


Fig. 2: XDSM for Eq. 1, with a gauss-siedel iteration and MDF solution architecture.

Alexandrov and Lewis introduced a graph based syntax called REMS which incorporates objectives and constraints into a graph, effectively combining FDT and DSM[2]. 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. 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[12]. 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 [13]. 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 [14].

Although XDSM captures part 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 provides some kind of solution strategy. The XDSM for Eq. 1 is given in Figure 2. This diagram is shown with an assumed gauss-siedel iteration scheme and a MDF solution architecture. Hence XDSM is too specific for use with a fundamental problem formulation.

Of all the methods, REMS comes the closest to fully describing a fundamental 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

while retaining the flexiblity to represent a fundamental problem formulation free of solution specific information.

4 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 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 that goal the graph needs to accomodate a number of constructs of MDAO problems:

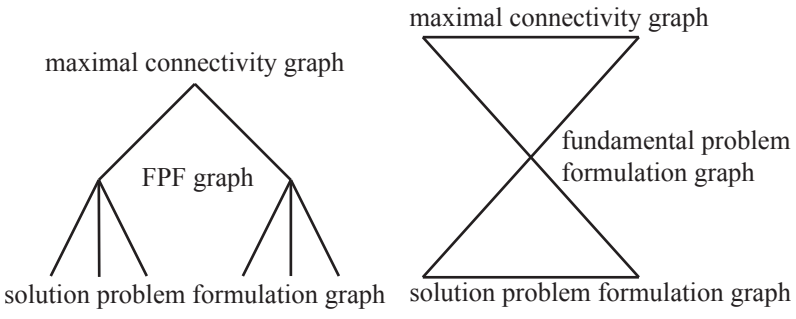
- Analysis tools and connections between them
- Design variables, objectives, and constraints
- Local and global properties
- Coupling between analyses
- Multi-fidelity analyses

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 specific 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 the following three graphs:

- Maximal Connectivity Graph (MCG)
- Fundamental Problem Graph (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 the *fundamental problem 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 until 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 of 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

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

5 Syntax Definition

In this section we present the necessary graph theory fundamentals to construct the graphs discussed above. The notation used in this work is adapted from Diestel [15]. 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. 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. 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

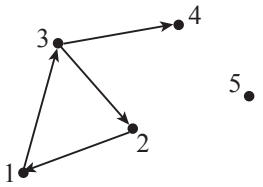


Fig. 4: Example directed graph.

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

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

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 $E = \{(x_0, x_1), (x_1, x_2), \ldots, (x_{k-1}, x_k)\}$. 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 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[16]. 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 $|B|$.

5.1 Node and Edge Types

We now define different types of nodes and edges with specific properties to compose graphs used to represent an MDAO problem. There are three node types:

variable: represents scalar or array data
model: responsible for mapping inputs to outputs
driver: control logic capable of managing iteration

In addition to the three 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).
driven edge: Passing of information from a driver node to a variable node. A single variable node can have many incomming driven 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[2]. We've chosen to rename the "function" node type to "model" to be more consistent with modern MDAO frameworks. The present work adds one more node types and one more edge type to the syntax to allow descriptive graphs for all three phases of the design process. In addition, we introduce the concept of fixed vs. free connection edges to represent the static relationship between variable and model nodes from a single analysis, vs the flexible relationship between variables from different analyses.

5.2 Rules for Nodes and Edges

There are specific rules for the usage of these nodes and edges. The driver node and the driven edge are only allowed to be present in PSGs. All other node and edge types can be present in any of the three graphs, subject to these 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.

5.3 indegree and outdegree

The *indegree* of a node is the number of connection edges directed in and is denoted as $\deg^-(v)$, and the *outdegree* 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 driven 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 driven edges from both optimizers but this would not affect the indegree of those variable nodes.

We also define the *upper indegree limit*

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

and the *lower indegree limit* as

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

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

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

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

This represents a lack of information being supplied to the node.

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

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

An algorithm for detecting and managing these violations if proposed in Sec. 7.7.3.

6 Graph Representation of MDAO Constructs

In this section we will make use of the Sellar problem, described in Eqn. 3. The graph representation of the Sellar problem, using the new stynax, is presented in Fig. 5. Throughout the rest of this section, different sub-graphs or slight modifications of the full graph will be used to demonstrate the important MDAO constructs within this graph syntax.

6.1 Analysis Blocks and Connections

Analysis tools take in a number input variables and then perform some work to calculate the values for their respective outputs. In an MDAO graph, this process is represented by a group of nodes and edges called an *analysis block*, shown in Fig. 6. 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. 6, the analysis block contains two model nodes, with a single edge connecting them. This edge represents the necessary

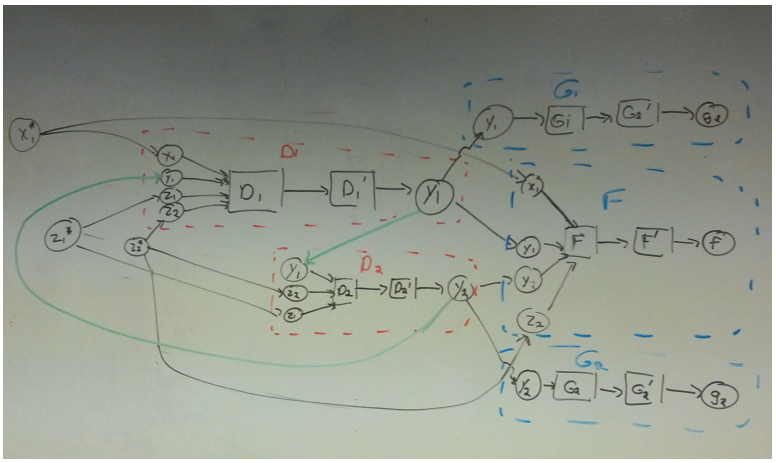


Fig. 5: Graph of the Sellar problem formulation

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. In this way, they cannot be altered but they can be removed as a group.

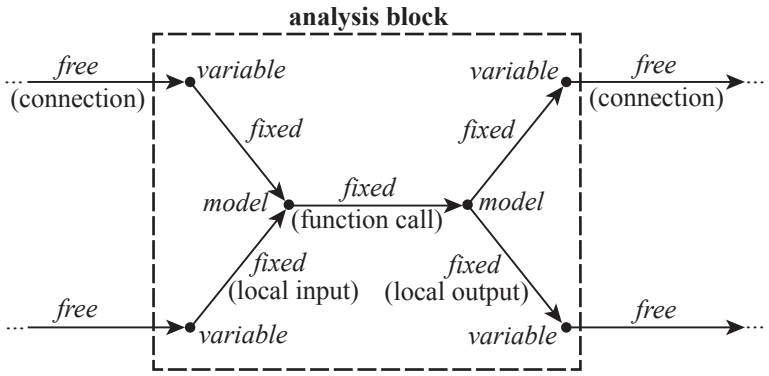


Fig. 6: Example analysis block. The each node type and edge type is labeled in italics and annotated parenthetically.

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 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.

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, can 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.

6.2 Design Variables

For the Sellar problem, there are three design variables, x_1^*, z_1^*, and z_2^*. In Fig. 5, these are the only three holes (nodes without out incomming edges) in the graph.

[NOTE: Re-work this one to match D_2 from Sellar]

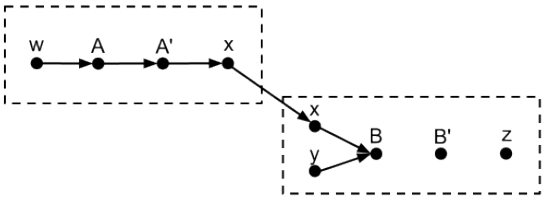


Fig. 7: Notional graph with two potential design variables

More formally stated, given a MCG with the $\deg_l^-(v) = 1$ for all nodes, there will likely be a set of holes preventing a valid FPG from being obtained. Each hole represents a node that could potentially become a design variable. It is up to the designer to examine each and decide if it is appropriate to consider it a design variable, in which case the designer shall set the lower indegree limit to be zero. However, some variable holes may not actually be suitable as a design variable. For instance, if an aircraft mission analysis code requires a drag input which ends up being a hole then it's likely best to leave$\deg_l^-(v) = 1$ and find a new analysis code to fill the hole in graph.

So a design variable is defined as any variable node with $\deg_l^-(v) = 0$. This, by definition, means that design variables are not holes in the graph, since they have not violated their lower indegree limit. Regardless, when working with a MCG or FPG, only connection edges are allowed, so a design variable in either graph will be a variable with no incomming edges edges. For a valid PSG any design variable node would have at least one, but possibly multiple, incomming driven edges from one or more driver nodes.

6.3 Objectives, and Constraints

Like design variables, objectives and constraints are also constructs that need to be identified when setting up a given design problem. In the case of objective functions single output values could be selected, but commonly multiple values are assembeled together via simple algebraic expressions to form a composite objective function. (e.g.

The objective function from Eqn. 3 is $x_1^2 + z_2 + y_1 + e^{-y_2}$). Constraints are always given in the form of either an inequality or an equality. (e.g. a constraint from Eqn. 3 is $\frac{y_1}{3.16} - 1 \geq 0$.) By convention, constraints are usually given such that some expression will be less than or equal to zero, so any positive value would violate the constraint. For both objective and constraints, these simple expressions take some inputs and map them to an output value of significance to the overall design problem.

These operations, though very simple, are effectively the same kind of task performed by an analysis block. A constraint or objective function can therfore be represented as another analysis block on the graph, with it's own input and output variable nodes. Although fundamentally no different than an analysis block, for clarity and convience it is useful to distinguish between regular analysis blocks and those that arise from the addition of objectives or constraints to the graph. Therefore, we define an *expression block*, as the collection of variable and model nodes related to a given objective or constraint function. Fig. 8 highlights the objective and constraint blocks from the Sellar problem.

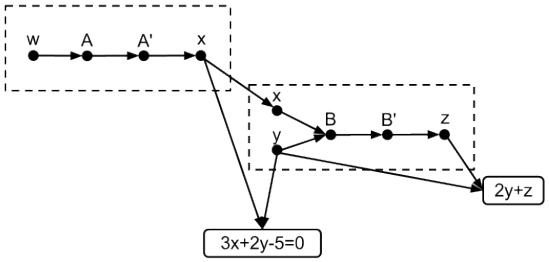


Fig. 8: Notional graph with and objective and constraint nodes

6.4 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. Global variables are a part of many MDAO problems and must be representable in the graph. Follwoing our simple definition any variable node that had multiple incomming or outgoing edges, would be global. In Fig. 5 the variable nodes for x_1^*, z_1^*, and z_2^* have multiple outgoing edges and would be considered global.

Note that the variable x_1 from the Sellar problem is usually refered to as a local variable, which is contradicted by the given graph. This discrepancy arises from the explicit treatment of objectives and constraints as separate expression blocks. Normally in the Sellar problem x_1 is considered a local variable because it only directly affects analysis block D_1. By forcing the expansion of F into an expression block with its own inputs, then F gets it's own copy of the x_1 variable. This necessitates the creation of the global x_1^* node in the graph with two outgoing edges to link the two x_1

[INSERT GRAPH HERE]

Fig. 9: Graph of a notional problem with simple coupling

inputs nodes in the different blocks. Interestingly, using this definition, where x_1^* is a global variable fits nicely within the structure of Braun's Colaborative Optimization (CO) archtiecture [17]. The rules of setting up a problem with CO require that each discipline be given an independent local copy of all global variables. All local variables are retained uniquely within their respective disciplines, except when a local design variable shows up explicitely in the objective or global constraint. In that case a global variable is created, and the discipline again gets an independent local copy of it. Given our definition of global vs local variable nodes, no such exception for local variables in global expressions is necessary. When you include a given variable in an expression block, it becomes global by definition and should be treated as such in CO.

Analysis and expression blocks can also have a locality defined for the set of nodes and edges within the block. Their locality is determined by the incomming and outgoing edges from the block as a whole. Fig. 5 has expression blocks for each constraint, G_1 and G_2. Both constraints are local to their respective analysis blocks. But the expression block for F has incomming edges from D_1 and D_2, so it would considered global.

Simply stating that any node or block with incomming or outgoing edges from more than one other node or block is global works glosses over a more subtle aspect of some MDAO problems. The locality of any node in a graph is a relative property. For instance, a single node might have outgoing edges to two separate analysis blocks, A and B, but none to a third, C. Then you could say that this node was global with respect to A and B separately, but local to the group A, B. This situation produces a natural hierarchy in a graph, where you could form increasingly smaller groups as you segment problem into finer localities.

When solving a problem using a heirarchical decomposition approach, there are various techniques for partitioning that are effective in different scenarios [18,19,20, 21,22]. Tosserams et al proposed a language based syntax for describing the partitioning of problems, named Ψ [4]. Ψ provides an opposing perspective to this work, where they compose hierarchies from the bottom up, into larger and larger groups. However, Ψ provides a compiler that can produce a standard problem representation of the assembled problem, and a number of example converters from that standard representation to application specific formats.

[[Could probably build this and include it in appendix]]

6.5 Coupling Between Analyses

Coupling exists in a design problem when a set of two or more analyses each depend on the others outputs. In the Sellar problem from Eqn 3 the two coupling constraints, H_1 and H_2 provide a reciprical dependence between D_1 and D_2. In a graph, these constraints show up as connection edges between the outputs of each analysis block. The relevant connection edges are highlighted in Fig. 9, where they form a cycle. Cycles of connection edges are the characteristic structure that represents coupling.

It is important to note that the coupling cycles do not contain any driver nodes. Solver, optimizer, or other iterative process is invovled in the coupling definition–

[INSERT GRAPH HERE]

Fig. 10: Graph of a notional multi-fidelity problem

though one will be necessary for a valid PSG. Additionally, a coupling cycle has no inherant start or end. It would be valid to pick any node in the cycle as a starting point, and proceed around the loop untill you get back to your starting point. For the Sellar Problem, picking D_1 as the starting point would yeild a problem as given in Eqn. 1, while picking D_2 would yield the problem as given in Eqn. 2.

Larger problems can contain more complex cycles in their FPG, indicating more complex coupling between analyses. A cycle can contain more than just two analysis blocks. Multiple independent cycles can exist, indicating independent coupling relationships. Cycles can also overlap, where 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. 8.8.1 will demonstrate how building an FPG from an MCG provides an opportunity to identify and potentailly reduce the number of cycles in a graph.

If large amounts of coupling are present it can help convergence rates to look 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 [8,9]. Rodgers work focused on the matrix form of the DSM for ordering optimization. Lu and Martins more recent work worked with a weighted form of the DSM and used an iterative clustering approach to perform a similar task to DEMAID [14].

However, when using a graph representation, the ordering problem can be greatly simplified. For any given coupling cycle, a Gauss-Seidel iteration pattern is automatically given simply folowing the cycle around the graph. If only once cycle is present, then the task reduces down to selecting a starting point in the cycle. When sub-cycles are present each sub-cycle ordering is also given automatically by following the graph, but now a there is freedom at which level to manage the iteration over the cycles. Ordering of many cycles and sub-cycles is still a challenge, but the total search space is reduced by always following the ordering from the graph from a selected starting node.

6.6 Multi–fidelity Problems

A multi–fidelity problem can be characterized by two or more different analyses each calculating the same data. Mapping this type of multi–fidelity situation onto a graph will yield two analysis blocks, each with their own output variable nodes, having outgoing edges to a single variable in a third analysis block that needs the data as input. Figure 10 shows a modified version of the Sellar problem with a new analysis, D_0, representing a low fidelity version of D_1.

When an input variable node has more than one incoming edge, designers must make some kind of decision about how to manage the flow of information in their problem. In Eqn. 8 we defined a collision as the situation where there are more incoming connection edges than the upper indegree limit. So desginers can indicate how to handle conflicting edges in a problem graph by setting the upper indegree limit 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 confliciting edges by introducing some mechanism to manage when each of the different fidelity analyses are run[23,24,25].

7 Building MCG and FPG

7.1 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 connection 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.

7.2 Fundamental Problem Graph

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

(1) $F \subset M$ and $V_{\text{out}} \subset V_F$
(2) $\forall i \in I$, if $F \cap A_i \neq \emptyset$ then $A_i \subset F$
(3) $\forall v \in V_F$ with $t_{\text{node}}(v) =$ 'variable,' $\deg_l^-(v) \leq \deg^-(v) \leq \deg_u^-(v)$
(4) $\forall v \in V_F$ there exists a reverse path $P \subset R_F$ from x to v with $x \in V_{\text{out}}$

Requirement (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 global output must be included. Requirement (2) requires that each analysis block must be entirely present or entirely absent. Requirement (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 a *hole*, and if $\deg^-(v) > \deg_u^-(v)$ the node is a *collision*. Lastly, requirement (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 global output to that node. The reverse graph R_F is obtained from F by simply switching the orientation of every edge in E_F.

7.3 Obtaining the Fundamental Problem Graph

In general, there may be multiple different graphs that satisfy the FPG conditions in Sec. 7.2, though there may be none at all. Here, we describe a process for obtaining an FPG by starting with the MCG and disconnecting connection edges until the FPG conditions are met. Then the problem is reduced to deciding which connection edges to remove.

((describe pruning as a subroutine enforcing step 4 and then step 2))

Subroutine: Pruning Given a digraph $G_1 = (V_1, E_1)$, let the subroutine denoted as P_{prune} operate on G_1 to produce a digraph $G_2 = (V_2, E_2)$ that satisfies requirements (2) and (4), and write $G_2 = P_{\text{prune}}(G_1)$. This is accomplished by first creating the reverse graph of G_1, $R = (V_1, E_R)$. Then adding a new node x to V_R and adding edges directed from this node to each of the global outputs, i.e.,

$$\forall y \in V_{\text{out}}, (x, y) \in E_R \tag{9}$$

The set of nodes that may be reached from the global outputs is constructed as

$$U = \{n \in V_1 \mid \exists P \text{ a path from } x \text{ to } n,\ P \subset R\}. \tag{10}$$

The list of analysis blocks with at least one node that may be reach is constructed as

$$I_1 = \{i \in I \mid V_{A_i} \cap U \neq \emptyset\} \tag{11}$$

The set of nodes to remove is

$$N = \{v \in V_1 \mid v \in V_{A_i} \text{ for } i \in I_1,\ \text{or } v \in V_{\text{in}} \setminus U\}, \tag{12}$$

which means that N is composed of unused analysis block nodes and unused global input nodes. Then the new set of nodes is created as

$$V_2 = V_1 \setminus N, \tag{13}$$

and edges involving the removed nodes are also deleted

$$E_2 = E_1 \setminus \{(a, b) \mid a \in N \text{ or } b \in N\}. \tag{14}$$

The set of connection edges can be obtained as

$$C_2 = \{(x, y) \in V_2 \mid \ \sim (x \in V_{A_i} \text{ and } y \in V_{A_i} \text{ for some } i \in I_1)\}, \tag{15}$$

which means $C_2 \cap V_{A_i} = \emptyset\ \forall i \in I_1$, and $E_2 = C_2 \cup \left(\bigcup_{i \in I_1} E_{A_i}\right)$.

Step 1: Holes The first step is to detect holes and disconnect the connection edges following them. These connection edges are removed because they represent variables which cannot be determined because the analysis function does not have adequate inputs.

To start, the an initial graph is created as a pruned copy of the MCG

$$F_0 = P_{\text{prune}}(M), \tag{16}$$

where $F_0 = (V_{F,0}, E_{F,0})$, and $C_{F,0}$ is also obtained as described. The set of variable nodes which are holes is created as

$$H = \{v \in V_{F,0} \mid t_{\text{node}}(v) = \text{'variable' and } \deg^-(v) < \deg_l^-(v)\}, \tag{17}$$

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 edges preceding or succeeding the analysis block:

$$C_{F,1} = C_{F,0} \setminus \{(x,y) \in C_{F,0} \mid x \in V_{A_i} \text{ or } y \in V_{A_i}, \text{ and } H \cap V_{A_i} \neq \emptyset\}, \tag{18}$$

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. If this step every finds a global output to be a hole, meaning $V_{\text{out}} \cap H \neq \emptyset$, then the algorithm stops because an FPG cannot be obtained. Otherwise, it is guaranteed that an FPG can be obtained.

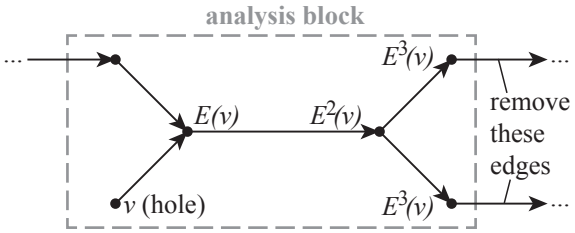


Fig. 11: Example variable node indicating a hole.

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

$$S_{\text{nodes}} = \{v \in V_M \mid t_{\text{node}}(v) = \text{'variable' and } \deg^-(v) > \deg_u^-(v)\}. \tag{19}$$

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

$$S_{\text{edges}} = \left\{ \{(x,y) \in E_M\} \;\middle|\; y \in S_{\text{nodes}} \right\} \tag{20}$$

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

$$B_j = \left\{ e_k \in S_{\text{edges},j} \mid k \in \{1, 2, \ldots, K\} \text{ with } K \leq \deg_u^-(v_j) \right\},\ j \in J, \tag{21}$$

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

$$C_{F,2} = \{e \in C_{F,1} \mid e \in B_j \text{ for some } j \in J\}, \tag{22}$$

$$E_{F,2} = E_M \setminus (C_M \setminus C_{F,2}) \tag{23}$$

and

$$F_2 = (V_M, E_{F,2}) \tag{24}$$

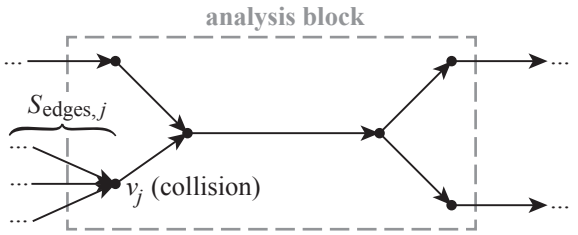


Fig. 12: Example variable node indicating a collision.

Step 3: Finalize The third and final step is to prune the graph one last time

$$F = P_{\text{prune}}(F_2), \tag{25}$$

and $F = (V_F, E_F)$. The indices of the used analysis blocks can then be found as

$$I_F = \{i \in I \mid A_i \cap F \neq \emptyset\} \tag{26}$$

This process will always provide an FPG if one exists. If an FPG does not exist, this process will reveal the limiting factors preventing a valid formulation.

8 Example Problem

This section presents an example problem to demonstrate the process of obtaining the FPG and the advantages of doing so. The algorithm from Sec. 7.7.3 was implemented in the NetworkX package of the Python programming language. Any language with the ability to manipulate a graph should be applicable. NetworkX was chosen due to the built–in features, such as cycle detection.

The example 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

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, engine sizing, and weight estimation
FLOPSb	mission performance only

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. This sort of representation is useful for "supercodes" that care capable of being used in different ways and with different sets of inputs and outputs.

In this example, the MCG is formed using a consistant variable naming convention and then connecting every variable 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 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.

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

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, which is directed into another model node, which is then directed into variable nodes corresponding to each output. A sample analysis block is shown for analysis code FLOPSb in Fig. 13.

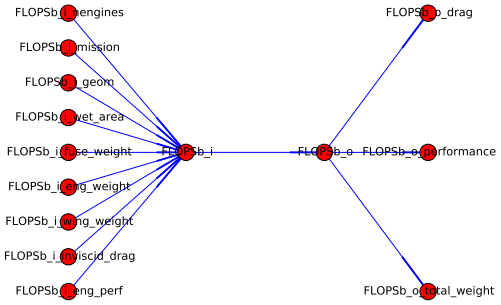


Fig. 13: Sample analysis block for analysis code FLOPSb using Table 2

2. Variable ((change)) 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 as global inputs.
4. Explicit edges are created from 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 model node (local input or local output).

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

8.1 Obtaining an FPG

The process of obtaining an FPG from the given MCG is expected to be an iterative process in which the user applies the algorithm to determine whether or not an FPG can be obtained, and then modifies the indegree limits or the MCG to obtain a satisfactory FPG. For example, after determining the location of a hole, the user can change the indegree limit to make the node a design variable, or the user could change the MCG by adding a new analysis code such that the hole is resolved.

As presented in Sec. 7.7.3, the first step to obtaining an FPG is to detect holes. To start, we set the lower indegree limit for every node to be unity to find any potential holes. 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, as shown in Fig. 15. In this figure, the green nodes

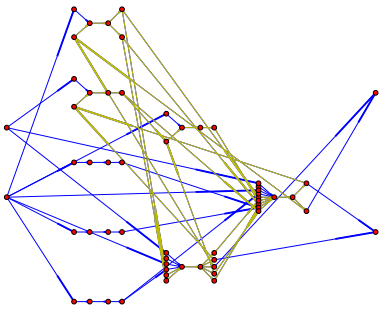


Fig. 14: Maximal connectivity graph for the subsonic transport example problem

indicate analysis blocks with a hole, and the cyan nodes indicate analysis blocks that became unused after the invalid analyses were removed ((connections were removed, color edges too)).

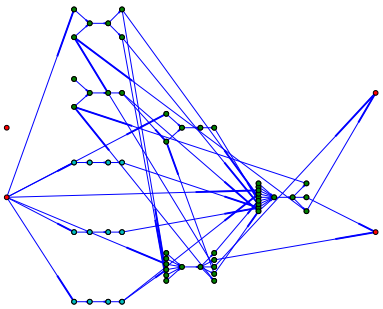


Fig. 15: The MCG with one global variable missing. The resulting holes are shown in green and unused analyses shown in cyan.

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. 7.7.3 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 [26] in the Python package NetworkX. The FPG with the fewest cycles is shown in Fig. 16, which indicates one cycle shown in yellow. A similar alternative would be to minimize the number of analysis blocks involved in cycles, counting multiplicity.

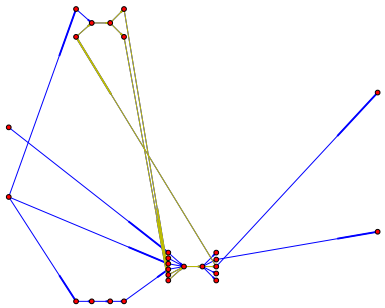


Fig. 16: FPG with the fewest number of cycles.

Alternatively, it may be desirable to resolving collisions by including the higher fidelity codes, such as NPSS in this case. 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. 7.7.3 (the resolution of collisions) is now conducted by selecting

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

the edges directed from the analysis blocks with the highest rank for each collision. The resulting FPG is shown in Fig. 17, 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, meaning that it is sought to use multiple analysis codes to calculate the same variable. This is done simply by setting the upper

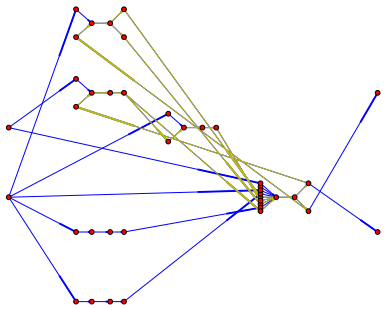


Fig. 17: FPG obtained using the ranking system.

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		

indegree limit for this 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.

9 Applications

10 Conclusions

References

1. Gray, J., Moore, K. T., Hearn, T. A., and Naylor, B. A., "A Standard Platform for Testing and Comparison of MDAO Architectures," *8th AIAA Multidisciplinary Design Optimization Specialist Conference (MDO)*, Honolulu, Hawaii, 2012, pp. 1–26.
2. Alexandrov, N. and Lewis, R., "Reconfigurability in MDO Problem Synthesis, Part 1 and Part 2," Tech. rep., Papers AIAA-2004-4307 and AIAA-2004-4308, 2004.
3. 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.
4. Tosserams, S., Hofkamp, A., Etman, L., and Rooda, J., "A specification language for problem partitioning in decomposition-based design optimization," *Structural and Multidisciplinary Optimization*, Vol. 42, No. 5, 2010, pp. 707–723.
5. Martins, J. R. R. A. and Hwang, J. T., "Review and Unification of Methods for Computing Derivatives of Multidisciplinary Systems," *In Proceedings of the 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, HI, April 2012, AIAA 2012-1589.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. Martins, J. R. R. A. and Lambe, A. B., "Multidisciplinary Design Optimization: Survey of Architectures," *AIAA Journal*, 2012, pp. 1–46.
14. 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.
15. Diestel, R., *Graph Theory*, Springer, 2010.
16. Smith, D., Eggen, M., and Andre, R. S., *A Transition to Advanced Mathematics*, Brooks/Cole Pub Co, 2006.
17. Bruan, R., *Collaborative Optimization: An Architecture for Large-Scale Distributed Design*, Ph.D. thesis, Stanford University, June 1996.
18. 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.
19. 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.
20. Sobieszczanski-Sobieski, J. and Haftka, R., "Multidisciplinary aerospace design optimization: survey of recent developments," *Structural and Multidisciplinary Optimization*, Vol. 14, No. 1, 1997, pp. 1–23.

21. Perez, R. E., Liu, H. H. T., and Behdinan, K., "Evaluation of Multidisciplinary Optimization Approaches for Aircraft Conceptual Design," *Proceedings of the 10th AIAA/ISSMO Muldisiciplinary Analysis and Optimization Conference*, No. September, 2004, pp. 1–11.
22. 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.
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.