A Graph Theory Based Approach to MDAO Problem Formulation

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

blah blah blah

Nomenclature

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

I. Introduction

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 since configuring a model with the larger nubmer of analyses, associated inputs and outputs can be very difficult. Even for models with 10's of analyses, there could be hundreds of variables that need to be managed. It is not hard to imagine that the task of combining all the analyses into a consistent system model capable of solving a relevant engineering design problem could easily become much more costly than createing the discipline analyses themselves. Thus, for a large system as 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.

When building a model, coupling represents the reciprical flow of information between two analyses. We propose the use of graph-based syntax to describe the flow of information from inputs, through analyses, to outputs. If you allow for any free inputs to become design variables and some set of the outputs to become design objectives and constraints, then the graph represents the problem formulation for a given design effort. The graph-based syntax presented has application in all phases of the design process. It can be used early on, to help construct a valid problem formulation from the set of available analysis tools. It also provides a consistent structure to describe problem formulation that provides a foundation to apply algorithmic methods for selecting and then implementing effective solution strategies inside an MDAO framework.

We've already identifed that a problem formulation graph should include design variables, analysis tools and their associated inputs and outputs, and objectives and constraints. Notably absent from that list are any kind of solvers, optimizers, or other iterative solution finding tools. At its most fundamental state, a problem formulation includes only information about what is being sought after in a design or what the goals of a given problem are. It need not contain any information about solution paths or strategies to reach those goals. While solution information can be represented with the graph syntax presented here, a unique aspect of this syntax is that such information always remains seperable from the underlying problem formulation. In other words, a given a graph representing a specific solution strategy for a specific problem formulation, it is possible to remove the solution information from the graph and reclaim the more basic problem formulation.

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

II. Specific vs Fundamental Problem Formulation

The Fundamental Problem Formulation (FPF) for any given problem will be constant regardless of which MDAO framework, optimization algorithm, iterative solver, or solution strategy is used to solve the problem. For example, consider the following notional problem:

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

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

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

Equation 2 differs only slightly from Eq. 1. A is now dependent on the output of B, and z will be iterated on to convergence with z^t. Now the problem can be solved by running B first and then A. Since the formulations in Eqs. 1 and 2 both describe the same problem and neither can be the FPF. They are both specific versions of the same 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. Existing Graph–Based Problem Formulation 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.[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 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.[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 is done through row operations on the DSM matrix and yields multiple specific problem formulations which all solve the same FPF. In other words, manipulation of the DSM does not fundamentally alter the problem formulation, which makes DSM an excellent foundation for specifying the FPF itself.

Despite it's attractive properties, a DSM by itself is insufficient to describe complete problem formulations. Traditional DSM only captures information about data dependency between analyses. Objective and constraint information is missing from the description of the problem. An alternative matrix-based syntax, called a Functional Dependency Table (FDT), was proposed by Wagner and Papalambros. FDT represents the relationship between functions, including objectives and constraints, and specific variables that affect them.[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 can not capture the coupled data dependency that DSM captures. For instance, we know from the FPF in Eq. 3 that the objective, f, is dependent on the output, m, of analysis A. You could not determine that from the FDT in Fig. 1 alone. This missing information means that, while FDT is very useful for partitioning problems, it is also not sufficient to contain a complete problem formulation.

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

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

Lamb and Martins 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.[8] XDSM retains the square adjacency matrix form from DSM, but by adding in the new 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.

Of the three methods, XDSM comes the closest to fully describing a problem formulation. Thought it does not contain specific variable information, this could be easily added back in simply by putting the

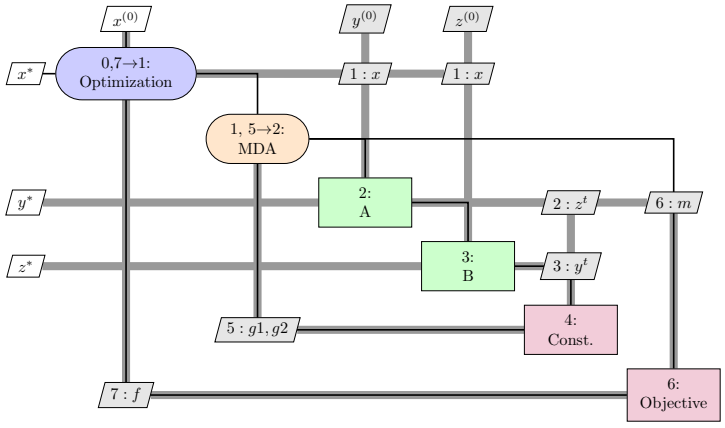


Figure 2: XDSM for Eq. 1, with a gauss-siedel iteration and MDF solution architecture.

variables back into the graph and drawing the apprpriat connections between them and the relevant analysis nodes. However, it's dependence on solution strategy specifics to represent couplings can not be removed without stripping it of the useful information contained in an FDT. In the following sections we describe a new graph syntax that retains the features of XDSM, without the dependence on solution strategy (although it does allow for that information to be present). This new syntax allows for the compelete specification of an FPF.

IV. Requirements for New Graph Syntax

The ultimate goal of the new graph–based syntax presented here is to be able to fully describe the fundamental problem formulation for a complex design task, while still being able to accomodate the more specific case when a solution strategy is applied to the problem formulation. In order to achieve that goal the graph needs to accomodate a number of constructs from MDAO problems:

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

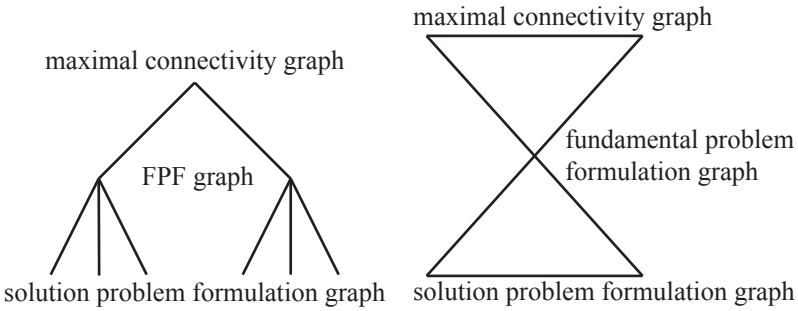
Beyond those basic constructs, there are also three phases of a design process that all need to be representable with the new syntax. Firstly there is the initial problem definition phase where the specific analysis tools and design goals are identified. At the end of this phase, a single formal problem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

- Maximal Connectivity
- Fundamental Problem Formulation
- Specific Solution Formulation

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

specific problem formulation may be represented by including additional edges and node types 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 FPFs 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 FPFs that could be reached untill only one is remaining. Then, a single FPF, different SPFs may be obtained by implementing different solution strategies. The hourglass shape in Fig. 3(b) illustrates how the MCG gets reduced to a single FPF, then multiple possible SPFs exist to solve the problem. In other words the FPF is obtained from the MCG by removing nodes and edges, and the SPF is obtained from the FPF by adding nodes and edges.



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

Figure 3: The relationship between the MCG, FPF, and SPF.

In the following sections we present some graph theory basics, then demonstrate how to build the relevant MDAO constructs from the given grah primatives. Next we show how to construct the MCG, FPF, and SPF graphs from those constructs and demonstrate methods to transition between them.

A. Graph Theory Basics

The notation used in this work 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. Edge $e = (x, y)$ may be referred to simply as xy and we say $y = E(x)$.

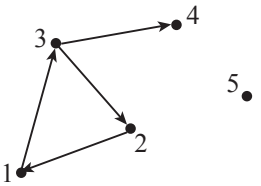


Figure 4: Example directed graph.

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

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

Let I be a nonempty set such that for each $i \in I$ there is a corresponding set A_i. The family $\{A_i \mid i \in I\}$ is an indexed family of sets with index i and indexing set I, and we can say $\mathcal{A} = \{A_i \mid i \in I\}$.? Then the union over this family of sets is

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

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

V. MDAO Graph Primatives

There are four node types:

variable: represents scalar or array data

model: responsible for mapping inputs to outputs

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

driver: constrol structures capable of managing iteration

Only the variable, model, and expression nodes are allowed to be used in an MCG or FPF. All four nodes are required for any SPF. In addition to the four node types, there are three edge types:

fixed edge: Fundamental, non-changeable property of a problem, such as connections between related variable nodes. These edges can never be removed from a graph.

explicit edge: Explicit exchange of information between two nodes. No node can have more than a single incomming explicit edge. These edges are free and can be added or removed from the graph.

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

A number of restrictions are placed on the model node with respect to the number and type of edges that can be connected to it:

1. A model node can only have one edge directed to or from another model node.
2. A model node can only have fixed 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.

A. Analysis Blocks and Connections

Analysis tools take in a number input variables, then perform some work to calculate the values for their respective outputs. In an MDAO graph, this process is represented by an group of nodes and edges called an *analysis block*, shown in Fig. 5. Within an analysis block each variable node represents a single input or output and is connected to a single model node via a fixed edge. Note that in Fig. 5, the analysis block contains two model nodes, with a single edge connecting them. This part of the graph represents the necessary calculations to map given inputs to proper output values. Although analysis blocks are comprised of multiple nodes and edges, since all the edges within them are fixed, they become fixed structures within a graph.

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

MDAO problems require that information be passed between sets of analyses. When information from the output of one analysis block is passed, or connected, to the input of another analysis block, a new free edge is added connecting the two variable nodes involved in the exchange. The edge is free because, unlike the edges within an analysis block, it could be added or removed depending on the demands of the specific problem.

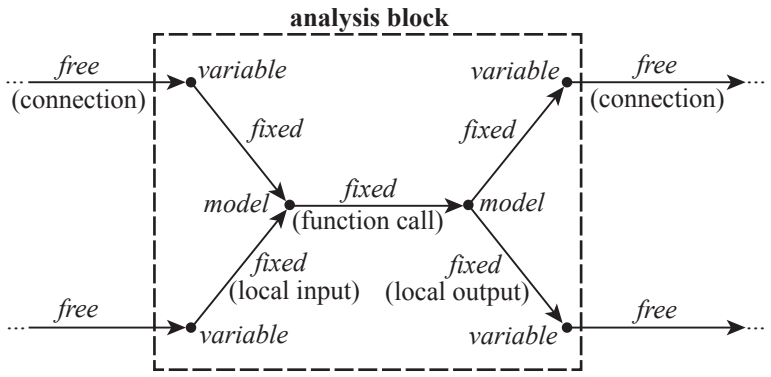


Figure 5: Example analysis block. The each node type and edge type is labeled in italics and annotated parenthetically.

B. Design Variables, Objectives, and Constraints

Design variables, objectives, and constraints all require external decisions to be made about a given system to identify them. Consider a notional problem given by the graph in Fig. 6 There are three input variables which, could potentially become design variables: w, x, and y. Because the output, x, of A is connected to the input, x, of B x can't actually be a design variable. The inputs w and y have no incomming edges and could each be selected as design variables.

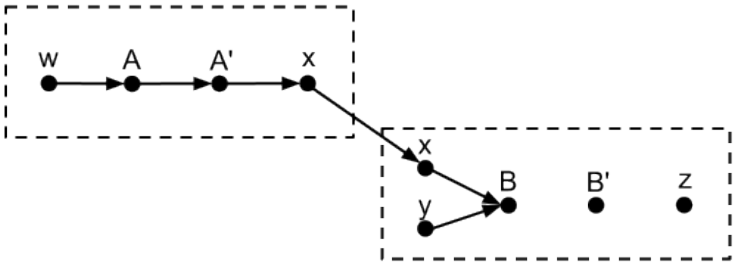


Figure 6: Simple notional problem with two potential design variables

More formally, all inputs that have no incomming edges

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

For the proposed syntax objectives and constraints are represented through the use of expression nodes. These nodes contain the actual mathematical expression used to define the objective or constraint value. You can distinguish between an objective or constraint node via the presence of a simple expression (objective) vs an inequality/equality (constraint). Fig. [insert new ref] shows the notional problem graph augmented with objective and constraint values.

[[INSERT ANOTHER GRAPH HERE]]

C. Local vs Global Properties

- what can be local or global? Variables, objectives, constraints,
- What does it really mean to be local vs global?

- What does the graph structure of local vs global look like?
- local is a frame of reference... a variable could be global when looking at two analyses, but local to the two when looking at the three

D. Coupling Between Analyses

E. Multi-fidelity Analyses

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. The *indegree* of a node is the number of edges directed in and is denoted as $\deg^-(v)$, and the *outdegree* is the number of edges directed out and it is denoted as $\deg^+(v)$. We also define the *upper indegree limit*

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

and the *lower indegree limit*

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

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.

To keep track of which nodes and edges are which type, let T_{node} and T_{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. Then, for example, for a variable node v we have $t_{\text{node}}(v) =$ 'variable'.

VI. Building MCG, FPF, and SPF Graphs

A. Maximal Connectivity Graph

To construct the maximal connectivity graph, we assume that a set of codes, global inputs, and objectives and constraints (collectively called global outputs). The codes are represented by analysis blocks $A_i = (V_{A_i}, E_{A_i}),\ i \in \{1, 2, \ldots, m\} = I$, the global inputs are represented 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 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

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

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

B. Fundamental Problem Formulation Graph

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

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

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

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

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

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

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

The set I_F is an index set containing the indices of the analysis blocks in F and it may only include the analysis blocks A_i that meet requirements (2) and (6). Requirement (2) stipulates that the analysis blocks in F must each have at least one local output that is beging used. Requirement (3) stipulates that only the global inputs that are being used should be included. Lastly, requirement (6) stipulates that the number of edges directed into the variable nodes must be within the lower and upper indegree limits; if $\deg^-(v) < \deg_l^-(v)$ the node is called a *hole*, and if $\deg^-(v) > \deg_u^-(v)$ the node is called a *collision*.

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

C. 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

$$H = \{v \in V \mid t_{\text{node}}(v) = \text{'variable' and } \deg^-(v) < \deg_l^-(v)\}, \tag{7}$$

   which is the set of variable nodes with few incoming edges than are allowed by the lower indegree limit. Then the updated set of edges is created by removing the first set of free edges following the hole:

$$C_{F,1} = C_{F,0} \setminus \{(x, y) \in E \mid x = E^3(v) \text{ for } v \in H\}, \tag{8}$$

   and this step is demonstrated by Fig. 7. Because removing these edges can create new holes, this step must be repeated until no more holes are found.

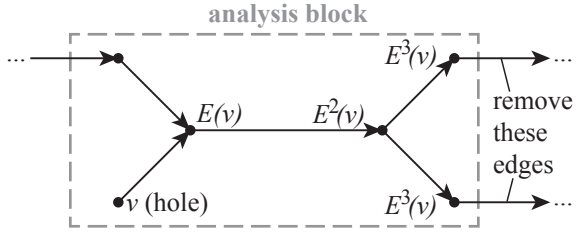


Figure 7: Example variable node indicating a hole.

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

$$S_{\text{nodes}} = \{v \in V \mid t_{\text{node}}(v) = \text{'variable' and } \deg^-(v) > \deg_u^-(v)\}. \tag{9}$$

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

$$S_{\text{edges}} = \left\{ \{(x,y) \in E\} \mid y \in S_{\text{nodes}} \right\} \tag{10}$$

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 examle collision is shown in Fig. 8 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}. Then we may construct sets of edges as

$$B_j = \left\{ e_k \in S_{\text{edges},j} \mid k \in \{1, 2, \ldots, K\} \text{ with } K \leq \deg_u^-(v_j) \right\},\ j \in J, \tag{11}$$

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 indegree 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

$$C_{F,2} = \{e \in C_{F,1} \mid e \in B_j \text{ for some } j \in J\}. \tag{12}$$

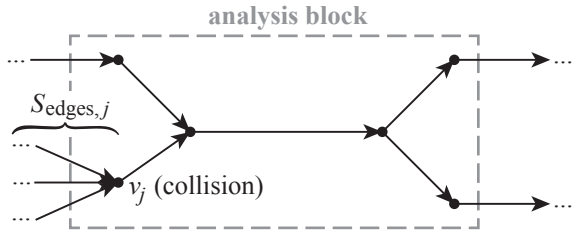


Figure 8: Example variable node indicating a collision.

3. The third and final step is to remove any unused nodes and edges. The set of unused variable nodes representing local outputs is

$$V_{\text{empty}} = \{v \in V_M \mid t_{\text{node}}(E^{-1}(v)) = \text{'model' and } \deg^+(v) = 0\}, \tag{13}$$

and so then the indexing set for only the analysis blocks with at least one used local output may be constructed as

$$I_F = \{i \in I \mid \nexists v \in V_{A_i} \text{ such that } v \in V_{\text{empty}}\}. \tag{14}$$

The set of used global inputs is constructed as

$$V_{\text{in},F} = \{v \in V_{\text{in}} \mid |\{(x,y) \in E(v) \mid y \in V_{A_i} \text{ for } i \in I_F\}| > 1\}, \tag{15}$$

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

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

$$V_F = V_{\text{in},F} \cup V_{\text{out}} \cup \left(\bigcup_{i \in I_F} V_{A_i} \right), \tag{16}$$

$$C_F = \{(x,y) \in C_{F,2} \mid x, y \in V_F\}, \tag{17}$$

$$F = (V_F, C_F). \tag{18}$$

D. 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?)

VII. Example Problem

VIII. Applications

IX. 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.