Implementation of topology optimization using openMDAO

Hayoung Chung *
University of California San Diego, San Diego, 92093, USA

John T. Hwang†
NASA Glenn Research Center, 21000 Brookpark Rd, Cleveland, OH 44135 and Peerless Technologies Corporation, 2300 National Rd, Beavercreek, OH 45324

Justin S. Gray‡
NASA Glenn Research Center, Cleveland, OH, 44139

H. Alicia Kim§
University of California San Diego, San Diego, California, 92093, USA, and Cardiff University, Cardiff, UK

Topology optimization is one of the widely known branches among the structural optimization, and it distinguishes itself being able to generate extremely lightweight structures. Recently it has drawn particular interest from both industry and academia because of its natural applicability to additive manufacturing. However, its implementation is often a daunting task for engineers in practice. In particular there can potentially be a large programming effort required to modify the method, even from subtle tweaks in the problem definition or solution algorithm. Changes to the code can leave to corresponding updates to relevant derivative calculations, further compounding the problem. Implementation, therefore, is not only time-consuming but also repetitive and susceptible to human-induced errors. In this regard, topology optimization implementations stand to benefit from changes that result in more code modularity, ease of restructuring, and more automated derivative calculations. In this work we propose using OpenMDAO, a computational framework for multidisciplinary design optimization, as a generic platform for to built topology optimization implementations with in order to achieve these implementation improvements. Two widely used topology optimization techniques—density-based and level-set—are implemented as to serve as reference code designs. These techniques are implemented in a decomposed manner, with the aid of the modular architecture of OpenMDAO as well as state-of-the-art numerical methods. To demonstrate the flexibility of the new topology optimization architecture, two variations on the density-based topology optimization approach are shown.

*Postdoctoral Researcher, Structural Engineering Department, UC San Diego, AIAA Member.
†Research engineer (contractor at NASA GRC), AIAA Member.
‡Aerospace Engineer, Propulsion Systems Analysis Branch, AIAA Member.
§Professor, Structural Engineering Department, Associate Fellow.

Nomenclature

f	=	objective function
$SIMP$	=	Solid Isotropic Material with Penalization
$LSTO$	=	Level Set Topology Optimization
λ_i	=	Lagrange multiplier for sensitivity of function i
ρ	=	element-wise material density
ϕ	=	signed distance function for level-set method
CFL	=	Courant-Frederichs-Lowy condition
$XDSM$	=	eXtended Design Structure Matrix

I. Introduction

Topology optimization is a numerical design method that computes an optimal structure for a specified objective and constraint. A common topology optimization problem formulation is to minimize compliance subject to a minimum constraint on material volume or mass. Alternatively the optimization problem can be formulated to minimize mass subject to a set of stress constraints.

Topology optimization traces its history can be traced back to shape optimization [1, 2], which is inspired by computational geometry [3]. Topology optimization methods can be classified into two categories based on the representation of the structural topology. One is the density-based formulation where the problem is formulated as a material distribution problem and the material densities are design variables. Typically, each element in the finite element mesh is separately parameterized as in Solid Isotropic Material with Penalization (SIMP) [4] method, which is the most common approach. Another category is the boundary-based formulation where the structure is defined by its boundary. The change of the structural topology is realized by the direct movement of the boundary encompassing merge, split, and creation operations. The introduction of the level-set method [5, 6] attracted much attention in recent years and is quickly becoming a viable alternative to SIMP due to its lack of a need for post-processing. Both methods employ the finite element method (FEM) as the backbone that computes the responses of the structure and the corresponding sensitivities with respect to the design variables. In recent years, topology optimization has been actively studied with Additive Manufacturing (AM) in mind, a novel manufacturing technique with far greater design freedom when compared to the classical methods. AM offers small-scale manufacturing capability with micron-scale resolution and multiple materials without additional costs, and leads to inherently multi-scale and multi-functional structures.

In practice topology optimization can be somewhat difficult to implement because it generally requires additional algorithms and code development beyond what is required for a basic FEM solver. One strategy to simplify the implementation is the use of object-oriented programming (OOP) to create a modular code base with a high degree of

code reuse. This approach helps a user easily change the constituent units of code and reorganize the overall program. These OOP traits reduce the repetitive programming tasks that are not only time-consuming but also make the code susceptible to human error during the implementation. While OOP does simplify the organization of the overall code, there are remaining difficulties associated with computing the derivatives of the non-linear code for use with gradient based optimization methods.

We propose OpenMDAO, a computing framework for Multidisciplinary Design Optimization (MDO), as a generic platform for topology optimization. OpenMDAO uses a modular software architecture for the MDO problem, by which the model is configured into smaller computational units that communicate to each other. This modular design provides the basis for a general OOP design of the code. Crucially, OpenMDAO can also automatically compute the total derivatives of the model defined by any arbitrary combination of the smaller computational units. The automatic derivatives feature is what enables a truly reconfigurable code design and leads to a significant decrease in the implementation effort. Moreover, OpenMDAO is equipped with a large library of nonlinear and linear solvers as well as visualization utilities that also reduce coding effort. Based on these attributes, potential users can easily implement and reconfigure their topology optimization workflow and more easily share their modules and algorithms.

These properties are also beneficial to researchers who are new to topology optimization. Once the framework is separated into smaller units of code and robustly programmed, external users can freely rearrange and tweak these components. These traits distinguish the present work from the previous educational publications on topology optimization [7, 8].

The first part of this paper defines the optimization problem and briefly describes the two topology optimization schemes implemented in this work: SIMP topology optimization and level-set topology optimization (LSTO). For both, we consider a linearly elastic compliance minimization problem. In the next section, the OpenMDAO implementations of SIMP and LSTO are explained with design structure matrix diagrams, and numerical examples are presented. The last part of the paper demonstrates an extension of the SIMP implementation to a third topology optimization formulation, the parametric level set approach. This extension provides a practical demonstration of the benefits of modularity, as we reuse existing components from the SIMP approach and integrate new components create the new topology optimization approach with with minimal effort.

II. Theory

A. Optimization problem

Compliance minimization is a classical problem in topology optimization [4, 6, 7], and the optimization problem formulation is given by

$$\begin{aligned} \min f(x,u) = u(x)^T F = \int_\Omega \nabla u(x) : E(x) : \nabla u(x) d\Omega \\ s.t. g(x) \leq g^* \end{aligned} \tag{1}$$

where Ω is the domain, f is the compliance of the structure, which is determined by design variable x and displacement function u. $g(x)$ is the material volume found within the structure, g^* is the prescribed volume constraint, and Young's modulus E. Note that u is also a function of x because linear elasticity as a boundary value problem is assumed herein. In this work, we look at a 2D cantilevered plate for demonstrating the implementations of both topology optimization methods. The design domain, boundary conditions, and the external loading are illustrated in Figure 1.

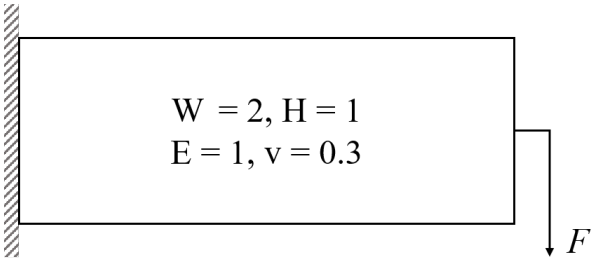


Fig. 1 The cantilevered plate for the compliance minimization problem. The dimensions of the design domain (width W and height H), Young's modulus (E) with Poisson ratio (v), and external force (F) are illustrated.

In this work, the length-to-width ratio (W/H) is fixed to 2, and the Young's modulus E and Poisson ratio v are set to 1.0 and 0.3, respectively. Force F is applied in the $-y$ direction with a value of 1 at the mid-point of the right edge. A linear elasticity assumption is employed, and the computation of the total derivative $\frac{df}{dx}$ exploits the self-adjoint characteristic of the compliance minimization problem. The constraint $g(x)$, the ratio of the total amount of material over the area of the whole domain, is set to be 40% (i.e. $g^* = 0.4$). These optimization settings are fixed throughout this work to simplify the verification of the numerical results and the demonstration of the reusability and the restructurability of the current implementation. Note that design variable x that represent the structure varies depending on the type of topology optimization method. In SIMP, discrete material density ρ is used and level-set method utilizes level set function phi.

B. SIMP: Density-based method

In SIMP, the topology of the structure is explicitly described by the distribution of the discrete element-wise material density ρ that parameterizes the material property. The material densities ρ for each element are the design variables of

SIMP, and therefore the number of design variables coincides with the number of elements. Due to its simplicity of implementation, the SIMP method has been widely used [7, 9] for topology optimization. Without losing generality, Eq. (1) can be reformulated as follows

$$\begin{aligned} \min c(\rho, u) &= \sum_{e}^{N_e} \rho_e^p u_e^T k_e^0 u_e \\ s.t. \sum_{e}^{N_e} \rho_e &\leq G^* \\ s.t. K(\rho) u &= F \\ s.t. 0 \leq \rho_e &\leq 1 \end{aligned} \tag{2}$$

where p is a penalizing parameter, u_e and ρ_e are the its discretized displacement and material density correspond to element e, and k_e^0 is the stiffness matrix, provided that the element is filled with material (i.e. $\rho_e = 1$). A typical workflow is described in many works[4, 7], and it is illustrated in Figure 2.

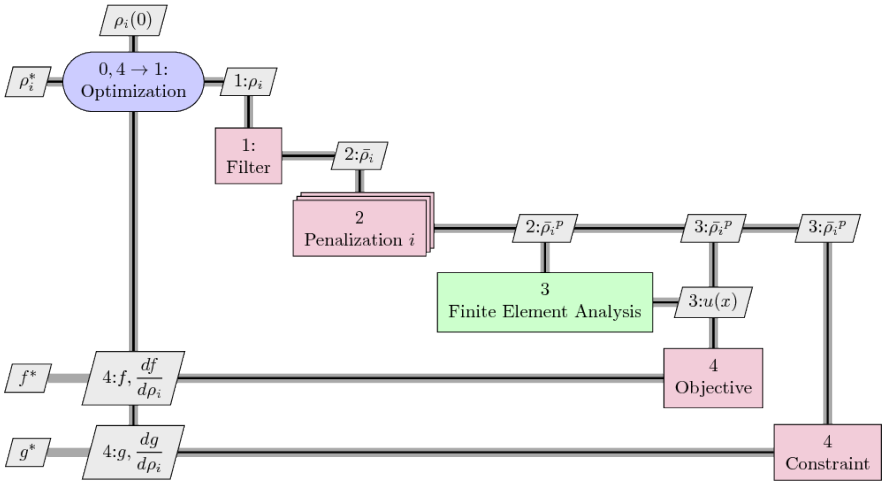


Fig. 2 An XDSM diagram of SIMP method

The simplest decomposition of SIMP consists of 5 modules: filter, penalization, finite element analysis, objective, and constraint. The finite element analysis is a computational backbone of the SIMP method, as all inputs and outputs of each module are present in the same discretized space. Whenever the material connectivity is required in the problem definition, either density or sensitivity filter is required. In this diagram, the density filter is shown.

As shown in the diagram, the first step in the SIMP method is an applying density filter to the material distribution ρ_i to generate locally uniform distribution $\bar{\rho}_i$. Then a filtered material density is penalized so that intermediate densities (i.e. gray material) are removed. The penalization is a crucial step in obtaining a well-defined topological solution. Without it, a converged solution is likely to contain gray areas that are open to interpretation during post-processing. In this work, the penalization parameter p is fixed to 3. The material properties (i.e., Young's modulus) are assumed to be

uniform within each element, and proportional to the specific material densities found on the element as

$$E = \rho^p E_0 + (1 - \rho^p)E^\epsilon \tag{3}$$

where E_0 is the original modulus of the material and $E^\epsilon = 10^{-6}E_0$ is a fictitious stiffness, which is introduced to prevent rank deficiency of the stiffness matrix and enforce nonzero sensitivity at the element. The objective and constraint functions are computed based on these variables.

Recalling that gradient-based optimization is utilized in this work, the update of the solution requires the derivatives of the objective function with respect to design variables. As there is a substantially higher number of design variables than quantities of interest, the adjoint method is employed for efficiency. Due to the self-adjoint characteristic of the compliance minimization problem, the total derivative calculation is greatly simplified. The corresponding sensitivity to the material density is

$$\frac{dc(\rho, u)}{d\rho_e} = -(p\rho_e^{p-1})u_e^T K_e(E(\rho))u_e \tag{4}$$

where ρ is a material density, u is a displacement vector, and K is a stiffness matrix. The subscript e refers the element index found in the finite element mesh.

C. LSTO: level-set topology optimization

Level-set topology optimization (LSTO) differs from SIMP, as the structural domain is implicitly represented. An advantage of LSTO over density-based optimization scheme is the resulting boundaries of the optimal solution can cut an element and smooth boundaries are obtained in contrast to jagged-edge boundaries found in SIMP. In this work, we implemented the LSTO method found in the reference [10].

For a given level set function $\phi(r)$ in the fixed grid space, the boundary is defined by its zero hypersurfaces $(\Gamma(r) : \{r \in \Omega | \phi(r) = 0\}$. A signed distance function, a distance measure from the boundaries of the design domain, is commonly used as a level set function, and it is implicitly updated by solving the Hamilton-Jacobi equation

$$\dot{\phi} + \nabla\phi \cdot \frac{\partial r}{\partial t} = \dot{\phi} + V_n|\nabla\phi| = 0 \tag{5}$$

where t is the pseudo-time in which the level set function advects, and V_n is the velocity of advection, normal to the structural boundary. An explicit integration method is often employed to solve the problem for the discretized space and time domain. Equation (1) is linearized with respect to shape perturbations (i.e., shape derivatives) at the boundary points.

$$\begin{aligned} \min \frac{\partial f(x)}{\partial \Omega^k} \cdot \Delta\Omega^k &= \Delta t \int_\Gamma S_f V_n d\Gamma \\ s.t. \frac{\partial g}{\partial \Omega^k} \cdot \Delta\Omega^k &= \Delta t \int_\Gamma S_g V_n d\Gamma \leq -g^{*k} \end{aligned} \tag{6}$$

where $\Delta\Omega^k$ is the update for the current domain at the k-th pseudo timestep, and S_f and S_G each represent the shape sensitivities for the objective and the constraint. The function change is now a function of the shape sensitivities and normal velocities evaluated using boundary integrals.

In contrast to SIMP, the shape sensitivities are evaluated at the discretized boundary points, and the consistency and the smoothness of the values are necessary to obtain the optimal solution with clear distinction of the boundary. The consistent compliance sensitivities [6] are

$$\begin{aligned} S_f &= -\int_\Gamma \epsilon(u) \cdot E(\rho) \cdot \epsilon(u) dl \\ S_g &= -1 \end{aligned} \tag{7}$$

where the weighted least squares method is employed based on sensitivities S_f and S_g evaluated at the point clouds. The Gauss quadrature points are selected as the point of evaluation, and an Ersatz material model (i.e., $\mathrm{E} = \mathrm{E}_0\rho$, where ρ is an area fraction that is a ratio of materials presence ($\phi > 0$) within each element) is employed. It is worth noting that the sensitivity found in Eq. (7) can be reduced to Eq. (4) when the sensitivity is evaluated at the midpoint of each element as it also exploits the self-adjoint trait of the given problem.

As suggested by Dunning et al., [11], the linearized equation (6) leads to Eq. (8) after the algebraic manipulations. Detailed procedures also can be found in the recent literature [10].

$$\begin{aligned} \frac{\partial [f,g]}{\partial \Omega^k} \cdot \Omega^k &= \sum_z^{nb} \Delta t V_{nj} [S_{f,j}, S_{g,j}] l_j \\ &= [C_f, C_g] \cdot V_n \Delta t \equiv [C_f, C_g] \cdot z \\ z(\lambda) &= \lambda_f S_f + \lambda_g S_g \end{aligned} \tag{8}$$

where subscript j indicates the index of the boundary points, and C_f and C_g refer the vectors obtained by boundary integration of the corresponding sensitivities, while l refers the length segment. Normal velocity V_n is multiplied by the t to produce advecting distance z that the boundary advects during given time, while z is assumed to be the sensitivities multiplied by Lagrange multiplier λ. By using Eq. (8), Eq. (6) is then written as:

$$
\begin{aligned}
\min \quad & C_f \cdot z \\
s.t. \quad & C_g \cdot z < g^* \\
s.t. \quad & z_{min} \leq z \leq z_{max}
\end{aligned}
\tag{9}
$$

where a bound for a distance vector $[z_{min}, z_{max}]$ satisfies two criteria: (1) the advection must not go beyond the boundary of the design domain, and (2) the distance cannot violate CFL (Courant-Frederichs-Lowy) condition. These set of equations leads to the sub-optimization of the LSTO method, by which optimal distance for the function minimization is obtained and remains within the range of constraints and bounds.

As a result, two sets of the optimization layers are separately dedicated to the Hamilton-Jacobi equation and the sub-optimization, and shown in the XDSM diagram shown in Figure 3.

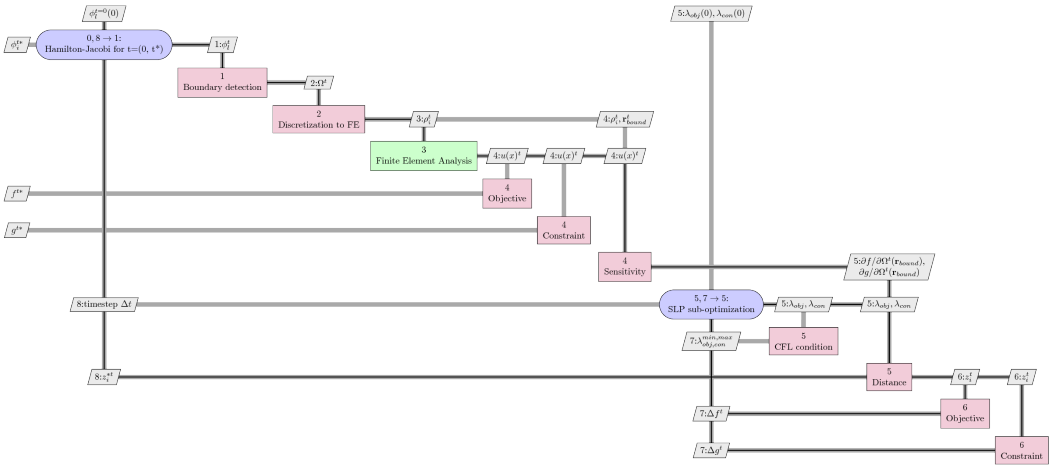


Fig. 3 An XDSM diagram of LSTO

For a given signed distance function ϕ, the boundary points are obtained by using a marching square algorithm[3]. The positions of the boundary points and the length segments of the boundary l that represent discretized design and the resulting boundary sensitivities S_f and S_g are the arguments of the sub-optimization.

In order to retain distance z within its bounds, the bounds of the multipliers λ are also computed based on the CFL condition and the geometric limit of the movement (i.e., side limit [12]). These are the conservative move limits for boundary advection. The sub-optimization problem (Eq. (9)) is then solved and the optimal distance z is computed as its solution. These boundary distances are extended to level-set fixed grid within the narrow band[3], and the structure is updated by solving the Hamilton-Jacobi equation. Furthermore, reinitialization of the function is required to satisfy the $|\nabla\phi| = 1$ condition as it is often violated after a few updates.

Note that the solution update scheme in LSTO requires time integration over pseudo time t, and involves sophisticated

numerical techniques such as the fast-marching method. Due to these traits, the first layer of the optimization found in the LSTO is not benefited from the numerical solvers found in OpenMDAO. In this work, therefore, only the sub-optimization step is exposed to the OpenMDAO, where the optimal advection velocity at each optimization steps is computed. The outer part of the optimization, on the other hand, is solved externally by using numerical libraries for a level set, such as the upwind scheme, fast marching methods, and the 5th order WENO [10, 13].

III. Implementations of topology optimization methods

A. OpenMDAO

OpenMDAO [14] is an MDO platform developed and maintained by researchers at NASA Glenn Research Center. OpenMDAO is open-source and is written in Python, which is a language known for its ease of interfacing to compiled languages such as C++ and Fortran. OpenMDAO has been used to solve MDO problems in satellite design [15], wind turbine design [14], aircraft design [16], and aircraft trajectory optimization [17].

OpenMDAO is unique among MDO frameworks because it is designed for gradient-based optimization, and assists in the computation of analytic total derivatives. OpenMDAO is designed around a collection of equations and algorithms, called the (MAUD) architecture [18], that enable it to compute total derivatives of a model given the partial derivatives of each smaller unit of code. MAUD provides a generalization the adjoint method that unifies all existing methods for computing derivatives using a matrix equation [19].

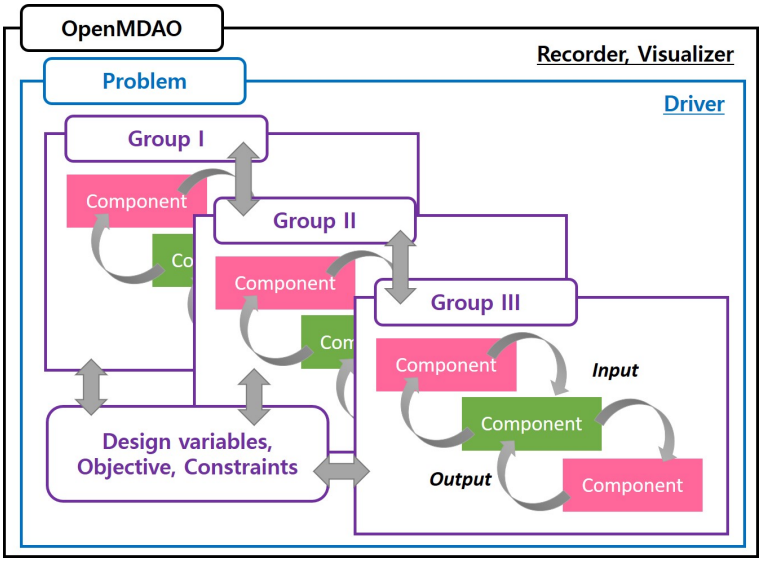


Fig. 4 Illustration of modularized architecture found in OpenMDAO. Three set of the layers are found at the different hierarchies: Component, Group, and Problem.

The basic unit of code in an OpenMDAO model is the Component, where the actual computations take place during execution. A Component can be as comprehensive as a single discipline, but more often it is a subset of an entire

discipline (e.g. the penalization equation found in the SIMP-based optimization). The Component object declares its variables (inputs and outputs), implements the governing equation that computes the outputs for given inputs, and optionally computes partial derivatives of its outputs with respect to its inputs. If analytic partial derivatives are not provided, OpenMDAO can approximate them using the finite differences or the complex-step method. Depending on the type of equation used to compute the outputs, a Component can be of either explicit or implicit type. Consider two variables from the SIMP-based formulation: penalization and displacements. The penalization calculation is an explicit function, whereas the displacement vector is implicitly solved for by converging the linear elastic residual equations.

In OpenMDAO models are built hierarchically from Groups that aggregate Components and other child Groups. The full model is contained within the Problem object, which also contains a Driver object. The Driver is responsible for execution of the model, typically using an optimization algorithm. For the present study, the Driver is an SLSQP optimizer with a tolerance of 10^{-9}.

It is a common practice to assign each individual discipline its own Group, and then implementing the governing equations as a set of smaller Components within the Group and arranging them in a logical order. This decomposition leads to a salient modularity due to the encapsulation provided by the the Component class. This modularity leads to both restructurability and reusability. Once the Component object is programmed, further modifications are is unnecessary as long as the governing equations remain intact (reusability). Also, these modules are freely arranged within the Group layer and therefore reformatting of the information requires only changes in connectivity between the Components, and the required changes to the global derivative are applied automatically (easy restructurability). These two traits are beneficial as these significantly reduce repetitive programming and user-induced error.

The decomposition of the model into smaller components has another major advantage. The task of computing the required partial derivatives is simplified because the smaller components each have less inputs and outputs and hence less to differentiate. The total derivatives can then be automatically computed by the framework. Such capability is especially useful when models contain implicit components because the user no longer needs to manually implement the adjoint method themselves. Thus OpenMDAO fundamentally reduces the amount of work necessary to built topology optimization implementations.

In this work, topology optimization is deemed as the one discipline of our MDO problem, and a single Group object is therefore assigned to each optimization method (e.g., SIMP, LSTO). In our case, there are no other Group objects in the model. Therefore, each Group is comprised solely of Component objects, including the finite element analysis and auxiliary computations such as the density filter found in the SIMP method. Figures 2 and 3 explain the topological optimization workflows visually using design structure matrix and hierarchy diagrams. Each Component is marked by the blue box, and its inputs (pink-colored) and outputs (either gray- or orange-colored) exposed to Group layer are also specified. An orange-colored output is an unknown from implicit function, while gray-colored one is an explicit output. Note that dependencies are marked by black off-diagonal squares.

B. SIMP

The visualization of SIMP is shown in Figure 5, where the Components and their connectivity in the SIMP Group are illustrated.

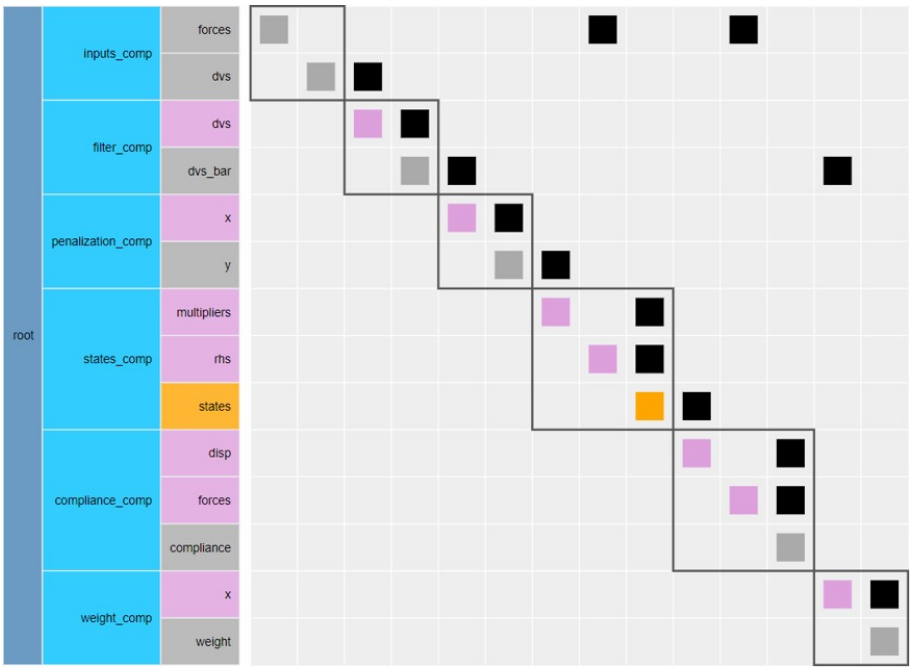


Fig. 5 Design structure matrix of SIMP method.

The workflow of the SIMP method is composed of six Components: specifying independent variables (*input_comp*), density filtering ($filter\_comp$), penalization (*penalization_comp*), linear elasticity (*states_comp*), the objective function (*compliance_comp*), and the constraint function ($weight\_comp$). Note that this configuration is analogous to the modules and their connectivities found in the corresponding XDSM diagram (Figure 2). For example, displacement ($disp$) and forces ($forces$) are inputs to the *compliance_comp*, of which output is a *compliance*.

After preprocessing that includes discretization and specifying boundary conditions, the independent inputs are specified for the given finite element mesh: discrete material density dvs as a design variable and external force vector $forces$. Design variables first go through a simple density filter, a low-pass filter that removes a checkerboard pattern often found within the solution:

$$\hat{\rho} = \sum_j w_{ij}\rho_j, \qquad w_{ij} = \frac{R - d(i,j)}{\sum_{k \in N_i}(R - d(i,k))} \qquad if \quad j \in N_i \tag{10}$$

where $\bar{\rho}$ is a filtered material density found at the element i, and w_{ij} is a conic weight based on the distance d between elements i and j. A Group N_i refers to the list of the neighboring elements of element i, within a radius R. Although the filtering is not visible in describing XDSM of SIMP method (Figure 3), it is commonly employed for a stable

convergence in SIMP. Choosing the best filter, however, relies heavily on the heuristics; therefore, iterative testing is often required [20]. Again, the restructurability of the present approach reduces the effort required.

The filtered design variables are then penalized with constant p. The computed material density $\bar{\rho}^p$ $(multipliers)$ leads to parameterized material stiffness (Eq. (3)) and global stiffness matrix. The displacement vector $states$ is a state variable by solving force equivalence ($R(\rho, u) = K(\rho)u - F = 0$). Within the $state\_comp$ Component, the partial derivatives of the residual R with respect to all inputs are computed, which are later used by OpenMDAO in the adjoint method. This is another advantage of using OpenMDAO as a framework of topology optimization as it greatly simplifies the calculation of the global gradient.

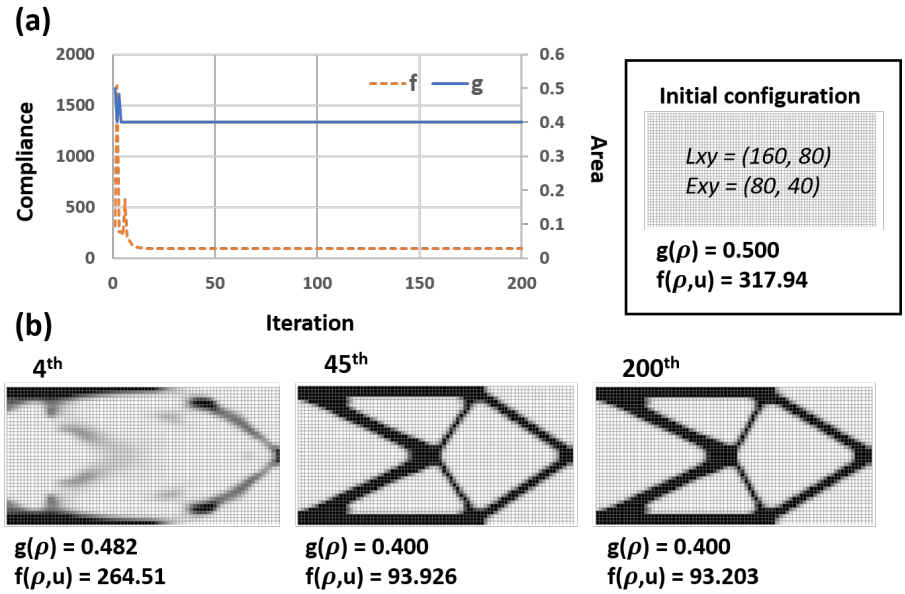


Fig. 6 Numerical results of SIMP. (a) a convergence profile with initial configuration where the number of meshes to each direction are specified (b) a material configuration, area fraction (g) and compliance (f) found during at the 4^{th}, 45^{th}, and 200^{th} iterations.

Figure 6(a) shows the convergence graph of the compliance $f(\rho)$, which is a objective function, the area fraction g, and the initial configuration of the design domain. The size of the rectangular design domain is prescribed to 160 × 80 (length × width), and the domain is discretized with 80 × 40 quadratic elements. The initial material densities are uniformly set to 0.5, by which the constraint is mildly violated. According to the convergence graph, a constraint ($g(\rho) < 0.4$) is always satisfied after 5th iteration and the norm of the changes of the compliance is smaller than 10^{-6} after 48^{th} iteration. However, the iteration continues until the 200^{th} iteration because of the small convergence criteria ($\Delta f(\rho, u) < 10^{-9}$). In Figure 6(b), the final structural topology is nearly obtained. Note that intermediate densities found in the early iterations are penalized so that the solutions at the later iterations do not exhibit the intermediate densities.

C. LSTO

As described in its XDSM diagram (Figure 3), LSTO is composed of two layers: (1) an update of the level-set function achieved by solving on Hamilton-Jacobi equation on the domain boundary, and (2) a sub-optimization, by which the optimal advection distance at each iteration is calculated. As mentioned earlier, only the sub-optimization step in Eq. (9) is implemented within OpenMDAO. On the other hand, the pre-processing operations and the post-processing operations, such as updating and extracting the design, are separately accomplished outside of OpenMDAO.

The operations prior to sub-optimization consist of initializing the level-set properties and execution of the finite element analysis. A signed distance function is firstly initialized, and the geometric properties of the structure, which includes the location of the boundary points, length segments, and side limits of the boundary movement, are extracted by external level-set libraries based on Ref. [12]. The structure in the finite element domain is represented by the Ersatz material model, and the boundary conditions are assumed to be constant during the iteration. As a result, the sensitivities S_f and S_g evaluated at the boundary points and the geometric properties are obtained, based on which the optimal distances at the structural boundaries are calculated through sub-optimization. The post-processing operations update the level set function based on the calculated distance. The given distances z at the boundary points are first extended to the level-set fixed grid using the 5th order WENO, and the signed distance function is updated the fast marching scheme.

The detailed description of the sub-optimization step other than pre-processing or post-processing operations, are found in the design structure matrix of the sub-optimization shown in Figure 7.

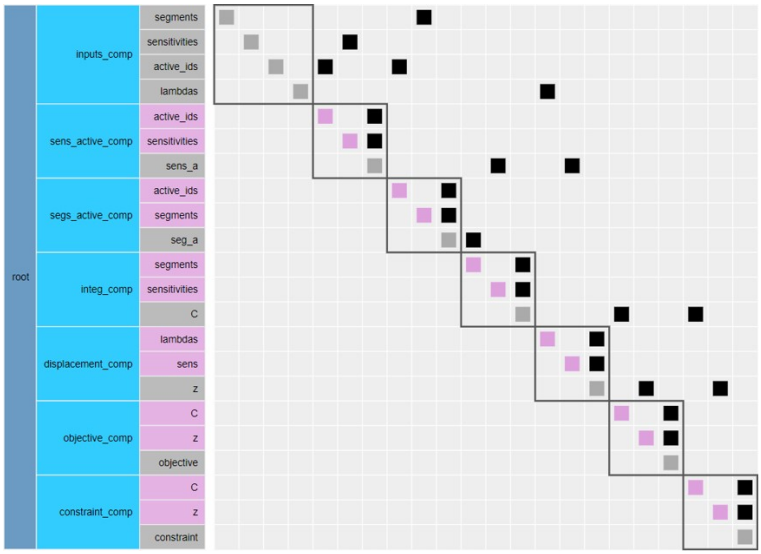


Fig. 7 Design structure matrix of sub-optimization found in LSTO

The workflow of the sub-optimization of the LSTO is composed of 6 Components. The first *input_comp* Component specifies the 4 independent variables: Lagrange multipliers (*lambdas*) as design variables, length segments for boundary integral (*segments*), sensitivities with respect to the objective and constraints (*sensitivities*), and the indices of the

active nodes (*active_ids*). Note that not all of nodes in the level-set grid are active during the iteration. For example, the nodes belonging to the domain boundary or non-designable regions and their level-set values remain fixed to their initial value. Filtering the values at active boundary points (*active_comp*) is, therefore, a necessary step in order to reduce the number of degree of freedoms. The subsequent Component computes boundary-integrated values (*integ_comp*) and its corresponding advection distances (*distance_comp*) based on Eq. (8). It is worth pointing out that a finite difference scheme is inevitably used to estimate the partial derivatives of the distances with respect to λ because the distances are not linear with respect to the input as the side limit and the CFL condition are possibly violated when the distances are linearized [11]. Therefore, a numerical differentiation scheme provided by OpenMDAO is utilized in the Component, which also demonstrates the efficiency and flexibility in practical implementations. The Components that follow are dedicated to the objective function (*objective_comp*) and constraint function (*constraint_comp*) based on Eq. (9). The resulting advection velocity is expected to be optimal for compliance minimization, while satisfying the constraints.

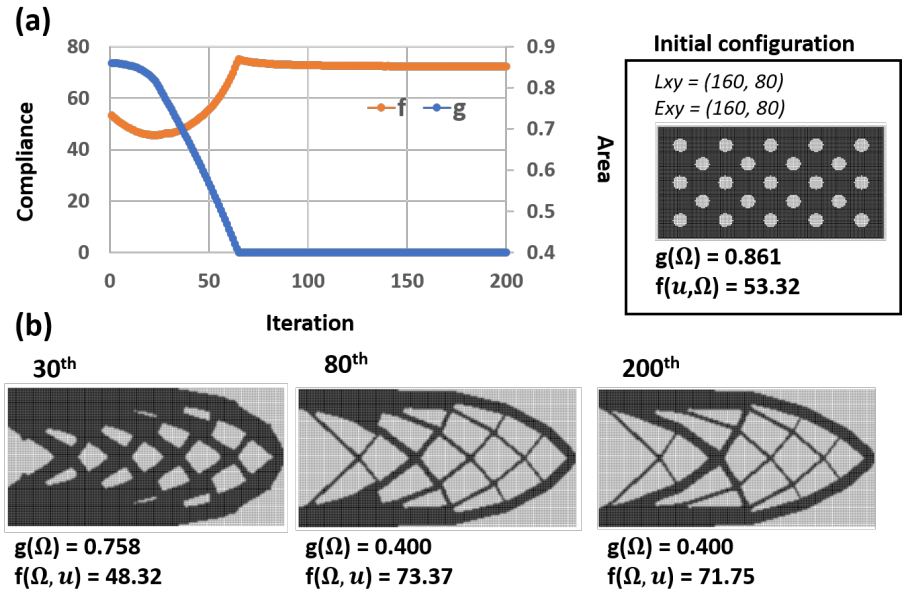


Fig. 8 Numerical results of LSTO. (a) a convergence profile with initial configuration where size (Lxy) and the number of meshes (Exy) to each direction are specified (b) a material configuration, area fraction (g) and compliance (f) found during at the 30^{th}, 80^{th}, and 200^{th} iterations.

Figure 8 shows a set of results and the convergence graph for the same cantilevered beam problem solved previously with SIMP. As the hole-creation method [13] is not employed herein, the initial design domain has seeded holes. Both the compliance and constraint values exhibit a smooth convergence as reported in the literature [11], which is in contrast to SIMP example. A topology similar to SIMP is found in the converged optimal solution, although with more holes.

IV. Extensions

In addition to the unique architecture of OpenMDAO, which enhances the modularity of the optimization schemes and the ease of the implementations, using OpenMDAO as a topology optimization platform is also advantageous in extending or modifying the existing implementations. In the present section, we demonstrate the ease of restructuring and the reusability of the modules by example.

A. Flexible arrangement of the module

The restructurability of the program is demonstrated by rearranging the order of the density filter Component with respect to other Components. Thanks to the encapsulation of the Component object, minimal changes are required, in the Group layer only. This is in contrasts to the topology optimization programs such as the minimalistic educational implementations [7]. No matter how simple the program might be, it is still true that small tweaks such as the removal of the filter require re-programming of the code; not only the functions but also the sensitivities must be changed accordingly.

One of the possible tweaks to the configuration are found in the Figure 9, where the corresponding design structure matrix and code snippet are presented.

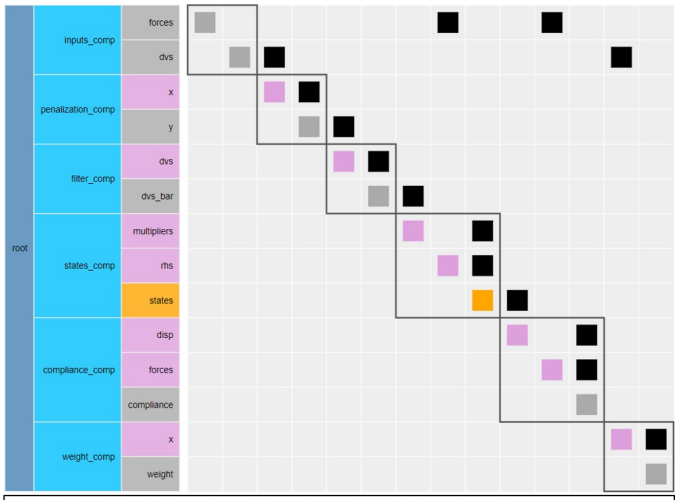
When compared with the design structure matrix of SIMP (Figure 5), the Component object for the density filter ($filter\_comp$) is shown to be rearranged relative to the penalization filter ($penalization\_comp$) (Figure 9(a)). The code snippet that realizes the rearrangement is also shown in Figure 9(b). The change of the code is proven to be minimal as the reconnection of the inputs and outputs within the Group object is the only required modification, thanks to the *connect* member function of the Group object and the thorough encapsulation of the Component object. Even though the whole code is not shown herein, one may easily see how the removal of the filter can be realized. It is worth mentioning that these two tweaks are presented only to succinctly demonstrate the possible extensions, and these extensions are not necessarily typical in the SIMP method; however, this example is in line with reducing the repetitive programming required in selecting the type of weight of the filter.

The numerical results are shown in Figure 10. As one may expect, either rearranging (Figure 10(a)) or removal (Figure 10(b)) of the existing filter each saliently generates an intermediate density or a checkerboard pattern in the converged solutions. This example also exhibits the educational benefits, as an effect of the filter as a way to remove the ill-posedness of the problem is demonstrated without the drastic changes of the code.

B. Parameterized level-set topology optimization

The reusability of the modules is also exemplified by extending the present Components to a new topology optimization formulation that bears similarities to the parameterized level-set topology optimization approach [5, 21, 22]. This method is, in essence, a density-based approach, but the densities are parametrized using a level set function

(a)



(b)

```
self.connect('inputs_comp.dvs', 'penalization_comp.x')
#self.connect('inputs_comp.dvs', 'filter_comp.dvs')

# penalization
comp = PenalizationComp()
self.add_subsystem('penalization_comp', comp)
# self.connect('filter_comp.dvs_bar', 'penalization_comp.x')
self.connect('penalization_comp.y', 'filter_comp.dvs')

# filter
comp = DensityFilterComp(…)
self.add_subsystem('filter_comp', comp)
# self.connect('filter_comp.dvs_bar', 'penalization_comp.x')
self.connect('filter_comp.dvs_bar', 'states_comp.multipliers')
```

Fig. 9 A demonstration of restructurability. (a) the design structure matrix of the SIMP method, where the filter object is rearranged with respect to that of the penalization step, and (b) a code snippet for the reconfiguration that exemplifies the easy reconfiguration; commented commands are marked by blue and corresponding changes are marked by red.

(a) (b)

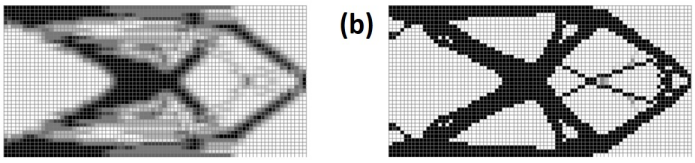


Fig. 10 Optimal solutions where the filter object is (a) rearranged or (b) removed.

that implicitly represents the boundary. Instead of tracking the boundary directly, the value of this function is used to control the densities in the elements, using a Heaviside function to map to the $(0, 1)$ interval and SIMP-type penalization to discourage intermediate densities. Further investigation is necessary to explore the potential of this method as a new approach for topology optimization, but our goal here is to highlight that the use of OpenMDAO as a topology optimization platform facilitated the development of this approach through reusability and restructurability. As shown in Figure 11, many Components from SIMP are reused, including the implicit Component that computes the displacements using linear elasticity (*states_comp*), the penalization Component (*penalizaion_comp*), and the objective (*objective_comp*) and constraint function (*weight_comp*) Components.

In this scheme, design variables are the localized values of the regularly spaced control points, by which a

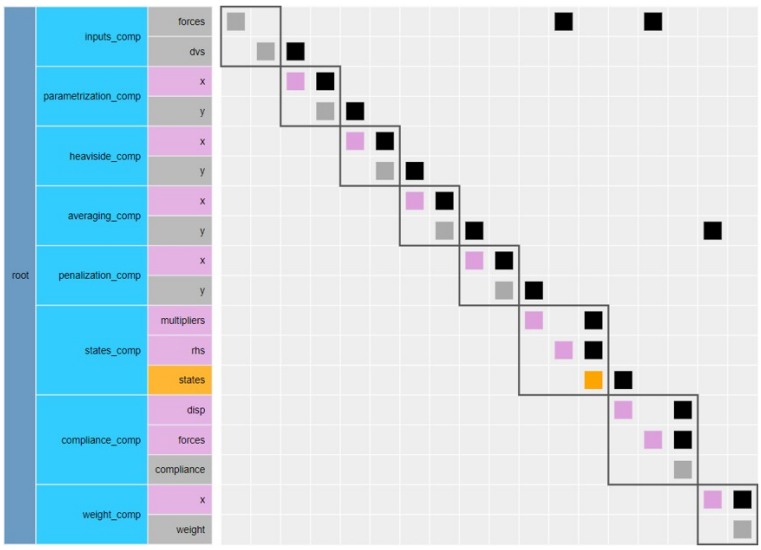


Fig. 11 Design structure matrix of the parametric level-set approach for topology optimization.

hypersurface is constructed via a parametric mapping using B-splines (*parametrization_comp*). The generated surface is then filtered into discrete values bound to (0, 1) by passing them into an analytic Heaviside function (*heaviside_comp*): the hyperbolic tangent function. The filtered values are interpreted as a material density (*averaging_comp*); if their spatial size coincides to that of a finite element mesh, as in the present case, the SIMP module without a filter can be reused for the remaining computation.

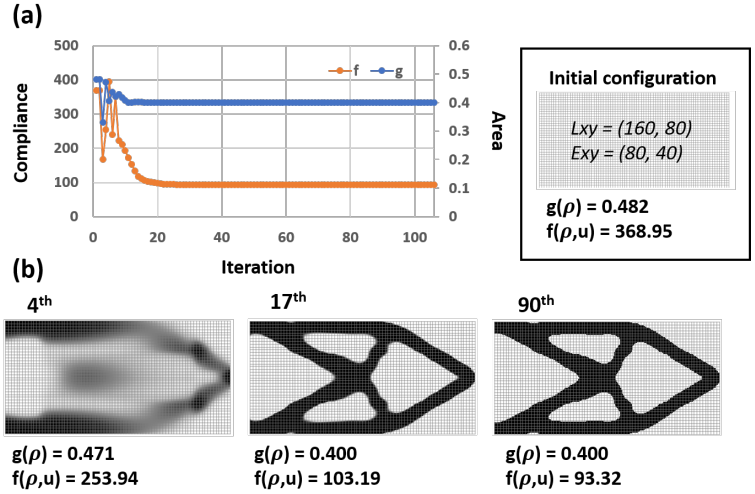


Fig. 12 Numerical results computed using the parametric level-set approach for topology optimization: (a) the convergence profile with the initial specifications same as in SIMP (b) the material configuration, area fraction (g) and compliance (f) at the 4^{th}, 17^{th}, and 90^{th} iterations.

The numerical results are presented in Figure 12. Although the results show oscillations in the boundary, they are sufficient to demonstrate that the optimal topology computed out of the present scheme and its compliance value

are comparable to those of SIMP. We note that the reusability demonstrated herein can enhance the productivity in implementation, as new ideas or different modules can be quickly implemented without repeated programming once the initial library is in place.

V. Conclusion

Topology optimization is a structural design method capable of computing the optimal layout among a large number of candidates. There is ongoing research in extending it to multiscale or multidisciplinary applications; however, its potential is not fully exploited due to the non-technical problems associated with the implementation time and effort. This is not only inefficient but also makes the program prone to human-induced errors.

In this work, we incorporate topology optimization into OpenMDAO, an computational framework for MDO. The primary benefit of using OpenMDAO comes from its modular architecture. By decomposing each topology optimization method into Components, the programming becomes more object-oriented and the derivatives across the multiple Components are automatically calculated. Also, the workflow can be easily visualized as connectivities between Components. Another benefit is easy application of standard nonlinear and linear solvers available in OpenMDAO. We demonstrate these benefits using the two most widely used topology optimization methods, SIMP and the level-set method. We also demonstrate the extensibility by modifying the SIMP implementation to perform parameterized level-set topology optimization, which is a third method. The modular approach enables this modification with minimal additional programming effort where most of the existing Components are reused.

The authors expect the current work to be applied to the different physics as the required change is minimal when compared with using approaches [7]. Moreover, OpenMDAO's suite of optimization methods and utilities potentially benefits in educating and promoting the topology optimization strategies to researchers in other disciplines.

Acknowledgement

We acknowledge the support of the NASA Transformational Tools and Technologies Project, contract number NNX15AU22A. H Alicia Kim also acknowledges the support of the Engineering and Physical Sciences Research Council Fellowship for Growth (grant number EP/M002322/2).

References

[1] Kim, H., Garcia, M., Querin, O., Steven, G., and Xie, Y., "Introduction of fixed grid in evolutionary structural optimisation," *Engineering Computations*, Vol. 17, No. 4, 2000, pp. 427–439.

[2] Eschenauer, H. A., Kobelev, V. V., and Schumacher, A., "Bubble method for topology and shape optimization of structures," *Structural and Multidisciplinary Optimization*, Vol. 8, No. 1, 1994, pp. 42–51.

[3] Fedkiw, S. O. R., and Osher, S., "Level set methods and dynamic implicit surfaces," *Surfaces*, Vol. 44, 2002, p. 77.

[4] Bendsøe, M. P., Sigmund, O., Bendsøe, M. P., and Sigmund, O., *Topology optimization by distribution of isotropic material*, Springer, 2004.

[5] van Dijk, N. P., Maute, K., Langelaar, M., and Van Keulen, F., "Level-set methods for structural topology optimization: a review," *Structural and Multidisciplinary Optimization*, Vol. 48, No. 3, 2013, pp. 437–472.

[6] Allaire, G., Jouve, F., and Toader, A.-M., "Structural optimization using sensitivity analysis and a level-set method," *Journal of computational physics*, Vol. 194, No. 1, 2004, pp. 363–393.

[7] Sigmund, O., "A 99 line topology optimization code written in Matlab," *Structural and multidisciplinary optimization*, Vol. 21, No. 2, 2001, pp. 120–127.

[8] Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B. S., and Sigmund, O., "Efficient topology optimization in MATLAB using 88 lines of code," *Structural and Multidisciplinary Optimization*, Vol. 43, No. 1, 2011, pp. 1–16.

[9] Liu, K., and Tovar, A., "An efficient 3D topology optimization code written in Matlab," *Structural and Multidisciplinary Optimization*, Vol. 50, No. 6, 2014, pp. 1175–1196.

[10] Kambampati, S., Jauregui, C., Museth, K., and Kim, H., "Fast level set topology optimization using a hierarchical data structure," *AIAA Aviation and Aeronautics Forum and Exposition 2018*, submitted.

[11] Dunning, P. D., and Kim, H. A., "Introducing the sequential linear programming level-set method for topology optimization," *Structural and Multidisciplinary Optimization*, Vol. 51, No. 3, 2015, pp. 631–643.

[12] Dunning, P., "Introducing loading uncertainty in level set-based structural topology optimisation," Ph.D. thesis, University of Bath, 2011.

[13] Dunning, P., and Kim, H., "A new method for creating holes in level-set function based topology optimisation," *International Journal for Numerical Methods in Engineering*, 2013.

[14] Gray, J. S., Hearn, T. A., Moore, K. T., Hwang, J., Martins, J., and Ning, A., "Automatic Evaluation of Multidisciplinary Derivatives Using a Graph-Based Problem Formulation in OpenMDAO," *15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, American Institute of Aeronautics and Astronautics, 2014. doi:doi:10.2514/6.2014-2042, URL `http://dx.doi.org/10.2514/6.2014-2042`.

[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*, Vol. 51, No. 5, 2014, pp. 1648–1663. doi:10.2514/1.A32751.

[16] Hwang, J. T., and Martins, J. R. R. A., "Allocation-mission-design optimization of next-generation aircraft using a parallel computational framework," *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, American Institute of Aeronautics and Astronautics, 2016. doi:10.2514/6.2016-1662.