# An Application of Graph Theory to MDAO Problem Formulation

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

**blah blah blah**

## Nomenclature

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

## I. Introduction

As the size and complexity of engineering systems grow, the time and expense for setting up analysis models grow with them. Multidisciplinary Design Analysis and Optimization (MDAO) frameworks such as OpenMDAO[1] and ModelCenter have enabled a new level of analysis tool integration and paved the way for models with more analyses and increasing numbers of interdisciplinary couplings. Such complex models present a distinct challenge for proper implementation of any given solution strategy.

We propose the use of graph-based problem formulation that generally describes a given problem, supports algorithmic manipulation to find beneficial solutions strategies, and enables automated implementation of those strategies inside an MDAO framework. In order to serve that purpose, the graph must include the following information:

- Discipline Analyses
- Discipline State Variables and Residuals
- Design Variables
- Constraints
- Objective or Objectives
- Coupling Constraints
- Parameters

Notably absent from the preceding list are any kind of solvers, optimizers, or other iterative solution finding tools. At its most basic, a problem formulation includes only information about what is being sought after in a given problem or what the goals of a given problem are. It need not contain any information about solution paths or strategies to reach those goals.

We define a problem specification striped down to its most essential parts to be the Fundamental Problem Formulation (FPF). By definition, the FPF for any given problem will be constant regardless of which MDAO framework, optimization architecture, optimization algorithm, or iterative solver is used to solve the problem.

The graph-based syntax used in this work is based on the Design Structure Matrix (DSM). This provides several key features that make it useful for working with large-scale MDAO problems. It provides a rigid

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

structure that can be easily manipulated with a wide range of well-established graph-theory algorithms for the purposes of problem decomposition. It also provides a means for testing a given problem formulation to check if it is the FPF, and if not, to reduce it to the FPF. Lastly, it lends itself well to interacting with MDAO frameworks.

## II. Specific vs Fundamental Problem Formulation

A simple, notional problem is given as follows:

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

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

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

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

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

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

## III. Using DSM for Problem Formulation

As shown above, the mathematical language for specifying problem formulations is very general and can be used both for fundamental and specific problem formulations. Tedford and Martins used the above

mathematical syntax to specify the FPF for a set of test problems and also to describe specific formulations for solving them with a number of optimization architectures.[3] Their work demonstrates clearly how multiple specific problem formulations can all relate back to a common FPF.

The challenge with using this traditional mathematical syntax is that it is not easily manipulated or analyzed. A number of matrix-based methods have been used successfully to translate the mathematical syntax into a more useful computational form. Steward's Design Structure Matrix (DSM) is a square adjacency matrix which captures the relationship between analysis tools where off diagonal elements of the matrix indicate coupling.[4] Since a DSM describes a square adjacency matrix, it can be represented in an equivalent directed graph where nodes represent analysis tools and edges represent information dependence between those tools. The ordering of elements in a DSM can be used to indicate execution order. For more complex problems, choosing the proper order to run analysis tools is a non-trival task. Rogers et. al developed DeMAID to manipulate a DSM to find an ordering for analysis tools that reduces the cost of solving highly coupled systems.[5] This re-ordering yields multiple specific problem formulations which all solve the same FPF. In other words, manipulation of the DSM does not fundamentally alter the problem formulation, which makes DSM an excellent foundation for specifying the FPF itself. Traditional DSM only captures information about data dependency between analyses. Objective and constraint information is missing from the description of the problem.

An alternative matrix-based syntax, called a Functional Dependency Table (FDT), was proposed by Wagner and Papalambros. FDT represents the relationship between functions, including objectives and constraints, and specific variables that affect them.[6] Similar to DSM, FDT also describes an adjacency matrix of a graph. Unlike the DSM graph, however, the graph is undirected and nodes can represent analysis tools, objectives, or constraints. Edges between nodes represent a dependence on the same variable. Michelena and Papalambros made use of the FDT to solve a graph partitioning problem that yielded more efficient optimization problem decompositions.[7] While FDT succeeds at capturing the information about objectives and constraints, it neglects the data dependency captured by DSM. For instance, although we know from the FPF in Eq. 3 that the objective, $f$, is dependent on the output of analysis $A$, you could not determine that from the FDT in Fig. 1 alone. This make FDT applicable for partitioning decisions, but not for ordering problems.

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

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

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

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

By taking XDSM, removing the requirement for solver/optimizer nodes, and adding nodes for each

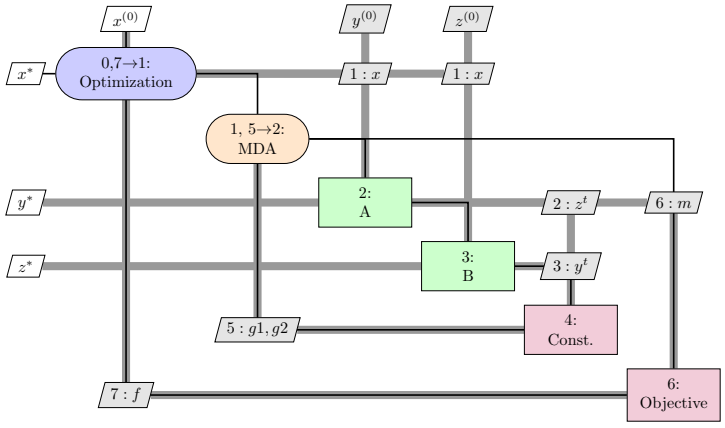


**Figure 2: XDSM for Eq. 1, with a gauss-siedel iteration and MDF solution architecture.**

variable, we are able to fully describe the problem formulation for a general MDAO problem. In the following sections we describe the necessary syntax for specifying a problem formulation within a DSM and show how to use the resulting graph to find the FPF from any given graph. For completeness we also present algorithms for converting between the general DSM and the more specific formulations discussed above.

## IV. Graph-based Problem Formulation Syntax

### A. Graph Theory Basics

We begin by discussing the general syntax of graph theory and then expand upon this syntax to represent the data flow of an MDAO problem. The present notation is adapted from Diestel.[10] A *graph* is a pair $G = (V, E)$ of sets such that $E \subseteq V \times V$, which means that the elements of $E$ are 2–element subsets of $V$. The set $V$ contains the *vertices* or *nodes* and the set $E$ contains the *edges*. For a *directed graph** we construct $E$ as a set of ordered pairs instead of a set of sets. Each ordered pair represents an edge starting at the node indicated by the first entry and directed to the node indicated by the second entry. As an example,

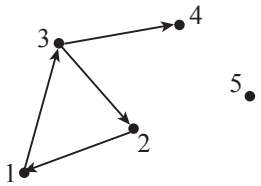


**Figure 3: Example directed graph.**

for the directed graph shown in Fig. 3 we have

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

The edges directed out of $v \in V$ are given by $E(v)$ and the edges directed into $v$ are given by $E^{-1}(v)$. The *indegree* 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 $\deg^+(v)$. Edge $e = (x, y)$ may be referred to simply as $xy$ and we say $y = E(x)$. Also, $E(E(v))$ is denoted as $E^2(v)$, and likewise for additonal levels.

A *path* is a nonempty graph $P = (V, E)$ with $V = \{x_0, x_1, \ldots, x_k\}$ and $E = \{x_0x_1, x_1x_2, \ldots, x_{k-1}x_k\}$, where $x_i$ are all distinct. A graph $G$ is *connected* if any two of its vertices are linked by a path in $G$. A *tree* is a connected and acyclic graph.

### B. Problem Formulation Concepts to Represent

The MDAO problem formulation concepts we wish to represent are:

**global input** A global input is a variable that is taken as given and is used by multiple analyses.

**design variable**

**function call** A function call requires a set of inputs to be fulfilled and has certain properties, such as run time, which are independent of the number of outputs actually being used.

**passing of distinct variables** Instead of connecting analyses, the individual variables as are connected

**objective & constraint** These are variables that must be produced by the problem formulation for it to be valid

**collision** A collision occurs wh  n input has multiple sources

**hole** A hole occurs when an input has no sources

**coupling** Couple is the mutual dependence between a set of analysis

**multi–fidelity** It may be desirable for the same variable to be calculated by separate analyses and to use both results.

### C. Graph Theory Representation of a Problem Formulation

In order to represent the listed problem formulation concepts with a graph, we assign properties and rules to nodes and edges. The first property assigned to nodes and edges is the *type*. The possible node types are:

**variable node** The variable node represents passing of data.

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

The possible edge types are:

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

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

To keep track of which nodes and edges are which type, let $T_{\text{node}}$ and $T_{\text{edge}}$ be sets containing the possible node types and edge types, respectively, and then define mappings $t_{\text{node}} : V \to T_{\text{node}}$ and $t_{\text{edge}} : E \to T_{\text{edge}}$ to assign a type to each node and edge. For these types of nodes and edges we impose the following rules:

- A model node can only have one edge directed to or from another model node.
- A model node can only have fixed edges directed in or out.
- A model node must have at least one edge directed in and at least one edge directed out.
- An objective or constraint is indicated by a variable node with no outgoing edges and with the incoming edges being directed from only other variable nodes.
- A global input is a variable node with no edges directed in and with at least two edges directed out to different variable nodes.

Additional problem formulation concepts represented by the edges are described in Table 1

Finally, any tree containing only fixed edges and at least one model node is an *analysis block*, which is used to represent a contributing analysis. An example analysis block is shown in Fig. 4 to demonstrate the basic structure. This structure is derived from the need to represent each input and output individually while representing the actual analysis with a single node or edge. The variable nodes each represent a single input or output of the analysis block. Each of these nodes is connected to a single model node via a fixed edge, which represents the gathering of the inputs to begin the analysis or the disseminating of outputs after the analysis. Finally, free edges are used to connect the variable nodes of the analysis block to other variables nodes outside of the analysis block. In this way, the analysis block graph is the fundamental building block of the MDAO problem data flow.

**Table 1: Problem formulation concept represented by an edge**

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

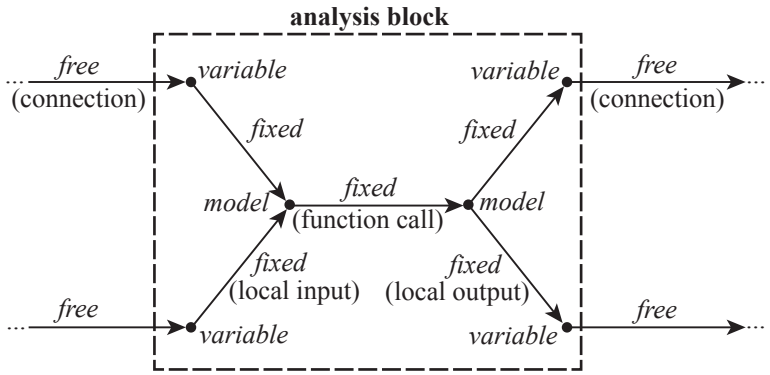


**Figure 4: Example directed graph. The each node type and edge type is labeled, with nodes being label in italics.**

### D. Maximal Connectivity Graph

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

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

The maximal connectivity graph contains all possible connections that may exist between the given analysis blocks.

### E. Fundamental Problem Formulation Graph

Before defining the the fundamental problem formulation graph, let us first define the *upper degree limit*

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

and the *lower degree limit*

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

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

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

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

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

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

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

### F. Obtaining the Fundamental Problem Formulation Graph

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

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

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

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

Then the updated set of edges is

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

This step is demonstrated by Fig. 5.

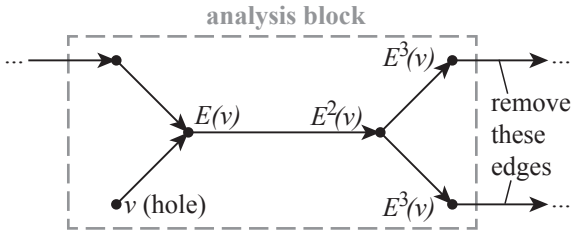


**Figure 5: Example variable node indicating a hole.**

### G. Classification of the FPF

### H. Solution Graph Syntax

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

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

## V. Example Problem

## VI. Applications

## VII. Conclusions

## References


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

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

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

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

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

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

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

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

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

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