Struct Multidisc Optim (2014) 50:861–882
DOI 10.1007/s00158-014-1085-z

EDUCATIONAL ARTICLE

GRAND — Ground structure based topology optimization for arbitrary 2D domains using MATLAB

Tomás Zegard · Glaucio H. Paulino

Received: 9 July 2013 / Revised: 5 March 2014 / Accepted: 17 March 2014 / Published online: 22 June 2014

Abstract The present work describes in detail an implementation of the ground structure method for non–orthogonal unstructured and concave domains written in MATLAB, called GRAND — *GRound structure ANalysis and Design*. The actual computational implementation is provided, and example problems are given for educational and testing purposes. The problem of ground structure generation is translated into a linear algebra approach, *which is inspired by the video–game literature*. To prevent the ground structure generation algorithm from creating members within geometric entities that no member should intersect (e.g. holes, passive regions), the concept of "restriction zones" is employed, *which is based on collision detection algorithms used in computational geometry and video–games*. The aim of the work is to provide an easy–to–use implementation for the optimization of least–weight trusses embedded in any domain geometry.

Keywords Ground structure method · Topology optimization of trusses · Optimal structures · Unstructured meshes · Linear programming

Electronic supplementary material The online version of this article (doi:10.1007/s00158-014-1085-z) contains supplementary material, which is available to authorized users.

T. Zegard · G. H. Paulino (✉)
Department of Civil and Environmental Engineering,
Newmark Laboratory, University of Illinois
at Urbana–Champaign, 205 N. Mathews Avenue,
Urbana, IL 61801, USA
e-mail: paulino@illinois.edu

1 Introduction

The ground structure method (Dorn et al. 1964) provides an approximation to an optimal Michell structure (Hemp 1973) composed of an infinite number of members, by using a reduced finite number of truss members. The optimal (least–weight) truss for a single load case, under elastic and linear conditions, subjected to stress constraints can be posed as a linear programming problem (Ohsaki 2010). The method removes unnecessary members from a highly interconnected truss (*ground structure*) while keeping the nodal locations fixed. (Hegemier and Prager 1969) showed that a truss with maximum stiffness is also fully stressed. In addition, the problem of a single load case considering equal stress limits in compression and tension, is equivalent to the minimization of compliance for a prescribed volume (Bendsøe and Sigmund 2003).

The analytical solution must satisfy some known conditions for structural optimization problems (Michell 1904; Hencky 1923). However, these conditions themselves do not provide means for obtaining the optimal analytical solution. Given a *candidate optimal structure*, these requirements can be used to check if the structure is indeed optimal or not.

The ground structure method has been refined, simplified and optimized, resulting in an easy–to–use implementation for truss topology optimization in structured orthogonal domains (Sokół 2011). The method has also been extended to support unstructured meshes (Smith 1998), where the initialization of the method (generation of the ground structure) is intricate. Recently, exact solutions for complicated domains have been numerically approximated and obtained (Lewiński et al. 2013), and there is ongoing work to extend the library of known analytical solutions for complicated domains.

The interest in unstructured non–orthogonal domains is reasonable because applied engineering problems are often not composed of *boxes*. The present work extends the ground structure method with a simple, flexible and effective methodology to generate the ground structures in non–orthogonal unstructured and concave domains. However, the method is restricted to piecewise polygonal boundaries (convex, concave and with the possibility of holes).

In fact, our goal is to move away from traditional "box–like" shapes by exploring innovative bio–inspired and natural designs using the ground structure approach. In this context, Fig. 1 illustrates our motivation: Fig. 1a shows the boundary value problem consisting of a circular domain, with a hole (zero prescribed displacements) and tangential loads applied on the outer boundary. Figure 1b is the base–mesh from which the ground structure is generated, leading to the design of Fig. 1c. We compare this pattern with that of the flower shown in Fig. 1d, not just in terms of appearance but also functionality.

The computer implementation, named GRAND, aims to be a balance between performance and legible code (educational). The objective is to provide future researchers in structural optimization with a ground structure implementation that serves as a starting point for future developments. Limitations and assumptions of the present implementation are:

– single static load case scenario
– constant forces (design independent)
– small deformations
– two–dimensional (2D) problems

It can, however, address different limits in tension σ_T and compression σ_C (Sokół 2011).

Throughout this paper, the terms *truss member* and *bar* are used interchangeably. The manuscript is organized as follows: Section 2 describes the formulation used. Section 3 explains details of the implementation. Selected examples and their convergence are analyzed in Section 4. Finally, conclusions and findings are summarized in Section 5. Appendix A has the nomenclature and symbols used in the manuscript (and in the MATLAB implementation). Appendix B includes notes on the usage and the library of examples provided in GRAND. The MATLAB implementation for GRAND is provided in Appendix C.

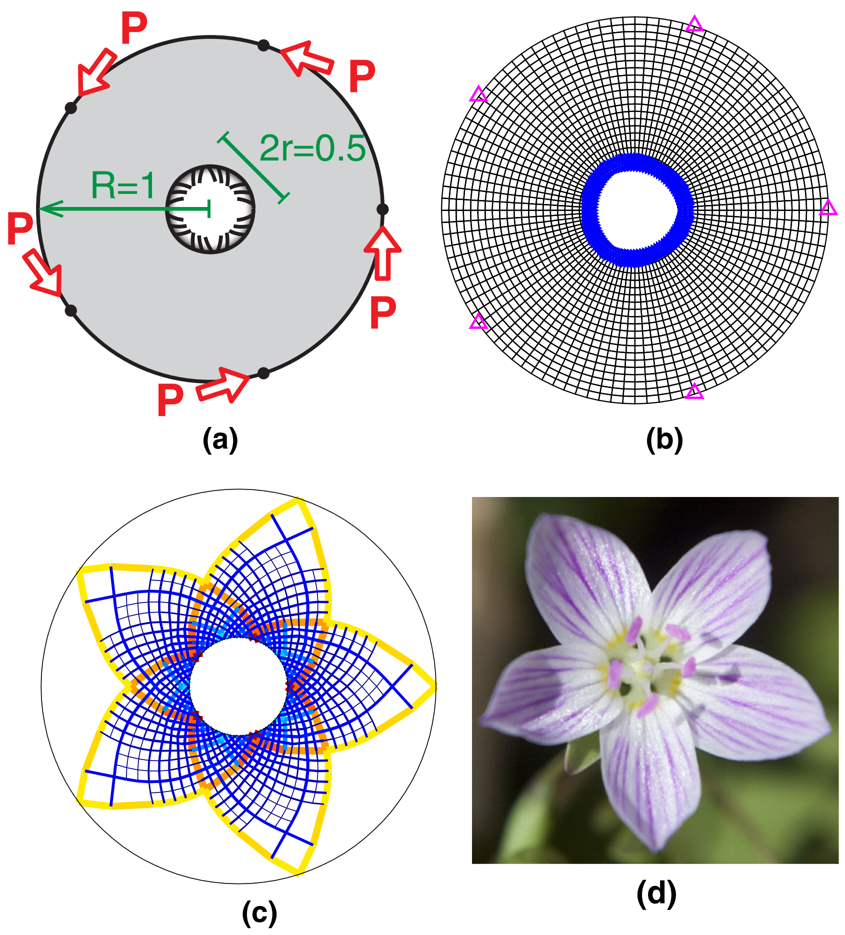


Fig. 1 Tangentially loaded donut–shaped domain. **a** Domain, loading and boundary conditions. **b** Base–mesh (discretization) used to construct the ground structure. **c** Optimized ground structure. **d** Photo of *Claytonia caroliniana* [© Nathan Masters — Masters Imaging]

Finally, Appendix D provides examples of the modifications required in the main script to analyze different problems. We note that the present manuscript doubles as the documentation for the computational implementation. In this context, special care has been taken to maintain the naming convention consistent between both the manuscript and the computational implementation.

2 Formulations

Michell (1904) derived the conditions necessary for a minimum volume truss subjected to stress constraints (Ohsaki 2010; Hemp 1973): given stress limits in tension $\sigma_T > 0$ and compression $\sigma_C > 0$, and the average limit stress $\sigma_0 = (\sigma_T + \sigma_C)/2$, the truss is optimal if:

1. The truss is in equilibrium
2. The stress is equal to either σ_T or σ_C for all members
3. There exists a compatible deformation field such that the strains are equal to $\varepsilon_t = \sigma_0\varepsilon_0/\sigma_T$ and $\varepsilon_c = \sigma_0\varepsilon_0/\sigma_C$ for members in tension and compression, respectively

As a consequence, the members in the resulting structure are arranged in the directions of the principal strains for the displacement field. Unfortunately, Michell's solutions encompass infinitely dense members. Nonetheless, a reasonable approximation to this solution can be obtained using a (finite) large number of members within the prescribed domain.

The problem formulation used in this work is based on *plastic analysis*: no stiffness matrices, compatibility equations or stress–strain relations are used (Hemp 1973). However, for the sake of completeness and clarity of presentation, the *elastic* and some issues present in this method will also be discussed (Christensen and Klarbring 2009).

2.1 Elastic formulation

Consider a rigid truss (no mechanisms), with N_{dof} nodal forces $\mathbf{f}$ (excluding the components with supports), and assume that the supports are sufficient to prevent the structure from having rigid body motions. The basic formulation for the minimum volume truss is then (Ohsaki 2010):

$$\begin{aligned} \min_{\mathbf{a}} \quad & V = \mathbf{l}^T\mathbf{a} \\ \text{s.t.} \quad & \mathbf{K}\mathbf{u} = \mathbf{f} \\ & -\sigma_C \le \sigma_i \le \sigma_T \ \text{ if } a_i > 0 \\ & a_i \ge 0 \quad i = 1, 2 \ldots N_b \, , \end{aligned} \tag{1}$$

with a_i, l_i and σ_i the cross–sectional area, length and stress of the ith member (for all N_b members). The parameters N_n and N_{sup} are the number of nodes and components with supports, respectively, and $N_{dof} = 2N_n - N_{sup}$ for a two–dimensional ground structure. Here, $\mathbf{K}$ denotes the global stiffness matrix and $\mathbf{u}$ denotes the nodal displacements associated with the N_{dof} free nodal components. Theoretically, a member is absent (removed) from the truss if $a_i = 0$. This issue has received significant attention in the literature and is further discussed in the next paragraph. The redundancy of the ground structure is $N_b - N_{dof}$ and should be greater than zero to provide optional layouts.

This formulation considers the equilibrium and compatibility conditions, and is thus an *elastic analysis* formulation (Hemp 1973; Kirsch 1993). The stress constraint may be violated if its corresponding member is absent, i.e. $a_i = 0$ (Sved and Ginos 1968). This phenomena of is known as *vanishing constraints* or *design–dependent constraints*. For the case of multiple loads, the optimal solution may become a *singular topology*, and thus obtaining the global optimum becomes quite challenging (Rozvany 2001). Fortunately, the single load case does not suffer from this problem.

2.2 Plastic formulation

Compared to (1), a formulation based on *plastic analysis* enforces equilibrium and no explicit compatibility or stress–strain relations (Kirsch 1993):

$$\begin{aligned} \min_{\mathbf{a}} \quad & V = \mathbf{l}^T\mathbf{a} \\ \text{s.t.} \quad & \mathbf{B}^{\mathbf{T}}\mathbf{n} = \mathbf{f} \\ & -\sigma_C a_i \le n_i \le \sigma_T a_i \quad i = 1, 2 \ldots N_b \, , \end{aligned} \tag{2}$$

where $\mathbf{B}^{\mathbf{T}}$ is the nodal equilibrium matrix of size $N_{dof} \times N_b$, built from the directional cosines of the members, and $\mathbf{n}$ is a vector with the internal (axial) force for all members in the ground structure. *The stress constraint (in tension or compression) must be active for all members at the optimum.* An intuitive proof is that if the ith member has $n_i < \sigma_T a_i$ and $n_i > -\sigma_C a_i$, then a_i can be reduced (reducing the total volume) without violating the constraints. The stress constraint is expressed in terms of member force, thus simplifying its treatment. Incorporating *slack variables* in the stress constraints (Hemp 1973; Achtziger 2007), one converts the inequalities into equalities:

$$\begin{aligned} n_i + 2\frac{\sigma_0}{\sigma_C}s_i^- &= \sigma_T a_i \\ -n_i + 2\frac{\sigma_0}{\sigma_T}s_i^+ &= \sigma_C a_i \, , \end{aligned} \tag{3}$$

where the (positive) coefficients in the slack variables simplify the resulting expressions for cross–sectional area and axial force:

$$\begin{aligned} a_i &= \frac{s_i^+}{\sigma_T} + \frac{s_i^-}{\sigma_C} \\ n_i &= s_i^+ - s_i^- \end{aligned} \tag{4}$$

The optimization problem in (2) becomes then a linear programming problem as:

$$\begin{aligned} \min_{\mathbf{s}^+,\mathbf{s}^-} \quad & V = \mathbf{l}^T \left(\frac{\mathbf{s}^+}{\sigma_T} + \frac{\mathbf{s}^-}{\sigma_C} \right) \\ \text{s.t.} \quad & \mathbf{B}^\mathbf{T} \left(\mathbf{s}^+ - \mathbf{s}^- \right) = \mathbf{f} \\ & s_i^+, s_i^- \geq 0 \end{aligned} \tag{5}$$

Note that for any active member, only one of s_i^+ and s_i^- is non–zero. The member is in tension if $s_i^+ > 0$, and in compression if $s_i^- > 0$. If the truss structure is stable, has no repeated and no overlapping members, then the rank of matrix $\mathbf{B}^\mathbf{T}$ is N_{dof} (i.e. the solution does not lie on the edge of the feasible domain). The solution of the linear programming problem (5) yields at most N_{dof} non–zero *basic variables*, with the remainder *non–basic variables* being absent from the optimal structure (i.e. $a_i = 0$). Therefore, the optimal truss is statically determinate and the solution is also globally optimal (Sved 1954; Kicher 1966). The elastic design (1) has to satisfy additional compatibility conditions, and thus a higher optimal volume is expected compared to the plastic design (2). But because the optimal structure for a single load case is statically determinate, then the *plastically admissible structure*, based on force equilibrium, also satisfies the kinematic compatibility and stress–strain relation, and is thus also an *elastically admissible structure*: the optimal solution for both methods are equal for the given assumptions (Dorn et al. 1964; Hemp 1973).

If the ratio in the stress limits is defined as $\kappa = \sigma_T / \sigma_C$, then the formulation becomes:

$$\begin{aligned} \min_{\mathbf{s}^+,\mathbf{s}^-} \quad & V^\star = \frac{V}{\sigma_T} = \mathbf{l}^T \left(\mathbf{s}^+ + \kappa \mathbf{s}^- \right) \\ \text{s.t.} \quad & \mathbf{B}^\mathbf{T} \left(\mathbf{s}^+ - \mathbf{s}^- \right) = \mathbf{f} \\ & s_i^+, s_i^- \geq 0 \end{aligned} \tag{6}$$

This final form of the *plastic layout optimization* problem is utilized in this work (Sokół 2011; Achtziger 2007; Gilbert and Tyas 2003), and can be efficiently solved using the interior–point algorithm (Karmarkar 1984; Wright 2005). The optimal volume $V^\star$ is calculated for $\sigma_T = 1$, and should be scaled by $1/\sigma_T$ for values other than unity.

The resulting optimal structure will be in equilibrium. However, the equilibrium may be unstable due to the existence of members with zero cross–sectional areas belonging to the *basic variables* (degenerate LP problem). As recommended by (Dorn et al. 1964), the structure should be post–processed to become a *reduced optimal structure* (ROS) with:

- Nodes connecting two collinear members are all replaced with a single long member (collinear hinges).
- Nodes where all members have $a_i = 0$ are removed, i.e. nodes not participating in the resulting structural configuration are removed.
- Members associated with basic variables (LP) having zero cross–sectional area should have a minimum value $a_{min} > 0$ to account for imperfections and small variations in the geometry and loads.

The resulting (stable) structure maintains all the properties of the original one: equilibrium, statically determinate and fully stressed. This post–processing however, is beyond the educational scope of this manuscript.

3 Implementation

3.1 Domain definition — Base mesh

To define the domain and boundaries, four variables must be specified. These in turn will be used to generate the ground structure, and are described in detail in Table 1. The number of nodes, elements, nodes with prescribed boundary conditions and nodes with prescribed loads are N_n, N_e, N_f and N_l respectively. It should be noted that GRAND makes no assumption on the type of elements (lines, triangles, quads, polygonal or combinations of these), nor on the element numbering. This information is only used to define the nodal connectivity, the domain's extension and boundaries. GRAND has three options available for importing or generating the base mesh: loading an external mesh following the guidelines from Table 1, generating the mesh with the bundled polygonal mesher called PolyMesher (Talischi et al. 2012a), and generating an orthogonal structured domain using a subroutine provided with GRAND.

3.2 Ground structure generation

The generation of the ground structure is the main contribution in GRAND. The ground structure should have no

Table 1 Domain definition (base mesh) input variables

Variable Name	Type & Size	Description
`NODE`	`array` $N_n \times 2$	Each row p has the nodal coordinates x and y for node p.
`ELEM`	`cell` $N_e \times 1$	Every element in the list is a row vector containing the node numbers for a particular element.
`SUPP`	`array` $N_f \times 3$	Each row consists of a node number, fixity x and fixity in y. Any value other than `NaN` specifies fixity. The total number of specified fixities is N_{sup}.
`LOAD`	`array` $N_l \times 3$	Each row consists of a node number, load in x and load in y. A zero or `NaN` specify no force in that direction.

overlapping truss members (members connecting collinear nodes), and a single member connecting the same nodes p and q (Fig. 2). The starting point of this process is the base mesh (defined as in Table 1). In addition, a connectivity level Lvl and a collinearity tolerance $ColTol$ need to be specified. To efficiently generate the ground structure on modern computer architectures, the problem of generating the ground structure should be translated to linear algebra and matrix operations when possible, thus taking advantage of the optimizations in numerical computing frameworks (Heath 1998; Olson 2013).

The user defined *connectivity level* determines the level of redundancy, or inter–connectedness, of the initial ground structure. If the connectivity level Lvl is sufficiently high, the ground structure generation algorithm will interconnect all nodes; this is often referred to as a *full level ground structure*. The analytical solution (Michell 1904), is typically composed of curved members. The ground structure method will have a tendency to retain short members, so as to more accurately try to represent these curved members in a piecewise fashion. Thus, the full level ground structure, containing long (edge–to–edge) candidate members, is not the best from a cost–effective point of view. If two nodes belong to the same element in the base mesh (Fig. 3a), then they are considered *neighbors*. From this idea of neighbors, the connectivity level can be explained as follows:

- Level 1 connectivity will generate members between all neighboring nodes (Fig. 3b).
- Level 2 connectivity will generate members up to the neighbors of the neighbors (Fig. 3c).
- Level 3 connectivity will generate members up to the neighbors of the neighbors of the neighbors (Fig. 3d).

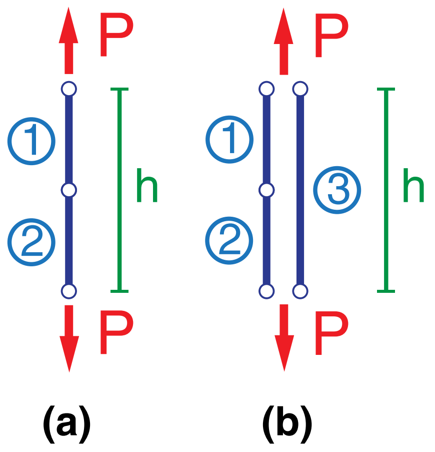


Fig. 2 Overlapping members example assuming $P = 1$, $h = 1$ and $\sigma_T = 1$. **a** Problem with a unique solution: optimal volume is $V = 1$ and $a_1 = a_2 = 1$. **b** Problem with a non–unique solution: optimal volume is $V = 1$, but $a_1 = a_2 = [0, 1]$ and $a_3 = 1 - a_1$

This process scales quickly as illustrated in Fig. 3, and only decelerates when the member generation is reaching the full level connectivity. The example in Fig. 3 achieves a *full level ground structure* at level 5, and the difference between levels 4 and 5 is minimal. Note that no members are generated in the domain's concave region; this desirable feature will be discussed in detail in the following sections.

The nodal connectivity matrix (symmetric i.e., bi-directional) for the base mesh is named $\mathbf{A}_1$, and is defined as follows:

$$[\mathbf{A}_1]_{p,q} = \begin{cases} 1 \text{ or } \texttt{true} & \text{if nodes } p,q \text{ share an element} \\ 0 \text{ or } \texttt{false} & \text{otherwise or if } p = q \end{cases} \tag{7}$$

The second level connectivity is simply $\mathbf{A}_2 = \mathbf{A}_1\mathbf{A}_1$. However, it should be noted that this matrix is likely to have entries > 1 and a non–zero diagonal (see the example in Fig. 4). Thus, the diagonal is set to zero and the matrix is again converted to *logical*. The nodal connectivity matrix for some level $n > 1$ is then:

$$[\mathbf{A}_n]_{p,q} = \begin{cases} 0 \text{ or } \texttt{false} & \text{if } p = q \\ 1 \text{ or } \texttt{true} & \text{if } \left[\mathbf{A}_1^n\right]_{p,q} > 0 \\ 0 \text{ or } \texttt{false} & \text{otherwise} \end{cases} \tag{8}$$

3.3 Collinearity check

The following assumptions are made in the collinearity check:

1. New bars added at level n are deleted if found collinear with bars from previous levels.
2. New bars added at level n are assumed not to be collinear between them.

The first assumption is logical if we consider that new bars are *longer* than those from previous levels. The second assumption may be violated if elements are not strictly convex, or in domain shapes that curl; where a new level may have two collinear nodes viewed from the starting node. The new (candidate) bars for level n can be obtained as:

$$\mathbf{G}_n = \mathbf{A}_n - \mathbf{A}_{n-1}, \tag{9}$$

with $\mathbf{G}_1 = \mathbf{A}_1$. The non–zero entries in $\mathbf{G}_n$ that will be included in the ground structure are those that have no angle close to zero with previously accepted bars. The directional cosines vector $\hat{\mathbf{d}}_{p,q}$ from node p to node q is:

$$\begin{aligned} \mathbf{d}_{p,q} &= \mathtt{NODE}_{q,:} - \mathtt{NODE}_{p,:} \\ \hat{\mathbf{d}}_{p,q} &= \frac{\mathbf{d}_{p,q}}{\|\mathbf{d}_{p,q}\|}, \end{aligned} \tag{10}$$

where $\mathtt{NODE}_{i,:}$ stands for row i of the nodal coordinates array (i.e. the coordinates x and y of node i), as defined in

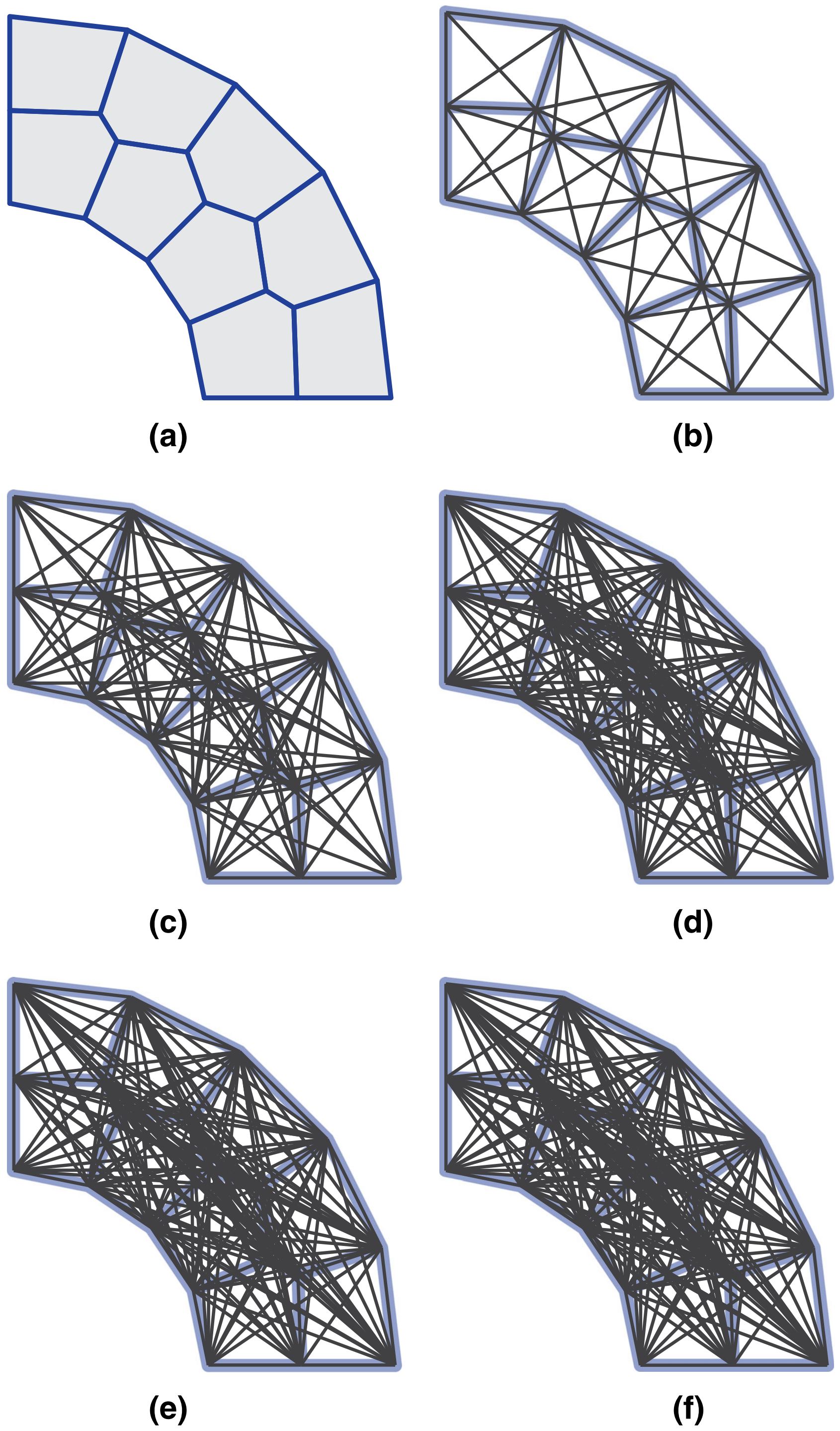


Fig. 3 Ground structure connectivity level generation example. **a** Base mesh composed of 9 polygonal elements. **b** Level 1 connectivity. **c** Level 2 connectivity. **d** Level 3 connectivity. **e** Level 4 connectivity. **f** Level 5 connectivity

Table 1. The angle between two directional cosines vectors is:

$$\cos(\angle qpr) = \hat{\mathbf{d}}_{p,q} \cdot \hat{\mathbf{d}}_{p,r} \tag{11}$$

Assume that m new candidate bars originating from a specific node i, have to be checked against previously accepted n bars from the same node. The directional vectors of the new (candidate) bars can be grouped into $\mathbf{D}_{new}$ of size $m \times 2$, and the previously accepted bars into $\mathbf{D}_{old}$ of size $n \times 2$. The dot product of new (candidate) and old bars can be (efficiently) computed by:

$$\mathbf{C} = \mathbf{D}_{old}\mathbf{D}^{\mathbf{T}}{}_{new}, \tag{12}$$

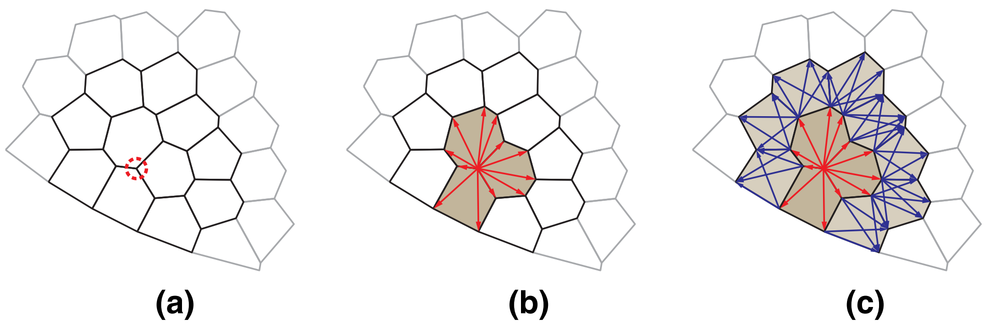


Fig. 4 Connectivity matrix calculation. **a** Base mesh and starting node. **b** Level 1 connectivity obtained from $\mathbf{A}_1$. **c** Level 2 connectivity. Note that the entries of $\mathbf{A}_2 = (\mathbf{A}_1)^2$ are typically > 1 due to the existence of more than one path to the new set of nodes

where a new (candidate) bar j is found to be collinear if any entry in column j is equal to 1. In reality, a collinearity tolerance $ColTol < 1$ is used instead, and a bar j is removed if:

$$\mathbf{C}_{i,j} > ColTol \qquad \forall\, i = 1 \ldots n \tag{13}$$

The bars that pass the collinearity test are then appended to the ground structure.

The currently accepted bars at a level are stored in matrix $\mathbf{H}$, with:

$$\mathbf{H}_{p,q} = \begin{cases} 1 \text{ or } \texttt{true} & \text{if there } \exists \text{ member } \overline{pq} \\ 0 \text{ or } \texttt{false} \text{ otherwise} \end{cases} \tag{14}$$

Matrix $\mathbf{H}$ loses symmetry when a bar has one angle below $ColTol$ and one above; in the example of Fig. 5 this means $\cos(\beta_2) < ColTol < \cos(\beta_1)$, and thus $\mathbf{H}_{p,q} = 0$ but $\mathbf{H}_{q,p} = 1$. If the member has one angle that passes the $ColTol$ requirement, it will be spared from deletion by forcing matrix $\mathbf{H}$ to be symmetric again:

$$\mathbf{H}^{\star} = \mathbf{H} + \mathbf{H}^{\mathbf{T}}, \tag{15}$$

after each level calculation. The entries $\mathbf{H}^{\star} > 0$ contain the bars of the *current* ground structure to be used in the next level iteration if any.

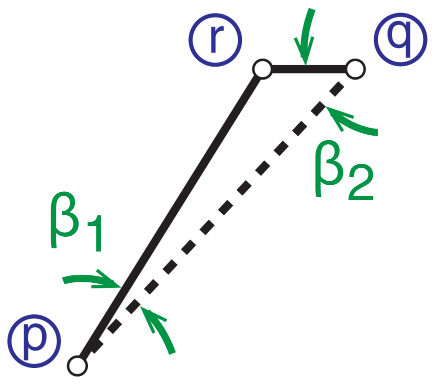


Fig. 5 Collinearity test between three bars. The long bar (*dashed line*) between nodes p and q is candidate for deletion

Once the ground structure has been generated, only bars linking nodes p and q with $p < q$ are returned (i.e., the upper triangular form of $\mathbf{H}^{\star}$). This prevents duplicate bars from appearing in the ground structure.

The sample base mesh in Fig. 6a results in the matrices $\mathbf{A}_1 = \mathbf{G}_1$ and $\mathbf{G}_2$ detailed in (16).

$$\mathbf{A}_1 = \mathbf{G}_1 = \begin{bmatrix} 0&1&0&0&1&1&0&0&0&0&0&0 \\ &0&1&0&1&1&1&1&0&0&0&0 \\ &&0&1&0&1&1&1&1&0&0&0 \\ &&&0&0&0&1&0&1&0&0&0 \\ &&&&0&1&0&0&0&1&1&0 \\ &&&&&0&1&1&0&1&1&0 \\ &&&&&&0&1&1&0&0&1 \\ &&&&&&&0&1&0&1&1 \\ &&&&&&&&0&0&0&1 \\ &\text{symm}&&&&&&&&0&1&0 \\ &&&&&&&&&&0&1 \\ &&&&&&&&&&&0 \end{bmatrix}$$

$$\mathbf{G}_2 = \begin{bmatrix} 0&0&1&0&0&0&1&1&0&1&1&0 \\ &0&0&1&0&0&0&0&1&1&1&1 \\ &&0&0&1&0&0&0&0&1&1&1 \\ &&&0&0&1&0&1&0&0&0&1 \\ &&&&0&0&1&1&0&0&0&1 \\ &&&&&0&0&0&1&0&0&1 \\ &&&&&&0&0&0&1&1&0 \\ &&&&&&&0&0&1&0&0 \\ &&&&&&&&0&0&1&0 \\ &\text{symm}&&&&&&&&0&0&1 \\ &&&&&&&&&&0&0 \\ &&&&&&&&&&&0 \end{bmatrix} \tag{16}$$

The members from level 1 are always included in the ground structure (no additional checking), resulting in the ground structure in Fig. 6b. The candidate bars at level 2 are tested against the previously accepted bars at level 1, and thus not all members in $\mathbf{G}_2$ will be retained. The accepted bars for levels 1 and 2 are detailed in matrix $\mathbf{H}^{\star}$ as follows:

$$\mathbf{H}^{\star} = \begin{bmatrix} 0&1&0&0&1&1&1&0&0&0&1&0 \\ &0&1&0&1&1&1&1&0&1&1&1 \\ &&0&1&1&1&1&1&1&1&1&1 \\ &&&0&0&1&1&0&1&0&0&0 \\ &&&&0&1&0&1&0&1&1&1 \\ &&&&&0&1&1&1&1&1&1 \\ &&&&&&0&1&1&1&0&1 \\ &&&&&&&0&1&1&1&1 \\ &&&&&&&&0&0&1&1 \\ &\text{symm}&&&&&&&&0&1&0 \\ &&&&&&&&&&0&1 \\ &&&&&&&&&&&0 \end{bmatrix} \tag{17}$$

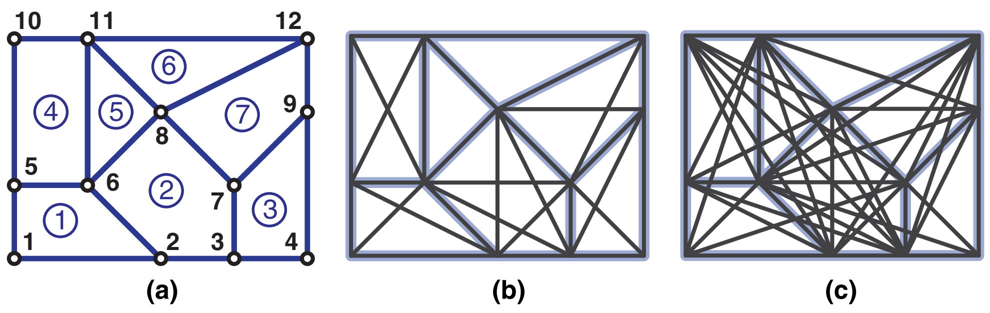


Fig. 6 Ground structure generation example. **a** Sample base mesh with 7 elements and 12 nodes. **b** Resulting ground structure for a level 1 connectivity. **c** Resulting ground structure for a level 2 connectivity

Note, for example, that node 1 does not link to nodes 3, 8 or 10 because these (level 2) bars are collinear with members from previous levels. The output of the ground structure algorithm for the base mesh in Fig. 6a, for a level 2 connectivity with no collinear bars results in the following 48 members:

$$\mathtt{BARS} = \left[\begin{array}{cc} 1 & 2 \\ 1 & 5 \\ 1 & 6 \\ 1 & 7 \\ 1 & 11 \\ 2 & 3 \\ 2 & 5 \\ 2 & 6 \\ 2 & 7 \\ 2 & 8 \\ 2 & 10 \\ 2 & 11 \\ 2 & 12 \\ \vdots & \vdots \end{array}\right. \quad \begin{array}{|cc|} \vdots & \vdots \\ 3 & 4 \\ 3 & 5 \\ 3 & 6 \\ 3 & 7 \\ 3 & 8 \\ 3 & 9 \\ 3 & 10 \\ 3 & 11 \\ 3 & 12 \\ 4 & 6 \\ 4 & 7 \\ \vdots & \vdots \end{array} \quad \begin{array}{|cc|} \vdots & \vdots \\ 4 & 9 \\ 5 & 6 \\ 5 & 8 \\ 5 & 10 \\ 5 & 11 \\ 5 & 12 \\ 6 & 7 \\ 6 & 8 \\ 6 & 9 \\ 6 & 10 \\ 6 & 11 \\ \vdots & \vdots \end{array} \quad \left.\begin{array}{cc} \vdots & \vdots \\ 6 & 12 \\ 7 & 8 \\ 7 & 9 \\ 7 & 10 \\ 7 & 12 \\ 8 & 9 \\ 8 & 10 \\ 8 & 11 \\ 8 & 12 \\ 9 & 11 \\ 9 & 12 \\ 10 & 11 \\ 11 & 12 \end{array}\right] \tag{18}$$

This connectivity information results in the ground structure in Fig. 6c.

3.4 Restriction zones

The restriction zone idea is inspired by known collision detection algorithms used in video–games and in computational geometry (Ericson 2004).

The domain may have concave regions or holes where no bar should be present. To prevent the ground structure generation algorithm from laying out members in these regions, the user defines *restriction zones* (i.e. hitboxes): geometric entities that no bar should intersect. The current implementation provides the following restriction primitives:

- rectangle
- circle
- segment (line)
- convex polygon

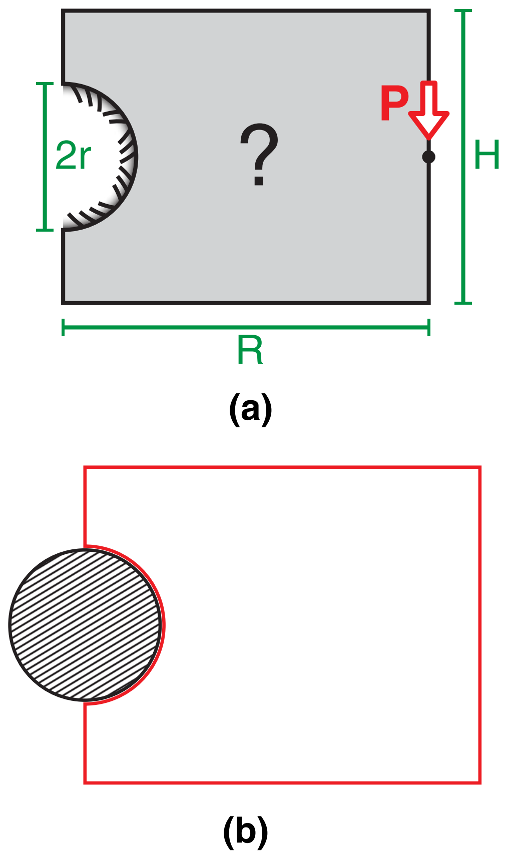
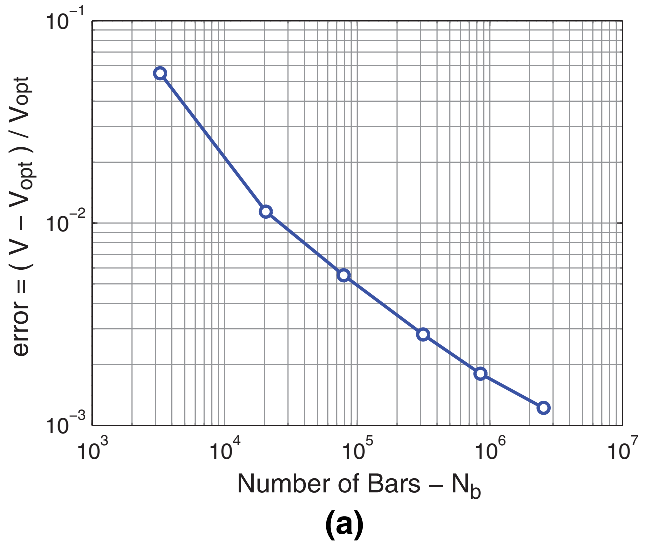


Fig. 7 Cantilever with circular support. **a** Domain, loading and boundary conditions. **b** Restriction zone for the cantilever with circular support



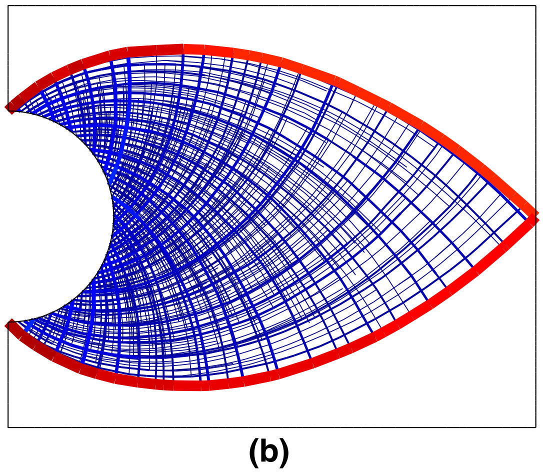


Fig. 8 Cantilever with circular support. **a** Convergence with ground structure refinement. The analytical solution is given by (Michell 1904), provided that the height H is large enough to develop the complete solution. **b** Solution obtained for $N_b = 851,511$, generated from a non–symmetric (unstructured) polygonal mesh with $N_e = 5,000$, $N_n = 9,889$ and $Lvl = 5$

Table 2 Cantilever with circular support. Problem parameters: $r = 1$, $R = 5$, $H = 4$ and $P = 1$

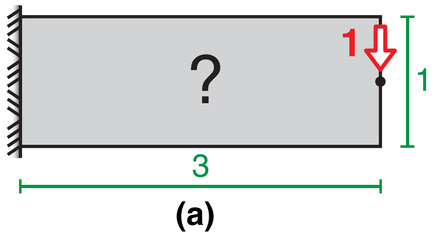
N_e	N_n	Lvl	N_b	Volume V
300	593	1	3,264	16.9824
625	1,239	2	20,447	16.2777
1,200	2,369	3	78,807	16.1834
2,800	5,527	4	313,830	16.1396
5,000	9,889	5	851,511	16.1234
10,500	20,761	6	2,548,545	16.1140

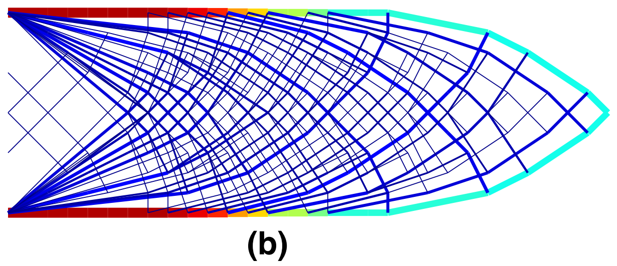
Table 3 Square cantilever beam comparison

30 × 10 mesh with Level 10 connectivity	GRAND	Structured implementation[a]
DOFs	660	660
Bars	19,632	19,632
Volume	13.6857	13.6857

[a]The structured implementation refers to (Sokół 2011)

These primitives can be combined to create complicated regions or boundaries, and the user can easily implement new collision primitives in GRAND. The problem of a cantilever with a circular support (Fig. 7a), is well known in the field of structural optimization (Michell 1904). The restriction zone for such domain is illustrated in Fig. 7b as an example. Candidate bars for levels > 1 are tested against the restriction zones. Level 1, however, is not tested because it is assumed that the base input mesh is completely contained within the feasible domain.





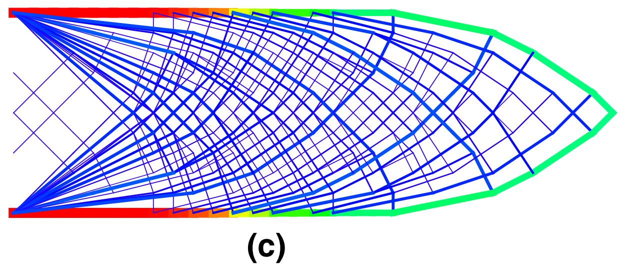


Fig. 9 Rectangular cantilever clamped at the left side and loaded at the right by a vertical force applied at the middle—central point. **a** Domain definition, discretized with 30 × 10 elements and level 10 connectivity. **b** Solution from GRAND. **c** Solution from a structured ground structure implementation (Sokół 2011)

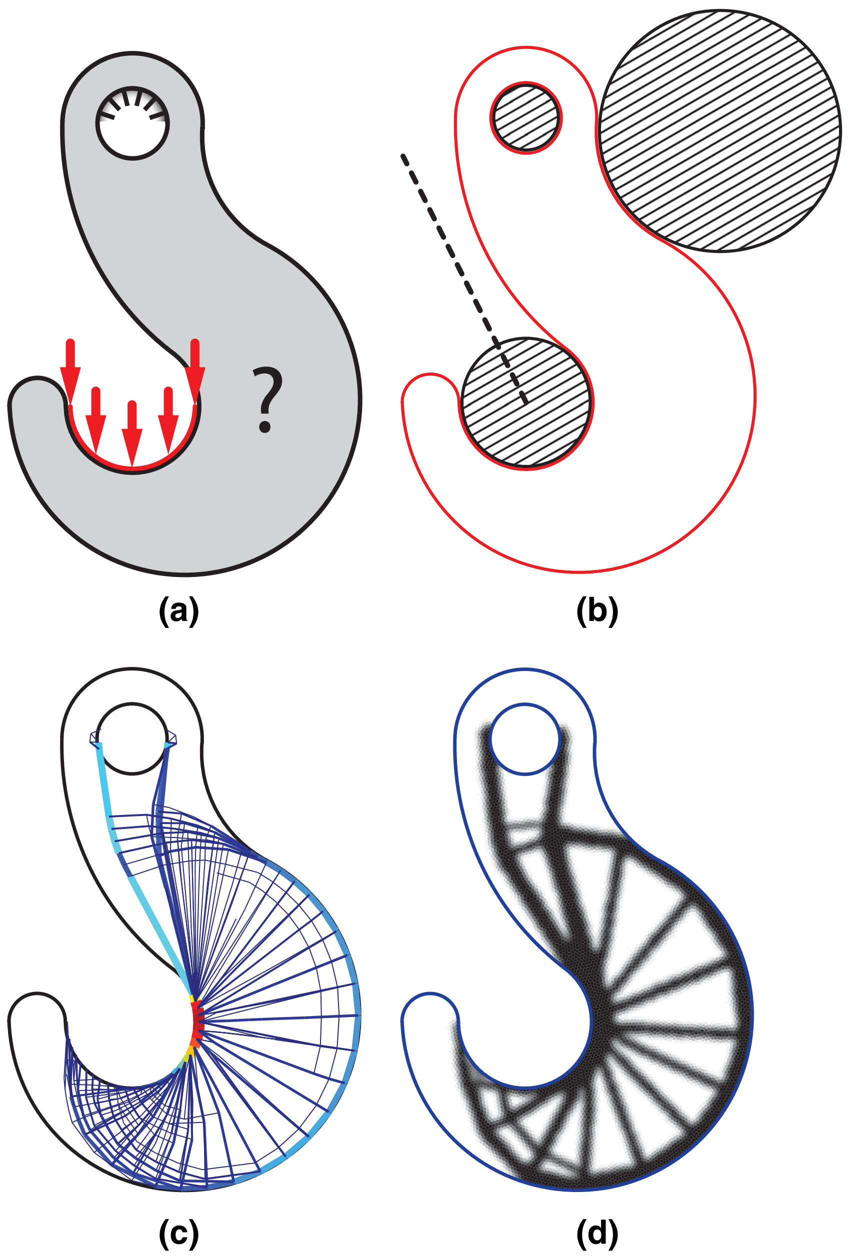


Fig. 10 Hook problem: **a** Domain, loading and boundary conditions. **b** Restriction zone composed of three circles and one segment. **c** Solution obtained from GRAND with $N_b = 72,589$ using an externally generated mesh and level 10 connectivity. **d** Solution from a density method with $N_e = 10,000$ (continuum polygonal elements) for comparison

3.5 Linear program input/output

The matrix $\mathbf{B}^\mathbf{T}$ in (6) is assembled from the directional cosines $\hat{\mathbf{d}}$, and the member length vector denoted as $\mathbf{l} = \|\mathbf{d}_i\|$. The nodal force vector $\mathbf{f}$ is generated with the information supplied in the LOAD array.

With no loss of generality, the implementation assumes $\sigma_T = 1$. The case of $\sigma_T \neq 1$ is scaled from the results obtained with the previous assumption, as follows:

$$\begin{aligned}\mathbf{a}_{\sigma_T \neq 1} &= \mathbf{a}/\sigma_T\\ V_{\sigma_T \neq 1} &= (V_{\sigma_T = 1})/\sigma_T\end{aligned} \tag{19}$$

4 Examples and verification

The following problems were selected to showcase features and issues in GRAND. Unless stated otherwise, the stress limit ratio is $\kappa = 1.0$ with $\sigma_T = 1$, the collinear tolerance is $ColTol = 0.999999$ and the plotting cutoff is $Cutoff = 0.002$ using 50 plotting groups.

4.1 Cantilever with circular support

The analytical solution of a cantilever supported on a circle (Fig. 7a) was obtained by Michell (Michell 1904), and later generalized to different geometries and conditions (Graczykowski and Lewiński 2005). This problem, for example, cannot be directly solved with an orthogonal structured domain and requires an unstructured mesh to be modeled accurately. If the height of the domain H is sufficiently large to allow the full solution to develop, the (analytical) optimal volume is:

$$V_{opt} = P \log\left(\frac{r}{R}\right)\left[\frac{1}{\sigma_T} + \frac{1}{\sigma_C}\right] \tag{20}$$

The convergence to the analytical solution with refinement is shown in Fig. 8a, with the values from Table 2. One of these solutions is shown in Fig. 8b as an example.

4.2 Structured square cantilever

The ground structure generation algorithm for unstructured meshes (detailed in this manuscript), must generate the same ground structure as an implementation of the method for structured orthogonal meshes. A rectangular cantilever beam clamped at the left side and loaded at the right by a vertical force applied at the middle–central point is used for comparison: the domain size is 3×1, discretized using 30×10 elements with a unit downward vertical load and $Cutoff = 10^{-6}$. Table 3 and Fig. 9 summarize the number of truss elements, degrees–of–freedom (DOFs), optimal volume and resulting structure for both methods: the solution using GRAND is exactly the same as the reference implementation (Sokół 2011).

4.3 Hook problem

The hook problem (Fig. 10a) initially introduced by (Talischi et al. 2012b) has no analytical solution, yet the complexity of the domain makes it a good example to showcase the capabilities of the proposed method. The domain has a restriction zone composed of four primitives as in Fig. 10b: three circles and one segment. For comparison, the solution from a polygonal density–based method (Talischi et al. 2012b) is also provided: there is a qualitative agreement of the solutions from both methods (Fig. 10c and d).

5 Conclusions

The ground structure method provides insight into the optimal solution for a given problem. This information can be used to further refine the solution, and in some cases, to obtain an analytical solution (or benchmark). The present method and implementation extends the ground structure method to domains of any shape with a simple and efficient technique.

This work impacts the field of topology optimization using ground structures by means of an integrated approach involving engineering, linear algebra, and the video–game literature. As illustrated by Fig. 1, this approach can lead to *natural designs*, which search for natural systems by reacting to loads and constraints, just as organic systems evolve in response to the surrounding environment.

To guarantee quality solutions, the ground structure should have linear independence among truss members (no collinearity), and no two members connecting the same nodes (repeated members). Concave domains and holes are addressed by defining *restriction zones*. These zones flag colliding members so that they can be removed when generating the ground structure. The member generation, collinearity check, restriction calculation and member removal are translated into linear algebra operations that can be efficiently calculated in modern linear algebra systems.

The MATLAB implementation is provided to encourage future research in the field and an in–depth understanding of the method. A library of examples (some with analytical solutions available) is provided in the appendix to further facilitate learning. The resulting code was created with a balance of performance and readability in mind, and can be further improved for speed and flexibility if required. In particular, methods that adaptively generate, modify and/or reduce the ground structure have been successful at achieving greater level of detail, while maintaining a reasonable computational cost (Gilbert and Tyas 2003; Rozvany and Sokół 2013).

Acknowledgments The authors appreciate constructive comments and insightful suggestions from the anonymous reviewers. We are thankful to the support from the US National Science Foundation under grant CMMI #1335160. We also acknowledge the support from SOM (Skidmore, Owings and Merrill LLP) and from the Donald B. and Elizabeth M. Willett endowment at the University of Illinois at Urbana–Champaign. Any opinion, finding, conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

Appendix A: Nomenclature

$\mathbf{A}$	Connectivity matrix
$\mathbf{a}$	Cross–sectional areas vector
$\mathbf{B}^{\mathrm{T}}$	Force equilibrium (geometric) matrix
$\mathbf{C}$	Dot product matrix of accepted (old) and new bars
$ColTol$	Tolerance in the collinearity check
Cutoff	Cross–section cutoff value for plotting
$\mathbf{D}$	Directional cosines matrix

$\mathbf{d}, \hat{\mathbf{d}}$	Member direction vector. Directional cosines vector
$\mathbf{f}$	Nodal load vector
$\mathbf{G}$	Candidate member matrix
$\mathbf{H}, \mathbf{H}^{\star}$	Ground structure members matrix
$\mathbf{K}$	Stiffness matrix
$\mathbf{l}$	Member's length vector
Lvl	Ground structure connectivity level
$\mathbf{n}$	Internal (axial) forces vector
N_b	Number of bars (truss members)
N_{dof}	Number of degrees–of–freedom of the structure
N_e	Number of elements in the base mesh
N_f	Number of nodes with fixities
N_l	Number of nodes with loads
N_n	Number of nodes in the domain
N_{sup}	Number of fixed nodal components
$\mathbf{s}^+, \mathbf{s}^-$	Stress constraint (positive) slack variables
$\mathbf{u}$	Nodal displacement vector
V	Volume
κ	Tension to compression stress limit ratio
$\boldsymbol{\sigma}$	Member's (axial) stresses vector
σ_T, σ_C	Stress limits in tension and compression

Appendix B: Notes on using GRAND

The user script to run GRAND is `GRANDscript.m`. Problem definition and parameters are all set within this file: `NODE`, `ELEM`, `SUPP`, `LOAD`, `Lvl`, `RestrictDomain` (if applicable), κ, `ColTol` and `Cutoff`. The *Optimization Toolbox* in MATLAB is required to run GRAND, otherwise the script will break with an error.

There are three ways to define the base mesh:

1. Using `PolyMesher` to generate a polygonal mesh
2. Using `StructDomain` to generate an orthogonal structured mesh
3. Loading an externally generated mesh with the `load` command in MATLAB

Examples using all three input methods are provided in Examples using all three input options are provided in the comments within `GRANDscript.m`. Additional details for three problems (one with each available input option) are detailed in Appendix D A brief description of all files (and folders) in GRAND is available in Table 4, and details for all 12 problems bundled in GRAND are given in Table 5.

B1: Discussion on plotting

Solutions with a large number of members may have plotting issues. To prevent this, only the members with cross–sectional areas $a_i > (Cutoff)\,(a_{max})$ are plotted; this may cause a truss cord to abruptly end mid–air in the resulting figure, which is not accurate, but is merely a plotting artifact. In addition, in order to reduce the plot function calls, members of similar cross–sectional areas are grouped and plotted with average visual characteristics (thickness and color), in a single function call. The user can specify the number of groups used in this plot grouping technique.

Table 4 Description of function files in GRAND

Filename	Description
`GRANDscript`	Main user script. Contains commented examples for all three available input options. Parameters are set in the file's preamble.
`GenerateGS`	Generates the ground structure for a given base mesh up to a specified level, with an optional restriction zone.
`GetMatrixBT`	Builds the nodal equilibrium matrix $\mathbf{B^T}$. The row entries corresponding to supports are removed.
`GetSupports`	Returns the global numbering of supported nodal components based on the data in `SUPP`. These are in turn used to remove these equations from the nodal equilibrium matrix $\mathbf{B^T}$ and force vector $\mathbf{f}$.
`GetVectorF`	Returns the nodal force vector based on the data in `LOAD`. The row entries corresponding to supports are removed.
`PlotBoundary`	Identifies the boundary edges and plots them.
`PlotPolyMesh`	Takes the resulting cross–sectional areas, the nodal coordinates and member definitions to plot all members with cross–sectional areas above the specified cutoff.
`rCircle`	Restriction zone primitive to test collision against a circle.
`rLine`	Restriction zone primitive to test collision against a line segment.
`rPolygon`	Restriction zone primitive to test collision against a polygon.
`rRectangle`	Restriction zone primitive to test collision against a rectangle.
`StructDomain`	Generates structured orthogonal domains and can optionally also return boundary conditions for a cantilever problem, the MBB beam and a half–space bridge.
`PolyMesher/`	Folder containing the polygonal mesher with all its related files (Talischi et al. 2012a).

Appendix C: MATLAB code

The following files comprise the problem–independent modules of GRAND.

GRANDscript.m

```
%GRAND - Ground Structure Analysis and Design Code.
%   Tomas Zegard, Glaucio H Paulino - Version 1.0, Dec-2013

%% === MESH GENERATION LOADS/BCS ==========================================
kappa = 1.0; ColTol = 0.999999;
Cutoff = 0.002; Ng = 50; % Plot: Member Cutoff & Number of plot groups

% --- OPTION 1: POLYMESHER MESH GENERATION --------------------------------
% addpath('./PolyMesher')
% [NODE,ELEM,SUPP,LOAD] = PolyMesher(@MichellDomain,600,30);
% Lvl = 5; RestrictDomain = @RestrictMichell;
% rmpath('./PolyMesher')

% --- OPTION 2: STRUCTURED-ORTHOGONAL MESH GENERATION ---------------------
% [NODE,ELEM,SUPP,LOAD] = StructDomain(60,20,3,1,'MBB');
% Lvl = 6; RestrictDomain = []; % No restriction for box domain

% --- OPTION 3: LOAD EXTERNALLY GENERATED MESH ----------------------------
% load MeshHook
% Lvl = 10; RestrictDomain = @RestrictHook;

% load MeshSerpentine
% Lvl = 5; RestrictDomain = @RestrictSerpentine;

% load MeshMichell
% Lvl = 4; RestrictDomain = @RestrictMichell;

load MeshFlower
Lvl = 4; RestrictDomain = @RestrictFlower;

%% === GROUND STRUCTURE METHOD ============================================
PlotPolyMesh(NODE,ELEM,SUPP,LOAD) % Plot the base mesh
[BARS] = GenerateGS(NODE,ELEM,Lvl,RestrictDomain,ColTol); % Generate the GS
Nn = size(NODE,1); Ne = length(ELEM); Nb = size(BARS,1);
[BC] = GetSupports(SUPP);                 % Get reaction nodes
[BT,L] = GetMatrixBT(NODE,BARS,BC,Nn,Nb); % Get equilibrium matrix
[F] = GetVectorF(LOAD,BC,Nn);             % Get nodal force vector

fprintf('Mesh: Elements %d, Nodes %d, Bars %d, Level %d\n',Ne,Nn,Nb,Lvl)
BTBT = [BT -BT]; LL = [L; kappa*L]; sizeBTBT = whos('BTBT'); clear BT L
fprintf('Matrix [BT -BT]: %d x %d in %gMB (%gGB full)\n',...
        length(F),length(LL),sizeBTBT.bytes/2^20,16*(2*Nn)*Nb/2^30)

tic, [S,vol,exitflag] = linprog(LL,[],[],BTBT,F,zeros(2*Nb,1));
fprintf('Objective V = %f\nlinprog CPU time = %g s\n',vol,toc);

S = reshape(S,numel(S)/2,2);  % Separate slack variables
A = S(:,1) + kappa*S(:,2);    % Get cross-sectional areas
N = S(:,1) - S(:,2);          % Get member forces

%% === PLOTTING ===========================================================
PlotGroundStructure(NODE,BARS,A,Cutoff,Ng)
PlotBoundary(ELEM,NODE)
```

GenerateGS.m

```
function [BARS]=GenerateGS(NODE,ELEM,Lvl,RestrictDomain,ColTol)

if nargin<5, ColTol=0.9999; end
if (nargin<4 || isempty(RestrictDomain)), RestrictDomain=@(~,~)[];
elseif nargin<3, error('Not enough input arguments.'), end

% Get element connectivity matrix
Nn = max(cellfun(@max,ELEM)); Ne = length(ELEM);
A1 = sparse(Nn,Nn);
for i=1:Ne, A1(ELEM{i},ELEM{i}) = true; end
A1 = A1 - speye(Nn,Nn); An = A1;

% Level 1 connectivity
[J,I] = find(An); % Reversed because find returns values column-major
BARS = [I J];
D = [NODE(I,1)-NODE(J,1) NODE(I,2)-NODE(J,2)];
L = sqrt(D(:,1).^2+D(:,2).^2);  % Length of bars
D = [D(:,1)./L D(:,2)./L];      % Normalized dir

% Levels 2 and above
for i=2:Lvl
    Aold = An; An = logical(An*A1); Gn = An - Aold; % Get NEW bars @ level 'n'
    [J,I] = find(Gn-diag(diag(Gn)));
    if isempty(J), Lvl = i - 1; fprintf('-INFO- No new bars at Level %g\n',Lvl); break, end

    RemoveFlag = RestrictDomain(NODE,[I J]); % Find and remove bars within restriction zone
    I(RemoveFlag) = []; J(RemoveFlag) = [];

    newD = [NODE(I,1)-NODE(J,1) NODE(I,2)-NODE(J,2)];
    L = sqrt(newD(:,1).^2+newD(:,2).^2);
    newD = [newD(:,1)./L newD(:,2)./L];

    % Collinearity Check
    p = 1; m = 1; RemoveFlag = zeros(size(I)); Nb = size(BARS,1);
    for j=1:Nn
        % Find I(p:q) - NEW bars starting @ node 'j'
        for p=p:length(I), if I(p)>=j, break, end, end
        for q=p:length(I), if I(q)>j, break, end, end
        if I(q)>j, q = q - 1; end

        if I(p)==j
            % Find BARS(m:n) - OLD bars starting @ node 'j'
            for m=1:Nb, if BARS(m,1)>=j, break, end, end
            for n=m:Nb, if BARS(n,1)>j, break, end, end
            if BARS(n,1)>j, n = n - 1; end

            if BARS(n,1)==j
                % Dot products of old vs. new bars. If ~collinear: mark
                C = max(D(m:n,:)*newD(p:q,:)',[],1);
                RemoveFlag(p-1+find(C>ColTol)) = true;
            end
        end
    end

    % Remove collinear bars and make symmetric again. Bars that have one
    % angle marked as collinear but the other not, will be spared
    ind = find(RemoveFlag==false);
    H = sparse(I(ind),J(ind),true,Nn,Nn,length(ind));
    [J,I] = find(H+H');
    fprintf('Lvl %2g - Collinear bars removed: %g\n',i,(length(RemoveFlag)-length(I))/2);

    BARS = sortrows([BARS; I J]);
    D = [NODE(BARS(:,1),1)-NODE(BARS(:,2),1) NODE(BARS(:,1),2)-NODE(BARS(:,2),2)];
    L = sqrt(D(:,1).^2+D(:,2).^2);  % Length of bars
    D = [D(:,1)./L D(:,2)./L];      % Normalized dir
end

% Only return bars {i,j} with i<j (no duplicate bars)
A = sparse(BARS(:,1),BARS(:,2),true,Nn,Nn);
[J,I] = find(tril(A)); BARS = [I J];
```

GetMatrixBT.m

```
function [BT,L]=GetMatrixBT(NODE,BARS,BC,Nn,Nb)
% Generate equilibrium matrix BT and get member lengths L
D = [NODE(BARS(:,2),1)-NODE(BARS(:,1),1) NODE(BARS(:,2),2)-NODE(BARS(:,1),2)];
L = sqrt(D(:,1).^2+D(:,2).^2);
D = [D(:,1)./L D(:,2)./L];
BT = sparse([2*BARS(:,1)-1 2*BARS(:,1) 2*BARS(:,2)-1 2*BARS(:,2)],...
            repmat((1:Nb)',1,4),[-D D],2*Nn,Nb);
BT(BC,:) = [];
```

GetSupports.m

```
function [BC]=GetSupports(SUPP)
% Return degrees-of-freedom with fixed (prescribed) displacements
Nf = sum(sum(~isnan(SUPP(:,2:3))));
BC = zeros(Nf,1); j = 0;
for i=1:size(SUPP,1)
    if ~isnan(SUPP(i,2)), j = j + 1; BC(j) = 2*SUPP(i) - 1; end
    if ~isnan(SUPP(i,3)), j = j + 1; BC(j) = 2*SUPP(i);     end
end
if j~=Nf, error('Parsing number mismatch on BCs.'), end
```

GetVectorF.m

```
function [F]=GetVectorF(LOAD,BC,Nn)
% Return nodal force vector
Nl = sum(sum(~isnan(LOAD(:,2:3))));
F = sparse([],[],[],2*Nn,1,Nl);
for i=1:size(LOAD,1)
    n = LOAD(i,1);
    if ~isnan(LOAD(i,2)), F(2*n-1) = LOAD(i,2); end
    if ~isnan(LOAD(i,3)), F(2*n) = LOAD(i,3);   end
end
F(BC) = [];
```

PlotBoundary.m

```
function []=PlotBoundary(ELEM,NODE)

% Get number of nodes, elements and edges (nodes) per element
Nn = size(NODE,1); Ne = length(ELEM); NpE = cellfun(@numel,ELEM);

FACE = sparse([],[],[],Nn,Nn,sum(NpE));
for i=1:Ne
    MyFACE = [ELEM{i}; ELEM{i}(2:end) ELEM{i}(1)];
    for j=1:NpE(i)
        if FACE(MyFACE(1,j),MyFACE(2,j))==0 % New edge - Flag it
            FACE(MyFACE(1,j),MyFACE(2,j)) = i;
            FACE(MyFACE(2,j),MyFACE(1,j)) =-i;
        elseif isnan(FACE(MyFACE(1,j),MyFACE(2,j)))
            error(sprintf('Edge [%d %d] found in >2 elements',MyFACE(:,j)))
        else % Edge belongs to 2 elements: inside domain. Lock it.
            FACE(MyFACE(1,j),MyFACE(2,j)) = NaN;
            FACE(MyFACE(2,j),MyFACE(1,j)) = NaN;
        end
    end
end
[BOUND(:,1),BOUND(:,2)] = find(FACE>0);
BOUND(:,3) = FACE(sub2ind(size(FACE),BOUND(:,1),BOUND(:,2)));
plot([NODE(BOUND(:,1),1) NODE(BOUND(:,2),1)]',[NODE(BOUND(:,1),2) NODE(BOUND(:,2),2)]','k')
```

PlotGroundStructure.m

```
function []=PlotGroundStructure(NODE,BARS,A,Cutoff,Ng)

figure('Name','GRAND v1.0 -- Zegard T, Paulino GH','NumberTitle','off')
hold on, axis equal, axis off, color=jet(Ng);

A = A/max(A); % Normalize to [0,1] areas
ind = find(A>Cutoff);
MyGroup = ceil(Ng*A(ind)); % Round up to the closest group of bars
Groups = cell(Ng,1);       % Store the indices of similar bars
for i=1:Ng, Groups{i} = ind(find(MyGroup==i)); end
for i=Ng:-1:1 % Plot each group of similar bars in a single plot call
    if ~isempty(Groups{i})
        XY = [NODE(BARS(Groups{i},1),:) NODE(BARS(Groups{i},2),:)];
        GroupArea = mean(A(Groups{i})); % Mean area for this group
        plot(XY(:,[1 3])',XY(:,[2 4])','LineWidth',5*sqrt(GroupArea),'Color',color(i,:))
    end
end
fprintf('-PLOT- Cutoff %g, Groups %g, Bars plotted %g\n',Cutoff,Ng,length(ind))
```

PlotPolyMesh.m

```
function []=PlotPolyMesh(NODE,ELEM,SUPP,LOAD)
figure, hold on, axis equal, axis off
MaxNVer = max(cellfun(@numel,ELEM));         %Max. num. of vertices in mesh
PadWNaN = @(E) [E NaN(1,MaxNVer-numel(E))];  %Pad cells with NaN
ElemMat = cellfun(PadWNaN,ELEM,'UniformOutput',false);
ElemMat = vertcat(ElemMat{:});               %Create padded element matrix
patch('Faces',ElemMat,'Vertices',NODE,'FaceColor','w');
if (nargin==4 && ~isempty(SUPP) && ~isempty(LOAD))
    plot(NODE(SUPP(:,1),1),NODE(SUPP(:,1),2),'b>','MarkerSize',8);
    plot(NODE(LOAD(:,1),1),NODE(LOAD(:,1),2),'m^','MarkerSize',8);
end
axis tight, drawnow
```

rCircle.m

```
function flag=rCircle(C,r,NODE,BARS)
% Circle with center point C and radius R
Nb = size(BARS,1);
U = NODE(BARS(:,1),:) - repmat(C,Nb,1);
V = NODE(BARS(:,2),:) - repmat(C,Nb,1);
D = V - U;
L = sqrt(D(:,1).^2 + D(:,2).^2);
D = [ D(:,1)./L D(:,2)./L ];
flag = any( [ ( sum(D.*V,2)>=0 ) .* ( sum(D.*U,2)<=0 ) .*...
              ( abs(D(:,1).*U(:,2)-D(:,2).*U(:,1))<r ) , ...
              ( U(:,1).^2+U(:,2).^2<=r^2 ) , ...
              ( V(:,1).^2+V(:,2).^2<=r^2 ) ] , 2);
```

rLine.m

```
function flag=rLine(A,B,NODE,BARS)
% Line segment between points A and B
P = NODE(BARS(:,1),:); D = NODE(BARS(:,2),:) - P; V = B - A;
C = D(:,1)*V(2) - V(1)*D(:,2);                          % cross(d,v)
Ct = (A(1)-P(:,1)).*D(:,2) - (A(2)-P(:,2)).*D(:,1); % cross(a-p,d)
Cu = (A(1)-P(:,1))*V(2) - (A(2)-P(:,2))*V(1);       % cross(a-p,v)
Ct = Ct./C; Cu = Cu./C;
% If intersection is between A-B and P-Q
flag = (Ct>0).*(Ct<1).*(Cu>0).*(Cu<1);
```

rPolygon.m

```
function flag=rPolygon(A,NODE,BARS)
% Polygon with N edges defined by A of size [N x 2]
% Get normals for each half-space (A are poly nodes in CCW)
Np= size(A,1); % Number of half-spaces
N = zeros(Np,2);
N(1:Np-1,:) = A(2:Np,:) - A(1:Np-1,:); N(Np,:) = A(1,:) - A(Np,:);
N = [ N(:,2) -N(:,1) ]; % Normal vectors for all half-spaces
% Get number of bars and initialize T
Nb= size(BARS,1);
D = NODE(BARS(:,2),:) - NODE(BARS(:,1),:);
Tmin = zeros(Nb,1); Tmax = ones(Nb,1);
% Loop through all halfspaces
for i=1:Np
    deno = D * N(i,:)';
    dist = ( repmat(A(i,:),Nb,1) - NODE(BARS(:,1),:) ) * N(i,:)';
    T = dist ./ deno;
    ind = find( (T>Tmin) .* (deno<0) ); Tmin(ind) = T(ind);
    ind = find( (T<Tmax) .* (deno>0) ); Tmax(ind) = T(ind);
end
% No intersection if Tmin>Tmax
flag = (Tmin<=Tmax);
```

rRectangle.m

```
function flag=rRectangle(Amin,Amax,NODE,BARS)
% Amin and Amax are the rectangle's limit coords: minimum and maximum
Nb= size(BARS,1);
Tmin = zeros(Nb,1); Tmax = ones(Nb,1);
D = NODE(BARS(:,2),:) - NODE(BARS(:,1),:);
for i=1:2 % Check on X (i=1) and Y (i=2)
    T1 = ( Amin(i) - NODE(BARS(:,1),i) ) ./ D(:,i);
    T2 = ( Amax(i) - NODE(BARS(:,1),i) ) ./ D(:,i);
    ind = find(T1>T2); % We require T1<T2, swap if not
    [T1(ind),T2(ind)] = deal(T2(ind),T1(ind)); % Swap operation
    Tmin = max(Tmin,T1); Tmax = min(Tmax,T2);
end
% No intersection with rectangle if Tmin>Tmax
flag = (Tmin<=Tmax);
```

StructDomain.m

```
function [NODE,ELEM,SUPP,LOAD]=StructDomain(Nx,Ny,Lx,Ly,ProblemID)
% Generate structured-orthogonal domains
[X,Y] = meshgrid(linspace(0,Lx,Nx+1),linspace(0,Ly,Ny+1));
NODE = [reshape(X,numel(X),1) reshape(Y,numel(Y),1)];
k = 0; ELEM = cell(Nx*Ny,1);
for j=1:Ny, for i=1:Nx
        k = k+1;
        n1 = (i-1)*(Ny+1)+j; n2 = i*(Ny+1)+j;
        ELEM{k} = [n1 n2 n2+1 n1+1];
end, end

if (nargin==4 || isempty(ProblemID)), ProblemID = 1; end
switch ProblemID
    case {'Cantilever','cantilever',1}
        SUPP = [(1:Ny+1)' ones(Ny+1,2)];
        LOAD = [Nx*(Ny+1)+round((Ny+1)/2) 0 -1];
    case {'MBB','Mbb','mbb',2}
        SUPP = [Nx*(Ny+1)+1 NaN 1;
                (1:Ny+1)' ones(Ny+1,1) nan(Ny+1,1)];
        LOAD = [Ny+1 0 -0.5];
    case {'Bridge','bridge',3}
        SUPP = [     1      1 1;
                Nx*(Ny+1)+1 1 1];
        LOAD = [(Ny+1)*round(Nx/2)+1 0 -1];
    otherwise
        SUPP = []; LOAD = [];
        disp('-INFO- Structured domain generated with no loads/BC')
end
```

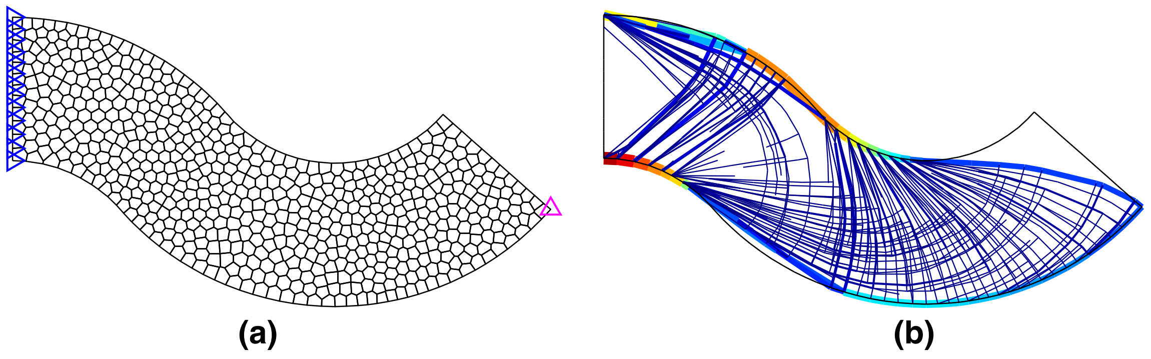


Fig. 11 Usage example of GRAND with input method 1. **a** Serpentine domain discretized with PolyMesher: $N_e = 600$ and $N_n = 1,192$. **b** Optimized ground structure for the serpentine domain: $Lvl = 5$ and $N_b = 1,192$

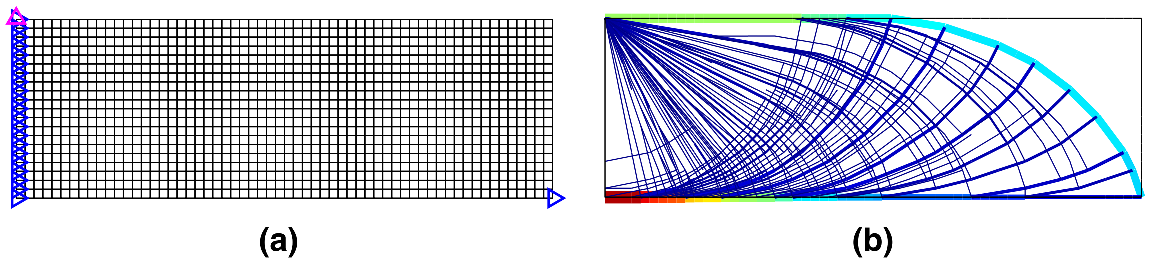


Fig. 12 Usage example of GRAND with input method 2. **a** MBB domain discretized using `StructDomain` in GRAND: $N_e = 1,200$ and $N_n = 1,281$. **b** Optimized ground structure for the MBB domain: $Lvl = 6$ and $N_b = 49,888$

Table 5 Examples provided in GRAND

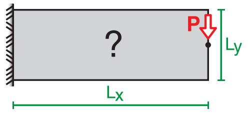
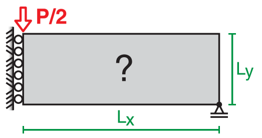
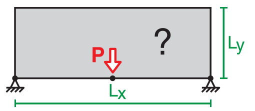
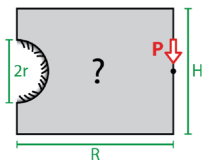
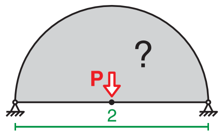
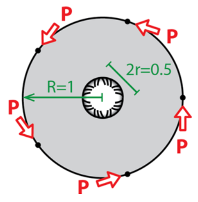
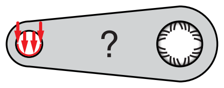
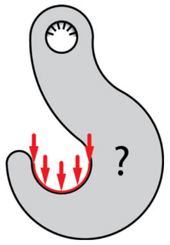
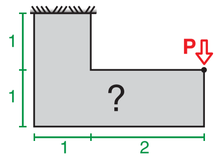
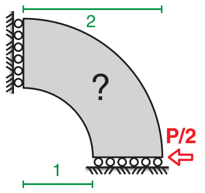
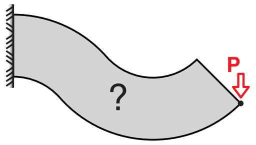
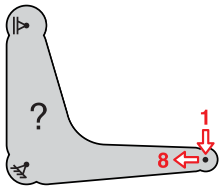
Domain	Base mesh definition	Restriction zone	Comments
	`StructDomain(Nx,Ny,Lx,Ly,'Cantilever')` or PolyMesher with `@CantileverDomain`	—	Horizontal and Vertical lengths `Lx` and `Ly`, using `Nx` and `Ny` elements in each direction. By default the load is $P = 1$
	`StructDomain(Nx,Ny,Lx,Ly,'MBB')` or PolyMesher with `@MbbDomain`	—	Horizontal and Vertical lengths `Lx` and `Ly`, using `Nx` and `Ny` elements in each direction. By default the load is $P/2 = 0.5$
	`StructDomain(Nx,Ny,Lx,Ly,'Bridge')` or PolyMesher with `@BridgeDomain`	—	Dimensions `Lx` and `Ly`, with `Nx` and `Ny` elements in each direction, with a default load $P = 1$. The analytical solution for this problem (if $L_y \geq \sqrt{2}L_x/4$) is: $V_{opt} = P\left(\frac{L_x}{2}\right)\left(\frac{1}{2}+\frac{\pi}{4}\right)\left[\frac{1}{\sigma_T}+\frac{1}{\sigma_C}\right] = 1.2854L_x$
	PolyMesher with `@MichellDomain` or `load MeshMichell`	`@RestrictMichell`	The default parameters are $r = 1$, $R = 5$, $H = 4$ and $P = 1$. The analytical solution for this problem is: $V_{opt} = PR\log\left(\frac{R}{r}\right)\left[\frac{1}{\sigma_T}+\frac{1}{\sigma_C}\right] = 16.0944$
	PolyMesher with `@HalfcircleDomain`	—	The default load is $P = 1$. The analytical solution for this problem is: $V_{opt} = Pr\left(\frac{1}{2}+\frac{\pi}{4}\right)\left[\frac{1}{\sigma_T}+\frac{1}{\sigma_C}\right] = 2.5708$
	PolyMesher with `@FlowerDomain` or `load MeshFlower`	`@RestrictFlower`	The default load is $P = 1$. The analytical solution for this problem is: $V_{opt} = 5PR\log\left(\frac{R}{r}\right)\left[\frac{1}{\sigma_T}+\frac{1}{\sigma_C}\right] = 13.8629$

Table 5 (continued)

Domain	Base mesh definition	Restriction zone	Comments
	PolyMesher with `@WrenchDomain`	`@RestrictWrench`	Loads are mesh dependent: volume will not converge to a specific value Reference: (Talischi et al. 2012b)
	PolyMesher with `@HookDomain` or `load MeshHook`	`@RestrictHook`	Loads are mesh dependent: volume will not converge to a specific value Reference: (Talischi et al. 2012b)
	PolyMesher with `@LshapeDomain`	`@RestrictLshape`	By default the load is $P = 1$
	PolyMesher with `@RingDomain`	`@RestrictRing`	By default the load is $P/2 = 0.5$
	PolyMesher with `@SerpentineDomain` or `load MeshSerpentine`	`@RestrictSerpentine`	By default the load is $P = 1$ Reference: (Talischi et al. 2012b)
	PolyMesher with `@SuspensionDomain`	`@RestrictSuspension`	Reference: (Talischi et al. 2012b)

Appendix D: Usage examples

The examples here discussed are taken from the library of examples distributed with GRAND. The complete library of examples with each of the three possible input options will be addressed.

The base mesh input section encompasses lines 8–29 in `GRANDscript.m`. In these examples $\kappa = 1$, $ColTol = 0.999999$ and $Cutoff = 0.002$. These parameters are defined in lines 5–6 in `GRANDscript.m`. The modifications required to run each problem will be clearly outlined, and the reader can refer to the complete source code available in Appendix C.

D1: Input option 1 – Domain meshed with PolyMesher

To run the serpentine problem with a base mesh of 600 polygonal elements and level 5 connectivity, lines 9–12 in `GRANDscript.m` should be modified to:

```
addpath('./PolyMesher')
[NODE,ELEM,SUPP,LOAD] = PolyMesher(@SerpentineDomain,600,30);
Lvl = 5; RestrictDomain = @RestrictSerpentine;
rmpath('./PolyMesher')
```

The PolyMesher main function is called with the distance function `SerpentineDomain` as the first argument, specifying 600 elements in the second and 30 Lloyd's iterations in the third (Talischi et al. 2012a). The next line specifies the connectivity level and sets the restriction zone to the function handle of `RestrictSerpentine`. All the other input options must be commented out. The output for these settings are Fig. 11a and b.

D2: Input option 2 – Structured orthogonal domains

To run the MBB beam problem with a structured orthogonal mesh of dimensions 3×1, with 60×20 elements, and level 6 connectivity, lines 15–16 in `GRANDscript.m` should be modified to:

```
[NODE,ELEM,SUPP,LOAD] = StructDomain(60,20,3,1,'MBB');
Lvl = 6; RestrictDomain = []; % No restriction for box domain
```

The function `StructDomain` is called to generate a domain with the required elements and dimensions (inputs 1 through 4). In addition, the boundary conditions (loads and supports) for the MBB beam problem are requested with the 5^{th} argument set to `'MBB'` (with quotes). The next line specifies the connectivity level and sets the restriction zone to *empty*. All the other input options must be commented out. The output for these settings are Fig. 12a and b.

5.1 D3: Input option 3 – Externally generated mesh

To run the flower problem with level 4 connectivity, lines 28–29 must be uncommented from within the examples provided in `GRANDscript.m`. These lines are:

```
load MeshFlower
Lvl = 4; RestrictDomain = @RestrictFlower;
```

The first line loads the base mesh and boundary conditions in the format specified by Table 1. The next line specifies the connectivity level and sets the restriction zone to the function handle of `RestrictFlower`. All the other input options must be commented out. The base–mesh has $N_e = 2,000$ and $N_n = 2,100$ as shown in Fig. 1b; generating the ground structure for $Lvl = 4$ connectivity results in $N_b = 69,400$. The resulting optimal structure is shown in Fig. 1c. This example illustrates the capability of GRAND for generating a biologically inspired structure, displaying the patterns of a flower, as the one shown in Fig. 1d.

References

Achtziger W (2007) On simultaneous optimization of truss geometry and topology. Struct Multidiscip Optim 33(4–5):285–304

Bendsøe M, Sigmund O (2003) Topology optimization: theory, methods and applications, 2nd edn. Springer, Berlin

Christensen P, Klarbring A (2009) An introduction to structural optimization. Springer, Berlin

Dorn W, Gomory R, Greenberg H (1964) Automatic design of optimal structures. J Mecanique 3(1):25–52

Ericson C (2004) Real-time collision detection. Morgan Kaufmann, San Francisco

Gilbert M, Tyas A (2003) Layout optimization of large-scale pin-jointed frames. Eng Comput 20(8):1044–1064

Graczykowski C, Lewiński T (2005) The lightest plane structures of a bounded stress level transmitting a point load to a circular support. Control Cybern 34(1):227–253

Heath M (1998) Scientific computing. An introductory survey, 2nd edn. McGraw Hill, New York

Hegemier G, Prager W (1969) On Michell trusses. Int J Mech Sci 11:209–215

Hemp W (1973) Optimum Structures. Oxford University Press, Oxford

Hencky H (1923) Über einige statisch bestimmte Fälle des Gleichgewichts in plastischen Körpern. Z Angew Math Mech 747:241–251

Karmarkar N (1984) A new polynomial-time algorithm for linear programming. Combinatorica 4(4):373–395

Kicher TP (1966) Optimum design-minimum weight versus fully stressed. ASCE J Struct Div 92(ST 6):265–279

Kirsch U (1993) Structural optimization: fundamentals and applications. Springer, Berlin

Lewiński T, Rozvany GIN, Sokół T, Bołbotowski K (2013) Exact analytical solutions for some popular benchmark problems in topology optimization III: L-shaped domains revisited. Struct Multidiscip Optim 47(6):937–942

Michell AGM (1904) The limits of economy of material in frame-structures. Phil Mag Ser 6 8(47):589–597

Ohsaki M (2010) Optimization of finite dimensional structures. CRC Press, Boca Raton

Olson L (2013) Personal communication

Rozvany G (2001) On design-dependent constraints and singular topologies. Struct Multidiscip Optim 21(2):164–172

Rozvany G, Sokół T (2013) Validation of numerical methods by analytical benchmarks, and verification of exact solutions by numerical methods. In: Topology optimization in structural and continuum mechanics. Springer, Vienna, Austria

Smith ODS (1998) Generation of ground structures for 2D and 3D design domains. Eng Comput 15(4):462–500

Sokół T (2011) A 99 line code for discretized Michell truss optimization written in Mathematica. Struct Multidiscip Optim 43(2):181–190

Sved G (1954) The minimum weight of certain redundant structures. Aust J Appl Sci 5:1–9

Sved G, Ginos Z (1968) Structural optimization under multiple loading. Int J Mech Sci 10:803–805

Talischi C, Paulino GH, Pereira A, Menezes IFM (2012a) PolyMesher: a general-purpose mesh generator for polygonal elements written in Matlab. Struct Multidiscip Optim 45(3):309–328

Talischi C, Paulino GH, Pereira A, Menezes IFM (2012b) PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. Struct Multidiscip Optim 45(3):329–357

Wright M (2005) The interior-point revolution in optimization: history, recent developments, and lasting consequences. Bull Am Math Soc 42(1):39–56