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[?] 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.[?]

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.[?] 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.[?] 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.[?] 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.[?] 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.[?] 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.[?] 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.[?]

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

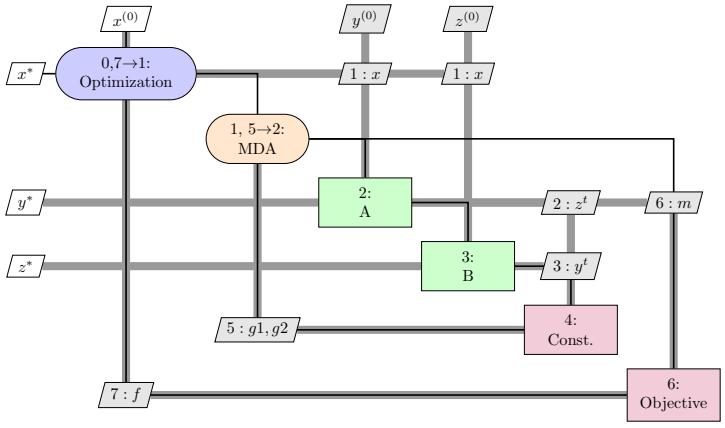

Figure 2: XDSM for Eq. 1, with a gauss-siedel iteration and MDF solution architecture.

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

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. 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.[?] 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. Therefore, E is a mapping from V to V. As an example, for the directed graph shown in Fig. 3 we have

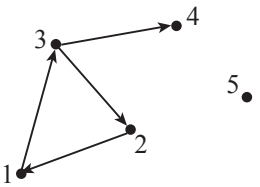

Figure 3: Example directed graph.

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

We now assign additional properties to the nodes and edges so that they may represent the data flow of an MDAO problem. The first property assigned to nodes and edges is the *type*. The possible node types are:

- global input
- global output
- local input
- local output
- analysis start
- analysis finish

and the possible edge types are

- connection
- in
- analysis
- out
- ((consider eliminating 'in' and 'out' and using connection))

Then, considering sets T_{node} and T_{edge} to be sets containing the possible node types and edge types, respectively, we use 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. Nodes and edges are named according to their type (e.g. connection edge). Additional properties will be discussed is subsequent sections.

Next, we introduce the graph A which represents a single analysis block broken down into its individual inputs and outputs. An example graph 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 'local input' nodes each represent a single input of the analysis block. Each of these nodes is connected to a single 'analysis start' node via an 'in' edge. Next the analysis start node is directed to the 'analysis finish' node via the 'analysis edge', which represents the actual analysis. Finally, the analysis finish node is directed to 'local output' nodes via individual 'out' edges. 'Connection' edges connect

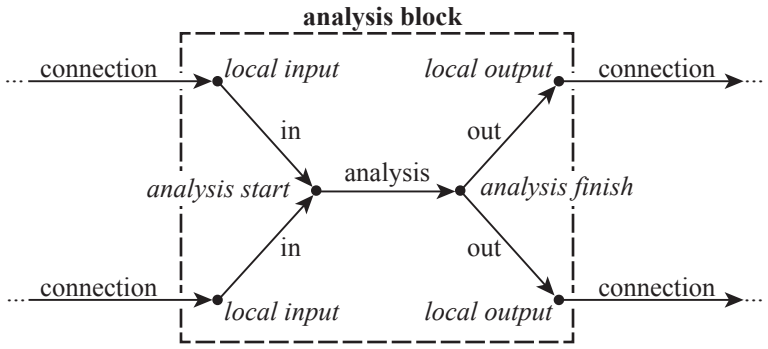

Figure 4: Example directed graph. The each node type and edge type is labeled, with nodes being label in italics.

local output nodes from one analysis block to either the local input nodes of other analysis blocks or global output node. In this way, the analysis block graph is the fundamental building block of the MDAO problem data flow.

A. Maximal Connectivity Graph

Now we define the node property *variable name*, which represents the unique identity of the variables passed between analysis blocks as represented by nodes of type local input, local output, global input, and global output. Let N be the set containing each unique variable name (no multiplicity), and let $n : V \to N$ be a mapping that assigns a variable name to a node.

Next, Consider an MDAO problem defined by a set of codes, global inputs, and global outputs, as represented by analysis blocks $A_i = (V_{A_i}, E_{A_i}), i = 1, \ldots, m$, the set of nodes $\mathcal{I}$, and the set of nodes $\mathcal{O}$, respectively. Then we may construct the *maximal connectivity graph* $M = (V_M, E_M)$ as

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