A C# code for solving 3D topology optimization problems using SAP2000

Nikos D. Lagaros[a,b] [1], Nikos Vasileiou[a] and Georgios Kazakis[a]

[a] Institute of Structural Analysis & Antiseismic Research,
Department of Structural Engineering,
School of Civil Engineering,
National Technical University of Athens,
9, Heroon Polytechniou Str., Zografou Campus,
GR-15780 Athens, Greece,
E-mail: nlagaros@central.ntua.gr, nickvasileiou93@gmail.com, kzkgeorge@gmail.com

[b] ACE-Hellas,
6, Aigaiou Pelagous Str., Agia Paraskevi
GR-15341 Athens, Greece,
E-mail: nikos.lagaros@ace-hellas.com

Abstract: SAP2000 is a well-known commercial software for analysis and design of structural systems that is equipped with an open application programming interface (OAPI). A code written in C# to solve three-dimensional topology optimization problems is presented in this work, where taking advantage of the OAPI feature, a topology optimization framework was integrated into SAP2000. The code is partially based on the 99 and 88 line codes written by Sigmund and co-workers (Struct Multidisc Optim 21(2):120–127, 2001 and Struct Multidisc Optim 43(1):1–16, 2011). The code solves the problem of minimum compliance while through OAPI it takes advantage of all modelling capabilities of SAP2000. The paper covers the theoretical aspects of topology optimization incorporated in the code and provides detailed description of their numerical implementations. Special effort was made to the latter one, describing in detail all numerical aspects of the code, in order to facilitate the reader to understand the code, and therefore being able to further enhance its capabilities. The complete code can be downloaded from http://users.ntua.gr/nlagaros/TOCP/projectsSMO.zip.

Keywords: Topology optimization; SAP2000 OAPI; C# code; optimality criteria; method of moving asymptotes; conceptual design.

1. INTRODUCTION

Since 1970 structural optimization has been the subject of intensive research and several methodologies for achieving optimized structural designs have been advocated [1-5]. Topology optimization represents a material distribution numerical procedure for synthesizing structural layouts without any preconceived shape. In this study we present the C# code that was developed in the framework of the research project "OPTARCH: Optimization Driven Architectural Design of Structures" (H2020-MSCA-RISE-2015) aiming to facilitate the needs of the project. In the rest of

[1] Corresponding author

the paper this code is abbreviated as TOCP standing for topology optimization computing platform. One of the objectives of the project is to integrate architectural design criteria into shape and topology (S&T) optimization procedures. In this direction, a first goal of the OPTARCH project is to exploit the use of S&T optimization techniques in computer aided architectural design. This scope, among others, will be fulfilled by formulating S&T problems as well as developing procedures for dealing with. According to OPTARCH project, it is anticipated that specially tailored topology optimization design tools will equip the engineer with the capacity to directly define various design alternatives. As a result, it is of importance to develop a platform that will provide a common ground with computer aided design (CAD) tools both for defining the architectural design constraints and for depicting the optimized results. In this direction, TOCP code was developed to become the basis and to serve OPTARCH objectives. Moreover, exchange of ideas is considered as an issue of major importance and therefore the gold model for open access was considered as an essential pillar for better communication and dissemination of OPTARCH's results. Consequently, it was decided that students and newcomers to the field of topology optimization to be able to download TOCP code from the webpage http://users.ntua.gr/nlagaros/TOCP/projectsSMO.zip. During the four years of the project, the code will further be enhanced and new capabilities incorporated will also be provided for free through the above mentioned webpage. Among different S&T optimization techniques that exist in the literature, the solid isotropic material with penalization (SIMP) [6,7] and level-set [8,9] methods are the most popular ones. Although, in the framework of OPTARCH project both methods will be examined with respect to their suitability for the needs of conceptual design for architectural criteria, in the current version of TOCP only SIMP was implemented.

In literature, various free source codes are provided, among others by the pioneering researchers in the field of S&T optimization, like the Matlab codes by O. Sigmund [10,11] and the Scilab ones by G. Allaire [12]. The applicability of most of these codes is limited to 2D elastic structural systems. Indicatively, the two Matlab codes written by O. Sigmund and his co-workers [10,11] both for solving 2D topology optimization problems based on SIMP. G. Allaire and his co-workers released a set of Scilab routines [12] that perform S&T optimization for 2D elastic structures based on the level set method. Talischi *et al.* [13] presented a 2D Matlab code for structural topology optimization that includes a general finite element routine based on isoparametric polygonal elements, which can be considered as an extension of linear triangles and bilinear quads. As far as 3D topology optimization problems only limited number of codes can be found, like the 169-line Matlab code by Liu and Tovar [14] where SIMP is used and the 100-line Python code by Zuo and Xie [15] that is based on the bi-directional evolutionary structural optimization method. All these codes are limited to implementations of S&T optimization using interpret programming languages (like Matlab, Python and Scilab) while the solution of the system of equations resulted

from the finite element (FE) modelling is also part of these codes. More exceptional geometries and structural systems are not easy to deal with these codes, while only linear analysis is support by most of these codes. In this work a C# source code that is a general-purpose, object-oriented programming language, is presented. C# language was developed by Microsoft within its .NET initiative and standardized by ECMA (ECMA-334) and ISO (ISO/IEC 23270:2006), however it is not limited to just Microsoft platforms; C# compilers exist for FreeBSD, Linux and Macintosh computing platforms. C# language was chosen because it is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines.

The basic advantage of the C# code presented here in, is that it provides a fully functional integration of the S&T optimization concept into a commercial structural and analysis software, specially tailored for structural engineering purposes. Therefore, the users of TOCP code will be able to benefit from all modelling and analysis-design capabilities that commercial software provide. SAP2000 [16], that was introduced over 30 years ago, is equipped with an advanced and rather simple to use open application programming interface (OAPI) making ease the communication of external developers with the capabilities of the software. In addition, a second reason for choosing SAP2000 [16] is because it is available for academic purposes almost free of charge.

The rest of the paper is organized as follows: In order to make the manuscript readable by unfamiliar to S&T optimization researchers, a rather extensive theoretical background regarding the optimization problem formulation and the solution methods implemented in TOCP code, are presented in Section 2. Section 3 discusses the general framework of SAP2000 OAPI implementation in order to facilitate future developers. Section 4 is devoted to the description of TOCP code, component by component, representing the numerical implementation of the theoretical parts of Section 2. Finally, some simple optimization test examples are presented in order to assist the understanding both of the code and also its integration into SAP2000, as well as how to take advantage of a commercial software capabilities and use those in research practice.

2. TOPOLOGY OPTIMIZATION

In order to present the components of TOCP code and make the paper self-contained several theoretical parts of S&T optimization are provided in this section. Structural topology optimization can be considered as a procedure for optimizing the topological arrangement of material into the design domain, eliminating the material volume that is not needed. It can also be seen as the problem of finding the structural layout that best transfers specific loading conditions to supports. It can be employed in order to generate an acceptable initial layout of the structural system, which is then refined by means of shape optimization procedure. Therefore, it can be used to assist de-

signer to define the structural system that satisfies best the operating conditions. Topology optimization procedure operates with gradual "removal" of low stressed material. This is treated as typical structural reanalysis problem where small variations are encountered on the stiffness matrix between subsequent optimization steps.

2.1 Problem Formulation

In topology optimization problem formulations the quantities provided are the domain Ω, where the optimized layout will be created, the required volume fraction of the optimized layout, the boundary Γ and loading conditions F. As shown in Figure 1, the boundary conditions Γ consist of Γ_o, Γ_s, Γ_t and Γ_u, parts where $\Gamma = \Gamma_u \cup \Gamma_o \cup \Gamma_t \cup \Gamma_s$. The surface tractions t are applied at region Γ_t; regions Γ_s denote the non-optimizable areas; Γ_u represent the support conditions and Γ_o are the geometric boundaries of Ω.

The problem can be solved using material distribution methods [6] for finding the optimum layout of a structural system composed by linearly elastic isotropic material. Therefore, the question under investigation is how to distribute material volume into domain Ω in order to minimize a specific criterion; compliance C is a commonly used criterion. The distribution of the material volume in domain Ω is controlled by the density values x distributed over the domain. More specifically, it is controlled by design parameters that are represented by the densities x_e assigned to the FE discretization of domain Ω. The densities (x or x_e) take values in the range [0,1], where zero denotes no material in the specific element. The mathematical formulation of the topology optimization problem can be expressed as:

$$\begin{cases} \min\limits_{x} C(x) = F^T \bar{u}(x) \\ s.t. \begin{cases} \dfrac{V(x)}{V_0} = f \\ F = K(x)\bar{u}(x) \\ 0 < x_{\min} \le x \le 1 \end{cases} \end{cases} \tag{1}$$

where $C(x)$ is the compliance of the structure, F is a force vector and $\bar{u}(x)$ is the corresponding global displacement vector. The second expression of Eq. (1) refers to a volume constraint where f is the volume fractions of domain Ω that the optimized layout should occupy. The third equality of Eq. (1) corresponds to the equilibrium equation where $K(x)$ is the global stiffness matrix of the structural system. The inequality of Eq. (1) denotes the definition set of the density values.

2.2 Solid Isotropic Material with Penalization

SIMP method was proposed by Bendsøe and Sigmund [6] aiming to deal with the problem of Eq. (1) and is implemented in TOCP code presented herein. According to SIMP method the finite

elements' density values are correlated with the corresponding Young modulus value *E*, through the following expression:

$$E_e(x_e) = x_e^p E_e^0 \Leftrightarrow k_e(x_e) = x_e^p k_e^0 \tag{2}$$

where the parameter *p* is usually taken equal to 3. This power law correlation is implemented in SIMP to achieve density values closer to the lower and upper bounds of the design variables (i.e. 0 and 1). The calculation formula of compliance can be written as follows:

$$C(x) = F^T \bar{u}(x) = \bar{u}(x)^T K(x) \bar{u}(x) \tag{3}$$

Therefore, based on Eq. (2), compliance calculation formula can be expressed as:

$$C(x) = \bar{u}(x)^T K(x) \bar{u}(x) = \sum_{e=1}^{N} (x_e)^p u_e^T k_e^0 u_e \tag{4}$$

where *N* is the number of finite elements and u_e is the displacement vector in the elements' local coordinate system. Thus, using Eqs. (2) and (4) the formulation of the topology optimization problem of Eq. (1) can now be expressed as follows:

$$\begin{cases} \min\limits_x C(x) = \bar{u}(x)^T K(x) \bar{u}(x) = \sum\limits_{e=1}^{N} (x_e)^p u_e^T k_e^0 u_e \\ \quad s.t. \begin{cases} \dfrac{V(x)}{V_0} = f \\ F = K(x)\bar{u}(x) \\ 0 < x_{\min} \le x \le 1 \end{cases} \end{cases} \tag{5}$$

Among others, the optimization problem formulated according to SIMP in Eq. (5) can be solved using either the optimality criteria (OC) method or the method of moving asymptotes (MMA), both described in the following sections.

2.3 Optimality Criteria Method

Calculating the derivative of *C(x)* represents an important part of both OC and MMA algorithms. In order to avoid calculating the derivative of the displacement vector $\bar{u}(x)$ with respect to design vector (x_e) a zero part is subtracted from *C(x)*:

$$C(x) = F^T \bar{u}(x) - \lambda^T \left(K(x)\bar{u}(x) - F \right) \tag{6}$$

Therefore, the partial derivatives of *C(x)* become:

$$\begin{aligned} \frac{\partial C(x)}{\partial x_e} &= F^{\mathrm{T}} \frac{\partial u_e(x)}{\partial x_e} - \lambda \frac{\partial k_e(x)}{\partial x_e} u_e(x) - \lambda k_e(x) \frac{\partial u_e(x)}{\partial x_e} \\ &= -\lambda \frac{\partial k_e(x)}{\partial x_e} u_e(x) + (F - \lambda k_e(x)) \frac{\partial u_e(x)}{\partial x_e} \end{aligned} \tag{7}$$

Since vector λ takes an arbitrary value, the value $\lambda = u_e(x_e)$ is used. Thus, the derivatives become:

$$\frac{\partial C(x)}{\partial x_e} = -u_e(x)^T \frac{\partial k_e(x)}{\partial x_e} u_e(x) \tag{8}$$

OC is an iterative search algorithm where the solution vector is updated in very iteration until convergence. First a linear approximation of *C(x)* is defined close to the design variable vector x^k:

$$C(x) = C(x^k) + (y_e - y_e^k)\sum_{e=1}^{N}\left.\frac{\partial C(x)}{\partial y_e}\right|_{x=x^k} = \\ C(x^k) + y_e\sum_{e=1}^{N}\left.\frac{\partial C(x)}{\partial y_e}\right|_{x=x^k} - \mathrm{y}_e^k\sum_{e=1}^{N}\left.\frac{\partial C(x)}{\partial y_e}\right|_{x=x^k} \tag{9}$$

where $y_e = x_e^{-a}$ and the derivatives of *C(x)* with respect to y_e are calculated as follows:

$$\frac{\partial C}{\partial y_e} = \frac{\partial C}{\partial x_e}\frac{\partial x_e}{\partial y_e} = \frac{\partial C}{\partial x_e}\frac{1}{\frac{\partial x_e^{-a}}{\partial x_e}} = -\frac{x_e^{1+a}}{a}\frac{\partial C}{\partial x_e} \tag{10}$$

Then *C(x)* becomes:

$$C(x) = C(x^k) + \sum_{e=1}^{N}\mathrm{y}_e^k\left(\frac{(x_e^k)^{1+a}}{a}\frac{\partial C(x)}{\partial x_e}\right) + \sum_{e=1}^{N} b_e^k x_e^{-a} \tag{11}$$

where

$$b_e^k = -\frac{(x_e^k)^{1+a}}{a}\left.\frac{\partial C}{\partial x_e}\right|_{x=x^k} \tag{12}$$

since the derivatives of the compliance (*C(x)*) can take negative values only:

$$\sum_{e=1}^{N}\mathrm{y}_e^k\left(\frac{(x_e^k)^{1+a}}{a}\frac{\partial C}{\partial x_e^k}\right) < 0 \quad \text{and} \quad \sum_{e=1}^{N} b_e^k x_e^{-a} > 0 \tag{13}$$

Therefore, in order to maximize the subtracting part of the objective function *C(x)* only the positive part need to be minimized:

$$\sum_{e=1}^{N} b_e^k x_e^{-a} > 0 \tag{14}$$

The following subproblem can now be formulated as follows:

$$\begin{cases} \min\limits_{x_e} C(x) = \sum\limits_{e=1}^{N} b_e^k x_e^{-a} \\ s.t. \begin{cases} \sum\limits_{e=1}^{N}\alpha_e x_e = V \\ 0 \le x_e \le 1 \end{cases} \end{cases} \tag{15}$$

In order to solve this problem the Lagrangian Duality method is applied where the augmented Lagrangian function is expressed as follows:

$$L(x_e, \lambda) = \sum_{e=1}^{N} b_e^k x_e^{-a} + \lambda\left(\sum_{e=1}^{N}\alpha_e x_e - V\right) \tag{16}$$

The minimum value of x_e resulted from the solution of the subproblem of Eq. (15) is obtained minimizing $L(x_e,\lambda)$ with respect to x_e and maximizing $L(x_e,\lambda)$ with respect to λ. In order to calculate x_e, the derivatives of $L(x_e,\lambda)$ with respect to x_e are defined first:

$$\frac{\partial L}{\partial x_e} = -ab_e^k x_e^{-a-1} + \lambda\alpha \tag{17}$$

and therefore the values of x_e are obtained:

$$\frac{\partial L}{\partial x_e} = 0 \Leftrightarrow x_e = \left(\frac{ab_e^k}{\lambda\alpha_e}\right)^{\frac{1}{1+a}} \tag{18}$$

Since x_e takes values in the range [0,1] and large changes should be avoided, x_e is updated according to the following rules:

$$x_e^{new} = \begin{cases} max(0, x_e - m) & \text{if } \left(\frac{ab_e^k}{\lambda\alpha_e}\right)^{\frac{1}{1+a}} < max(0, x_e - m) \\ \left(\frac{ab_e^k}{\lambda\alpha_e}\right)^{\frac{1}{1+a}} & \text{if } max(0, x_e - m) < (\frac{ab_e^k}{\lambda\alpha_e})^{\frac{1}{1+a}} < min(1, x_e + m) \\ min(1, x_e + m) & \text{if } \left(\frac{ab_e^k}{\lambda\alpha_e}\right)^{\frac{1}{1+a}} > min(1, x_e + m) \end{cases} \tag{19}$$

m is the maximum change allowed for x_e. Similar to x_e, in order to calculate λ, the derivatives of $L(x,\lambda)$ in respect to λ are defined:

$$\frac{\partial L}{\partial \lambda} = \sum_{e=1}^{N} \alpha_e x_e - V \tag{20}$$

The calculation of λ is achieved by iteratively choosing values of λ for each x_e until satisfaction of the following equality:

$$\frac{\partial L}{\partial \lambda} = 0 \Leftrightarrow \sum_{e=1}^{N} \alpha_e x_e - V = 0 \Leftrightarrow \sum_{e=1}^{N} \alpha_e x_e = V \tag{21}$$

2.4 Method of Moving Asymptotes

Similar to OC, the method of moving asymptotes is an iterative procedure proposed for solving nonlinear optimization problems, in particular of the following form:

$$\begin{cases} \min\limits_{x \in R^N} f_0(x) \\ s.t. \begin{cases} f_i(x) \le f, & \text{for } i = 1, 2, \ldots, n \\ \underline{x}_j \le x_j \le \bar{x}_j, & \text{for } j = 1, 2, \ldots, N \end{cases} \end{cases} \tag{22}$$

For every iteration k, the values of functions $f_0(x^k)$, $f_1(x^k)$ to $f_m(x^k)$ together with their partial derivatives need to be calculated. MMA rely on first order approximations, where the required estimates for design x^k are defined using the values of previous steps. Afterwards, Lagrangian Duality method is implemented in order to derive design x^{k+1} using these approximations. Specifically functions $f_0(x^k)$, $f_1(x^k)$ to $f_m(x^k)$ are approximated through the following expression:

$$f_i^{(k)}(x) = r_i^{(k)} + \sum_{j=1}^{N} \left[\frac{p_{ij}^{(k)}}{(U_j^{(k)} - x_j)} + \frac{q_{ij}^{(k)}}{(x_j - L_j^{(k)})} \right] \tag{23}$$

where

$$p_{ij}^{(k)} = \begin{cases} (U_j^{(k)} - x_j^{(k)})^2 \dfrac{\partial f_i}{\partial x_j}, \ if \quad \dfrac{\partial f_i}{\partial x_j} > 0 \\ 0 \qquad\qquad\qquad if \quad \dfrac{\partial f_i}{\partial x_j} \le 0 \end{cases} \tag{24a}$$

$$q_{ij}^{(k)} = \begin{cases} 0 \qquad\qquad\qquad if \quad \dfrac{\partial f_i}{\partial x_j} > 0 \\ -(x_j^{(k)} - L_j^{(k)})^2 \dfrac{\partial f_i}{\partial x_j}, if \quad \dfrac{\partial f_i}{\partial x_j} \le 0 \end{cases} \tag{24b}$$

$$r_i^{(k)} = f_i(x^{(k)}) - \sum_{j=1}^{n} \left[\frac{p_{ij}^{(k)}}{(U_j^{(k)} - x_j^{(k)})} + \frac{q_{ij}^{(k)}}{(x_j^{(k)} - L_j^{(k)})} \right] \tag{24c}$$

and the second order partial derivatives of f_i with respect to x_j are calculated through the following expressions:

$$\frac{\partial^2 f_i^k}{\partial x_j^2} = \begin{cases} \dfrac{2\dfrac{\partial f_i}{\partial x_j}}{U_j^k - x_j^k} & if \ \dfrac{\partial f_i}{\partial x_j} > 0 \\ -\dfrac{2\dfrac{\partial f_i}{\partial x_j}}{x_j^k - L_j^k} & f \ \dfrac{\partial f_i}{\partial x_j} < 0 \end{cases} \tag{25}$$

where, as it can be observed for Eq. (25) they are positive definite, and their values increase when U_j and L_j, called "moving asymptotes", tend to x_j. The implementation of MMA into the topology optimization problem of Eq. (5) requires some modifications, i.e. only compliance $C(x)$ needs to be approximated, while since its partial derivatives are always negative then $p^k = 0$. Thus, the topology optimization problem of Eq. (15) becomes:

$$\begin{cases} g(x) = \min\limits_{x_e} \left[r_i^k + \sum\limits_{e=1}^{N} \dfrac{q_e^{(k)}}{x_e - L_e^{(k)}} \right] \\ \qquad s.t. \begin{cases} \sum\limits_{e=1}^{N} x_e a = V \\ 0 \le x_e \le 1 \\ L_e^{(k)} \le x_e^{(k)} \le U_e^{(k)} \end{cases} \end{cases} \tag{26}$$

where Lagrangian Duality method is applied also for solving this problem and the augmented Lagrangian function is expressed as follows:

$$L(x_e, \lambda) = r_i^k + \sum_{e=1}^{N} \left(\frac{q_e^{(k)}}{x_e - L_e^{(k)}} \right) + \lambda \, (\sum_{e=1}^{N} x_e \, a_e - V) \tag{27}$$

The values of x_e and λ are obtained by minimizing $L(x_e,\lambda)$ with respect to x_e and maximizing $L(x_e,\lambda)$ with respect to λ. In order to calculate x_e, the derivative of $L(x_e,\lambda)$ with respect to x_e is defined first:

$$\frac{\partial L(x_e,\lambda)}{\partial x_e} = -\frac{q_e^{(k)}}{(x_e - L_e^{(k)})^2} + \lambda \tag{28}$$

thus the values of x_e are obtained as follows:

$$\frac{\partial L}{\partial x_e} = 0 \Leftrightarrow x_e = \sqrt{\frac{q_e^{(k)}}{\lambda}} + L_e^{(k)} \tag{29}$$

implementing a maximum movement factor m and a, b as move limits, x_e is updated according to the following rules:

$$x_e^{new} = \begin{cases} max(0, x_e - m, a_e^{(k)}) & \text{if } \sqrt{\frac{q_e^{(k)}}{\lambda}} + L_e^{(k)} < max(0, x_e - m, a_e^{(k)}) \\ \sqrt{\frac{q_e^{(k)}}{\lambda}} + L_e^{(k)} & \text{if } max(0, x_e - m, a_e^{(k)}) < \sqrt{\frac{q_e^{(k)}}{\lambda}} + L_e^{(k)} < min(1, x_e + m, \beta_e^{(k)}) \\ min(1, x_e + m, \beta_e^{(k)}) & \text{if } \sqrt{\frac{q_e^{(k)}}{\lambda}} + L_e^{(k)} > min(1, x_e + m, \beta_e^{(k)}) \end{cases} \tag{30}$$

To obtain λ, $L(x_e,\lambda)$ is maximized with respect to λ:

$$\frac{\partial L(x_e,\lambda)}{\partial \lambda} = \sum_{e=1}^{N} x_e\, a_e - V = 0 \tag{31}$$

This is achieved using the bisection method. The "moving limits" are used to avoid division by zero and can be calculated as follows:

$$\begin{aligned} & a_j^{(k)} = 0.9 L_j^{(k)} + 0.1\, x_j^{(k)} \\ & \text{and} \\ & \beta_j^{(k)} = 0.9 U_j^{(k)} + 0.1 x_j^{(k)} \end{aligned} \tag{32}$$

Moving asymptotes $L_j^{(k)}$ and $U_j^{(k)}$ are selected based on the rule proposed by Svanberg [17]: In case of oscillation values for the moving asymptotes closer to $x_j^{(k)}$ are selected, otherwise, in case of slow convergence values far away from $x_j^{(k)}$ are used. According to Liu and Tovar [14] the moving asymptotes are calculated using an update scheme that is based on three successive iterations. In particular, for k=1 and 2:

$$\left.\begin{aligned} U_j^{(k)} + L_j^{(k)} &= 2x_j^{(k)} \\ U_j^{(k)} - L_j^{(k)} &= 1 \end{aligned}\right\} \begin{aligned} L_j^{(k)} &= x_j^{(k)} - 0.5 \\ U_j^{(k)} &= x_j^{(k)} + 0.5 \end{aligned} \tag{33a}$$

for k≥3:

$$\left.\begin{aligned} U_j^{(k)} + L_j^{(k)} &= 2x_j^{(k)} \\ U_j^{(k)} - L_j^{(k)} &= \gamma_j^{(k)} \end{aligned}\right\} \begin{aligned} L_j^{(k)} &= x_j^{(k)} - 0.5\gamma_j^{(k)} \\ U_j^{(k)} &= x_j^{(k)} + 0.5\gamma_j^{(k)} \end{aligned} \tag{33b}$$

where

$$\gamma_j^{(k)} = \begin{cases} 0.7 \text{ if } (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) < 0 \\ 1.2 \text{ if } (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) > 0 \\ 1.0 \text{ if } (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) = 0 \end{cases} \tag{33c}$$

3. SAP2000 OAPI IMPLEMENTATION

In computer programming, an application programming interface (API) refers to a set of subroutine definitions, protocols, and tools for building application software. In general terms, it's a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programming developer. The SAP2000 open application programming interface allows developers to automate many of the processes required to build, analyse, and design models and to obtain customized analysis and design results. It also allows developers to link SAP2000 with third-party tools, providing a path for two-way exchange of model information with other applications. Based on these capabilities provided by SAP2000 OAPI, in this section details on the TOCP C# implementation of minimum compliance topology optimization framework, interfacing with SAP2000, is presented. Although, the code described herein is based on SAP2000 version 15 OAPI documentation, the corresponding code for the latest version 19 will also be provided and can be downloaded from the following site: http://users.ntua.gr/nlagaros/TOCP/projectsSMO.zip, users are free to use these codes as long as they acknowledge their source.

Before describing how parts of topology optimization described previously are modified in order to take advantage of SAP2000 OAPI, the basic issues related to the use of OAPI are provided in this section along with justification for choosing SAP2000 over other also well-known software for analysis and design of structural systems.

3.1 Why Choosing SAP2000

Over the years, SAP2000 has proven to be one of the most integrated, general-purpose structural software on engineering practice today. It is considered as one of the easiest, most productive software for structural analysis and design needs (from a simple small 2D static frame analysis to a large complex 3D nonlinear dynamic analysis). The interface of SAP2000 allows creating structural models rapidly and intuitively. Complex models can be generated and meshed using built in templates. This is the reason of less importance we have decided to use the specific software to integrate with the S&T optimization concept. Worth mentioning also that in case of academic purposes needs, multiple licenses for unlimited usage are provided almost for free.

However, the main reason for choosing SAP2000 is due to the fact that through OAPI it’s modelling, analysis-design capabilities are fully accessible. By means of its OAPI, integrated design code features can be used to automatically generate wind, wave, bridge, and seismic loads with automatic steel and concrete design code checks per European, US, Canadian, Chinese and other international design standards. In addition, various analytical techniques that allow for step-by-step large deformation analysis, Eigen and Ritz analyses based on stiffness of nonlinear cases,

catenary cable analysis, material nonlinear analysis with fibre hinges, buckling analysis, progressive collapse analysis, or plasticity and nonlinear segmental construction analysis along with specialised modelling capabilities using multi-layered nonlinear shell element, velocity-dependent dampers, or base isolators, are also accessible through OAPI.

3.2 SAP2000 OAPI basic features

In order to access SAP2000 from an external application using its OAPI the following steps need to be followed: The first step is to reference SAP2000 from the application. Specifically, as soon as a new visual studio (VS) C# console application project was created a reference (COM) is added in the project to SAP2000.EXE. Next an instance of the SAP2000 object needs to be created (also known as instantiating the object) within the developed application. This is accomplished as follows:

```
L1  SAP2000v15.SapObject Object;
L2  Object = new SAP2000v15.SapObject();
```

where the command of first line #L1 creates the object variable and the second line #L2 creates an instance of the SAP2000 object. Now that an instance of the SAP2000 object has been created in the application, SAP2000 can be started using the following command:

```
L3  Object.ApplicationStart(SAP2000v15.eUnits.kN_m_C,true,"");
```

At this point an existing model can be opened, or a new one to be created and perform any action required. In general, OAPI commands are accessed through `Object.SapModel`. It might be helpful to define a `SapModel` object so that the OAPI commands are accessed through `Model` instead of `Object.SapModel`. For example, a new SAP2000 is initiated with the following steps, a `Model` object is created first:

```
L4  SAP2000v15.cSapModel Model;
L5  Model = Object.SapModel;
```

and then the model is initialized as follows:

```
L6  int ret = Model.InitializeNewModel(Sap2000.eUnits.kip_in_F);
```

When finished with a model, it might be needed to close SAP2000 application. This can be accomplished using the following command:

```
L7  Object.ApplicationExit(false);
```

As a last step, `Model` and `Object` objects should be set to `null`. This is accomplished as follows:

```
L8  Model = null;
L9  Object = null;
```

Setting the objects to `null` is a very important step. It breaks the connection between developer's application and SAP2000 and frees up system resources. If the objects are not set to `null`, the SAP2000 application will not completely close (it will still be seen running in Windows task manager).

3.3 Specific Features of SAP2000 OAPI Used in Topology Optimization Implementation

Most of the functions of SAP2000 OAPI that are used in this study in order to integrate the topology optimization implementation into SAP2000 are described in this section. GetNameList (`ret = Model.SolidObj.GetNameList();`), this function retrieves the names of all defined solid objects. SetLocalAxes (`ret = Model.SolidObj.SetLocalAxes();`), this function sets the local axes angles for solid objects. AddMaterial (`ret = Model.PropMaterial.AddMaterial();`) that is a function that adds a new material property to the model based on the pre-defined material properties. GetMPIsotropic (`ret = Model.PropMaterial.GetMPIsotropic();`) that is a function that retrieves the mechanical properties for a material with an isotropic directional symmetry type. SetMPIsotropic (`ret = Model.PropMaterial.SetMPIsotropic();`) that is a function that sets the material directional symmetry type to isotropic, and assigns the isotropic mechanical properties. SetProp (`ret = Model.PropSolid.SetProp();`), this function assigns a solid property to solid objects. ClearSelection (`ret = Model.SelectObj.ClearSelection();`), this function deselects all objects in the model. CoordinateRange (`ret = Model.SelectObj.CoordinateRange();`), this function selects or deselects objects inside the box defined by the X_{Min}, X_{Max}, Y_{Min}, Y_{Max}, Z_{Min} and Z_{Max} coordinates. GetSelected (`ret = Model.SelectObj.GetSelected();`), this function retrieves a list of selected objects. SetModelIsLocked (`ret = Model.SetModelIsLocked();`) this function locks or unlocks the model. SolidJointForce (`ret = Model.Results.SolidJointForce();`), this function reports the joint forces for the point elements at every corner of the specified solid elements. JointDispl (`ret = Model.Results.JointDispl();`), this function reports the joint displacements for the specified point elements. The displacements reported by this function are relative displacements. GetProperty (`ret = Model.AreaObj.GetProperty();`), this function retrieves the area property assigned to an area object. GetType (`ret = Model.PropArea.GetType();`), this function retrieves the property type for the specified area property. Delete (`ret = Model.SolidObj.Delete();`), this function deletes solid objects. SetRunCaseFlag (`ret = Model.Analyze.SetRunCaseFlag();`), this function sets the run flag for load cases. RunAnalysis (`ret = Model.Analyze.RunAnalysis();`), this function runs the analysis. The analysis model is automatically created as part of this function. DeselectAllCasesAndCombosForOutput (`ret = Model.Results.Setup.DeselectAllCasesAnCombosForOutput();`), this function deselects all load cases and response combinations for output. SetCaseSelectedForOutput (`ret = Model.Results.Setup.SetCaseSelectedForOutput();`), this function sets a load case selected for output flag.

4. C# IMPLEMENTATION-TOCP: CODE DESCRIPTION

The current study is accompanied by four visual studio C# projects that can be downloaded from http://users.ntua.gr/nlagaros/TOCP/projectsSMO.zip. All four VS projects correspond to different integrations of the S&T optimization concept into SAP2000 software using its OAPI. In particular, the first two projects labelled as TOCP1 and TOCP2 refer to the implementations of OC and MMA algorithms, respectively, and the mesh of the optimizable domain is provided by the user; while the latter two projects labelled as TOCP3 and TOCP4 denote also the implementations of the two algorithms however the mesh of the optimizable domain is generated automatically and the user needs to provide also the representative element's discretization *nlx*×*nly*×*nlz*. In this section, the C# code corresponding to the implementation of OC is outlined while description of the numerical implementation of MMA is also provided. Once this code is understood the rest ones can easily be comprehended. The term method that is used in the current and next sections refers to a group of statements that together perform a task. In this section ten methods that are used in OC and MMA visual studio projects are described, together with the methods written for implementing OC and MMA algorithms. All these methods are described in detail below:

4.1 ReadModel method

`ReadModel` refers to the method that is used first by the four projects. Taking advantage of the modelling capabilities of SAP2000, a structural system can be modelled using 1D, 2D and/or 3D finite elements. Using an instance of this method several arrays are returned containing the nodes, the finite elements (solid, frame and area ones) of the model and the load cases considered. Part of the `ReadModel` method is provided in the code below, where indicatively in line #30 the 3D finite elements (solid ones) used in the model are retrieved using function `GetNameList` and in line #34 each solid element's local axes angles are set the same as the global ones. Similar procedure is implemented both for 1D and 2D finite elements and their corresponding nodes.

```
28  SolidName = new string[0];
29  SolidNumb = 0;
30  ret = Model.SolidObj.GetNameList(ref SolidNumb, ref SolidName);
31  Console.WriteLine("{0} solid objects", SolidNumb);
32  for (i = 0; i <= SolidNumb - 1; i++)
33  {
34    ret = Model.SolidObj.SetLocalAxes(SolidName[i],a,b,c, SAP2000v15.eItemType.Object);
35  }
```

4.2 CreateMatSolid method

In order to define 3D finite elements in SAP2000, solid sections need to be defined first. Each solid section property, containing among others the material characteristics, is assigned to groups of solid finite elements. According to SIMP, the elements' modulus of elasticity are penalized through density values x_e using the formula of Eq. (2). This is carried out by the `CreateMatSolid` method, where a number of material classes are generated first (for example 101 different material

classes as described in lines #72 to #77 of the following code); initially, having identical properties and they are stored in array `MatName`. This is described in the following part of `CreateMatSolid` method:

```
72    for (i = 0; i <= 100; i++)
73    {
74       temp_name = "OptiMat" + Convert.ToString(i);
75a      ret = Model.PropMaterial.AddMaterial(ref temp_mat,
75b            SAP2000v15.eMatType.MATERIAL_CONCRETE, Region, Standard, Grade, temp_name);
76       MatName[i] = temp_mat;
77    }
```

In subsequent part of `CreateMatSolid` method (i.e. lines #79 to #87), the modulus of elasticity is modified according to the following formula (see Line #84):

$$E_i = E_i + 0.01 \times E^0 \tag{34}$$

In order to avoid section properties having modulus of elasticity values equal to zero, that might generate numerical instability issues during the analysis, the first material property corresponding to the lowest modulus of elasticity value becomes equal to $E_0 = 10.0\text{E-}07 \times E^0$ (where E_0 corresponds to the first material class while E^0 is the initial value of the modulus of elasticity). This is described in lines #79 and #80 of the code provided below along with the generation of the rest of the material properties (i.e. lines #81 to #87):

```
79  ret = Model.PropMaterial.GetMPIsotropic(MatName[0], ref e, ref u, ref amat, ref g);
80  ret = Model.PropMaterial.SetMPIsotropic(MatName[0], e * 0.0000001, u, amat);
81  double metr = 0.01;
82  for (i = 1; i <= 100; i++)
83  {
84     ret = Model.PropMaterial.SetMPIsotropic(MatName[i], e * metr, u, amat);
85     metr = metr + 0.01;
86     metr = Math.Round(metr, 2);
87  }
```

In the last part of the code of `CreateMatSolid` method (i.e. lines #89 to #96) each material property defined according to Eq. (34) is assigned to a solid section property. Each solid section is labelled as OptiSolidProp#, where # stands for an id value that denotes the modulus of elasticity value of the specific section (i.e. OptiSolidProp0.01 stands for the solid section property whose modulus of elasticity is equal to $E_i = 0.01 \times E^0$). This is described in the code below:

```
89  metr = 0.00;
90  for (i = 0; i <= 100; i++)
91  {
92      temp_name = "OptiSolidProp" + string.Format("{0:N2}", metr);
93      ret = Model.PropSolid.SetProp(temp_name,MatName[i],0.0,0.0,0.0,0.0,true,-1,"","");
94      metr = Math.Round(metr, 2) + 0.01;
95      metr = Math.Round(metr, 2);
96  }
```

Using an instance of `CreateMatSolid` method, modulus of elasticity values that will be used in the following steps of the topology optimization procedure are assigned to solid section properties.

4.3 *CreateOptiSolidName method*

The use of an instance of the `CreateOptiSolidName` method creates the array `OptiSolidName` containing the ids of the model's solid elements whose density values will be the design variables of the topology optimization problem. The optimizable solid finite elements are generated into a rectangular domain (optimizable domain) that is defined by the user providing the bounds of its three dimensions (i.e. xmin, xmax, ymin, ymax, zmin and zmax, see Figure 2). In addition, the dimension of the quadratic finite elements needs also to be provided by the user.

`CreateOptiSolidName` method based on the nodal coordinates of the solid finite elements, in line #119 selects which elements will be optimized using OAPI function `Model.SelectObj.CoordinateRange`. Next, in line #124 the method obtains the id (i.e. `ObjNameSel`) of the selected solid element and stores it in the proper location of array `OptiSolidName` (see line #134). The arrangement of the solid elements' ids in the array `OptiSolidName` is the same with that used in [14]. An example of the numbering implementation for a mesh 5×3×5 of optimizable solid finite elements (along x, y and z axes, respectively) can be seen in Figure 3.

```
112a  for (x_double = Math.Round(xmax,3); x_double >= Math.Round(xmin,3) +
112b  Math.Round(diakr/2, 3); x_double = Math.Round(x_double, 3) - Math.Round(diakr, 3))
113   {
114a     for (z_double = Math.Round(zmax,3); z_double >= Math.Round(zmin,3) +
114b     Math.Round(diakr/2,3); z_double = Math.Round(z_double,3) - Math.Round(diakr,3))
115      {
116a         for (y_double = Math.Round(ymax,3); y_double >= Math.Round(ymin,3)+
116b         Math.Round(diakr/2,3); y_double = Math.Round(y_double,3)-
116c         Math.Round(diakr, 3))
117          {
118              ret = Model.SelectObj.ClearSelection();
119a             ret = Model.SelectObj.CoordinateRange(Math.Round(x_double,3) -
119b                   Math.Round(diakr,3), Math.Round(x_double,3), Math.Round(y_double,3)-
119c                   Math.Round(diakr,3), Math.Round(y_double,3), Math.Round(z_double,3)-
119d                   Math.Round(diakr,3), Math.Round(z_double,3), false, "Global", false,
119e                   false, false, false, true, false);
124a             ret = Model.SelectObj.GetSelected(ref NumbSel, ref ObjTypeSel,
124b                   ref ObjNameSel);
133              count = count + 1;
134              OptiSolidName[count - 1] = ObjNameSel[0];
136           }
137       }
138   }
```

4.4 *CreateMapDist method*

In order to avoid the checkerboard problem filtering needs to be applied both to new density values and derivatives. In every iteration the filter is applied to each solid finite element based on its neighbouring elements. To facilitate this process an instance of the `CreateMapDist` method needs to be used first before initiating the iterations. This method is used for creating the array `Map` that contains the ids of each solid finite element's neighbouring elements. The first three iterative loops (that begin in lines #175, #177 and #179) are implemented in order to identify in line #181 each solid finite element (`e_int`) while the next three iterative loops (that begin in lines #189, #191 and

#193) are employed in order to find in line #195 all its neighbouring located into a sphere having radius equal to `rmin`. In addition, `CreateMapDist` method calculates the distances $H_{e,i} = max\left(0, rmin - D(e,i)\right)$ for each solid finite element (see line #196), where *D(e,i)* is the distance between the solid finite element *e* and every of its neighbouring solid finite elements *i*, and stores them in array `Dist` (see line #198).

```
175   for (x_int = 1; x_int <= nlx; x_int++)
176   {
177     for (z_int = 1; z_int <= nlz; z_int++)
178     {
179        for (y_int = 1; y_int <= nly; y_int++)
180        {
181           e_int = (x_int - 1) * nlz * nly + (z_int - 1) * nly + y_int;
182           x_double_min = Math.Truncate(Math.Max(x_int - rmin, 0.0)) + 1;
183           z_double_min = Math.Truncate(Math.Max(z_int - rmin, 0.0)) + 1;
184           y_double_min = Math.Truncate(Math.Max(y_int - rmin, 0.0)) + 1;
185           x_double_max = Math.Min(Math.Truncate(x_int + rmin), nlx);
186           z_double_max = Math.Min(Math.Truncate(z_int + rmin), nlz);
187           y_double_max = Math.Min(Math.Truncate(y_int + rmin), nly);
188           j = 0;
189           for (x_double = x_double_min; x_double <= x_double_max; x_double++)
190           {
191             for (z_double = z_double_min; z_double <= z_double_max; z_double++)
192             {
193               for (y_double = y_double_min; y_double <= y_double_max; y_double++)
194               {
195                  e_double = (x_double - 1) * nlz * nly + (z_double - 1) * nly + y_dou-
196a  ble;
196b                 H = (rmin - Math.Sqrt(Math.Pow(x_int - Convert.ToInt32(x_double), 2)
196c                     + Math.Pow(y_int - Convert.ToInt32(y_double), 2)
197                      + Math.Pow(z_int - Convert.ToInt32(z_double), 2))) * diakr;
198                  Map[e_int - 1, j] = Convert.ToInt32(e_double);
199                  Dist[e_int - 1, j] = Math.Max(H, 0);
200                  j = j + 1;
201               }
202             }
203           }
204        }
205     }
       }
```

The procedure of finding neighbouring elements that is implemented with the formulas of lines #181 and #195 is based on the elements' names that Sigmund [10] proposed and Liu and Tovar [14] also applied in their 3D code and is currently adopted in `CreateOptiSolidName` method.

4.5 UpdateModel method

In this part of the VS project, the solid section properties with the corresponding modulus of elasticity values are assigned to the optimizable solid finite elements. This is performed with the use of an instance of `UpdateModel` method as described in the lines of the code below:

```
214  ret = Model.SetModelIsLocked(false);
215  for (i = 0; i <= OptiSolidNumb - 1; i++)
216  {
217     temp_double = Math.Round(xkfil[i] * xkfil[i] * xkfil[i], 2);
218     temp_string = "OptiSolidProp" + string.Format("{0:N2}", temp_double);
219     ret = Model.SolidObj.SetProperty(OptiSolidName[i], temp_string, 0);
220  }
```

The rounded in the second decimal digit of the 3rd power of the density value of the ith solid finite element is calculated in line #217, over the total number of solid finite elements, whose density values represent variables of the problem. In line #218, this density value is converted into a string variable and is stored in the temporary variable `temp_string` composed by the prefix "`OptiSolidProp`" and followed by the value (i.e. OptiSolidProp0.01). In line #219, a solid section property is assigned to every optimizable solid finite element based on the name that is stored in temporal variable `temp_string`.

4.6 CalcCompliance method

In this part of the code, the compliance of the structure is calculated. In line #341 OAPI function `SolidJointForce` is used over the solid elements in order to obtain from SAP2000 joint forces F1, F2, F3, M1, M2 and M3 for every node and they are stored in arrays. In the next for loop, function `JointDispl` is used in order to obtain the displacements U1, U2,U3, R1, R2 and R3 also for every node. The cross product between the arrays of displacements and forces over the nodes of each element results into the total compliance of the structural system (see line #347). Worth mentioning that the calculation of compliance is not a required part for the implementation of the optimization proccess, however, it is provided as a measure for the progress of the search procedure.

```
339   for (i = 0; i <= SolidNumb - 1; i++)
340   {
341a    ret = Model.Results.SolidJointForce(SolidName[i],
341b          SAP2000v15.eItemTypeElm.ObjectElm,
341c          ref nresultS, ref ObjeS, ref elmS, ref pointelmS, ref LoadCaseS, ref
341d          StepTypeS, ref StepNumS, ref F1S, ref F2S, ref F3S, ref M1S,
341e          ref M2S, ref M3S);
342     if (nresultS == 8)
343     {
344        for (j = 0; j <= nresultS - 1; j++)
345        {
346a          ret = Model.Results.JointDispl(pointelmS[j],
346b                SAP2000v15.eItemTypeElm.ObjectElm, ref nresult, ref Obje,
346c                ref elm, ref LoadCase, ref StepType, ref StepNum, ref U1,
346d                ref U2, ref U3, ref R1, ref R2, ref R3);
347a          comp = comp + U1[0] * F1S[j] + U2[0] * F2S[j] + U3[0] * F3S[j] +
347b                  R1[0] * M1S[j] + R2[0] * M2S[j] + R3[0] * M3S[j];
348        }
349     }
350     else
351     {
352a      Console.WriteLine("Error: while calculating the compliance of a solid element
352b                         (Method CalcCompliance). check nresultS");
353       Console.WriteLine("number of results: {0}", nresultS);
354       testbool = false;
355     }
356   }
```

4.7 *CalcDerivatives method*

The calculation of the partial derivatives with respect to x_e is the most important part of the iterative steps of the optimization procedure. This is performed with an instance of `CalcDerivatives` method, whose input argument is the array `xkfil`. According to the theoretical part provided previously the partial derivatives are calculated using Eq. (8). However, the finite elements' stiffness matrices $k_e(x)$ are not available through the OAPI of SAP2000. In order to overcome this difficulty a modified expression is proposed in this study. According to SIMP changing the modulus of elasticity of the optimizable solid finite elements, results into modified stiffness matrices:

$$k_e(x_e) = x_e^3 k_e^0 \Leftrightarrow k_e^0 = \frac{k_e(x_e)}{x_e^3} \tag{35}$$

and

$$\frac{\partial k_e(x_e)}{\partial x_e} = 3x_e^2 k_e^0 = 3x_e^2 \frac{k_e(x_e)}{x_e^3} = 3\frac{k_e(x_e)}{x_e} \tag{36}$$

Thus, Eq. (8) can be rewritten as follows:

$$\begin{aligned} \frac{\partial C(x)}{\partial x_e} &= -u_e(x)^T \frac{\partial k_e(x_e)}{\partial x_e} u_e(x) = \\ -u_e(x)^T 3\frac{k_e(x_e)}{x_e} u_e(x) &= -3\frac{F^T u_e(x)}{x_e} = -\frac{3C_e}{x_e} \end{aligned} \tag{37}$$

`CalcDerivatives` method is used to calculate the work produced by the optimizable solid finite elements based on the procedure described in the previous method (`CalcCompliance`) and then to calculate the partial derivatives according to Eq. (37). The method returns the array `der` containing the derivatives arranged in a similar fashion with the rest ones that have been created already.

```
492   for (i = 0; i <= OptiSolidNumb - 1; i++)
493   {
494      compder = 0.0;
495a    ret = Model.Results.SolidJointForce(OptiSolidName[i],
495b          SAP2000v15.eItemTypeElm.ObjectElm, ref nresultS, ref ObjeS, ref elmS,
495c          ref pointelmS, ref LoadCaseS, ref StepTypeS, ref StepNumS, ref F1S, ref F2S,
495d          ref F3S, ref M1S, ref M2S, ref M3S);
496      if (nresultS == 8)
497      {
498          for (j = 0; j <= nresultS - 1; j++)
499          {
500a           ret = Model.Results.JointDispl(pointelmS[j],
500b                 SAP2000v15.eItemTypeElm.ObjectElm,ref nresult,ref Obje,ref elm,ref
500c                 LoadCase, ref StepType, ref StepNum, ref U1, ref U2, ref U3, ref R1,
500d                 ref R2, ref R3);
501a           compder = compder + (U1[0] * F1S[j]) + (U2[0] * F2S[j]) + (U3[0] * F3S[j])
501b                     +(R1[0] * M1S[j]) + (R2[0] * M2S[j]) + (R3[0] * M3S[j]);
502          }
503       }
504       else
505       {
506a         Console.WriteLine("Error: while calculating the compliance of a solid element
506b                            (Method CalcDerivatives). check nresultS");
507          Console.WriteLine("number of results: {0}", nresultS);
508          testbool = false;
509       }
510       der[i] = -3 * compder / Math.Max(xkfil[i], Math.Pow(10,-3));
```

```
511         comp2 = comp2 + compder;
512     }
```

4.8 FilterDer method

Given that the derivatives have been calculated, their values need to be filtered in order to avoid the checkerboard problem. The input arguments of `FilterDer` method are the arrays `der`, `xkfil`, `Map` and `Dist` defined previously by an instance of `CreateMapDist` method. For each optimizable solid finite element, the ids of its neighbouring elements are retrieved through array `Map`, the corresponding distances `Hei` are obtained through array `Dist` and then the summations of the following expression are calculated:

$$\sum_{i \in N_e} H_{e,i} x_i \frac{\partial C(x)}{\partial x_i} \text{ and } \sum_{i \in N_e} H_{e,i} \tag{38}$$

where $H_{e,i}$ are distances calculated for each solid finite element over its neighbouring solid finite elements N_e. Line #547 describes the implementation of derivatives' filtering procedure that is performed according to the following expression:

$$\frac{\partial C(x)}{\partial x_e} = \frac{1}{\max(10^{-3}, x_e) \sum_{i \in N_e} H_{e,i}} \sum_{i \in N_e} H_{e,i} x_i \frac{\partial C(x)}{\partial x_i} \tag{39}$$

Finally, `FilterDer` method returns the array `derfil[i]` containing the filtered derivatives.

```
530 for (i = 0; i <= OptiSolidNumb - 1; i++)
531 {
532     SHXC = 0.0;
533     SH = 0.0;
534     for (j = 0; j <= 26; j++)
535     {
536        if ((Map[i, j] - 1) >= 0)
537        {
538            SHXC = SHXC + Dist[i, j] * xkfil[Map[i, j] - 1] * der[Map[i, j] - 1];
539            SH = SH + Dist[i, j];
540        }
541        else if ((Map[i, j] - 1) != -1)
542        {
543            Console.WriteLine("problem:  Map[i,j]-1 is {0}", Map[i, j] - 1);
544            testbool = false;
545        }
546     }
547     derfil[i] = SHXC / (SH * Math.Max(xkfil[i],Math.Pow(10,-3)));
548 }
549 return derfil;
```

4.9 OptimalityCriteria method

An instance of `OptimalityCriteria` method preforms the process described in the theoretical part of the study. In order to calculate the value of λ the bisection interval algorithm is applied, where the definition set of λ is defined first:

```
571 al = 0.0;
572 bl = Math.Pow(10, 10);
```

For each value of λ defined as the average value of the corresponding interval, a new density value (variable `xnew[i]`) is calculated according to Eq. (19), using the unfiltered density value of the previous iteration (variable `xk[i]`).

```
573   do
574   {
575      lamda = (al + bl) / 2;
576      for (i = 0; i <= OptiSolidNumb - 1; i++)
577      {
578a        xnew[i] = Math.Max(0, Math.Max(xk[i] - 0.2, Math.Min(1, Math.Min(xk[i] + 0.2,
578b        Math.Sqrt(be[i] / lamda)))));
579       }
```

In order to avoid the checkerboard problem filtering is performed over the optimizable solid elements' density values x_e according to the following expression:

$$x_e = \frac{1}{\sum_{i \in N_e} H_{e,i}} \sum_{i \in N_e} H_{e,i} x_i \tag{40}$$

This is described in the code that follows:

```
580  metrx = 0.0;
581  for (i = 0; i <= OptiSolidNumb - 1; i++)
582  {
583     SHX = 0.0;
584     SH = 0.0;
585     for (j = 0; j <= 26; j++)
586     {
587        if ((Map[i, j] - 1) != -1)
588        {
589           SHX = SHX + Dist[i, j] * xnew[Map[i, j] - 1];
590           SH = SH + Dist[i, j];
591        }
592     }
593     xnewfil[i] = SHX / SH;
594     metrx = metrx + xnewfil[i];
595  }
```

In the code of lines #596 to #608 the constraint described in Eq. (21) is verified. If the sum (variable `metrx`) of filtered density values (x_e) over all optimizable solid finite elements is greater than the required volume fraction value (v), then density values need to be reduced, and therefore λ (variable `lamda`) that lies in the interval $\left(0.5(a+b), b\right)$ needs to be increased. Accordingly, if the sum of density values (x_e) is less than the required volume fraction value, density values need to be increased and the value of λ that lies in the interval $\left(a, 0.5(a+b)\right)$ needs to be decreased. The division process continues until a suitable value of λ is achieved.

```
596     if ((bl - al) / (al + bl) < Math.Pow(10, -3))
597     {
598        testlamda = true;
599     }
600     else if (metrx > V)
601     {
602        al = lamda;
603     }
604     else
605     {
606        bl = lamda;
607     }
608  } while (testlamda == false);
```

4.10 MMA method

Similar to the previous section, an instance of `MMA` method preforms the steps of moving asymptotes algorithm described in the theoretical part of the current study. In particular, the commands in lines #641 to #695 are used to calculate L and U for each solid element according to Eq. (33) and then α and β according to Eq. (32). As it can be observed in lines #645 and #649 the method requires information from two previous iterations (i.e. `xkm1` stands for the element's density value in the previous iteration and `xkm2` stands for the element's density in before previous iteration).

```
641  if (epanal >= 3)
642  {
643      for (i = 0; i <= OptiSolidNumb - 1; i++)
644      {
645          if ((xk[i] - xkm1[i]) * (xkm1[i] - xkm2[i]) < 0)
646          {
647             gamma = 0.7;
648          }
649          else if ((xk[i] - xkm1[i]) * (xkm1[i] - xkm2[i]) > 0)
650          {
651             gamma = 1.2;
652          }
653          else
654          {
655             gamma = 1.0;
656          }
657          Lk[i] = xk[i] - 0.5 * gamma;
658          Uk[i] = xk[i] + 0.5 * gamma;
659
660          ak[i] = 0.9 * Lk[i] + 0.1 * xk[i];
661          bk[i] = 0.9 * Uk[i] + 0.1 * xk[i];
673       }
674  }
675  else
676  {
677      for (i = 0; i <= OptiSolidNumb - 1; i++)
678      {
679          Lk[i] = xk[i] - 0.5;
680          Uk[i] = xk[i] + 0.5;
681          ak[i] = 0.9 * Lk[i] + 0.1 * xk[i];
682          bk[i] = 0.9 * Uk[i] + 0.1 * xk[i];
694      }
695  }
```

In the following lines of the method, the calculation formulas of Eqs. (24a) and (24b) for the calculation of quantities p and q are implemented. As it is explained in the theoretical part, the

derivatives of compliance can take negative values only; thus, lines #709 and #710 of condition `if` are selected, while in case of positive values an error message is returned (see line #704).

```
698  for (i = 0; i <= OptiSolidNumb - 1; i++)
699  {
700      if (derfil[i] > 0)
701      {
702          pk[i] = (Uk[i] - xk[i]) * (Uk[i] - xk[i]) * derfil[i];
703          qk[i] = 0;
704          Console.WriteLine("Error: positive derivative");
705          testbool = false;
706      }
707      else if (derfil[i] < 0)
708      {
709          pk[i] = 0;
710          qk[i] = -(xk[i] - Lk[i]) * (xk[i] - Lk[i]) * derfil[i];
711      }
712  }
```

Subsequently, the new density values are calculated according to the formula of Eq. (30), instead of that of OC algorithm that is provided by Eq. (19):

```
714   al = 0.0;
715   bl = Math.Pow(10, 10);
716   testlamda = false;
717   do
718   {
719       lamda = (al + bl) / 2;
720       metrx = 0;
721       for (i = 0; i <= OptiSolidNumb - 1; i++)
722       {
723a          xnew[i] = Math.Max(Math.Max(0, ak[i]), Math.Max(xk[i] - 0.2,
723b                    Math.Min(Math.Min(1, bk[i]), Math.Min(xk[i] + 0.2,
723c                   (Math.Sqrt(qk[i] / lamda) + Lk[i])))));
```

Then, filtering is implemented on the density values and the constraint of Eq. (31) is verified, similarly to the procedure implemented for OC algorithm.

4.11 TestProcedure method

In this method the convergence criterion is verified based on the difference of the elements' density values between two successive iterations. An instance of the `TestProcedure` method receives as input arguments the arrays of unfiltered density values of the previous and current iterations, denoted in the following code (lines #757 to #769) with variables `xk` and `xnew`. When the maximum difference over the solid finite elements, whose density value represent design variable, is less than 0.001 (see line #766), then variable `testpro` that is returned by the method becomes equal to true (see line #768) and the iterative process is terminated. Otherwise, `testpro` variable value remains equal to false and the process continues.

```
757  int i;
758  max = -1.0;
759  for (i = 0; i <= OptiSolidNumb - 1; i++)
760  {
761     if (Math.Abs(xk[i] - xnew[i]) > max)
```

```
762     {
763         max = Math.Abs(xk[i] - xnew[i]);
764     }
765 }
766 if (max < Math.Pow(10, -3) || (epanal > fixedepanal))
767 {
768     testpro = true;
769 }
```

4.12 DeleteSolid method

At the end of topology optimization procedure, an instance of `DeleteSolid` method is used in order to remove those solid finite elements having density values lower than the threshold value set by the user. In most of the implementations this value was set equal to 0.6. If the element's density value is greater than this threshold value 0.6 then the procedure described in lines #782 to #784 is performed where its solid section property is updated based on its density value, otherwise the element is deleted (see line #788). When this threshold checked is performed for all optimizable solid finite elements, the optimized layout can be seen through SAP2000 environment.

```
777 ret = Model.SetModelIsLocked(false);
778 for (i = 0; i <= OptiSolidNumb - 1; i++)
779 {
780     if (xkfil[i] > 0.6)
781     {
782         temp_double = Math.Round(xkfil[i] * xkfil[i] * xkfil[i], 2);
783         temp_string = "OptiSolidProp" + string.Format("{0:N2}", temp_double);
784         ret = Model.SolidObj.SetProperty(OptiSolidName[i], temp_string, 0);
785     }
786     else
787     {
788         ret = Model.SolidObj.Delete(OptiSolidName[i], 0);
789     }
790 }
```

5. TEST EXAMPLES

In order to present the capabilities and to provide more information about the use of TOCP source code several test examples are considered in this section. Initially a couple of simple test examples such as the cantilever and simply supported beams are examined. In the second group of tests indicative issues related to conceptual design are integrated into the optimum layout design of moment resisting frames (MRFs), that represent an important representative of the structural members in case of building structures. For all optimization runs that will follow the value of p used in Eq. (2) was set equal to 3 and density filtering was applied having filter size $r_{\min} = \sqrt{3}$.

5.1 Simple examples

In the collection of simple examples a cantilever beam and two simply supported ones (beam-A and -B) are studied. For all optimization runs the value of the volume fraction (*volfrac*) used to obtain the optimized domains was set equal to 30%.

The dimensionless lengths and height, of the cantilever beam of Figure 4(a) are 1.2 along the longitudinal axis *x*, 0.5 along the transverse axis *y* and 0.8 along the vertical axis *z*. For the finite element discretization of the cantilever beam shown in Figure 4(a), the following parameters were used: *nlx = 12*, *nly = 5* and *nlz = 8* resulting into 480 cubic solid elements where each edge of the basic cubic solid is equal to 0.1 dimensionless length size. All nodes on the right side of the cantilever beam were restrained, while a uniformly distributed load is applied along the upper part of the left side at the vertical direction (i.e. axis z). Both OC and MMA algorithms were used in order to solve the topology optimization problem converging to similar results, both methods converged based on the maximum number of iterations criterion that was set to 150 finite element analyses. The optimized domain obtained by OC is shown in Figure 4(b), a similar one was obtained by MMA as well.

For the simply supported beam-A of Figure 5(a) the corresponding dimensionless lengths and height are 3.0 along the longitudinal axis *x*, 0.8 along the transverse axis *y* and 1.0 along the vertical axis *z*. For the finite element discretization of the simply supported beam-A shown in Figure 5(a), the following parameters were used: *nlx = 30*, *nly = 8* and *nlz = 10* resulting into 2400 cubic solid elements where each edge of the basic cubic solid is equal to 0.1 dimensionless length size. The nodes of the two lower extreme edges were restrained, while a uniformly distributed load is applied along the centre of the beam at the vertical direction i.e. axis z (as shown in Figure 5(a), corresponding to loading case 1). Both OC and MMA algorithms were used in order to solve the topology optimization problem converging again to similar results; however, OC method converged earlier when the stopping criterion of the maximum difference of the density values between two successive iterations was satisfied at the 90^{th} iterations while MMA converged when the maximum number of iterations criterion was achieved that was set to 150 finite element analyses. The optimized domain obtained by OC is shown in Figure 5(b), a similar one was also obtained by MMA. For the simply supported beam-A a second loading case was also considered, where additional uniformly distributed loads are applied along the upper extreme edges of the beam at the vertical direction (i.e. axis z, see Figure 6(a)), the optimized domain obtained by OC for loading case 2 is shown in Figure 6(b), a similar one was obtained also by MMA algorithm.

The last simple example is the second simply supported beam-B, where the dimensionless lengths and height are equal to 1.2 along the longitudinal axis *x*, 0.5 along the transverse axis *y* and 0.8 along the vertical axis *z*. For the finite element discretization of the simply supported beam-B shown in Figure 7, the following parameters were used: *nlx = 24*, *nly = 10* and *nlz = 16* resulting into 3840 cubic solid elements where each edge of the basic cubic solid is equal to 0.05 dimensionless length size. The nodes of the lower extreme edges were restrained, while a uniformly distributed load is applied along the upper left edge at the horizontal direction of axis x. Similar to the previous examples both OC and MMA algorithms were used in order to solve the

topology optimization problem converging, however, to rather different results both after satisfying the maximum number of iterations criterion that was set to 150 finite element analyses. In particular, OC resulted into the optimized domain presented in Figure 8(a), while MMA algorithm's result is shown in Figure 8(b).

5.2 MRF Design Test Example

Moment-resisting frames are rectilinear assemblages of beams and columns, with the beams rigidly connected to the columns. Resistance to lateral loads is provided primarily by rigid frame action, i.e. by the development of bending moments and shear forces in the frame members and joints. By virtue of the rigid beam-column connections, a moment frame cannot displace laterally without bending the beams or columns depending on the geometry of the connection. The bending rigidity and strength of the frame members is therefore the primary source of lateral stiffness and strength for the entire frame. The 1994 Northridge earthquake revealed a common flaw in the construction, and building design codes were revised to strengthen them.

In the context of OPTARCH research project, among others, topology optimization problems are formulated for deriving multiple alternatives. In this direction, topology optimization is used as a tool to design aesthetically acceptable layouts of MRFs used in the design of high-rise buildings. In order to present the integration of topology optimization formulations in the conceptual design of civil structures, the MRF shown in Figure 9(a) is employed, where the domain and finite element mesh discretization is also depicted. The dimensionless lengths and height of the MRF are equal to 2.0 along the longitudinal axis *x*, 0.3 along the transverse axis *y* and 9.0 along the vertical axis *z*. For the optimization runs of MRF test example various values of the volume fraction were used to obtain variants of optimized domains. For the finite element discretization of the MRF shown in Figure 9(a), the following parameters were used in all cases examined, i.e. *nlx = 20*, *nly = 3* and *nlz = 90* resulting into *5400* cubic solid elements. Each edge of the basic cubic solid is equal to 0.1 dimensionless length size and the bottom extreme edges of the domain were fully fixed.

For the MRF test example two loading cases have been applied. Among others, these two loading cases included, several issues concerning the conceptual design of high-rise buildings have been examined by Stromberg *et al*. [18]. According to the first case horizontally loads were applied on the left side of MRF (at the level of the three storeys of the structural system, loading case-1) and in the second one on its both sides (loading case-2). For the first loading case, OC algorithm was adopted and the volume fraction used to obtain the optimized domain was set equal to 40%. The optimized domain obtained for loading case-1 by OC is shown in Figure 9(b). For loading case-2, shown in Figure 10(a), both search algorithms were implemented, and OC resulted in the optimized domain of Figure 10(b) where the volume fraction was set equal to 55% whereas

MMA algorithm resulted in the optimized domain of Figure 10(c) where the volume fraction was set equal to 47%. For all optimization runs performed for this test example the maximum number of iterations criterion was set equal to 400 finite element analyses.

6. CONCLUSIONS

In this work we present a code implementing the integration of a minimum compliance shape and topology optimization framework into the SAP2000 structural analysis and design software. The code is written in C# programming language and exploiting the open application programming interface provided by SAP2000, it provides the possibility to take advantage of all its modelling, analysis and design capabilities. The paper covers all theoretical aspects of topology optimization incorporated in the code and provides detailed description of their numerical implementation. For purposes of understanding several simple test examples are also provided in the current study.

In particular, four visual studio projects implementing variants of a simple minimum compliance based topology optimization frameworks, are available from the webpage http://users.ntua.gr/nlagaros/TOCP/projectsSMO.zip which can be used for educational purposes. The codes can easily be extended to include additional optimization criteria and also to handle problems requiring nonlinear and dynamic analyses. Using the four projects in C# are not independent, since the finite element analysis part required by topology optimization is performed by SAP2000; this is considered as the basic advantages of the TOCP codes provided herein. This is because, the analysis and design features that the commercial software provides will enhance significantly the capabilities of future researchers that will extend the TOCP codes' capacities. Furthermore, the user needs not to care for issues like sparsity in the assembly of the global stiffness matrix, acceleration of the finite element equations solution, modelling and analysis issues. Since they will benefit from the acceleration capability and memory management that an advanced commercial software has inherently implemented in its computational core. Worth mentioning that in case of academic purposes usage the specific software provides multiple licenses almost for free.

The authors would be happy to receive suggested improvements that can be implemented in the public domain TOCP codes (please address to the first author by his e-mail address nlagaros@central.ntua.gr).

ACKNOWLEDGEMENTS

This research has been supported by the OptArch project: "Optimization Driven Architectural Design of Structures" (No: 689983) belonging to the Marie Skłodowska-Curie Actions (MSCA) Research and Innovation Staff Exchange (RISE) H2020-MSCA-RISE-2015.

REFERENCES

[1] Gallagher R.H., Zienkiewicz O.C., *Optimum Structural Design: Theory and Applications*, John Wiley & Sons, New York, USA, 1973.

[2] Haug E.J., Arora J.S. Optimal mechanical design techniques based on optimal control methods, ASME paper No 64-DTT-10, *Proceedings of the 1st ASME design technology transfer conference*, New York, 65-74, October, 1974.

[3] Moses F., Mathematical programming methods for structural optimization, *ASME Structural Optimisation Symposium AMD*; 7, 35-48, 1974.

[4] Sheu C.Y., Prager W. Recent development in optimal structural design, *Applied Mechanical Reviews*; 21(10): 985-992, 1968.

[5] Spunt L., *Optimum Structural Design*, Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1971.

[6] Bendsøe, M.P., Sigmund, O. Material interpolations in topology optimization. *Archive of Applied Mechanics*; 69: 635-654, 1999.

[7] Bendsøe, M.P., Sigmund, O. *Topology Optimization-Theory, Methods and Applications*. Springer Verlag, Berlin Heidelberg, 2003.

[8] Allaire, G. *Shape Optimization by the Homogenization Method*, Applied Mathematical Sciences, Volume 146. Springer, New York, USA, 2002.

[9] Allaire, G., Jouve, F., Toader, A.M. Structural optimization using sensitivity analysis and a level-set method. *Journal of Computational Physics*; 194(1):363-393, 2004.

[10] Sigmund, O. A 99 line topology optimization code written in Matlab, *Structural and Multidisciplinary Optimization*; 21(2): 120-127, 2001.

[11] Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B.S., Sigmund, O. Efficient topology optimization in Matlab using 88 lines of code, *Structural and Multidisciplinary Optimization*; 43(1): 1-16, 2011.

[12] Allaire, G. codes: http://www.cmap.polytechnique.fr/~allaire/levelset_en.html.

[13] Talischi, C., Paulino, G.H., Pereira, A., Menezes, I.F.M. Polytop: a matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Structural and Multidisciplinary Optimization*; 45(3): 329-357, 2012.

[14] Liu, K., Tovar, A. An efficient 3D topology optimization code written in Matlab, *Structural and Multidisciplinary Optimization*; 50(6): 1175-1196, 2014.

[15] Zuo, Z.H., Xie, Y.M. A simple and compact Python code for complex 3D topology optimization, *Advances in Engineering Software*; 85, 1-11, 2015.

[16] Wilson, E.L., Habibullah, A. *SAP 2000 software, version 19*. Computer and Structures, Inc. (CSI), Berkeley, CA, USA, 2017.

[17] Svanberg, K. The method of moving asymptotes-a new method for structural optimization, *International Journal for Numerical Methods in Engineering*; 24 (2): 359-373, 1987.

[18] Stromberg L.L., Beghini A., Baker W.F., Paulino G.H. Topology optimization for braced frames: Combining continuum and beam/column elements. *Engineering Structures*; 37:106-124, 2012.

FIGURES

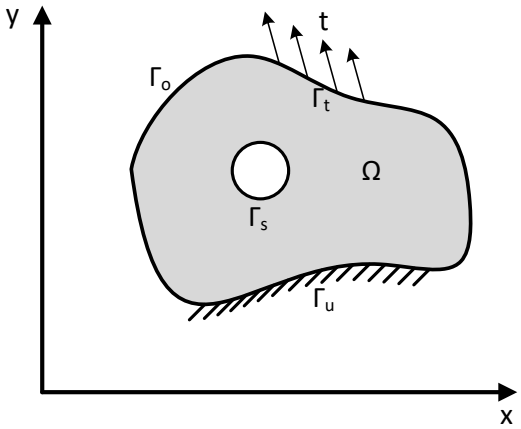


Figure 1. The generalized shape design problem of finding the optimal material distribution in 2D domain.

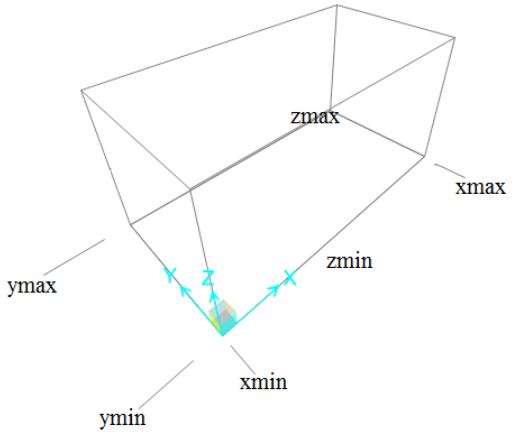
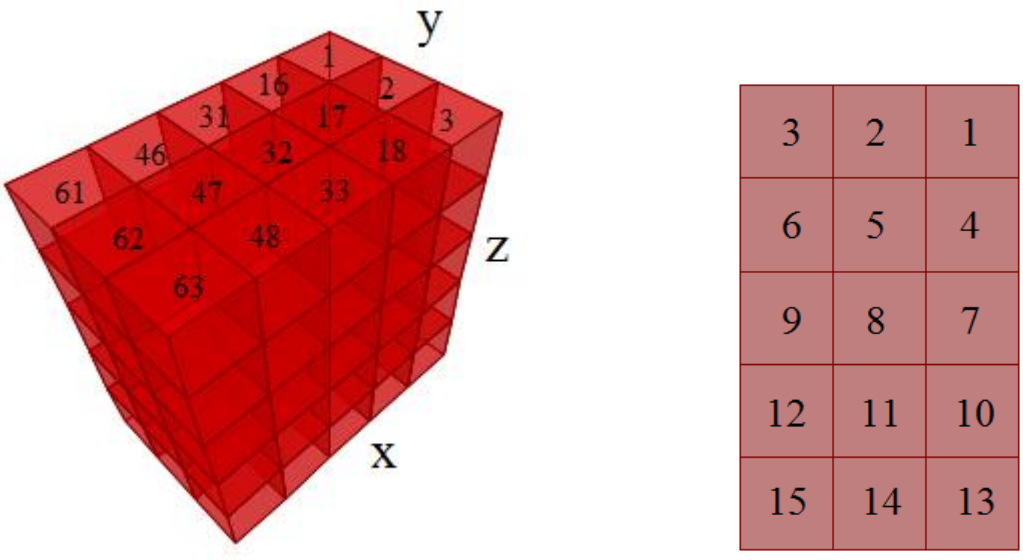
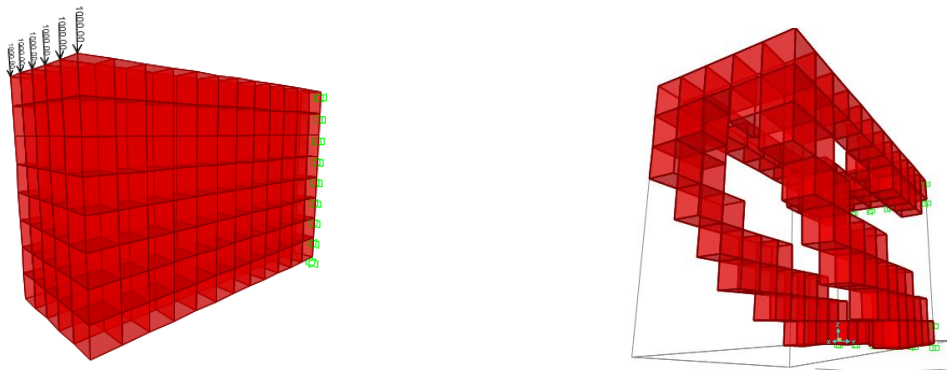


Figure 2. Optimizable domain.



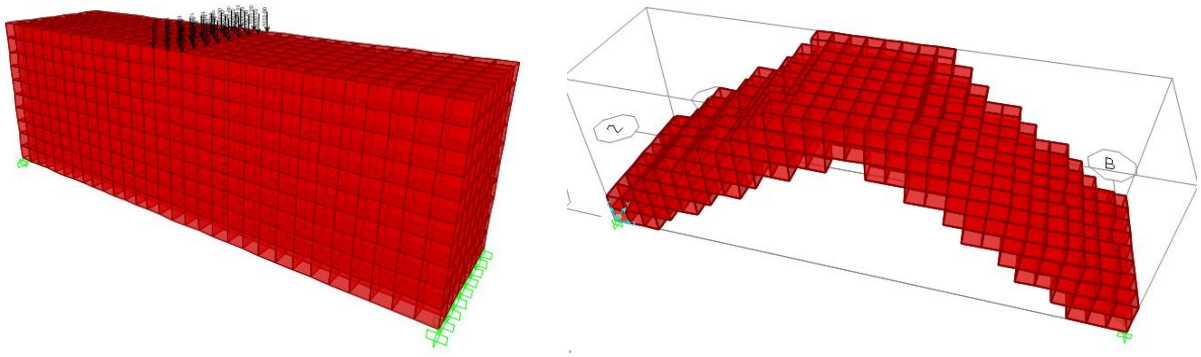
(a) (b)

Figure 3. Arrangement of the optimizable solid finite elements.



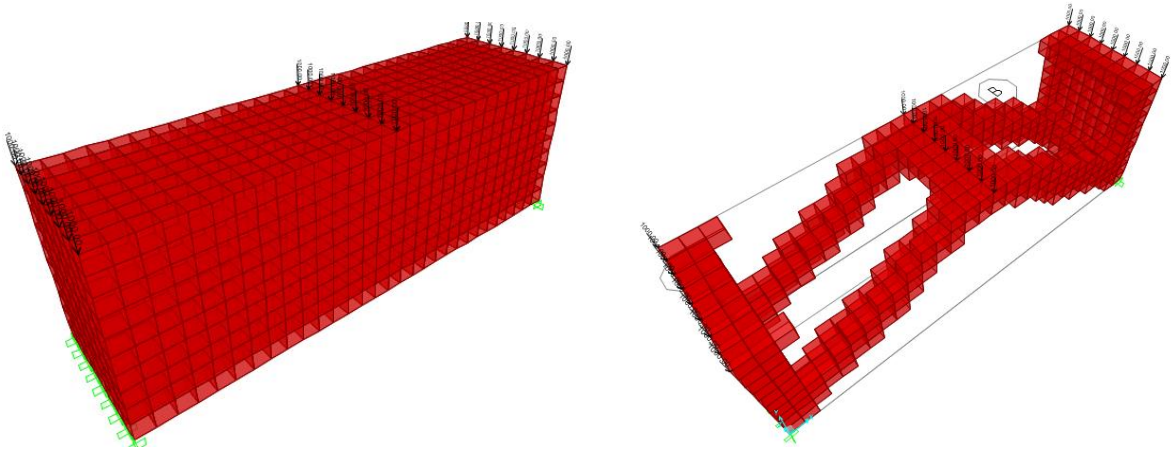
(a) (b)

Figure 4. Simple test examples – cantilever beam: (a) domain - mesh discretization and (b) optimized layout obtained by OC (volume fraction equal to 30%).



(a) (b)

Figure 5. Simple test examples – simply supported beam A (loading case 1): (a) domain - mesh discretization and (b) optimized layout obtained by OC (volume fraction equal to 30%).



(a) (b)

Figure 6. Simple test examples – simply supported beam-A (loading case 2): (a) domain - mesh discretization and (b) optimized layout obtained by OC (volume fraction equal to 30%).

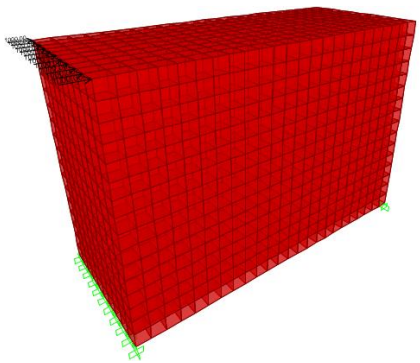
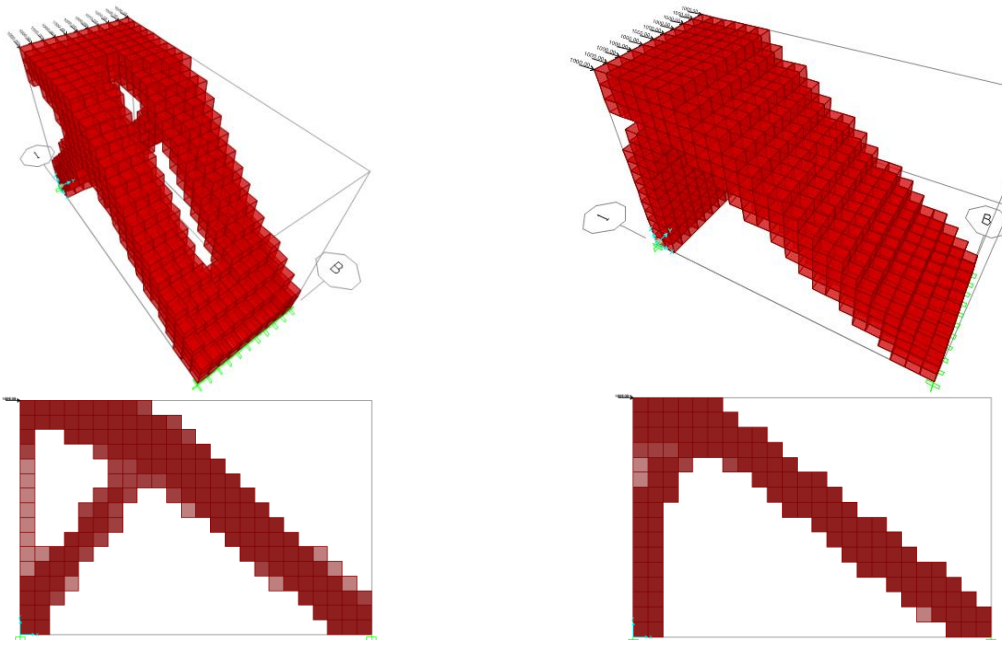
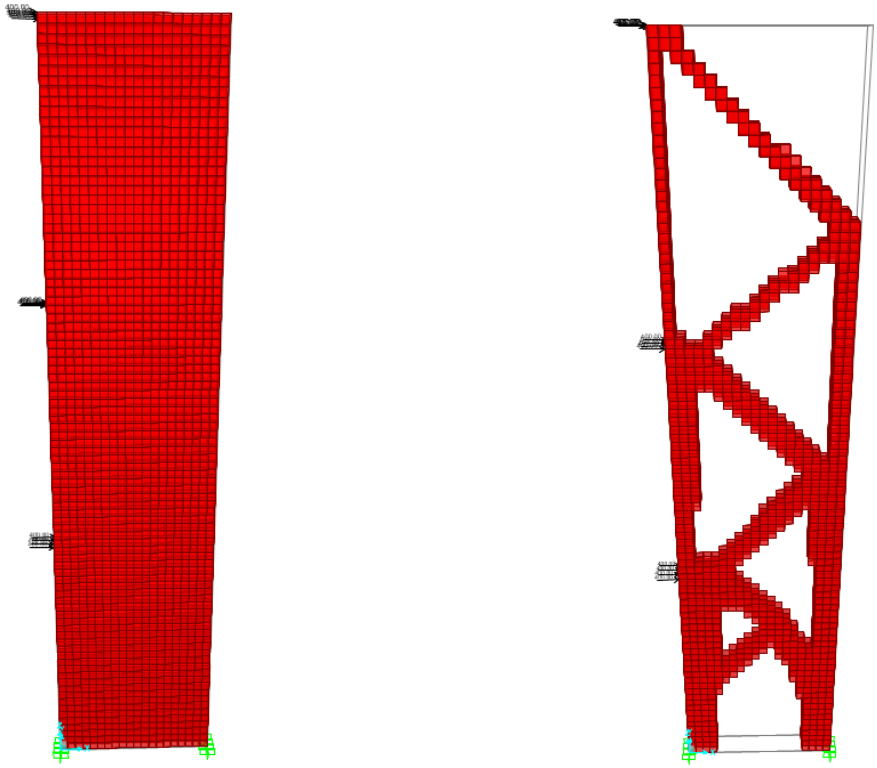


Figure 7. Simple test examples – simply supported beam-B: domain and mesh discretization.



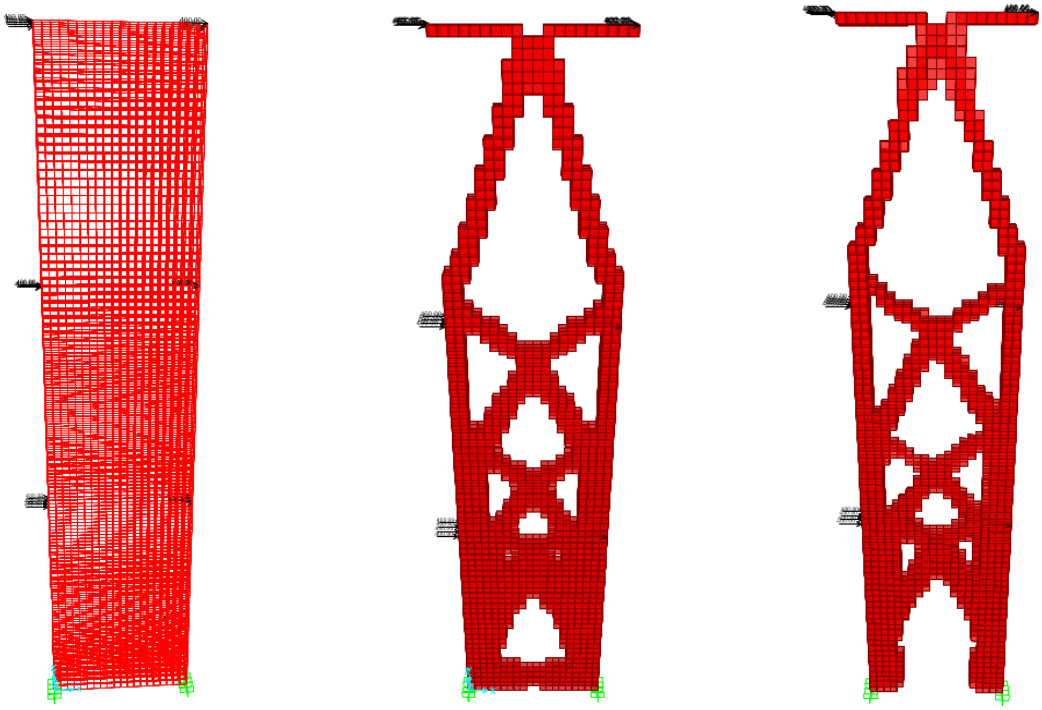
(a) (b)

Figure 8. Simple test examples – simply supported beam-B, 3D and side views of the optimized domain obtained by (volume fraction equal to 30%): (a) OC and (b) MMA.



(a) (b)

Figure 9. MRF test example: (a) domain - mesh discretization and (b) optimized layout obtained by OC (volume fraction equal to 40%).



(a) (b) (c)

Figure 10. MRF test example: (a) domain - mesh discretization and (b) optimized layout obtained by OC (volume fraction equal to 55%) and (c) optimized layout obtained by MMA (volume fraction equal to 47%).