Using Graph Coloring To Compute Total Derivatives More Efficiently in OpenMDAO

Justin S. Gray, Tristan A. Hearn
NASA Glenn Research Center, Cleveland, OH, 44139

Bret A. Naylor
DB Consulting Corp

When they are applicable, gradient based optimization algorithms are the most efficient way to solve design optimization problems. Although gradient based methods are generally efficient, they can be made significantly more so through the usage of analytic techniques to compute the necessary total derivatives. The traditional forward (direct) and reverse (adjoint) analytic techniques have computational costs that scale linearly with the number of design variables and the number of constraints, respectively. In this work, we present an application of a graph coloring algorithm to the analytic techniques for computing total derivative Jacobians in order to achieve much better computational scaling than the pure analytic methods can provide alone. A detailed theoretical explanation of how coloring algorithms interact with analytic derivative methods is presented that illustrates specific types of sparsity patterns that must be present in total derivative Jacobians in order for this coloring technique to be effective. The new technique has been implemented as a feature in the OpenMDAO framework and the implementation is demonstrated on two example problems. The performance on the example problems up to 50% reduction in compute cost for optimizations with bi-directional coloring compared to traditional constraint aggregation. Additionally, the results show how coloring technique alleviates some of the numerical difficulties that constraint aggregation can cause, leading to the ability to solve larger problems. It is expected that the new method will have wide applicability to multidisciplinary optimization problems, and that its availability in OpenMDAO will offer significant computational savings for users without the need for them to implement the coloring algorithm themselves.

I. Introduction

Gradient based optimization is an extremely efficient tool for solving high dimensional optimization problems with continuous design variables. Gradient based optimizers require functions that compute the objective and constraints as well as the derivatives of those functions with respect to the design variables. A general optimization problem formulation is given in Table 1.

Table 1 Generic optimization problem formulation

	Variable/Function	Description
minimize	$\mathcal{F}(x)$	objective
with respect to	x	design variables
such that	$\mathcal{R}(x, y) = 0$	governing equations
	$\mathcal{G}(x, y) < 0$	inequality constraints

In Table 1, the governing equations $\mathcal{R}(x, y) = 0$ must be solved for any given value of x. While you could converge the governing equations with the optimizer, more commonly a nonlinear solver is used so that an evaluation of $\mathcal{F}(x)$ and $\mathcal{G}(x)$ and internal solver iterations to find the values of y that satisfy the governing equations. Computing the functions of interest — i.e. $\mathcal{F}(x)$ and $\mathcal{G}(x)$ — is a fundamental goal of any analysis tool and hence can be essentially taken for granted in the context of optimization. Not all analysis tools provide built functionality for computing analytic derivatives [1], so this is generally one of the more challenging computational requirements when using gradient based optimization. There are two general approaches taken to compute derivatives:

1) monolithic numerical approximation using finite-difference or complex-step [2]
2) direct or adjoint analytic methods [3]

The monolithic numerical approach is easier to implement, but far more costly and potentially less accurate. The analytic method, while more difficult to implement, offers significantly greater computational performance and higher accuracy. In most cases, optimization problems with more than 100 design variables demand the use of the analytic derivative methods to achieve reasonable computational efficiency, although there are exceptions to that rule of thumb for problems with specific kinds of sparsity patterns or when parallel computational resource can be used for numerical approximations.

In the majority of cases, the computational cost of computing the gradients is the dominant factor in the overall cost of an optimization. Monolithic numerical approximations require performing one complete nonlinear evaluation for each design variable, so the cost scales linearly with the number of variables being optimized. The analytic derivative methods rely on the solution to a linear system of equations, contructed with partial derivatives from the nonlinear analysis, which must be solved once per design variable for the direct method and once per function of interest for the adjoint method.

The cost of the direct method scales the same as the monolithic numerical approximations — linearly with the number of design variables — but the cost of the adjoint method scales linearly with the number of quantities of interest. This means that if the optimization has a relatively small number of constraints the adjoint method offers significant computational savings, and that the compute cost for derivatives can be nearly independent of the number of design variables. The computational efficiency of the adjoint analytic derivative method has made it possible to solve optimization problems with very large design spaces and that require very expensive nonlinear analyses (e.g. aerodynamic shape optimization, structural optimization, and coupled aerostructural optimization).

Not all optimization problems naturally have a small number of constraints, which would seem to limit the applicability of the adjoint method. However, constraint aggregation techniques [4–7] offer a way to manipulate the problem definition into a form more suitable to apply the adjoint method. Though constraint aggregation has been widely and successfully employed in a huge range of different cases, it is not universally applicable. We will discuss the types of situations where aggregation can not be effectively applied in more detail in Section II, but for now it is sufficient to note that you can not always use it. In these cases, the computational advantage of the adjoint method breaks down, and the cost savings of the analytic derivative methods applied in their canonical form are significantly reduced.

In this work, we present an alternative to constraint aggregation via new technique to simultaneously compute analytic derivatives for multiple variables (direct) or multiple quantities of interest (adjoint) at the same time. The new technique uses a bidirectional graph-coloring algorithm for computing separable sets of variables and quantities of interest which can then be used with a combination of direct and adjoint analytic derivative methods to achieve a scalable analytic derivative approach that can be applied to a generic optimization problem. In Section III we discuss why this graph-coloring approach offers advantages when the optimization problem formulation contains a specific sparsity structure in the matrix of total derivatives, called the total derivative Jacobian, used by the optimizer.

We also demonstrate that the new technique offers practical — and not just theoretical – computational savings by implementing the new technique within the OpenMDAO framework[1] and comparing its performance to a traditional constraint aggregation approach on an example problem with a large number of constraints. The results from the example problem show that the graph-coloring approach out performs the constraint aggregation approach both in terms of wall time and number of total optimization iterations.

OpenMDAO is a open-source computational framework specializing in gradient based optimization with analytic derivatives, with support for efficient serial and parallel model execution. It is the framework's support for analytic derivatives that makes it particularly relevant for this work. The framework's automatic analytic derivative features are implemented in implemented in a generalized and problem-independent fashion, following the theoretical and algorithmic work of Hwang and Martins [8]. The generalized implementation is what allowed the coloring technique presented in this paper to be implemented in a similarly problem-independent manner. The new technique has been integrated into the OpenMDAO version 2.6 release and can now be employed by OpenMDAO users without requiring them to implement it themselves.

Although the results presented in this paper show that total derivative coloring has computational advantages for problems large block-diagonal sections in their Jacobian, we do not mean to imply that coloring will always be a better option then constraint aggregation. Nonetheless, the computational savings presented here validate the usefulness of the total derivative coloring, and with the generalized implementation in OpenMDAO users can easily test its effectiveness on their own problems.

II. Analyzing the Computational Cost of Analytic Derivative Methods

The central concept behind analytic derivative methods is that partial derivatives (i.e. $\partial\mathcal{F}/\partial x$, $\partial\mathcal{G}/\partial x$, $\partial\mathcal{G}/\partial y$, $\partial R/\partial y$) can be computed relatively inexpensively, but total derivatives (i.e. $d\mathcal{F}/dx$, $d\mathcal{G}/dx$) are both expensive and difficult to compute directly. So, analytic derivative methods compute total derivatives indirectly via the solution to a linear system constructed using only partial derivatives. Detailed derivations and theoretical analysis of the analytic derivative methods was presented by Martins and Hwang [3, 8], and more importantly they present a new generalization of these methods called the Unified Derive Equations (UDE) which are relied on heavily by the OpenMDAO framework to implement the analytic derivative functionality. There are two basic forms of the analytic derivative approach: direct and adjoint. The direct analytic method requires one linear solve per design variable in the x array. This method is given as

$$\begin{aligned}\frac{\mathrm{d}\mathcal{F}}{\mathrm{d}x} &= \frac{\partial\mathcal{F}}{\partial x} + \frac{\partial\mathcal{F}}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x}\,,\\ \frac{\mathrm{d}y}{\mathrm{d}x} &= -\left[\frac{\partial\mathcal{R}}{\partial y}\right]^{-1}\left[\frac{\partial\mathcal{R}}{\partial x}\right]\,.\end{aligned}\tag{1}$$

The adjoint analytic method requires one linear solve per objective and constraint variable. This method is given as

$$\begin{aligned}\frac{\mathrm{d}\mathcal{F}}{\mathrm{d}x} &= \frac{\partial\mathcal{F}}{\partial x} + \psi^T\left[\frac{\partial\mathcal{R}}{\partial x}\right]\,,\\ \psi &= -\left[\frac{\partial\mathcal{R}^T}{\partial y}\right]^{-1}\left[\frac{\partial\mathcal{F}^T}{\partial y}\right]\,.\end{aligned}\tag{2}$$

Equations (1) and (2) are the canonical form of the analytic derivative methods, but both can be generalized into the single UDE:

$$\left[\frac{\partial\bar{\mathcal{R}}}{\partial u}\right]\left[\frac{\mathrm{d}u}{\mathrm{d}r}\right] = [\mathcal{I}] = \left[\frac{\partial\bar{\mathcal{R}}}{\partial u}\right]^T\left[\frac{\mathrm{d}u}{\mathrm{d}r}\right]^T\,.\tag{3}$$

Equation (3) relies on a mathematical transformation that converts the explicit functions $\mathcal{F}(x)$, $\mathcal{G}(x,)$ and implicit function $\mathcal{R}(x, y) = 0$ into a single combined implicit function $\bar{\mathcal{R}}(u) = 0$. The left hand side of Equation (3) is called the forward mode — analogous to the direct method — and requires one linear solution per design variable. The right hand side of Equation (3) is called the reverse mode — analogous to the adjoint method— and requires one linear solution per objective or constraint. In the context of this work, the inherent value of the UDE formulation is that it offers the necessary flexibility to implement forward (direct) and reverse (adjoint) methods for any model with out any customized implementation by the user. OpenMDAO handles the mathematical transformation necessary to apply the UDE for the user, so that they may work in the more natural explicit formulation they are used to, and then uses the UDE to automatically solve for analytic derivatives of any model in either the forward or reverse forms without any customized implementation by the user.

Although the UDE is a useful generalization of the direct and adjoint methods, the formulation still requires the same number of linear solutions as those methods and hence has the same overall computational cost. So, on its own it provides not additional help for attacking problems that have a large number of design variables and constraints, since that would still require a large number of linear solves to compute the total derivatives. Two commonly addressed optimization problems provide good examples of situations where there are large numbers of both design variables and constraints: structural optimization and pseudo-spectral optimal control.

Structural optimization problems are an example where constraint aggregation works very well. For typical problem formulation we would seek to size the thickness of panels throughout a structure in order to minimize the mass, subject to maximum allowable stress and buckling constraints on each panel. If a finite element analysis is used as the structural analysis tool, then even a moderately coarse model may have 100's or 1000's of panels and consequently a similar number of design variables and constraints. Kreisselmeier-Steinhauser (KS) aggregation can be used to combine all separate stress constraints into a single scalar constraint, and similarly for the buckling constraints[9, 10]. The KS function is given as:

$$\mathrm{KS}\left[\mathcal{G}_j(x,y)\right] = \frac{1}{\rho}\ln\left[\sum_{j}^{n_g} e^{\rho g_j(x,y)}\right] \tag{4}$$

A simple way to think of the KS function is that is behaves like a smooth — and hence differentiable — maximum function that returns an approximation of the worst constraint violation in the aggregated set. Another key aspect of the KS function is that it is slightly conservative, meaning that when the aggregated constraint is satisfied the input constraint values will be slightly inside the actual constraint boundary. KS aggregation works well when you have a large set of constraints of which only a small sub-set are active. However, if most or all of the constraints are active the derivatives of the KS function with respect to each input constraint grow very large and can create numerical instabilities in the optimization. In the best case, these numerical instabilities will cause an optimizer to take more iterations to find an answer. In the worst case, they will prevent the optimizer from converging at all. KS aggregation is useful for structural optimization because we can reasonably expect that only a small subset of the structure will actually see the maximum stress or be subject to buckling in the final design.

On the other hand, KS aggregation does not work well for pseudo-spectral optimal control problems. These problems seek to find some control schedule that will give the best performance (e.g. lowest fuel burn, shortest time, etc.) for a system that is modeled via the numerical integration of an ordinary differential equation. Pseudo-spectral optimal control formulations involve the use of collocation methods that introduce a large number of equality constraints (i.e. governing equations) to the problem [11, 12]. The optimal control community has generally sought to use the optimizer itself to converge the governing equations, which results in an optimization problem with a large number of equality constraints that will all be active at the converged solution. Since all of the collocation constraints will be active at the final design, a KS aggregation poses unacceptable numerical trade-offs for this kind of problem and aggregation is not effective.

When aggregation is not applicable, but there remains a large number of design variables and constraints, a new technique is required to efficiently compute the necessary derivatives for optimization. The graph-coloring technique we present in this paper was inspired by work the work of Betts and Huffman to combine simple graph-coloring algorithms with the monolithic numerical technique to solve optimal control problems [13, 14]. They found that pseudo-spectral problems created a block-diagonal structure in the total derivative Jacobian that could be easily exploited to perform monolithic numerical approximations for multiple design variables simultaneously, thus dramatically reducing the computational cost for computing derivatives. As we will show in the next section, this very same idea can be applied to the forward (direct) analytic derivative method to achieve similar savings. However, if you rely solely on the forward method, then the range of problems for which graph-coloring offers any advantage is very limited. To work around this, we also combine graph-coloring on the reverse (adjoint) formulation to create a bi-directional coloring approach that is much more broadly applicable. Next we present the theoretical details of how this new approach works.

III. Combining Graph-Coloring with Analytic Derivative Methods

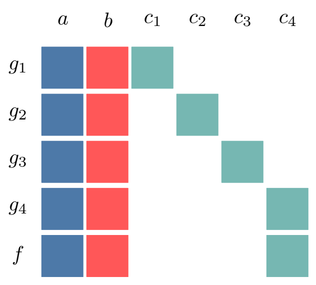
This section provides a short description of how a coloring works, how it can be applied to computing total derivatives with the UDE, and why bi-directional coloring is especially advantageous in this application. This is a very brief introduction to the concepts, intended to provide a common frame of reference for the readers who may not be familiar with coloring. For a much more complete treatment of the subject of graph-coloring, we refer you to the works of Coleman and Verma [15] and Gebremedhin et al [16]. The key concept to understand from this section is how specific sparsity patterns in a total derivative Jacobian enable the use of either single directional or bi-directional coloring.

A. Applying Coloring to the Unified Derivative Equation

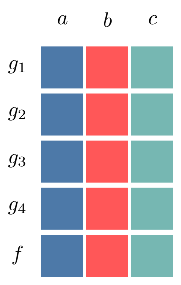
Consider the optimization problem defined in Table 2, with one objective function ($f = \mathcal{F}$), a set of inequality constraints ($g_i = \mathcal{G}_i$), and a set of design variables (a, b, c_i). Here the $()_i$ subscript indicates that there are some number (N), greater than one, of constraints/variables. For the purposes of explaining coloring, we'll assume that $N = 4$. An important detail in this notional problem is that the constraint functions, g_i are each a function of a, b and only their associated c_i variable.
Equally important to note is that the objective function is only a function of the last entry in the c vector, c_N. This

creates a block-diagonal structure in the total derivative Jacobian, or put more formally

$$
\begin{aligned}
\mathrm{d}g_i/\mathrm{d}c_j &= 0 \text{ for } i \neq j \,, \\
\mathrm{d}f/\mathrm{d}c_i &= 0 \text{ for } i \neq N \,.
\end{aligned} \tag{5}
$$



(a) Six individual solutions



(b) Three forward colored solutions

Fig. 1 Total derivative Jacobian structure for notional model where the model outputs are all forward colorable with respect to the c_i inputs. The left image shows the full Jacobian structure with coloring to indicate the potential for simultaneous solutions. The right image shows the collapsed Jacobian structure that takes advantage of the coloring.

The total derivative Jacobian for this optimization problem is shown in Figure 1(a), where the block-diagonal pattern is clearly illustrated. This is the general structure — seen in most pseudo-spectral optimal control formulations — that Betts and Huffman [14] exploited with their forward coloring technique. Forward coloring operates with respect to the design variables. For the problem in Table 2, a and b both affect all the outputs so they create dense columns in the Jacobian which each require their own color as shown in Figure 1. The c variable, thanks to the diagonal structure, can be grouped into a single color meaning that all of those variables can be worked with simultaneously in the forward mode of the UDE. Graphically, this is represented in Figure 1(b), where the whole c vector is collapsed into a single column. This collapsing is only possible since none of the green blocks overlap each other when combining the columns together. The collapsed total derivative Jacobian in Figure 1(b) is now only three columns wide, indicating that it requires three solutions of the forward form of Equation (3) — the left equation — where as the non-collapsed form would require six forward solutions.

When combining multiple total derivative Jacobian columns together, the associated columns of the identity matrix in Equation (3) also get combined. Normally this would result in an undesirable linear combination of total derivatives. For example, the combined forward mode solves for c_1, c_2, c_3, c_4 from the notional problem in Table 2 would give

$$
\begin{bmatrix}
\vdots \\
\frac{\mathrm{d}g_1}{\mathrm{d}c_1} + \frac{\mathrm{d}g_1}{\mathrm{d}c_2} + \frac{\mathrm{d}g_1}{\mathrm{d}c_3} + \frac{\mathrm{d}g_1}{\mathrm{d}c_4} \\
\frac{\mathrm{d}g_2}{\mathrm{d}c_1} + \frac{\mathrm{d}g_2}{\mathrm{d}c_2} + \frac{\mathrm{d}g_2}{\mathrm{d}c_3} + \frac{\mathrm{d}g_2}{\mathrm{d}c_4} \\
\frac{\mathrm{d}g_3}{\mathrm{d}c_1} + \frac{\mathrm{d}g_3}{\mathrm{d}c_2} + \frac{\mathrm{d}g_3}{\mathrm{d}c_3} + \frac{\mathrm{d}g_3}{\mathrm{d}c_4} \\
\frac{\mathrm{d}g_4}{\mathrm{d}c_1} + \frac{\mathrm{d}g_4}{\mathrm{d}c_2} + \frac{\mathrm{d}g_4}{\mathrm{d}c_3} + \frac{\mathrm{d}g_4}{\mathrm{d}c_4} \\
\frac{\mathrm{d}f}{\mathrm{d}c_1} + \frac{\mathrm{d}f}{\mathrm{d}c_2} + \frac{\mathrm{d}f}{\mathrm{d}c_3} + \frac{\mathrm{d}f}{\mathrm{d}c_4}
\end{bmatrix}
=
\left[\frac{\partial \bar{\mathcal{R}}}{\partial u}\right]^{-1}
\begin{bmatrix} \vdots \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} . \tag{6}
$$

Table 2 Notional optimization problem formulation that is suitable for total derivative coloring

	Variable/Function	Description
minimize	$f = \mathcal{F}(a, b, c_{n_i})$	objective
with respect to	a, b, c_i	design variables
subject to	$g_i = \mathcal{G}_i(a, b, c_i) < 0$	constraints

However, from Equation (5) we know that all cross terms are 0 (i.e. terms where $i \neq j$), so the problematic linear combinations simplify down to just the needed total derivatives:

$$\begin{bmatrix} \vdots \\ \frac{dg_1}{dc_1} + \frac{dg_1}{dc_2} + \frac{dg_1}{dc_3} + \frac{dg_1}{dc_4} \\ \frac{dg_2}{dc_1} + \frac{dg_2}{dc_2} + \frac{dg_2}{dc_3} + \frac{dg_2}{dc_4} \\ \frac{dg_3}{dc_1} + \frac{dg_3}{dc_2} + \frac{dg_3}{dc_3} + \frac{dg_3}{dc_4} \\ \frac{dg_4}{dc_1} + \frac{dg_4}{dc_2} + \frac{dg_4}{dc_3} + \frac{dg_4}{dc_4} \\ \frac{df}{dc_1} + \frac{df}{dc_2} + \frac{df}{dc_3} + \frac{df}{dc_4} \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{dg_1}{dc_1} + 0 + 0 + 0 \\ 0 + \frac{dg_2}{dc_2} + 0 + 0 \\ 0 + 0 + \frac{dg_3}{dc_3} + 0 \\ 0 + 0 + 0 + \frac{dg_4}{dc_3} \\ 0 + 0 + 0 + \frac{df}{dc_4} \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{dg_1}{dc_1} \\ \frac{dg_2}{dc_2} \\ \frac{dg_3}{dc_3} \\ \frac{dg_4}{dc_4} \\ \frac{df}{dc_4} \end{bmatrix} \tag{7}$$

So after this single combined linear solve all of the green entries in the total derivative Jacobian shown in Figure 1 are known.

B. Forward versus Reverse Coloring

As discussed in the previous section, the total derivative Jacobian illustrated in Figure 1(a) is well suited to forward coloring. The Jacobian has two dense columns, and then a large block-diagonal section which results in just three colors. The block-diagonal structure is key to be able to get a large benefit from a coloring technique, but the presence of dense columns is also significant. Although a dense column forces you to have a single-variable color, it does not otherwise preclude forward coloring.

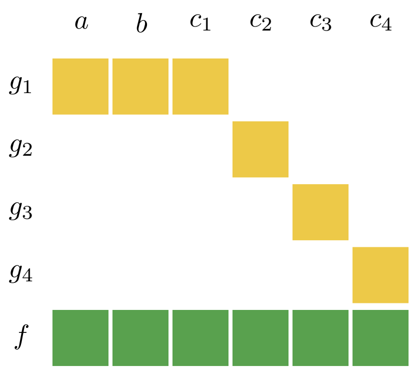
If, instead of dense columns, an optimization problem had dense rows in the total derivative Jacobian, then forward coloring no longer works. Examples of problems that would exhibit some dense rows, combined with large block-diagonal sections would be multipoint problems where the same analysis is run at different conditions (e.g. multiple CFD analyses that are run for the same geometry, but at different flow conditions [17–19]). A notional total derivative Jacobian for an optimization problem with dense rows is shown in Figure 2. Visually you can see that forward coloring doesn't work with dense rows, because if you were to collapse columns of the total derivative Jacobian then there would be overlapping blocks. Mathematically, this means that you can no longer take advantage of off-diagonal terms in the forward solutions being 0 to simplify the linear combinations that appear when columns are combined. However, using the reverse mode of Equation 3 — the right hand side of that equation — allows for solutions that give derivatives, corresponding to a specific row of the total derivative Jacobian. Therfore if a block-diagonal structure exists in the Jacobian, but there are also dense rows, then we can use reverse mode coloring to combine multiple rows together. Figure 2(b) illustrates the reverse mode coloring, with the five rows from Figure 2(a) collapsed down to just two combined rows. This means that we can compute the complete total derivative Jacobian with just two linear solutions, instead of five.

C. Bi-directional Coloring

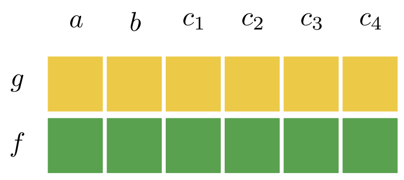
Problems that have a block-diagonal structure in the total derivative Jacobian, combined with either only dense rows or only dense columns are very convenient because they allow for uni-directional coloring. Unfortunately, in general you can not count on problems that exhibit such useful sparsity structures in their total derivative Jacobian. It is likely that both dense rows and dense-columns will exist within the same total derivative Jacobian, so that even if there was also a large block-diagonal section you could not take advantage of it with either pure forward of reverse coloring.

However, one of the most fundamental advantages of using the UDE form of analytic derivative methods is that both forward and reverse solves can be used interchangeably once the partial derivatives are known. Thus, it becomes possible to use both the forward and reverse modes together to solve for different parts of the total derivative Jacobian. This enables the use of bi-directional coloring schemes — ones that use a mixture of forward and reverse coloring — to reduce the total number of linear solves required to compute a full total derivative Jacobian. A notional optimization problem — different than the ones given for the two uni-directional examples — is presented in Figure 3. This total derivative Jacobian is broken up into two pieces, one solved in forward mode and the other in reverse mode. The single forward colored column is shown in Figure 3(b) and the two reverse colored rows are shown in Figure 3(c). The whole Jacobian can be solved for with a total of three linear solutions, compared to five for a pure reverse mode or six for a pure forward mode.

There are several details from Figure 3 that are important to note. First, note that df/da is computed twice; once in the forward solution and once in the reverse solution. This duplication results in a small overhead for the bidirectional

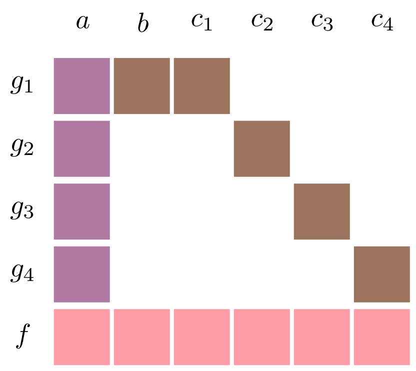


(a) Six individual solutions

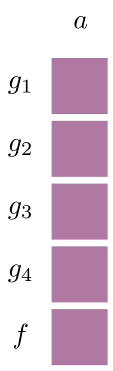


(b) Two reverse colored solutions

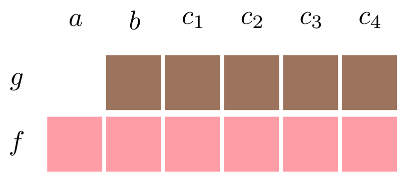
Fig. 2 Total derivative Jacobian structure for notional model where the model outputs are all reverse colorable with respect to the g_i outputs. The top image shows the full Jacobian structure with coloring to indicate the potential for simultaneous solutions. The bottom image shows the collapsed Jacobian structure that takes advantage of the coloring.



(a) Six individual solutions



(b) One forward colored solution



(c) Two reverse colored solutions

Fig. 3 Total derivative Jacobian structure for notional model where the model outputs are all reverse colorable with respect to the g_i outputs. The top image shows the full Jacobian structure with coloring to indicate the potential for simultaneous solutions. The bottom image shows the collapsed Jacobian structure that takes advantage of the coloring.

coloring since the same value is computed multiple times, but it is unavoidable whenever there is an intersection of a dense row and a dense column in the total derivative Jacobian. The wasted computational effort must be balanced against the reduction in the total number of linear solves to determine if a bi-directional coloring will yield savings, but in most cases the overhead from this issue is likely to be negligible.

Second, note that the block-diagonal section was chosen to be colored in reverse mode. A block-diagonal section could be colored in either direction, but one direction may offer fewer overall colors. For this notional problem, forward coloring of this section would have given two colors while reverse coloring gave only one, so reverse was preferred.

D. Graph Coloring Algorithms

In all three of the preceding notional optimization problems, the total derivative Jacobians were intentionally ordered in such a way as to make the potential colorings obvious. In order for a coloring algorithm to be applied, a sparsity pattern in the form of one of the three shown in Figures 1, 2 or 3 must exist. If it doesn't exist, you can not apply coloring. Unfortunately for a generic problem formulation, the ordering of the total derivative Jacobian may not make the block-diagonal structure obvious. Even worse, in general you can not know a priori if a block-diagonal pattern exists at all to know if it is worth re-ordering it to find the coloring.

To solve this difficulty, many coloring algorithms have been developed to compute a small number of colors for a given Jacobian. The very first coloring algorithm dates back to the 1800's when it was proven that any two dimensional map could be colored with no more than five colors without any two adjacent regions sharing the same color. The advent of modern computers enabled Appel and Haken [20] to develop a computer-assisted-proof — the first ever example of this kind of proof — of a more efficient four-color theorem. The computer-assisted-proof was controversial and not widely accepted at the time, though in 1996 [21] and 2005 [22] more rigorous proofs were presented that led to its general acceptance.

Modern graph coloring algorithms for Jacobians represent a generalization of the map coloring problem, and can be applied to a much wider range of problems. However, the difficulty of proving the four-color theorem is illustrative of an important point. Modern graph coloring algorithms are, except for some special cases, heuristic in nature and thus can not provably find the minimum possible number of colors for any given problem. For many years manual coloring of maps showed that only four colors were necessary, but it was not possible to prove that this would always be the case for any two dimensional map until very recently. Similarly, Jacobian coloring algorithms have not been proven to find the best possible coloring for any given problem. Moreover multiple different coloring algorithms exist and work on certain special classes of Jacobian. The development of coloring algorithms for Jacobians, to date, has focused primarily on applications for partial derivative Jacobians. An excellent overview of coloring algorithms is provided by Gebremedhin et al. [16] in their paper titled "What Color Is Your Jacobian? Graph Coloring for Computing Derivatives". A recent example of an efficient parallel coloring algorithm was presented by He et al. [23] in their work on developing adjoint analytic derivatives for the OpenFOAM computational fluid dynamics solver.

The coloring of total derivative Jacobians is, computationally speaking, a much easier task than partial derivative Jacobian coloring. This is due to the relative size of the two types of Jacobians, with the total derivative Jacobian being much smaller. Due to the relatively lower computational challenge, parallel coloring algorithms are not necessary for total derivative Jacobians and the simpler serial algorithms can be relied on. The OpenMDAO framework implements the direct determination flavor of the bi-directional coloring algorithm developed by Coleman and Verma [15].

IV. Testing the Effectiveness of Total Derivative Coloring

Since the idea for the bi-directional coloring of total derivative Jacobians was originally inspired by the forward-coloring applications to pseudo-spectral optimal-control problems, it is natural to examine the effectiveness of the new method on those type or problems. In a companion paper to this one, "Optimal Control within the Context of Multidisciplinary Design Analysis and Optimization"[24], Falck and Gray present Dymos, a general optimal-control library, built on top of the OpenMDAO framework, that uses the analytic derivatives and generalized bi-directional coloring functionality[24]. They examined several canonical optimal control problems, including the famous brachistochrone problem. The brachistochrone, originally posed by Johann Bernoulli and solved analytically by Isaac Newton, Jakob Bernoulli, Gottfreid Leibnitz, Ehrenfried Walther von Taschirnhaus, and Guillaume de l'Hopital in the late 1690's, serves as one of the basic benchmarks for any pseudo-spectral optimal control solution method. Falck and Gray present a detailed examination of bi-directional coloring applied to a modified brachistochrone problem that includes a constraint specifically designed to create a dense row in the total derivative Jacobian and prevent forward-coloring. Although the brachistochrone problem is an admittedly simple problem with very low computational cost, the results still showed a

40% reduction in wall time with a bi-directional coloring.

A. Defining an Air Traffic Control Test Problem

In this paper we focus on the usage of coloring as compared to constraint aggregation. As explained in Section II, constraint aggregation is not generally applicable to the collocation equality constraints present in a pseudo-spectral optimal control problems. However, it is possible to formulate pseudo-spectral problems with additional inequality constraints that can be aggregated. Here we present a simplified air-traffic-control problem, built with Dymos, that includes a set of aircraft separation constraints where aggregation can be effectively applied. The result is a problem that uses both coloring and constraint aggregation simultaneously, which can be used to study the relative impact of both methods.

The problem is formed by considering a circular field which a number of aircraft must cross along a pre-defined linear trajectory. Each aircraft has a specific departure time and a quadratic velocity profile chosen by the optimizer. The objective is to minimize the time requires for all aircraft to complete their trips, while ensuring that no two aircraft ever collide with each other. The problem can be scaled up or down by increasing the number of vehicles considered, which varies both the number of design variables and the number of overall constraints. The initial condition and linear trajectories for a 20 aircraft version of this problem is show in in Figure 4.

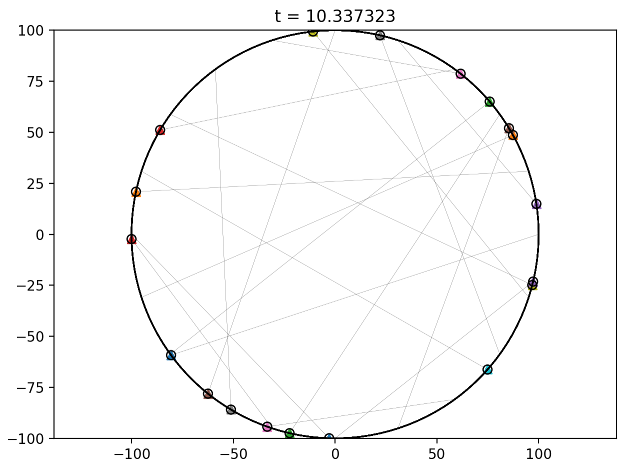


Fig. 4 Example of the simulated airspace linear trajectory problem.

The trajectory of each aircraft is solved with the standard pseudo-spectral formulation used in Dymos, and as such creates some inherent block-diagonal structure in the total derivative Jacobian that can be colored. We assume that each aircraft has constant mass, so the equations of motion are simply

$$\dot{x} = V\cos(\theta) \tag{8}$$

$$\dot{y} = V\sin(\theta)\,, \tag{9}$$

Where x and y are the positions in Cartesian space, V is the magnitude of the aircraft's velocity vector, $\dot{x}$ and $\dot{y}$ are the time-rate-of change of those positions, and θ is the pre-defined angle for the trajectory.

The aircraft separation constraints create a more challenging problem from an optimization standpoint, since a naive implementation that constrained the distance from each aircraft to all others would have N^2_{aircraft} constraints that form a totally dense block in the Jacobian for each time step in the time integrations. This obviously would scale very poorly, but since the trajectories are linear and pre-determined we can know with certainty that some of them can never possibly intersect. By accounting for this built in sparsity, we can reduce the number of constraints from N^2_{aircraft} to roughly N_{aircraft} and create an opportunity to color the aircraft separation constraint for better performance.

The formal optimization problem formulation for this simplified air traffic control problem is given in Table 3, where the subscripts i and j indicated loops over the set of all aircraft and the subscript k indicates loops over all times. The full separation constraint, $\mathcal{G}_{ijk} > 0$, an $i \times j$ set of constraints (though some are known to be exactly zero) for each of the k time steps. The aggregated separation constraint, $KS\left[\mathcal{G}_{ij}\right]_k > 0$, is a vector of k aggregated separation constraints so that each time step has a single scalar aggregated constraint value. Though both forms of the constraint are shown in Table 3, only one is used at a time.

Table 3 Notional optimization problem formulation that is suitable for total derivative coloring

	Variable/Function	Description
minimize	total duration	time to get all aircraft across the circle
with respect to	x_{ik}	x location vs time for each aircraft
with respect to	y_{ik}	y location vs time for each aircraft
	V_{ik}	velocities that define the quadratic velocity profile of each aircraft
subject to	$\mathcal{R}_{ik} = 0$	vector of pseudo-spectral collocation constraints for each aircraft
	$\mathcal{G}_{ijk} > 0$	aircraft separation constraints
	$KS\left[\mathcal{G}_{ij}\right]_k > 0$	vector aggregated separation constraints (one for each time)

Although this problem is obviously a contrived and simplified example, it has many relevant direct-analogs in real world problems. Any form of bin-packing problem or spatial arrangement problem[25] is an obvious use case. Additionally any problem that requires maintaining separation of a set of bodies is also directly applicable, such as maintaining a minimum separation distance between all aircraft within an airspace, all spacecraft in a set of intersecting orbits, or all wind turbines within a wind farm[26, 27].

B. Optimization Performance Results of Coloring vs Aggregation

The example aircraft trajectory problem defined above was tested for $N_{\text{aircraft}} = 5, 10, 20, 40, 80$ for both colored and aggregated versions of separation constraint — coloring for the pseudo-spectral constraints is used in both. Table 4 gives the number of design variables, constraints, and total linear solves required to the compute total derivative Jacobian for each case. The computational cost of these optimizations is governed by three key aspects:

1) Cost to evaluate the actual analysis
2) Cost to compute the total derivative Jacobian
3) Total number of iterations required by the optimizer

Table 4 Simple air traffic control optimization problem size for $N_{\text{aircraft}} = 5, 10, 20, 40, 80$

N_{aircraft}	No. Design Vars.	No. Total Constraints	No. Separation Constraints
5	267	755	$(10 \times 50) = 500$
10	532	2760	$(45 \times 50) = 2250$
20	1062	10520	$(190 \times 50) = 9500$
40	2122	41040	$(780 \times 50) = 39000$
80	4242	162081	$(3160 \times 50) = 158000$

The relative simplicity of this model makes the cost to compute the nonlinear function (the objective and constraints) effectively zero in this case. The cost of computing the total derivative Jacobian scales with the number of linear solves required and the cost of each linear solve. Table 4 shows that the raw optimization problem has hundreds to thousands of design variables, and thousands to hundreds-of-thousands of constraints. Hence, in its basic form (no coloring or aggregation) there would be a massive number of linear solves to perform to compute total derivatives. The total number of iterations will generally increase with increasing problem size, although a specific scaling rule is highly problem dependent and will vary with the specific optimizer used. For this work, we used the SNOPT optimizer, which uses a scalable and efficient sequential quadratic programming algorithm [28]. The focus of this work was on reducing the cost of computing total derivative Jacobians since that is where the coloring algorithm has its primary effect. However, our results show that the improved numerical stability of the colored problem formulation vs the aggregated one also provides benefits in terms of the number of major iterations as well.

Table 5 provides a detailed breakdown of the number of linear solves requires to compute the total derivative Jacobian for each case examined, including information on how many forward and reverse mode solves are required for each formulation of the problem. The computational scaling of each formulation can be more clearly seen when plotted on a log-log scale, as shown in Figure 5. The number of linear solves for the raw problem scaled linearly with the number of vehicles, giving roughly 53 constraints per aircraft. While linear scaling is certainly better than the quadratic scaling that would have occurred from a naive implementation of the separation constraints, it still results in an unacceptably large number of required linear solves. When we apply the bi-directional coloring algorithm, we can achieve far better scaling with a slope of just 0.5, meaning that the number of total linear solves required grows far slower than the number of aircraft. With constraint aggregation applied to the separation constraint — and coloring applied to the overall Jacobian as well — the total number of linear solves becomes completely independent of the number of aircraft. So, purely from the perspective of the number of linear solves, aggregation of the separation constraint provides a clear advantage. However, even for the 80 aircraft case, the colored formulation requires just 30 total linear solves — about 4 times more than the aggregated formulation but still about 100 times less than the raw problems. If we assume that the two formulations have roughly the same cost to perform a single linear solve, then both colored and aggregated formulations have reduced the problem to a feasible size to solve, though the aggregated problem would appear to be better.

Table 5 Number of linear solves requires to compute total derivative Jacobian for simple air traffic control optimization

N_{aircraft}	Full problem (fwd.)	Total colored (fwd. / rev.)	Total aggregated (fwd. / rev.)	Separation const. (colored / aggregated)
5	267	7 / 1	4 / 4	6 / 3
10	532	9 / 1	4 / 4	8 / 3
20	1062	15 / 1	4 / 4	14 / 3
40	2122	21 / 1	4 / 4	20 / 3
80	4242	29 / 1	4 / 4	28 / 3

As we stated earlier, the total cost of the optimization is a function of both the cost to compute the total derivative Jacobian, and also the number of major iterations required. From the number of linear solves alone, it would be reasonable to guess that the aggregated optimization would be faster than the colored one, although the conservative nature of the KS function would mean that it should yield a higher time required when the separation constraint is active. However, for this problem the numerical challenges posed by KS aggregation ultimately cause the overall optimization cost to be higher because it takes more iterations, especially for case with larger number of aircraft since these are where the separation constraint becomes more restrictive.

Table 6 gives the objective function value and number of major iterations required to find it for N_{aircraft} = [5,10,11,12,13,14,15,16,20,40]. For the 5 and 10 aircraft cases, both the colored and aggregated forms of the problem were solvable and gave the same final objective, because the separation constraint was not active enough to bring the conservativeness or the numerical instability of the KS function into play. Extra cases were run between 10 and 20 aircraft, because this is where the numerical challenges of the KS function start to impede the aggregated formulation. Both formulations were able to find a solution up to 15 aircraft, though in all cases above 10 aircraft the KS function is clearly impacting the quality of the solution to varying degrees and costing significant increase in the number of major iterations. There is a significant out lier for the 14 aircraft case where, although the optimization did converge for the aggregated case it clearly got stuck in a much less effective local optima.

The colored formulation was able to find decent results for the 16, 20, and 40 aircraft cases. However, the aggregated formulation was not able to converge for these cases and only the 16 and 20 aircraft cases got any results at all — though SNOPT reported the problem was ultimately infeasible. SNOPT was not able to find a meaningful solution for the 40 aircraft aggregated case.

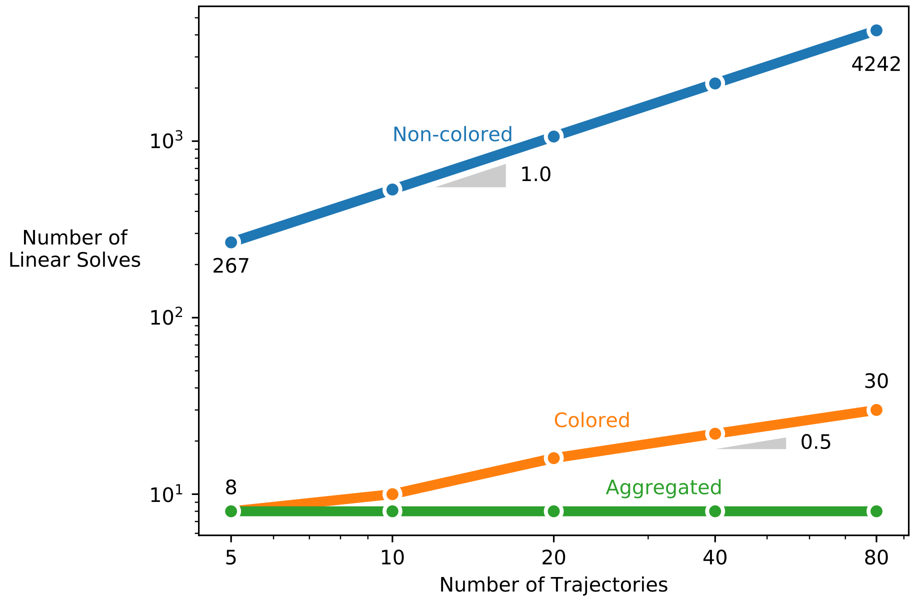


Fig. 5 Number of required linear solves to compute the total derivative Jacobian for the non-colored, colored, and aggregated problem formulations, plotted on a log-log scale.

Table 6 Optimization results comparing the colored and aggregated problem formulations ("*" indicated non-feasible results).

	Colored			Aggregated		
N_{aircraft}	Objective (sec.)	No. of Major iterations	wall time (sec.)	Objective (sec.)	No. of major iterations	wall time (sec.)
5	4.999	14	3.66	4.999	43	13.63
10	5.091	22	9.04	5.091	24	19.39
11	5.091	17	10.90	5.300	38	28.88
12	5.165	20	14.27	6.738	22	31.38
13	5.174	19	21.55	7.178	37	36.60
14	5.112	21	24.35	14.742	40	46.47
15	5.388	20	22.44	5.938	91	98.11
16	5.138	16	24.54	5.091*	91*	74.18
20	5.406	37	47.07	6.182*	62*	170.27
40	8.369	64	398.39	-	-	

V. Conclusions

The generalized implementation of the analytic derivative methods within the OpenMDAO framework makes them dramatically simpler to use, and opens up their range of possible applications for gradient based optimization with analytic derivatives to a much wider set of problems. Analytic derivatives provide a more accurate derivative computation with a potentially significantly lower cost, compared to monolithic finite difference calculations. While the improved accuracy is important, the computational cost of gradient based optimization is typically dominated by the cost of computing derivatives, so it is the computational savings offered by analytic derivatives that is their most important advantage. There are two analytic derivative methods — forward (direct) and reverse (adjoint) — that each perform best in different situations. The forward analytic method requires one linear solve per design variable, while the reverse analytic method requires one linear solve per objective and constraint.

For many optimization problems there are relatively fewer constraints than design variables, and when this is not the case constraint aggregation techniques can be employed to reduce the number of constraints. In general, reverse (adjoint) analytic derivative method has been the most widely applied of the two options. However, not all optimization problems interact well with constraint aggregation and in these cases the computational advantages of the standard analytic derivative methods decline significantly. New applications built in OpenMDAO focusing on models that require the numerical integration of ODE's, such as optimal-control problems, are a good example of situations where aggregation is not necessarily a good option for enabling the use of the reverse (adjoint) analytic derivative method.

In this paper we presented a new technique for combining a bi-directional coloring algorithm with both the forward and reverse analytic derivative methods to achieve good computational and numerical performance for problems where the total derivative Jacobian has a large block-diagonal structure that can be exploited. In companion work, Falck and Gray, showed how this new technique affords good computational performance to the general optimal-control library Dymos [24] (built on top of OpenMDAO). In this work we examined the performance of a simplified air traffic control problem, built with Dymos, that involved a large number of separation constraints to ensure that no aircraft got too close to any other aircraft. The performance results show that the bi-directional coloring technique was able to reduce the computational cost of computing the total derivative Jacobian by up to two orders of magnitude, which made it feasible avoid aggregating all of the separation constraints together. By not aggregating, the optimizer was able to find a better overall answer with the same computational budget than the aggregated problem formulation when up to 15 aircraft were considered simultaneously. For cases with more than 15 aircraft considered simultaneously, the optimizer could not find a valid solution with the aggregated separation constraint, but was able to solve it using the colored separation constraint.

The new bi-directional coloring technique has been implemented as part of the standard release of OpenMDAO V2.6. This work has shown its applicability and computational advantages compared to aggregation, hence we encourage practitioners to consider formulating their optimization problems to take advantage of this new technique.

VI. Acknowledgment

This work was performed with support from NASA's ARMD Transformational Tools and Technology Program. The authors would also like to thank Rob Falck for his many helpful discussions, advice, and input on the graph coloring capability in OpenMDAO.

References

[1] Gray, J. S., Hwang, J. T., Martins, J. R. R. A., Moore, K. T., and Naylor, B. A., "OpenMDAO: An Open-Source Framework for Multidisciplinary Design, Analysis, and Optimization," *Structural and Multidisciplinary Optimization*, Vol. 59, 2019, pp. 1075–1104. doi:10.1007/s00158-019-02211-z.

[2] 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. doi:10.1145/838250.838251.

[3] 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, 2013, pp. 2582–2599.

[4] Kreisselmeier, G., and Steinhauser, R., "Systematic Control Design by Optimizing a Vector Performance Index," *International Federation of Active Controls Symposium on Computer-Aided Design of Control Systems, Zurich, Switzerland*, 1979. doi:10.1016/S1474-6670(17)65584-8.

[5] Kennedy, G. J., and Hicken, J. E., "Improved Constraint-Aggregation Methods," *Computer Methods in Applied Mechanics and Engineering*, Vol. 289, 2015, pp. 332–354. doi:10.1016/j.cma.2015.02.017, URL `http://gkennedy.gatech.edu/wp-content/uploads/2015/03/aggregation_methods.pdf`.

[6] Lambe, A. B., Martins, J. R. R. A., and Kennedy, G. J., "An Evaluation of Constraint Aggregation Strategies for Wing Box Mass Minimization," *Structural and Multidisciplinary Optimization*, Vol. 55, 2017, p. 257–277.

[7] Jonsson, E., Mader, C. A., Kennedy, G. J., and Martins, J. R. R. A., "Computational Modeling of Flutter Constraint for High-Fidelity Aerostructural Optimization," *AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, AIAA, AIAA, San Diego, CA, 2019. doi:10.2514/6.2019-2354.

[8] Hwang, J. T., and Martins, J. R. R. A., "A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives," *ACM Transactions on Mathematical Software*, Vol. 44, No. 4, 2018. doi:10.1145/3182393.

[9] Kreisselmeier, G., and Steinhauser, R., "Systematic Control Design by Optimizing a Vector Performance Index," *International Federation of Active Controls Symposium on Computer-Aided Design of Control Systems, Zurich, Switzerland*, 1979. doi:10.1016/S1474-6670(17)65584-8.

[10] Martins, J. R. R. A., and Poon, N. M. K., "On Structural Optimization Using Constraint Aggregation," *Proceedings of the 6th World Congress on Structural and Multidisciplinary Optimization*, Rio de Janeiro, Brazil, 2005.

[11] Hargraves, C. R., and Paris, S. W., "Direct trajectory optimization using nonlinear programming and collocation," *Journal of Guidance, Control, and Dynamics*, Vol. 10, 1987, pp. 338–342. doi:10.2514/3.20223.

[12] Herman, A. L., and Conway, B. A., "Direct optimization using collocation based on high-order Gauss-Lobatto quadrature rules," *Journal of Guidance, Control, and Dynamics*, Vol. 19, 1996, pp. 522–529. doi:10.2514/3.21662.

[13] Betts, J. T., and Huffman, W. P., "Trajectory optimization on a parallel processor," *Journal of Guidance, Control, and Dynamics*, Vol. 14, No. 2, 1991, pp. 431–439. doi:10.2514/3.20656.

[14] Betts, J. T., and Huffman, W. P., "Application of Sparse Nonlinear Programming to Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 1, 1992, pp. 198–206. doi:10.2514/3.20819.

[15] Coleman, T. F., and Verma, A., "The Efficient Computation of Sparse Jacobian Matrices Using Automatic Differentiation," *SIAM Journal of Scientific Computing*, Vol. 19, No. 4, 1998, p. 1210–1233.

[16] Gebremedhin, A. H., Manne, F., and Pothen, A., "What Color Is Your Jacobian? Graph Coloring for Computing Derivatives," *SIAM Review*, Vol. 47, No. 4, 2005, pp. 629–705.

[17] Reuther, J. J., Jameson, A., Alonso, J. J., Rimlinger, M. J., , and Saunders, D., "Constrained Multipoint Aerodynamic Shape Optimization Using an Adjoint Formulation and Parallel Computers, Part 1," *Journal of Aircraft*, Vol. 36, No. 1, 1999, pp. 51–60. doi:https://doi.org/10.2514/2.2413.

[18] Nemec, M., Zingg, D. W., , and Pulliam, T. H., "Multipoint and Multi-Objective Aerodynamic Shape Optimization," *AIAA Journal*, Vol. 42, No. 6, 2004, pp. 1057–1065. doi:10.2514/1.10415.

[19] Gallard, F., Meaux, M., Montagnac, M., and Mohammadi, B., "Aerodynamic aircraft design for mission performance by multipoint optimization," *21st AIAA Computational Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, 2013. doi:10.2514/6.2013-2582, URL `http://dx.doi.org/10.2514/6.2013-2582`.

[20] Appel, K., and Haken, W., "Every planar map is four colorable. Part I: Discharging," *Illinois Journal of Mathematics*, Vol. 21, No. 3, 1977, pp. 429–490. URL `https://projecteuclid.org:443/euclid.ijm/1256049011`.

[21] Neil Roberston, P. S. R. T., Daniel P. Sanders, "A New Proof of the Four-Color Theorem," *Electronic Research Announcements of the American Mathematical Society*, Vol. 2, No. 1, 1996, pp. 17–25. doi:DOI: 10.1090/S1079-6762-96-00003-0, URL `http://www.ams.org/journals/era/1996-02-01/S1079-6762-96-00003-0/S1079-6762-96-00003-0.pdf`.

[22] Gonthier, G., "Formal Proof – The Four-Color Theorem," *Notices of the American Mathematical Society*, Vol. 55, No. 11, 2005, pp. 1382–1393. URL `https://www.ams.org/notices/200811/tx081101382p.pdf`.

[23] He, P., Mader, C. A., Martins, J. R. R. A., and Maki, K. J., "An Aerodynamic Design Optimization Framework Using a Discrete Adjoint Approach with OpenFOAM," *Computers & Fluids*, 2018. doi:10.1016/j.compfluid.2018.04.012, <p>in press</p>.

[24] Falck, R. D., and Gray, J. S., "Optimal Control within the Context of Multidisciplinary Design, Analysis, and Optimization," *AIAA Scitech 2019 Forum*, AIAA, San Diego, CA, 2019. doi:10.2514/6.2019-0976.

[25] Brelje, B. J., Anibal, J. L., Yildirim, A., Mader, C. A., and Martins, J. R. R. A., "Flexible Formulation of Spatial Integration Constraints in Aerodynamic Shape Optimization," *57th AIAA Aerospace Sciences Meeting (SciTech)*, AIAA, AIAA, San Diego, CA, 2019. doi:10.2514/6.2019-2355.

[26] Padrón, A. S., Thomas, J., Stanley, A. P. J., Alonso, J. J., and Ning, A., "Polynomial Chaos to Efficiently Compute the Annual Energy Production in Wind Farm Layout Optimization," *Wind Energy Science*, 2018. doi:10.5194/wes-2017-56, (in review).

[27] Tingey, E., and Ning, A., "Trading off Sound Pressure Level and Average Power Production for Wind Farm Layout Optimization," *Renewable Energy*, Vol. 114, No. B, 2017, pp. 547–555. doi:10.1016/j.renene.2017.07.057.

[28] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Journal of Optimization*, Vol. 12, No. 4, 2002, pp. 979–1006. doi:10.1137/S1052623499350013.