Automatic Evaluation of Multidisciplinary Derivatives Using a Graph-Based Problem Formulation in OpenMDAO

Justin Gray,* Tristan A. Hearn,† Kenneth T. Moore,‡
NASA Glenn Research Center, Cleveland, OH
John T. Hwang,§ Joaquim R. R. A. Martins¶
University of Michigan, Ann Arbor, MI
Andrew Ning ‖
National Renewable Energy Laboratory, Golden, CO

The optimization of multidisciplinary systems with respect to large numbers of design variables is best pursued using a gradient-based optimization together with a method that efficiently evaluates coupled derivatives, such as the coupled adjoint method. However, implementing such a method in a problem with more than a few disciplines is time consuming and error prone. To address this issue, we develop an automated procedure for assembling and solving the coupled derivative equations that takes into account the disciplinary couplings using the interdisciplinary dependency graph of the problem. The coupled derivatives can be computed completely analytically, if analytic derivatives are available for all disciplines; otherwise, the coupled derivatives are computed semi-analytically. The procedure determines the disciplinary analyses execution order, detects iterative cycles, and uses this information to converge the coupled analysis, and evaluate the coupled derivatives as efficiently as possible by exploiting sparsity. Sparsity can occur at two levels within a multidisciplinary problem: between disciplines, when certain analyses do not affect all outputs, and within a discipline when, the Jacobian of that discipline is sparse. The numerical procedures are implemented in NASA's OpenMDAO framework, providing a flexible API for declaring discipline-level derivatives that can handle sparsity within a discipline. The tool is demonstrated in two MDO problems: the design of a small satellite and its operation with the objective of maximizing downloaded data to a ground station, and the design of a horizontal-axis wind turbine with the objective of minimizing the cost of energy. In both cases, the method demonstrated improved efficiency by taking advantage of analytic gradients considering sparsity. This new capability in OpenMDAO greatly facilitates the implementation of system-level direct and adjoint coupled derivative evaluations, and is applicable for general problems.

I. Introduction

Gradient-based optimization with analytic gradients is an effective tool for solving problems with large dimensionality design spaces, and has been applied to a wide variety of design optimization problems, including aerodynamic shape optimization[1–3] and structural optimization.[4–6] To apply these methods to multidisciplinary problems, system-level derivatives must be constructed by combining the partial derivatives from each discipline to form the coupled derivative equations. Sobiesky[7] first formulated two versions of

*Aerospace Engineer, MDAO Branch, Mail Stop 5-11, AIAA Member
†Aerospace Engineer, MDAO Branch, Mail Stop 5-10, AIAA Member
‡Senior Systems Engineer, MDAO Branch, Mail Stop 5-11, AIAA Senior Member
§Ph.D. Candidate, Department of Aerospace Engineering, AIAA Student Member
¶Associate Professor, Department of Aerospace Engineering, AIAA Associate Fellow
‖Senior Engineer, National Wind Technology Center, AIAA Member

a direct coupled approach: the residual approach, which is based on the discipline governing equations residuals, and the functional approach, which is based on the coupling variables. Martins et al.[8,9] proposed an adjoint approach to computing system-level derivatives, and used it to perform coupled aerostructural design optimization of aircraft wings.[10,11] The approach is also applicable to other multidisciplinary design problems, such as aero-acoustic design.[12] Extending these gradient-based techniques to problems with more disciplines has proved to be difficult. When design problems grow to include tens of disciplines, it becomes increasingly challenging to construct the necessary derivatives. Even assuming all of the disciplines provide analytic partial derivatives, constructing the system-level derivatives depends heavily on the structure of the data connections between disciplines. Manual implementations for these problems are time consuming and discourage modifications to the problem formulation in the future. To overcome these issues, Moore[13] developed a method for automatically assembling the system level derivatives based on the data-dependency graph of a given problem formulation. This work represented the first implementation of the unified approach to computing system-level derivatives by Martins and Hwang[9] which combined forward- and adjoint-based derivatives into a single theoretical framework. An early attempt to the automatic implementation of coupled derivatives had also been achieved by Marriage and Martins,[14] but they did not use a graph to identify sparsity.. In addition to allowing greater flexibility in the problem formulation, a key feature of Moore's graph-based approach was the efficient handling of sparse problem formulations. By traversing the graph from design variables to quantities of interest, it was possible to consider only the subset of variables that were relevant to a specific system-level gradient, thus making gradient computations more efficient.

Although Moore's work demonstrated that a framework could compute system-level derivatives for arbitrary problem setups, the implementation assembled and inverted a complete system-level Jacobian for all derivative solutions. This approach made it unsuitable for larger problems involving high-fidelity tools, since enumerating a full Jacobian would be inefficient (if possible at all).

Hwang et al.[15] developed an alternative, graph-free method for computing automatic system level sensitivities that used a global design variable vector. Their method addressed the scalability challenges with Moore's work by using a matrix-free approach with components providing partial derivatives via a linear operator. They demonstrated the effectiveness of their method on a design optimization of a small satellite with over 25,000 design variables and over 1 million state variables. Despite its success, the global-vector-based approach required that all the variables from the system model be included when solving for system-level derivatives. This prevented the method from taking advantage of sparsity in the problem formulation and resulted in a less efficient gradient solving step.

The present work combines the graph-based approach with the matrix-free solution algorithm to efficiently compute derivatives for a wide range of sparse, large-scale engineering design problems. This new implementation is tested on the same small satellite design problem used in Hwang et. al's original work, showing a dramatic reduction in computational cost. In addition, the new method is applied to a wind turbine design study that has a more complex structure with stronger multidisciplinary couplings, as well as a mixture of analytic and finite-difference gradients. Comparisons between finite-difference and analytic gradient performance are made, demonstrating faster and tighter convergence with analytic gradients.

II. Unified Derivatives Computations

There are several methods for computing system-level derivatives that differ in accuracy, efficiency, and ease of implementation. The black-box finite-difference approximations involve the least amount of implementation effort, but they suffer from accuracy limitations caused by the combination of truncation and subtractive cancellation error that become worse with increasing nonlinearity in the model. Furthermore, the cost is at best equal to the cost of evaluating the full multidisciplinary model times the number of design variables. The complex-step method[16] eliminates the accuracy issue at the cost of increased implementation complexity, but it is typically two to three times slower than finite difference, which is already relatively inefficient.

The complex-step method can be somewhat invasive, but the remaining methods typically require an even greater level of source code access and modification. Algorithmic differentiation (AD) uses automated tools to symbolically differentiate at the line-of-code level, and combines the resulting partial derivatives to obtain numerically exact values for the system-level derivatives. AD operates in the forward and reverse modes, where the cost of the former is proportional to the number of design variables, and the cost of the latter is proportional to the number of output functions. Analytic methods have the same two modes of

operation and have the added advantage that only a linear system needs to be solved to obtain a vector of derivatives, which is cheaper than an evaluation of the model in some cases.

When implementing multidisciplinary problems, derivatives for each individual discipline are typically available and must be combined to compute system-level derivatives. The manner in which the derivatives are combined is highly problem-dependent. If there is no feedback among the disciplines and they are explicit functions, one can simply use the chain rule at the discipline-level. If all of the disciplines have residuals from the discretization of governing equations, the coupled adjoint or coupled direct method would be the most appropriate for computing system-level derivatives. If the disciplines are explicit but there is coupling among disciplines, the functional approach for the coupled derivative equations must be used.[7,9]

A framework that automatically computes system-level total derivatives from user-provided partial derivatives for each discipline must be able to apply the most appropriate method for the situation (explicit/implicit, coupled/sequential, direct/adjoint, etc.). This is greatly facilitated by a unification of derivative computation methods using a single equation from which all methods can be derived:[9]

$$\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}} \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}} = \mathcal{I} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}^{T} \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}^{T} \tag{1}$$

where $\boldsymbol{C}$ represents the vector of functions that constrain the variables $\boldsymbol{v}$ to their appropriate values.

By appropriately choosing $\boldsymbol{v}$ and $\boldsymbol{C}$, chain rule, black-box methods, analytic methods, coupled derivative equations, and algorithmic differentiation can be derived. For instance, including only the design variables and output variables in $\boldsymbol{v}$ yields a black-box method because all intermediate variables involved in computing the output variables are ignored in this situation. The significance of the above equation is that it enables a much simpler implementation of the derivative computation algorithm in a computational framework. Regardless of the specifics of a given problem, computing derivatives reduces to solving a linear system in the left (the forward or direct mode), or right equality (the reverse or adjoint mode) of the above equation.

III. Dependency Graph

In Moore's original work on computing derivatives from a dependency graph, he employed a discipline-based dependency graph, with a node for each discipline and edges describing dependency between the nodes.[13] A path-finding algorithm, from the NetworkX library,[17] computed the relevant set of disciplines for a given derivative. Although the dependency-based graph took advantage of sparsity at one level, by excluding any disciplines that did not directly contribute, it did not handle a second level of sparsity within disciplines. Even if a given discipline is relevant, some of its variables still may not directly affect the quantities of interest. To handle this problem, Moore utilized a secondary source of information outside the dependency graph. Pate et al.[18] proposed an alternative dependency graph structure that addressed this problem. Each discipline and each of its variables are represented by separate nodes with directed edges between them describing their dependencies on each other. Figure 1 shows a sample graph for a notional problem formulation.

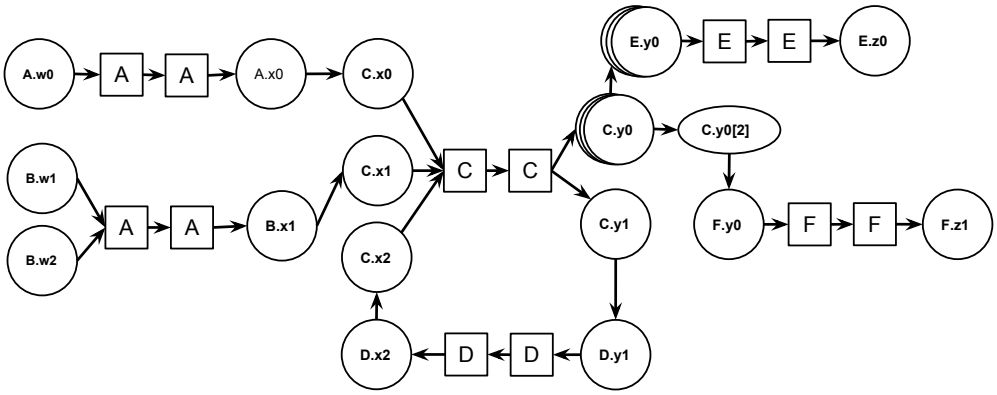

Figure 1: Dependency graph for a notional problem, using Pate et. al's proposed structure

In Pate et al.'s graph syntax, a single node is given for every variable, and two nodes are given for each discipline analysis. The dual discipline nodes were prescribed in order to create an edge that could be

assigned a weight in case weighted graph traversal algorithms were desired. However, they acknowledged that the dual nodes are often unnecessary and could be avoided for the sake of simplicity. Figure 2 shows the same notional problem, with the simplification of using only single model nodes made.

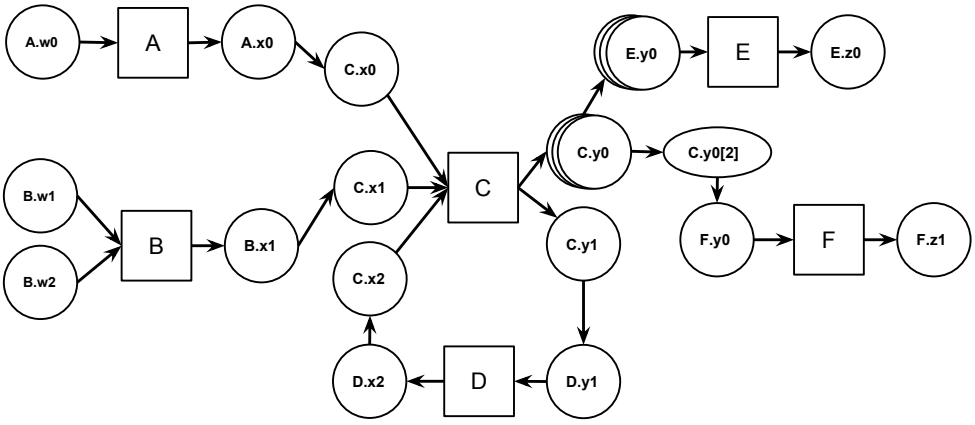

Figure 2: Dependency graph for a notional problem, using Pate et. al's proposed structure with a only one model node

This graph contains all the information needed to take full advantage of problem sparsity but suffers from one potential weakness. For a problem with millions of variables, such as the small satellite design problem, the graph would get very large and would be inefficient to operate on. To address this issue, related variables (i.e., arrays) can be aggregated into a single node to represent the overall variable. For problems with large arrays, this modification results in a significant reduction in the overall size of the graph. If any specific subvariable is referenced (e.g., some slice of an array) then a new node is added to represent that relationship. In Fig. 2 variable nodes C.y0 and E.y0 are both arrays and are directly connected. However, F.y0 is a scalar variable, which depends on a single entry form C.y0, so a subvariable node, C.y0[2], is added to the graph in between C.y0 and F.y0. By using the subvariable nodes, the computational advantages of problem sparsity are retained. Consider $\frac{\partial F.z1}{\partial C.y0}$. Given the graph, we know that this gradient will be sparse, with only the element relating to C.y0[2] having nonzero values as in Eq. (2).

$$\frac{\partial B.z}{\partial A.y} = \begin{bmatrix} 0 \\ 0 \\ \frac{\partial B.z}{\partial A.y[2]} \\ \vdots \\ 0 \end{bmatrix} \tag{2}$$

Because of the variable aggregation, solving for derivatives without the subvariable node would require all the elements of the C.y0 array to be included in the linear system. With the subvariable node, only the single relevant value from the array needs to be included. By representing hierarchical data as a single node the overall size of the graph remains manageable and the subvariable nodes are used to preserve the benefits of sparsity.

A. Graph Traversal Determining Relevance

OpenMDAO can calculate a Jacobian between any set of input and output nodes in a model by setting up the appropriate linear system. The size of the linear system is determined by the number of variables being considered, which means that the linear system can get very large. Although the matrix-free approach to solving this system can handle larger problems well, it is still desirable to minimize the size of the problem as much as possible. OpenMDAO was able to achieve significant reductions in the size of the linear system by traversing the dependency graph to find the subset of relevant variables. Consider solving for the derivative of $\frac{\partial F.z1}{\partial A.w0}$, given Fig. 2. Figure 3 highlights the relevant path through thre graph between A.w0 and F.z1. Only components A, C, D, and F are needed. Furthermore, only C.x0 and C.x2 are needed from C, because

of component-level sparsity. This means that only seven variables are needed to solve for the derivatives, instead of eleven if the graph is not reduced.

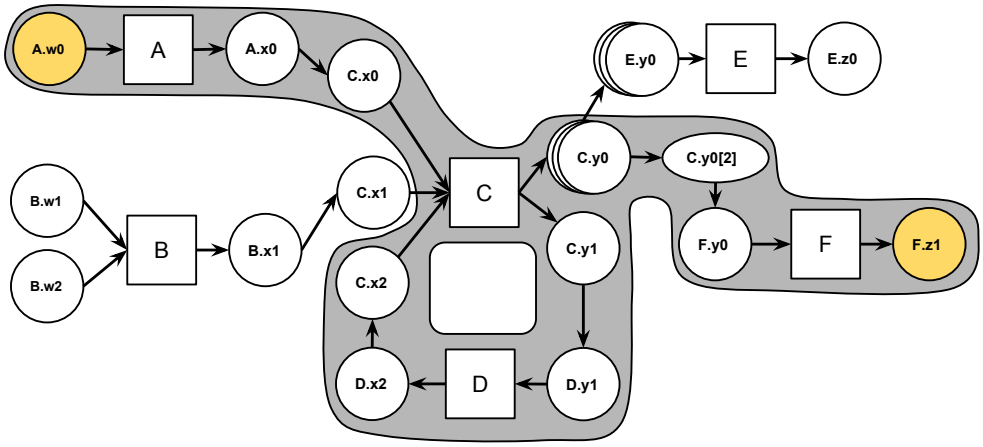

Figure 3: The reduced graph for the derivatives calculation includes components 1, 3, and 5 with their interconnecting variables.

A traversal from parameter to quantity of interest, as shown in Fig. 3, is used when solving for forward derivatives. In forward mode, the traversal searches for all of the variables that could be affected by a change in the parameter. One traversal is necessary for each design variable in the problem formulation. To solve for the derivatives using the adjoint method, a traversal must be made in the opposite direction. To do this, reverse all of the edges in the dependency graph and perform one traversal from each quantity of interest to all possible parameters.

B. Cycle Detection and Usage

Pate et al.[18] stated that within a dependency graph, the presence of cycles indicates coupling between the components in the cycle. Cycles can be induced by connections where data is passed directly from an output to an input, or implicitly where an input needs to be varied in order to drive a residual equation to zero. In Fig. 2 there is a cycle between components C and D. The presence of cycles has an impact on how models are solved. First, during normal execution, these cycles need to be converged with a numerical solver such as Gauss–Seidel or a Newton-based solver. In addition, these cycles need to be accounted for when solving for the derivatives. The unified gradient method by Martins and Hwang[9] allows for this situation, but requires the resulting residual equations be handled in a slightly different manner from explicit variable relationships (ones without cycles). Hence it is important to be able to efficiently identify all cycles in the graph. In formal terms, cycles exist in a graph when a group of nodes are strongly connected. Tarjan's algorithm provides an efficient means of finding the sets of strongly connected components.[19,20]

IV. API for Specifying Discipline Derivatives

All of the discussion up to this point has dealt with disciplines as if they were black boxes that are assumed to provide analytic derivatives of their outputs with respect to their inputs to the framework. This section describes the API for components to declare and provide said derivatives. The API is broken up into two parts: one that is mandatory for all components that declare derivatives, and another that is optional for components that wish to take full advantage of the graph-based sparse approach.

A. Mandatory API Methods

There are two mandatory methods that all components supporting derivatives must provide. The first is `list_deriv_vars()`. This method specifies the input and output variables that make up the columns and rows of its Jacobian, respectively. `list_deriv_vars()` returns a 2-tuple of inputs and outputs that have derivatives. A component is allowed to declare derivatives support for an arbitrary subset of all its variables.

The second method is `provideJ()`. This method serves two purposes. First, it provides the opportunity to perform any one-time work needed for linearization around the current point. Second, the method can optionally return the actual Jacobian, in the form of a $n \times m$ vector, where n is the number of outputs and m is the number of inputs. The ordering of the rows and columns of the Jacobian must match the order of the variables returned for `list_deriv_vars()`. When a Jacobian is returned from `provideJ()`, OpenMDAO will cache it and use it for all derivative computations around the current point for both forward and adjoint derivatives. This makes the `provideJ()` API the most straightforward and easiest to use in many cases. Figure 4 shows an example of an OpenMDAO component with user-specified derivatives, using the *provideJ()*method. This component has two input variables, x and y, and two output variables, f and g, where

$$f(x, y) = x^2 + \sin(y) \tag{3}$$
$$g(x, y) = x^3$$

```
from openmdao.main.api import Component
from openmdao.main.datatypes.api import Float
from math import sin, cos
from numpy import array

class ExampleComp(Component):

    # Inputs
    x = Float(0., iotype="in")
    y = Float(0., iotype="in")

    # Outputs
    f = Float(0., iotype="out")
    g = Float(0., iotype="out")

    def list_deriv_vars(self):
        return (('x', 'y',), ('f', 'g'))

    def provideJ(self):
        return array([[2 * self.x, cos(self.y)],
                      [3 * self.x ** 2, 0.0]])

    def execute(self):
        self.f = self.x ** 2 + sin(self.y)
        self.g = self.x ** 3
```

Figure 4: Example of an OpenMDAO component with user-specified Jacobian.

Although the `provideJ()` method is easy to implement, it has a few minor downsides. First, it requires that the component Jacobian be assembled in memory. In most cases, with tens or even hundreds of variables, this is reasonable, but if the Jacobian has a sparse structure, it may still be wasteful to allocate memory for a full matrix. Furthermore, for applications such as Computational Fluid Dynamics, where there are millions of variables partitioned across many CPUs, assembling the full Jacobian is simply not feasible. Lastly, depending on how the problem is configured, it is possible that not all of the variables will be needed for a given problem, as in Fig. 3. With `provideJ()`, the full Jacobian will still need to be provided though some of it will be irrelevant. So in effect, using the `provideJ()` method can partially negate some of the benefits of problem sparsity. In general, these inefficiencies are small and outweighed by the ease of implementation. However, in cases where these downsides become a significant concern, the `provideJ()` method should return nothing, and instead the optional API methods should be implemented for improved efficiency.

B. Optional API Methods

To overcome some of the downsides of constructing the full Jacobian, the components can provide a linear operator that computes the effect of multiplying the Jacobian with a given vector. This provides more freedom in how derivatives are implemented. It also enables the computing of only relevant quantities. The only significant downside is a small amount of added implementation complexity. Two different methods, `apply_deriv(arg, result)` and `apply_derivT(arg, result)`, must be implemented for forward and adjoint derivatives respectively. In each case, `arg` and `result` are dictionaries with relevant input and output

variable names as keys. If only forward or adjoint derivatives will be used, then only the necessary method needs to be implemented. Figure 5 provides an example of the `apply_deriv` and `apply_derivT` methods; using the same component shown above in Fig. 4.

```
class ExampleComp(Component):

    # Inputs
    x = Float(0., iotype="in")
    y = Float(0., iotype="in")

    # Outputs
    f = Float(0., iotype="out")
    g = Float(0., iotype="out")

    def list_deriv_vars(self):
        return (('x', 'y',), ('f', 'g'))

    def provideJ(self):
        pass

    def apply_deriv(self, arg, result):
        if "x" in arg:
            result["f"] += 2 * self.x * arg["x"]
            result["g"] += 3 * self.x ** 2 * arg["x"]
        if "y" in arg:
            result["f"] += cos(self.y) * arg["y"]


    def apply_derivT(self, arg, result):
        if "f" in arg:
            result["x"] += 2 * self.x * arg["f"]
            result["y"] += cos(self.y) * arg["f"]
        if "g" in arg:
            result["x"] += 3 * self.x ** 2 * arg["g"]

    def execute(self):
        self.f = self.x ** 2 + sin(self.y)
        self.g = self.x ** 3
```

Figure 5: Example of an OpenMDAO component with apply_deriv and apply_derivT methods implemented.

The *if* conditions in the sample implementation from Fig. 5 provide the mechanism for taking full advantage of sparsity. The usefulness of this feature can be readily demonstrated with a notional optimization using the Multidisciplinary Feasible (MDF) architecture.[21]

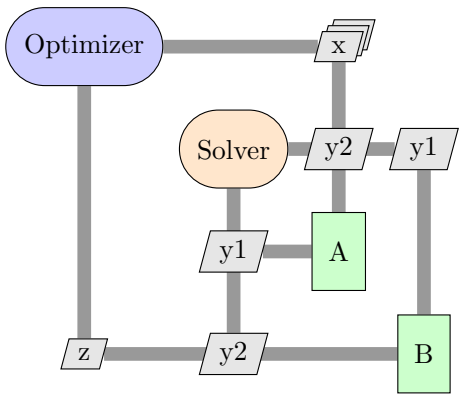

Figure 6: Notional design problem with MDF problem formulation

In Fig. 6, the optimizer varies the design variable $A.x$, a large vector. The solver converges variables y_1 and y_2 from disciplines A and B. Applying a Newton-based method requires $\frac{\partial A.y_1}{\partial A.y_2}$ and $\frac{\partial B.y_2}{\partial B.y_1}$ to converge, but $\frac{\partial A.y_1}{\partial A.x}$ is not relevant. By using the linear operator derivatives method, the cost of multiplying by $\frac{\partial A.y_1}{\partial A.x}$ can be avoided, while converging the solver and only computed as part of the outer optimization loop.

V. Small Satellite Design Problem

Cubesat investigating Atmospheric Density Response to Extreme driving (CADRE) is a mission funded by the National Science Foundation to study the response of the thermosphere to auroral phenomena.[22] The small satellite design problem optimizes the design of a cubesat for the CADRE mission. This problem was originally implemented and solved by Hwang et al.[15] using a matrix-free coupled derivative strategy. The analysis models the orbit of a cubesat as it performs a mission over the course of one year. The full mission is represented by 6 half-days of operation, computed at conditions 1, 3, 5, 7, 9, and 11 months after launch.

Table 1: Small satellite design problem formulation

	Variable/Function	Description	Quantity
maximize	$\sum_{i=1}^{6} D_i$	Data downloaded	
with respect to	$0 \leq I_{\text{setpt}} \leq 0.4$	Solar panel current	$300 \times 12 \times 6$
	$0 \leq \gamma \leq \pi/2$	Roll-angle profile	300×6
	$0 \leq P_{\text{comm}} \leq 25$	Communication power	300×6
	$0 \leq \text{cellInstd} \leq 1$	Cell vs. radiator	84
	$0 \leq \text{finAngle} \leq \pi/2$	Fin angle	1
	$0 \leq \text{antAngle} \leq \pi$	Antenna angle	1
	$0.2 \leq iSOC \leq 1$	Initial state of charge	6
		Total	25292
subject to	$I_{\text{bat}} - 5 \leq 0$	Battery charge constraint	6
	$-10 - I_{\text{bat}} \leq 0$	Battery discharge constraint	6
	$0.2 - SOC \leq 0$	Battery capacity constraint	6
	$SOC - 1 \leq 0$	Battery capacity constraint	6
	$fSOC - iSOC = 0$	SOC periodicity constraint	6
		Total	30

The objective of the optimization was to maximize the total amount of data transmitted to the ground station over these six design points. In their original work, Hwang et al. located the ground station for receiving data in Ann Arbor, Michigan. Because the satellite was launched into a polar orbit moving the ground station closer to one of the poles would increase the amount of data collected. McMurdo Station, Antarctica is located close to the South Pole and is an existing scientific research facility. So for this work, the McMurdo was selected as the ground station location.

There are seven design variables listed in Tab. 1. Three of them, γ, I_{setpt}, and P_{comm}, are arrays of time-varying schedules for the attitude, solar panel current, and communications power, respectively. The remaining four, $cellInstd$, $finAngle$, $antAngle$, and $iSOC$, are all physical design variables of the satellite. All together there are 25,292 design variables. The five constraints for the problem relate to the battery charge rate, battery discharge rate, minimum battery capacity, maximum battery capacity, and a battery state-of-charge (SOC) periodicity constraint. Each constraint was a length 6 vector with a value representing each of the 6 different orbits, yielding a total of 30 constraints. Since the problem has significantly more design variables than objectives and constraints, it the gradients are computed using the coupled adjoint method.

Overall, the problem can be broken down into seven distinct disciplines: orbit dynamics, attitude dynamics, cell illumination, temperature, solar power, energy storage, and communication. Figure 7 shows how each of the disciplines relate to each other in the optimization problem. The actual implementation of the problem further subdivided most of the disciplines into smaller subdisciplines. The true component-level dependency graph, given in Fig. 8, has 39 different components. The full graph, including all variable and subvariable nodes, is omitted for visual clarity.

From a problem structure standpoint, this problem has three significant features. First, the large number of disciplines, design variables, and the complex connections between them make assembling the linear system

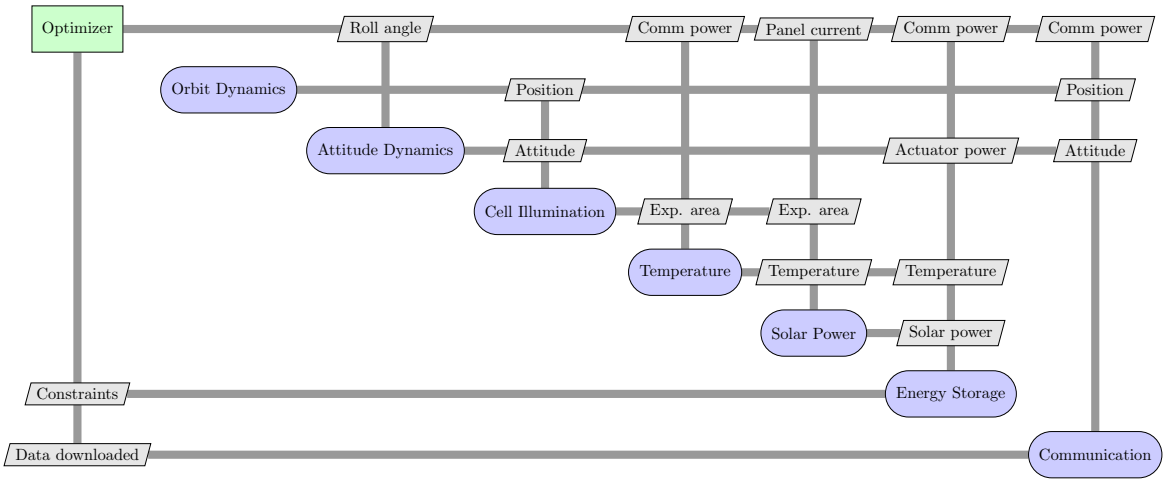

Figure 7: Extended Design Structure Matrix (XDSM)[23] showing the structure of the small satellite design problem

to solve for gradients challenging to do by hand. Second, although the problem does not have any explicit interdisciplinary coupling, there is coupling from the SOC periodicity constraint, $fSOC - iSOC = 0$, which is dependent on many of the disciplines. Third, all disciplines are implemented with analytic derivatives, so that no finite differencing is necessary.

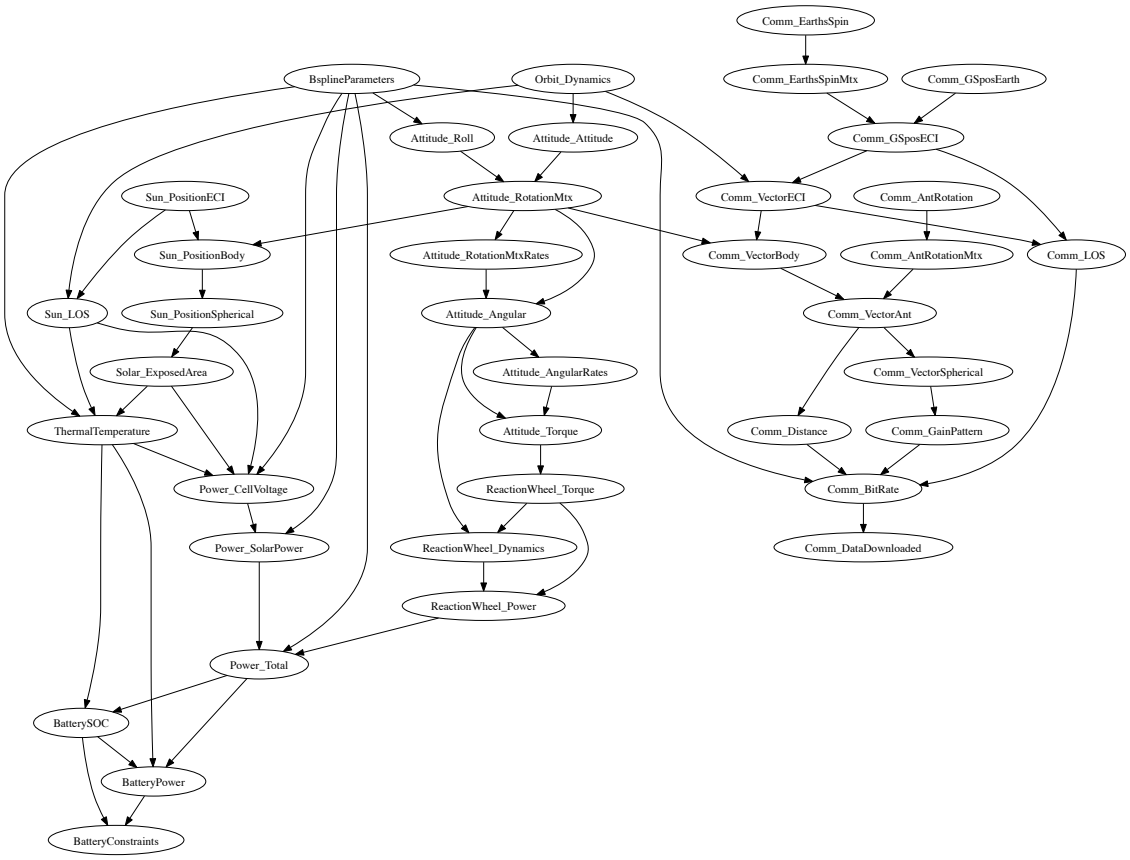

Figure 8: Dependency graph for the satellite problem.

A. Optimization Results

The final optimal design yielded 10343 GibaBytes (GB) total data downloaded over the 6 time points considered. Figure 9 shows ground trace for the satellite colored by communications bit rate for all 6 time points combined. Bit rate can be correlated directly with the communications power in Fig. 10, where large spikes in communications bit rate correspond to passes over the ground station requiring power to the communication system for transmission. Figure 10 shows key design and performance data for the design problem over the course of the optimization for all six points in time considered. Data is shown for the initial condition, at 100 function evaluations, and for the final optimum solution. At the initial condition the roll angle is a constant 45 degrees, the communication power is uniformly 0, and battery state of charge drops quickly below 0. This design is obviously terrible and indicates clearly why the optimizer had to work hard to find a feasible solution. By the 100th function evaluation, the optimizer found a reasonable design which refined smoothly for the rest of the optimization.

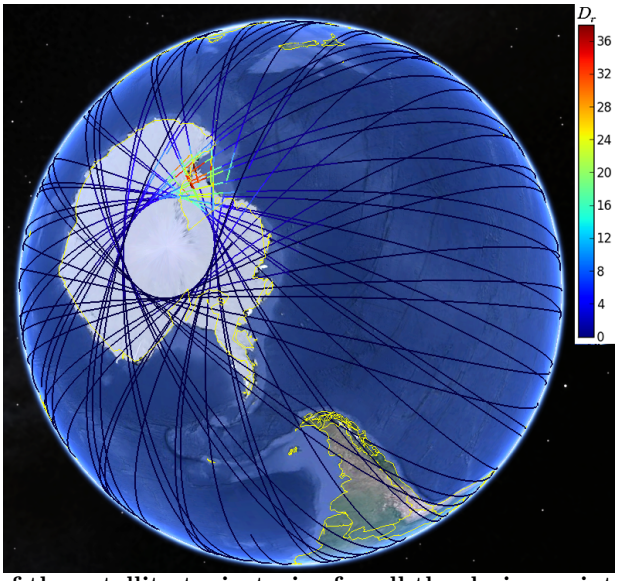

Figure 9: Ground trace of the satellite trajectories for all the design points colored by communications bit rate.

The roll angle variation in Fig. 10 roughly approximates a sine function with a period of 90 minutes (the approximate orbital period of the satellite), with short-term perturbations during the time periods, where line of sight is achieved with the ground station. That is, the optimizer converged to a solution with the satellite continuously turned to maximize exposure to the sun, except when turning to point its antenna towards the ground station. This dynamic is also reflected in the SOC data plotting in the bottom row, with the battery losing charge quickly during communication with the ground station but recharging while tracking with the sun.

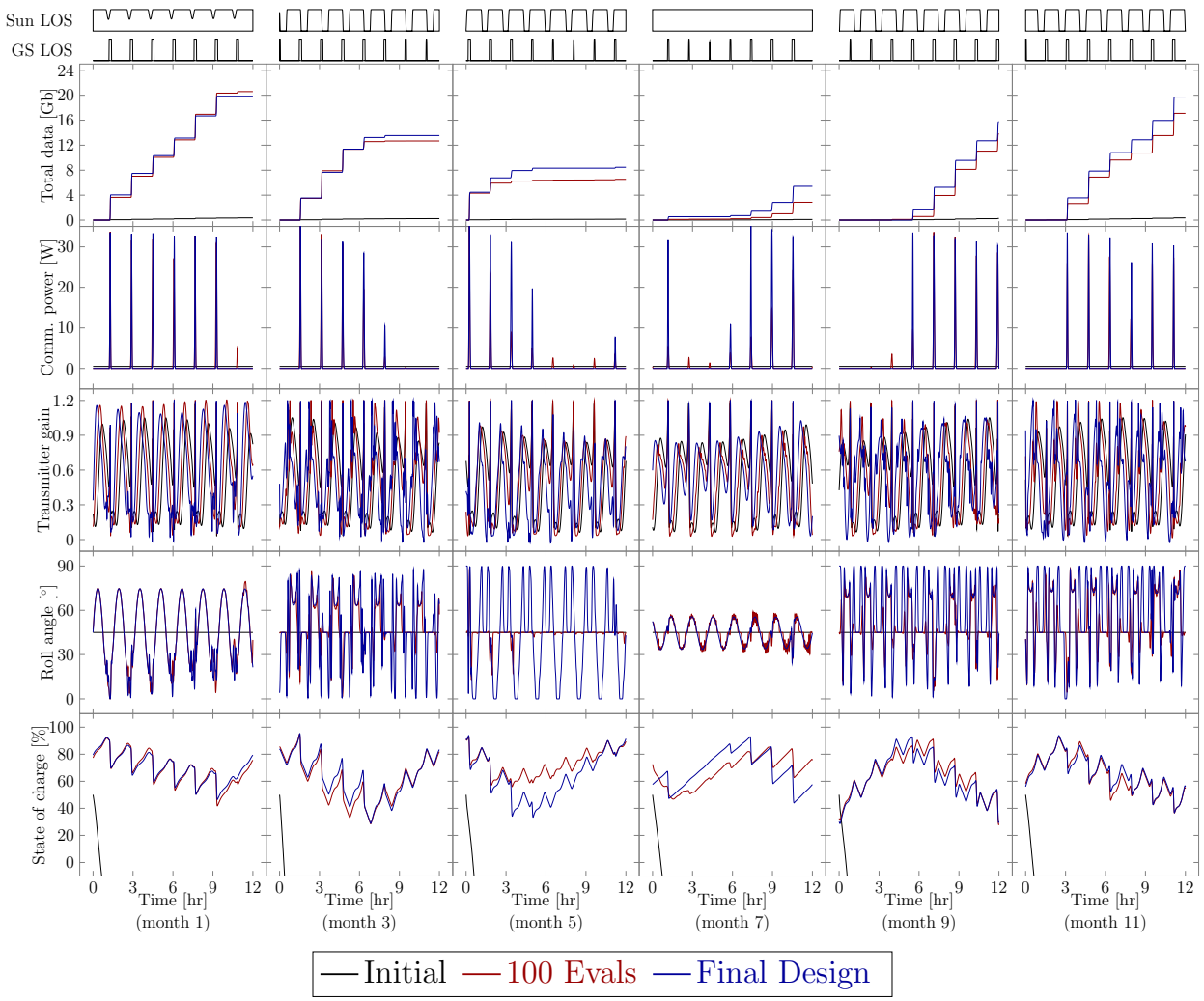

Figure 10: Plots of the SOC, roll angle, transmitter gain, communications power, and total data downloaded for each of the 6 time periods analyzed. Data is presented for the initial condition (black), at 100 function evaluations (red), and at the final optimal point (blue).

B. Performance Results

The OpenMDAO implementation of the satellite problem was executed on a Macbook Pro (2.6 GHz Core i7 processor, 16 GB 1600 MHz DDR3 memory, running OSX 10.8.5). The problem was converged to a termination tolerance of 10^{-4} using the SNOPT[24] optimizer. The problem was run twice, once with a full problem graph including all variables regardless of relevance, and a second where the graph was reduced based on the algorithms discussed in Section A. The effects of the graph reduction are shown in Table 2. The number of variables in the graph, and hence the size of the linear system that needs to be solved, was reduced by 18%. This reduction yielded a drop in runtime of about 21%.

Table 2: Effects of relevance-based graph reduction on optimization performance. Run times are quoted as time to 325 function evaluations

	Full Graph	Reduced Graph	% Reduction
# Variables	2.15e6	1.76e6	18%
# Components	39	29	25.6%
Derivative Solve Time (sec)	126	87	31%
Total Run Time (hours)	22.39	17.33	21.3%

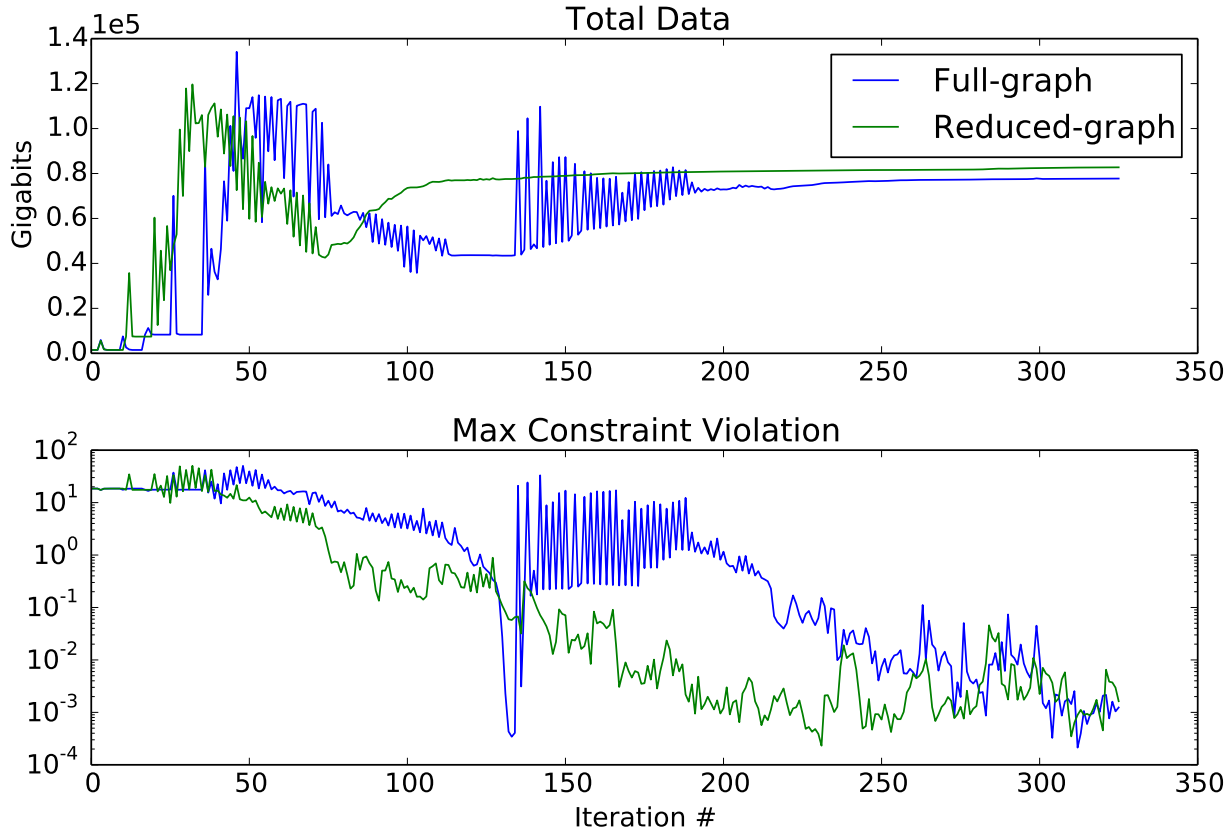

Figure 11: Iteration history for the full-graph and reduced-graph cases of the satellite problem.

Figure 11 compares function evaluation history over the course of the optimization between the full-graph and reduced-graph cases. The top plot shows the value of the objective function, the total data downloaded. The bottom plot shows the maximum constraint violation. In both cases the objective function oscillated greatly early in the optimization while the optimizer sought a feasible solution. The saw-tooth pattern in the results occurs when the optimizer performs large jumps during a line search. When the line search moves too far, the objective and convergence can both get worse, causing the optimizer to backtrack. The reduced-graph case shows significantly improved convergence, stabilizing to a feasible solution in around 80 iterations verses 200 for the full-graph case. The reduced-graph version also found a better optimum data

downloaded of 10343 GB versus 9718 GB for the full-graph case, a 6% improvement.

Table 3: Comparison between the full and reduced problem graphs for different-sized small satellite design problems

Full Graph	Reduced Graph	Reduction
1.08e6	8.8e5	18%
2.15e6	1.76e6	18%
4.30e6	3.52e6	18%
8.60e6	7.05e6	18%
1.72e7	1.41e7	18%

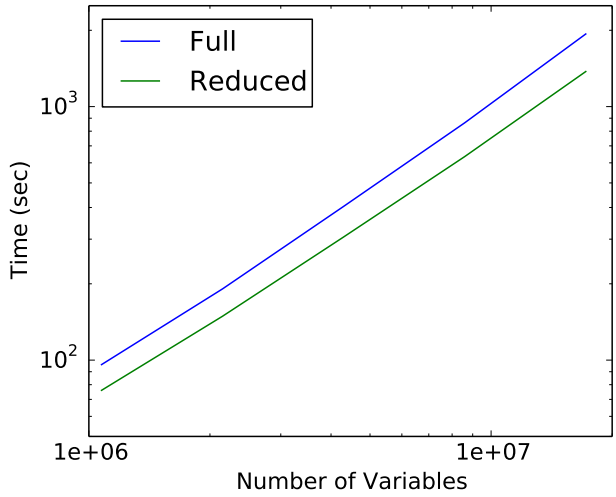

Figure 12: Adjoint solution cost as a function of number of variables in the problem formulation.

Figure 12 shows a log-log plot of computational cost for 31 adjoint solves (30 constraints + 1 objective) versus the number of variables in the full problem. The trend shows linear growth for both the full and reduced versions of the graph, with a constant ratio between them. This problem is scaled by refining the time discretization for the integration of ordinary differential equations. Although increasing the number of time steps taken does increase the sparsity of the individual discipline Jacobians, it does not have an impact on the sparsity of the problem formulation. In other words, although the size of the variable arrays grows, the structure of the dependency graph remains constant. Table 3 demonstrates the consequence of the constant problem formulation sparsity, which yields a fixed ratio between the full and reduced problem size.

Although the overall reduction in optimization cost was 21%, reduction of dependency graph provided a 30% reduction in the cost to compute adjoint derivatives.The reduction in computational cost for solving the gradients did not directly translate into equivalent savings in the overall optimization cost because of the computational cost of the optimizer. This problem requires a large number of minor iterations, where SNOPT solves a quadratic subproblem. The cost of solving these subproblems is not directly affected by the cost of computing the derivatives, and hence reduces the overall computational savings. This characteristic is highly problem-dependent, and is not expected to be so significant for all problems. Also, the overhead from the optimizer would get less significant as the computational cost of the analyses and derivative solves increased. Thus, the efficiency gain from the graph reduction would be more significant for larger, more expensive problems.

VI. Wind Turbine Design Problem

The goal of this design problem was to optimize a horizontal-axis wind turbine for minimum cost of energy. The problem includes the design of the rotor blades with a coupled aerostructural analysis as well as the structural design of the hub, nacelle, and tower. Cost models for the turbine and wind plant are included

to assess trade-offs in energy capture and capital and operational costs. The analyses for this model are part of the Wind-Plant Integrated Systems Design & Engineering Model (WISDEM) developed at the National Renewable Energy Laboratory (NREL) as part of a larger effort to develop a physics- and cost-based design optimization framework for wind turbines.[25–28]

Wind turbine design involves a large number of structural constraints specified by various standards. This optimization problem utilizes a subset of the design requirements specified by the International Electrotechnical Commission for land-based designs.[29] Structural considerations include maximum deflections, stress and strain limits, resonance, buckling, and manufacturing considerations. A typical problem formulation is given in Table 4. Bound constraints are chosen to be large enough to not be active, except for those that must be restricted because of manufacturing or transportation reasons. For simplicity, safety factors are not included in the list of constraints.

Table 4: Horizontal-axis wind turbine optimization problem description

	Variable/function	Description	Quantity
minimize	COE	Cost of energy	
with respect to	$0.4 \leq c \leq 20$	Chord distribution	5
	$-10 \leq \theta \leq 30$	Twist distribution	4
	$0.005 \leq t_{sc} \leq 0.2$	Spar-cap thickness distribution	5
	$0.005 \leq t_{te} \leq 0.2$	Trailing-edge panel distribution	5
	$3 \leq \lambda \leq 14$	Tip-speed ratio in Region 2	1
	$0.1 \leq L_{shaft} \leq 10$	Low-speed-shaft lengths	2
	$0.01 \leq h_{beam} \leq 10$	Bedplate I-beam sizing	2
	$3.87 \leq d \leq 6.3$	Tower diameter	3
	$0.005 \leq t \leq 0.2$	Tower wall thickness	3
	$0.25 \leq z_{waist} \leq 0.75$	Tower waist location	1
	$50 \leq H \leq 200$	Tower height	1
		Total	32
subject to	$\delta_{tip} \leq \delta_{max}$	Blade tower strike	1
	$\delta_{ground} \geq \delta_{min}$	Blade ground clearance	1
	$f_{blade} \geq 3\Omega$	Blade resonance avoidance	2
	$\epsilon \leq \epsilon_{ult}$	Ultimate strain in blades	30
	$\epsilon \leq \epsilon_{cr}$	Panel buckling in blades	15
	$\delta_{lss} \leq \delta_{max}$	Deflection limits in low-speed shaft	4
	$\delta_{bdp} \leq \delta_{max}$	Deflection limits in bedplate	2
	$\sigma_{bdp} \leq \sigma_{ult}$	Ultimate stress in bedplate	2
	$\sigma_{twr} \leq \sigma_{ult}$	Ultimate stress in tower	12
	$\sigma_{twr} \leq \sigma_{shell}$	Shell buckling in tower	14
	$\sigma_{twr} \leq \sigma_{global}$	Global buckling in tower	14
	$f_{tower} \geq \Omega_{rotor}$	Tower resonance avoidance	1
	$d/t \geq 120$	Weldability	3
	$d_{top}/d_{base} > 0.4$	Manufacturability	1
		Total	102

There are 32 design variables, 1 objective, and 102 constraints. An additional maximum tip-speed constraint of 80 m/s is imposed directly in the analysis. Because there are fewer design variables than quantities of interest, the problem was solved with forward derivatives. In addition, the relatively small number of design variables enabled a comparison between the use of analytic and finite-difference gradients. There were 10 disciplines involved in the top level of the model, as seen in Fig. 13. Half of these disciplines

represented the physics of the problem, aerodynamic and structural models of the turbine components, while the other half encapsulated the various cost components. The aerostructural coupling is achieved through a fixed point iteration around the rotor discipline, converging the deflected blade shape to a tolerance of 1×10^{-8}. OpenMDAO automatically computes the coupled derivatives around this convergence loop.

Similar to the small satellite design problem, many of the disciplines are further subdivided into more than one component for implementation. In the satellite problem, all components were left at the same level in the model, but for this problem actual subassemblies of components were made (e.g., Rotor, Hub, Nacelle, Tower, Turbine Capital Cost, and Balance of Station). This highly nested structure is designed to provide modularity, to allow a wide range of researchers to build discipline models that can be swapped in and out. It also greatly simplifies the complexity of the top level model. The presence of these subassemblies is relevant to how the derivatives are computed. They are treated as single discipline components from the perspective of the top level model, meaning that when derivatives are requested from the assembly, it will internally solve for its own boundary derivatives and report them to the top level assembly. Table 5 summarizes number of subassemblies and components in those subassemblies for the overall model. There are a total of 104 components spread out between 12 different subassemblies.

Table 5: Component and subassembly counts for the wind turbine design problem at different levels of the model

	# Components	# Subsssemblies
Top level	1	9
Rotor	33	0
Hub	4	0
Nacelle	10	0
Tower	16	0
Tower Strike	1	0
Annual Energy Production	1	0
Turbine Capital Cost	1	3
Nacelle Capital Cost	8	0
Tower Capital Cost	2	0
Rotor Capital Cost	6	0
Operational Expenses	1	0
Balance of Station	19	0
Financial	1	0
Total	104	12

The Rotor discipline in particular has an important internal feature with regard to computing derivatives. The rotor subassembly is responsible for modeling aerodynamics, using CCBlade,[30] a blade element momentum (BEM) tool specifically tailored to optimization. It includes an internal convergence on the PowerCurve discipline, with a Brent solver,[31] to find the rated speed, V_{rated}, for the wind turbine. This convergence loop requires the Rotor subassembly to compute coupled derivatives. Figure 14 shows the complete dependency graph for just the Rotor discipline, which has 33 components in total. The internal solver used to find rated speed is highlighted with a box in the figure.

The structures discipline for the blades is modeled with pBEAM,[32] a beam finite element code, PreComp,[33] a structural property tool for composite blades, and additional structural modules for panel buckling, load and mass transfer, and geometric modeling. In the Tower subassembly, aerodynamics is modeled with power-law wind profiles and cylinder drag theory. Tower structures is modeled with pBEAM and additional modules for hoop stress, shell buckling, and global buckling. Some drivetrain components are modeled with scaling relationships (bearings, yaw system, generator), while others used bottom-up physics models (bedplate, gearbox, low-speed shaft).

Analytic gradients are derived for all aerodynamic components, cost components, and many of the structural components using a mixture of hand-derivation and automatic differentiation. Some of the remaining structural components (pBEAM and PreComp) do not yet provide analytic gradients. For these components, the output was still continuously differentiable, and their Jacobians were estimated with finite differencing.

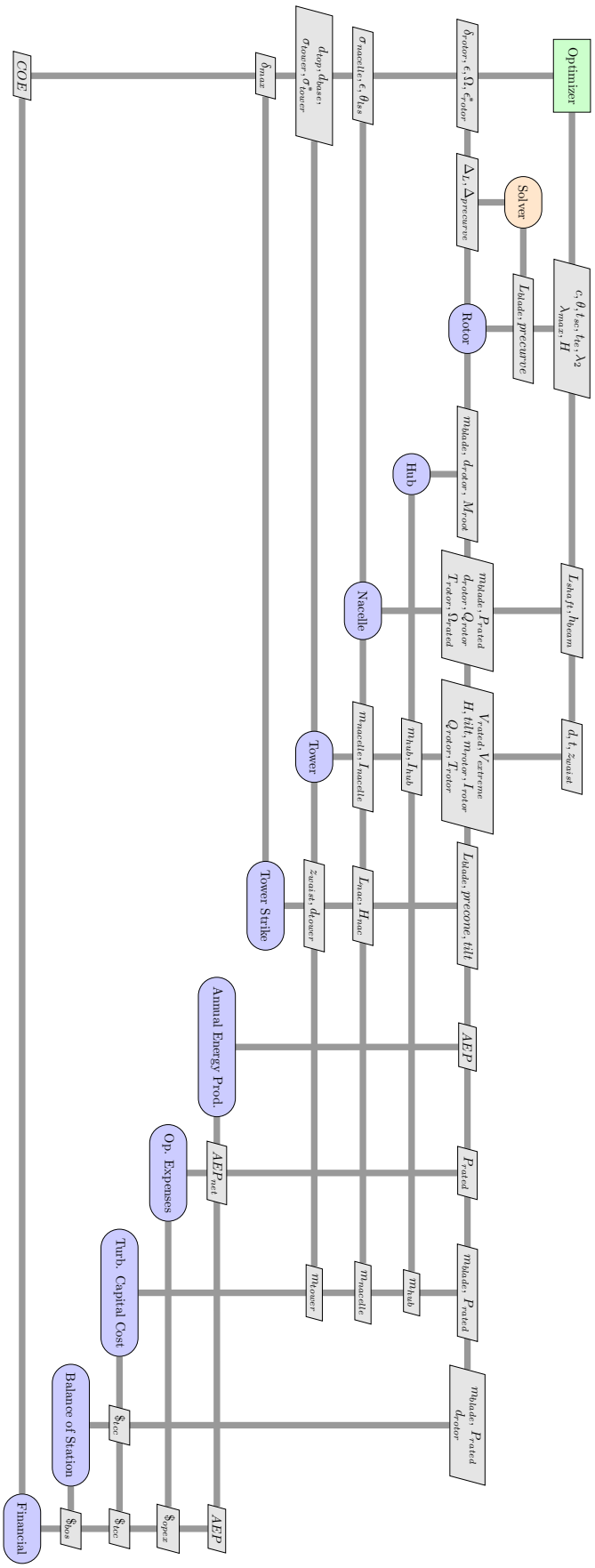

Figure 13: XDSM[23] diagram of the top level wind turbine design optimization problem

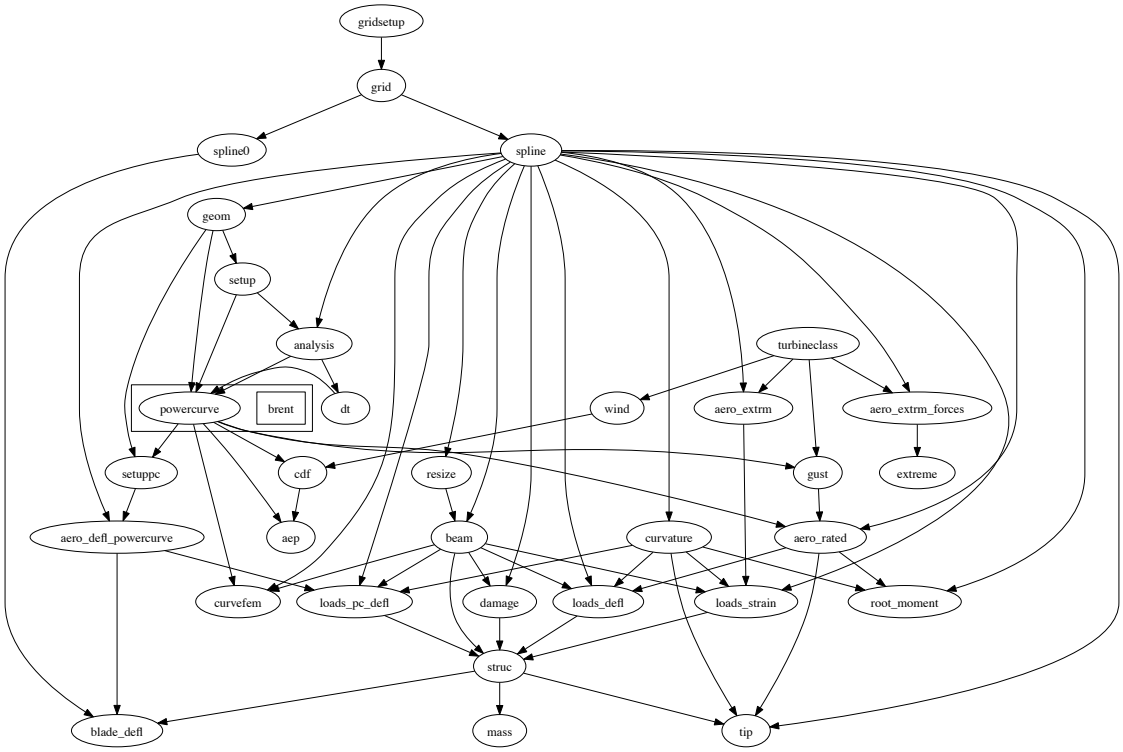

Figure 14: Rotor subassembly dependency graph

This led to a mixed derivative scenario where analytic Jacobians and the finite difference Jacobians were used together to solve for system-level derivatives.

This problem has one additional interesting characteristic in its problem formulation: a number of data connections throughout the dependency graph were never relevant to the derivatives calculations. These nonrelevant variables were used for various settings and initialization values throughout the model, such as the number of blades on the rotor or the density of air. For convenience and consistency, the variables still have connections so that the right values can be set in one place and passed to all necessary components automatically. Such variables, especially when they are integers (or otherwise nondifferentiable) necessitate the graph-based approach to identify irrelevant variables. Without that capability, these variables would be included in the linear systems for computing derivatives. At best, the continuous nonrelevant variables would only make the linear solve a bit slower. At worst, the discrete nonrelevant variables would prevent the derivatives computation from working properly at all.

A. Optimization Results

The baseline design for this problem is the NREL 5-MW reference model,[34] with a blade structural layup definition from Sandia National Laboratories.[35] The baseline model was not designed to be optimal in any sense, but is instead useful as a nominal reference configuration.

A high-level summary of wind turbine component masses, energy production, and plant costs are summarized in Table 6. The energy capture is quantified by the annual energy production (AEP). Plant costs include turbine capital costs (TCC), balance-of-station costs (BOS), and operating expenses (OPEX). The cost of energy is computed as

$$COE = \frac{FR(TCC + BOS) + (1 - T)OPEX}{AEP} \tag{4}$$

where FR is the financing rate, and T is the tax deduction rate on operating expenses.

The baseline design has significant structural margins in both the blade and the tower. In this case the optimization pushes towards a lower solidity blade, saving mass, and correspondingly operates at a

Table 6: Comparison between component masses, energy production, and plant costs for the baseline design and a minimum cost of energy design.

	Baseline	Optimized
blade mass (kg)	17,303	13,226
hub mass (kg)	42,126	38,482
nacelle mass (kg)	208,748	208,364
tower mass (kg)	349,649	333,492
AEP (MWh/turbine)	19,579	20,093
TCC ($\frac{\$}{kW}$)	1,698	1,587
BOS ($\frac{\$}{kW}$)	558	561
OPEX ($\frac{¢}{kWh}$)	1.22	1.21
COE ($\frac{¢}{kWh}$)	6.20	5.80

higher tip-speed ratio (tip-speed ratio increased from 7.55 to 8.52). The solidity and spar-cap stiffness was sized primarily by the out-of-plane tower strike requirement. Trailing-edge panels were sized primarily by maximum strain and buckling requirements at 50-year survival load conditions.

While the final tower design has a similar mass to the baseline, it is significantly taller in order to increase power capture (hub height increase from 90 m to 107 m). Because of the large design margins for the baseline design, the tower is able to increase in height without large mass penalties by using a larger base diameter and thinner shell sections. The tower design is sized primarily by shell buckling at maximum thrust load conditions.

The changes in the drivetrain and nacelle are more subtle, resulting in a similar mass. Overall, the mass reductions allow for over 9% savings in turbine capital costs. Simultaneously, the larger hub height, and more efficient blades increase the AEP by almost 3%. The net effect is a reduction in cost of energy of approximately 6.5%.

B. Performance Results

This optimization was run with both finite-difference and mixed analytic/finite-difference gradients. The problem was converged to a termination tolerance of 5×10^{-5} using SNOPT. For this problem, finite-difference achieved almost the same optimum design that the analytic gradients case found, and had the same level of feasibility. The major advantage of the analytic gradients was a 5X reduction in computational time, although a minor improvement in the objective function was achieved as well. Figure 15 shows optimization history for every function evaluation of the top level model. This outcome is significant in two ways. First, it demonstrates the value obtained by applying the extra effort to derive derivatives for discipline analyses. Second, it proves that it is possible to achieve some benefits without defining derivatives for every single analysis. While outside the scope of this paper, a wide range of variations on this optimization problem have been performed, and the use of the analytic gradients has shown to also be beneficial not only in terms of speed, but also in terms of solution robustness and quality.

Table 7: Comparison of the results between the finite-difference and analytic derivatives

	Finite-Difference	Analytic
Objective (COE in ¢/kWh)	5.8045	5.8042
Max Constraint Violation	2.62×10^{-6}	1.81×10^{-6}
# Major Iterations	143	113
Total Run Time (hours)	**5.43**	**1.11**

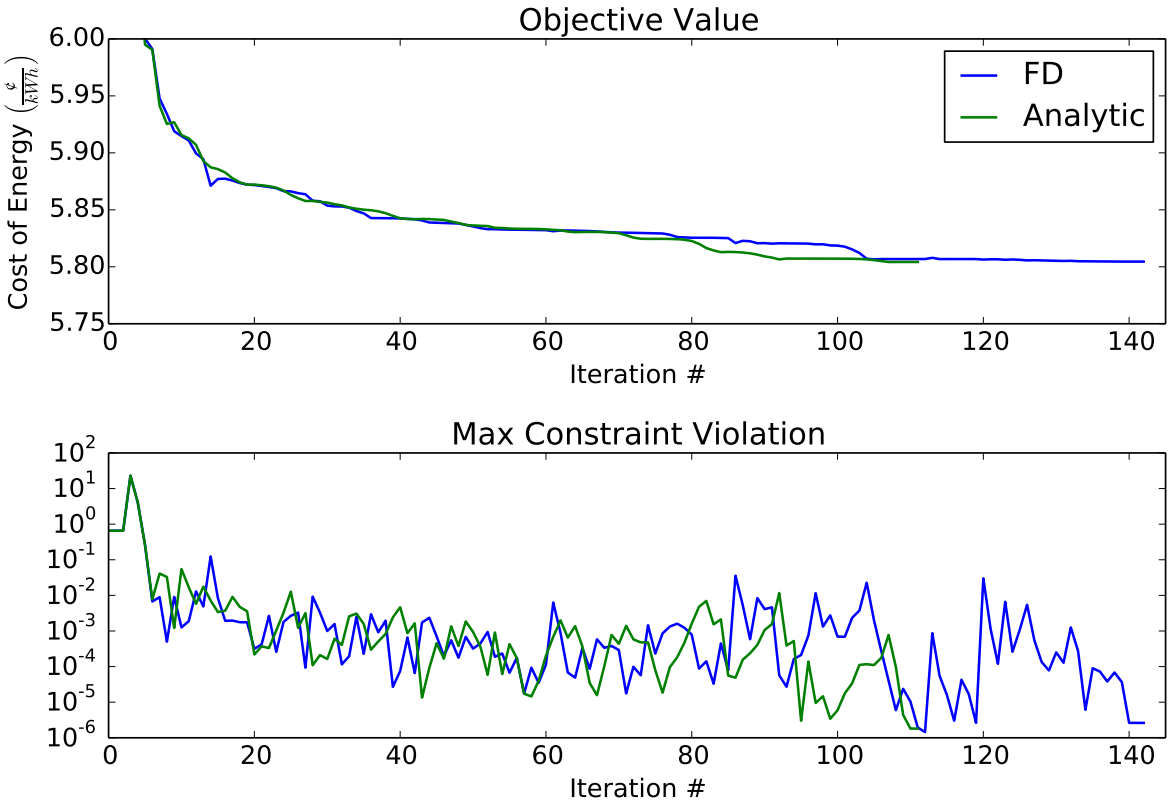

Figure 15: Comparison between optimization performance and finite-difference and analytic gradients.

VII. Conclusion

Multidisciplinary design problems can exhibit different types of sparsity. Certain disciplines may not be relevant to all objectives and constraints, not all variables within a discipline will be relevant, and the Jacobians for any given discipline may be sparse. We implemented a graph-based approach to identify the sparsity in any given formulation using a graph structure designed to allow for efficient usage even with millions of variables. We further developed a flexible API that allowed disciplines to declare their partial derivatives to OpenMDAO in a manner that takes full advantage of any sparsity within its Jacobian. Through the combination of the graph based method to find the minimum set of relevant variables, and the flexible API provided by OpenMDAO, we were able to take full advantage of the sparsity at all levels in a problem formulation.

The approach was applied to two different design problems, each with unique characteristics that tested different aspects of the method. The small satellite design problem had millions of variables and used analytic adjoint derivatives. This problem would have been very difficult to solve without analytic derivatives because of its large dimensionality design space. Assembling the system-level derivatives would have been challenging to do by hand due to the large number of components involved. The results from the optimization of the small satellite reveal that the implementation scales well with increasing number of variables, and a significant computational cost benefit was obtained by taking advantage of sparsity in the problem formulation. The wind turbine design problem had a design space with lower dimensionality, a significantly more complex problem formulation, and used a combination of finite-difference and analytic gradients. The problem was solved with forward gradients because it had more constraints than design variables. The optimization results demonstrated OpenMDAO's ability to solve highly nested problems with coupled derivatives. In addition, they showed a gain in computational efficiency by applying analytic derivatives, even though some major analyses still required finite differencing.

The combined results from the two design problems clearly demonstrate the value of the approach presented here: efficient and automatic computation of system-level derivatives, given partial derivatives from

the discipline analyses, even for very large problems. It also allows analyses with and without analytic derivatives to be used effectively within the same model.

VIII. Acknowledgments

This work was supported by the NASA Fundamental Aeronautics Program, Aeronautical Sciences Project. The authors would like to thank Dae Young Lee and Prof. James W. Cutler from the University of Michigan for contributing to the development of the analyses used in the small satellite design problem. The authors also gratefully acknowledge the contributions of Katherine Dykes, Yi Guo, and Ryan King from the National Renewable Energy Laboratory for their development efforts on the wind turbine nacelle and cost models.

References

[1]Lee, B. J., Liou, M.-S., and Kim, C., "Optimizing a Boundary-Layer-Ingestion Offset Inlet by Discrete Adjoint Approach," *AIAA Journal*, Vol. 48, No. 9, 2014/04/16 2010, pp. 2008–2016.

[2]Palacios, F., Alonso, J. J., Colonno, M., Hicken, J., and Lukaczyk, T., "Adjoint-based method for supersonic aircraft design using equivalent area distributions," *50th AIAA Aerospace Sciences Meeting, American Institute of Aeronautics and Astronautics, Nashville, TN*, Vol. 4, 2012.

[3]Lyu, Z., Kenway, G. K., and Martins, J. R. R. A., "Aerodynamic Shape Optimization Studies on the Common Research Model Wing Benchmark," *AIAA Journal*, 2014, (Accepted subject to revisions).

[4]Kennedy, G. J. and Martins, J. R. R. A., "A Parallel Finite-Element Framework for Large-Scale Gradient-Based Design Optimization of High-Performance Structures," *Finite Elements in Analysis and Design*, 2014, (In press).

[5]Venkataraman, S. and Haftka, R., "Structural optimization complexity: What has Moore's law done for us?" *Structural and Multidisciplinary Optimization*, Vol. 28, 2004, pp. 375–387.

[6]Adelman, H. M. and Haftka, R. T., "Sensitivity Analysis of Discrete Structural Systems," *AIAA Journal*, Vol. 24, May 1986, pp. 823–832.

[7]Sobieszczanski-Sobieski, J., "Sensitivity of complex, internally coupled systems," *AIAA Journal*, Vol. 28, No. 1, 04 1990, pp. 153–160.

[8]Martins, J. R. R. A., Alonso, J. J., and Reuther, J. J., "A Coupled-Adjoint Sensitivity Analysis Method for High-Fidelity Aero-Structural Design," *Optimization and Engineering*, Vol. 6, No. 1, March 2005, pp. 33–62.

[9]Martins, J. R. R. A. and Hwang, J. T., "Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models," *AIAA Journal*, Vol. 51, No. 11, November 2013, pp. 2582–2599.

[10]Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., "Scalable Parallel Approach for High-Fidelity Steady-State Aeroelastic Analysis and Derivative Computations," *AIAA Journal*, Vol. 52, No. 5, May 2014, pp. 935–951.

[11]Kenway, G. K. W. and Martins, J. R. R. A., "Multi-Point High-Fidelity Aerostructural Optimization of a Transport Aircraft Configuration," *Journal of Aircraft*, Vol. 51, No. 1, January 2014, pp. 144–160.

[12]Economon, T. D., Palacios, F., and Alonso, J. J., "A coupled-adjoint method for aerodynamic and aeroacoustic optimization," *12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSM Multidisciplinary Analysis and Optimization Conference*, Indianapolis, IN, September 2012, AIAA 2012-5598.

[13]Moore, K. T., "Calculation of Sensitivity Derivatives in an MDAO Framework," *14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA, Indianapolis, IN, September 2012.

[14]Marriage, C. J. and Martins, J. R. R. A., "Reconfigurable Semi-Analytic Sensitivity Methods and MDO Architectures Within the πMDO Framework," *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Victoria, BC, September 2008, AIAA 2008-5956.

[15]Hwang, J. T., Lee, D. Y., Cutler, J. W., and Martins, J. R. R. A., "Large-Scale Multidisciplinary Optimization of a Small Satellite's Design and Operation," *Journal of Spacecraft and Rockets*, 2014, (In press).

[16]Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262.

[17]Hagberg, A. A., Schult, D. A., and Swart, P. J., "Exploring network structure, dynamics, and function using NetworkX," *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.

[18]Pate, D., Gray, J., and German, B., "A graph theoretic approach to problem formulation for multidisciplinary design analysis and optimization," *Structural and Multidisciplinary Optimization*, 2013, pp. 1–18.

[19]Tarjan, R., "Depth-first search and linear graph algorithms," *SIAM journal on computing*, Vol. 1, No. 2, 1972, pp. 146–160.

[20]Nuutila, E. and Soisalon-Soininen, E., "On finding the strongly connected components in a directed graph," *Information Processing Letters*, Vol. 49, No. 1, 1994, pp. 9–14.

[21]Martins, J. R. R. A. and Lambe, A. B., "Multidisciplinary Design Optimization: A Survey of Architectures," *AIAA Journal*, Vol. 51, 2013, pp. 2049–2075.

[22]Cutler, J., Ridley, A., and Nicholas, A., "Cubesat Investigating Atmospheric Density Response to Extreme Driving (CADRE)," *Proceedings of the AIAA/USU Small Satellite Conference*, 2011.

[23]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*, Vol. 46, August 2012, pp. 273–284.

[24]Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM review*, Vol. 47, No. 1, 2005, pp. 99–131.

[25]Dykes, K., Ning, A., King, R., Graf, P., Scott, G., and Veers, P., "Sensitivity Analysis of Wind Plant Performance to Key Turbine Design Parameters: A Systems Engineering Approach," Tech. Rep. NREL/CP-5000-60920, National Renewable Energy Laboratory, Golden, CO, 2014.

[26]Ning, A., Damiani, R., and Moriarty, P., "Objectives and Constraints for Wind Turbine Optimization," *Journal of Solar Energy Engineering*, 2014, (In press).

[27]Ning, A. and Petch, D., "Design Optimization of Downwind Wind Turbines," *Wind Energy*, 2014, (Forthcoming).

[28]Ning, A. and Dykes, K., "Understanding the Benefits and Limitations of Increasing Maximum Rotor Tip Speed for Utility-Scale Wind Turbines," *Journal of Physics: Conference Series*, 2014, (In press).

[29]"Wind Turbines Part 1: Design requirements," Tech. Rep. IEC 61400-1, International Electrotechnical Commission, 2005.

[30]Ning, A., "A simple solution method for the blade element momentum equations with guaranteed convergence," *Wind Energy*, 2013.

[31]Brent, R. P., "An Algorithm with Guaranteed Convergence for Finding a Zero of a Function," *The Computer Journal*, Vol. 14, No. 4, 1971, pp. 422–425.

[32]Ning, A., "pBEAM Documentation," Tech. Rep. TP-5000-58818, National Renewable Energy Laboratory, September 2013.

[33]Bir, G., "User's Guide to PreComp (Pre-Processor for Computing Composite Blade Properties)," Tech. Rep. TP-500-38929, National Renewable Energy Laboratory, January 2006.

[34]Jonkman, J., Butterfield, S., Musial, W., and Scott, G., "Definition of a 5-MW Reference Wind Turbine for Offshore System Development," NREL/TP-500-38060, National Renewable Energy Laboratory, Golden, CO, Feb 2009.

[35]Resor, B. R., "Definition of a 5MW/61.5m Wind Turbine Blade Reference Model," SAND2013-2569, Sandia National Laboratories, April 2013.