Struct Multidisc Optim (2015) 51:125–143
DOI 10.1007/s00158-014-1117-8

RESEARCH PAPER

# Multiobjective topology optimization of energy absorbing materials

**Raymond A. Wildman · George A. Gazonas**

Received: 4 November 2013 / Revised: 21 April 2014 / Accepted: 13 May 2014 / Published online: 24 June 2014
© Springer-Verlag Berlin Heidelberg 2014

**Abstract** A method for the multiobjective optimization of local-scale material topology is presented. The topology optimization scheme is based on a constructive solid geometry-like representation, in which convex polygons—defined as the convex hull of arbitrary-length lists of points—are combined using an overlapping function. This data structure is tree-shaped and so genetic programming is used as the optimizer. The forward problem is solved with a multiscale finite element method with automatic cohesive zone insertion to model damage. As a multiscale method, loads and boundary conditions are applied and objective functions measured at a global scale, while the local scale material structure is optimized. The global scale geometry is assumed fixed. Pareto optimal designs are generated, representing optimal tradeoffs between conflicting goals: quasi-static displacement and dynamic strain energy. Results demonstrate the efficacy of the proposed algorithm.

**Keywords** Topology optimization · Pareto optimization · Genetic programming · Energy absorbing material

## 1 Introduction

Materials designed to absorb energy have applications in numerous areas: crashworthiness (Mozumder et al. 2012; Guo et al. 2011; Huang et al. 2007; Anghileri et al. 2005), head protection (Rueda et al. 2009), impact resistance (Qiao et al. 2008), and blast resistance (Main and Gazonas 2008; Qi et al. 2013). There are several ways to design an energy absorbing material or system of materials, for example, one could layer several different materials in a one dimensional sense, while optimizing for material placement and thickness. Another approach is to design the geometry or topology of a bi-material system or truss structure in two dimensions or three dimensions. In this work, we take the latter approach, and focus on designing energy absorbing materials at a local scale in a multiscale finite element (FEM) setting.

Our approach to designing materials will be to optimize the topology of a material at a local scale in two dimensions, while applying loads and boundary conditions and computing objectives at the global scale. A heterogeneous representative volume element (RVE) (strictly two materials in our case) will represent the local scale, and the global scale will be assumed to be statistically homogenous with material properties derived from a homogenization of the representative volume element (RVE). At the local scale, boundary conditions will be spatially homogenous using linear displacements, while at the global scale, a material sample will be fixed at one edge, with different loading conditions applied depending on whether we are considering static displacement or dynamic energy absorption. The different loading configurations are used to simulate possible scenarios in which such a material would be used, either as a structural element or an energy absorbing material for blast-like loading. As will be discussed in more detail later, these two loading conditions will be used as two separate goals in a multi-objective optimization problem. There are several options for optimizing such a configuration, the first choice being between a simple geometry or shape optimization method or a topology optimization method. A shape optimization method assumes a specific topology (i.e. each design is homeomorphic in that they must have the same

R. A. Wildman (✉) · G. A. Gazonas
U.S. Army Research Laboratory,
Attn: RDRL-WMM-B, Aberdeen Proving Ground,
MD, 21005, USA
e-mail: raymond.a.wildman.civ@mail.mil

number of shapes and holes), and parameterizes that shape with some type of function expansion, the simplest being a linear interpolation between a given number of vertices. An optimization method, either local, derivative-based or global heuristic, can then be used to solve for the defining parameters of the shape. An example of this approach can be found for the design of energy absorbing tubular structures in Chiandussi and Avalle (2002).

Shape optimization is overly restrictive, however, as one must fix the topology *a priori*. A popular method for topology optimization is known as the solid isotropic material with penalization (SIMP) method, introduced by Bendsøe (Duysinx and Bendsoe 1998; Bendsoe and Sigmund 2003). This method uses a square grid of unknowns (pixels), where each unknown is continuous value representing a graded material. A local optimization method is used to determine the optimal value of graded material at each pixel. Though a structure consisting of graded material is the result, methods exist to regularize the final result to a single material. Level-set methods have also been used to optimize topology (Sethian and Wiegmann 2000), which use contours of a higher dimensional function to represent topology in the plane. Finally, evolutionary methods have also been designed for topology optimization (Xie and Steven 1993, 1997).

To optimize topology, we use a constructive solid geometry (CSG) representation (Requicha and Voelker 1977), which uses Boolean operations to combine shape primitives. While typical implementations of CSG use canonical shapes such as rectangles and circles, we use convex polygons represented as the convex hull of an arbitrary length list of points. This encoding can represent any physically realizable (orientable) topology as a set of line segments. Though line segments are used, their length is not restricted, so arbitrarily small line segments can be used to approximate curves. Genetic programming (GP), a variant of genetic algorithms (GA) that uses a tree-based chromosome, will be used and is a natural fit for this topology representation as CSG can be readily expressed as a tree structure. This approach has been used in several applications including RF microwave inverse scattering (Wildman and Weile 2007, 2010), gravitational anomaly inversion (Wildman and Gazonas 2009), and phononic bandgap material design (Wildman and Gazonas 2011).

The approach presented here can be compared with current methods in two ways: geometry representation and optimization method. Our approach offers an alternative to the SIMP and level set methods in that it ultimately uses a set of line segments to approximate a topology rather than a grid (SIMP) or smooth curves (level-set). As will be described below, multiple materials are also easily represented in this approach. Further, as automatic meshing is used, the discretization of the topology as represented in the optimization method is uncoupled from the discretization used in the forward solver. SIMP and level-set methods typically use a local optimization method that requires gradient information. The genetic programming method used here does not require the computation of gradients and it is well-suited for multi-objective problems as it is a population-based method. The topology representation used here can also be adapted to a local search methodology, as described in Wildman and Gazonas (2009).

Previously, microstructural (or microscale) topology optimization has been performed for different goals: maximum stiffness of a periodic material (Huang et al. 2013), prescribed macroscale constitutive parameters (Sigmund 1995, 1994; Zohdi 2002; Mei and Wang 2004), and extremal microstructural properties (Allaire and Kohn 1993; Sigmund 2000). In this paper, we focus on balancing structural stability with energy absorption in a multiobjective setting. Typical engineering problems involve balancing conflicting goals, using the present example, if we wish to design an energy absorbing material and incorporate it into an overall structure, that material may not be structurally sound and could be unusable in our final design. We could place constraints on the optimization problem, such as a mass constraint, though this approach may leave out important information that could be garnered from the Pareto front, or set of multiobjective designs (Cohon 1978; Cohon and Marks 1975; Steuer 1989). The goal of Pareto optimization is to deliver a set of designs, rather than one single design, that represent the optimal trade-offs between two or more conflicting goals. This set of designs is *Pareto optimal* in that no other design simultaneously outperforms it in all goals. (The Pareto front can be defined as the boundary between the infeasible region of designs, and the dominated region.) Each design in the Pareto optimal set (Pareto front) can only outperform another design in the Pareto optimal set by at most one goal less than the total being optimized (or one goal in a two-goal problem, two in a three-goal problem, etc.).

While an individual design on the Pareto front may be determined using constraints on one or more goals, the shape of the Pareto front can provide information as well. For example, the front's shape may indicate areas of diminishing returns, whereby minuscule improvement in one goal only comes at the severe detriment of another. Here, we will optimize for two goals: a quasi-static loading problem representing a material's structural qualities, and a dynamic loading problem to measure energy absorption. These goals are conflicting because a structural material will be massive and stiff, while an energy absorbing material tends to be soft and compliant.

Three objectives, though only ever two simultaneously, will be considered in a multi-objective setting: Static displacement under uniaxial load, dynamic energy absorption under hydrostatic, blast-like loading, and total mass. Each goal is defined and measured at the global scale, with no objectives considered at the local scale. The forward solver for the static and dynamic problems is a multiscale FEM code, with a global scale representing a sample of the material and the local scale representing a heterogeneous, repeatable (though not necessarily periodic) RVE, consisting of one or more brittle, elastic materials and void. This method is described in Souza et al. (2008), and commercially available as MultiMech (MultiMech Research & Development 2012). In our approach, the global scale is a fixed, unchanging structure, on which the loads and boundary conditions will be applied. A single design for the local scale will be used (i.e. the global scale will be homogeneous), and it will be assumed to be made of up to two materials or a single material with a void. Damage can also be modeled in the form of automatically inserted cohesive zones. In both the dynamic and quasi-static case, cohesive zones can be inserted at the local scale, representing micro-cracking and damage. In the quasi-static case, these micro-cracks can coalesce into macroscale damage, being inserted at the global scale with an extended finite element method (XFEM) (Chessa et al. 2002). We will consider the different models in turn, showing the difference in results when incorporating damage in the form of cohesive zones.

In summary, we will use a constructive solid geometry-based topology representation in conjunction with genetic programming to optimize the local structure of a material. Loads are applied and objectives measured at the global scale in a multiscale FEM setting. A multiobjective design approach is used, with one goal being a material's static, structural response, and a second being its dynamic, energy absorbing capability. The remainder of this paper is organized as follows: Section 2 details the topology representation and GP optimization method. Section 3 then discusses the multiscale finite element method used as the forward problem to model energy absorbing materials. Section 4 then presents results of the optimization, and Section 5 discusses our conclusions.

## 2 Optimization method

The optimization of geometry/topology is complicated by the difficulty in representing topology numerically. Here, we use a combinatorial approach based on constructive solid geometry applied to convex polygon primitives. This representation forms a tree data structure, with Boolean operations as function nodes (operators) and convex polygons as terminal nodes (operands), and is most naturally optimized with genetic programming (GP). In this section, we detail the topological data structure (Subsection 2.1) and the optimization method (Subsection 2.2).

### 2.1 Topology representation

Our approach to topology optimization is based on a CSG representation (Requicha and Voelker 1977), in which complex topologies are generated by combining shape primitives using Boolean operations. Over the past decade, this type of approach has been used for several applications, including optimization of truss joints (Hamza and Saitou 2004), optimization of statically loaded beams (Ahmed et al. 2013), optimization of phononic bandgap structures (Wildman and Gazonas 2011), gravitational inversion (Wildman and Gazonas 2009), and imaging (Wildman and Weile 2007, 2008, 2010; Yamagiwa et al. 2010). Our approach differs in that the shape primitives are based on convex polygons or polyhedra, rather than canonical shapes such as rectangles and ellipses (or cubes, spheres, and cylinders in 3D). Subsection 2.1.1 discusses the use of convex polygons as shape primitives, and subsequently Subsection 2.1.2 discusses generating more complex topologies.

#### *2.1.1 Shape primitives*

Typically, CSG methods use a canonical set of shape primitives to generate a desired topology. While, as shown in previous work, this approach can be effective, we use a more flexible representation that can result in smaller tree sizes. Here, shape primitives are convex polygons represented by the convex hull of arbitrary-length lists of points. The convex hull can be defined as the intersection of all half-planes that contain the points, or more colloquially, as the shape that results from stretching an elastic membrane around the points. Figure 1 gives an example of the convex hull (dashed line) of a set of randomly generated points in the plane (black dots). The total number of points allowed in each convex polygon is not restricted, so that curved shapes

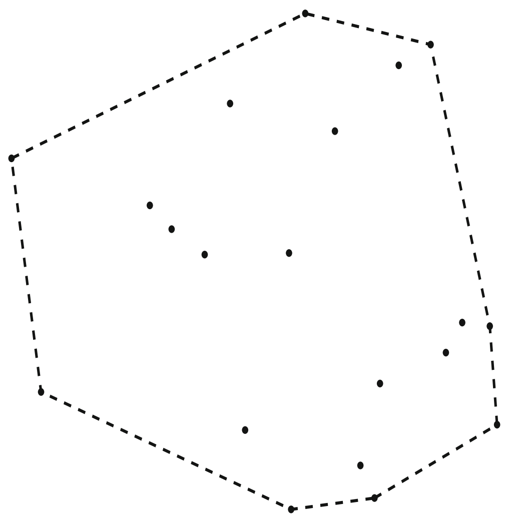


**Fig. 1** An example of the convex hull of a set of random points

can be well-approximated by small line segments, while retaining the ability to match sharp corners where necessary.

In CSG methods with canonical shape primitives, each shape may have a few defining parameters such as center location, bounding box size, or rotation. These may or may not be compatible between varying shapes making hybridization or crossover difficult with a GA approach. Here, each shape is described as a list of points, and though its length is arbitrary, we can easily perform crossover as the data has identical meaning across chromosomes. While this convex hull approach is capable of generating a wide variety of shapes, it is incomplete as it is incapable of generating concave shapes and varying topologies; the methodology for deriving concave shapes from convex hulls using Boolean operations is described in Subsection 2.1.2.

### *2.1.2 Constructive geometry*

The convex shape primitives described in the previous subsection can be combined to generate more complex geometries and topologies in a variety of ways. Boolean operations can be used in applications with a single material, such as the microwave imaging of perfect conductors (Wildman and Weile 2007). For problems with more than one material, an overlapping scheme can be used, in which each convex polygon also contains material properties and a priority value designating which operand is placed on top (Wildman and Weile 2010). Here, we will use a scheme similar to the overlapping scheme of (Wildman and Weile 2010), but somewhat simplified by removing priority values from the terminal nodes.

The construction scheme used here is rather simple: Given a binary tree with terminal nodes containing point lists and material properties, at each function node, always place the topology from the left operand on top of the topology from the right operand. This is somewhat equivalent to using only union functions, but assigning each terminal node a sign (essentially the material properties), so that subtraction can be generated with two oppositely signed operands (equivalent to differing material properties). While simple, this approach will generate any physically realizable topology including shapes with holes and multiple disjoint shapes. Consider a few examples, each assuming a two material system embedded in an infinite medium consisting of one of those materials.

First, a concave shape can be generated by overlapping two convex shapes of the same material, essentially resulting in the union of the two shapes. Figure 2 shows a three node binary tree representing the overlap of the convex polygon $C_1$ on convex polygon $C_2$. The two (randomly generated) polygons are shown in Fig. 3, with $C_1$ represented as the solid line resulting from the convex hull of the points marked as circles, and $C_2$ being the dashed line resulting

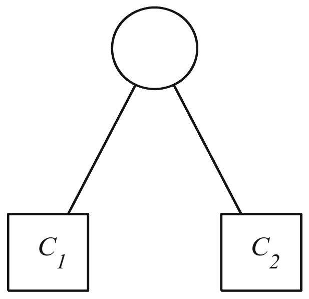


**Fig. 2** A three node binary tree

from the points shown as squares. The result of applying the tree of Fig. 2 to the convex polygons of Fig. 3 is shown in Fig. 4. Disjoint topologies can be generated if $C_1$ and $C_2$ do not overlap, thus generating two convex polygons in the plane.

Next, a topology with a hole can be generated if a shape contained entirely within another and of a different material is overlapped. Say $C_1$ is contained within shape $C_2$ and is made of the same material as the background medium, as shown in Fig. 5. Generating polygons from the convex hulls and overlapping using the tree shown in Fig. 2, the topology in Fig. 6 is generated. These three basic operations described can then be combined with more complex tree structures, generating more complex topologies.

The computational geometric operations used to evaluate a CSG tree are available in the Computational Geometry Algorithms Library (CGAL) (CGAL 2007) as Boolean operations on Nef polyhedra (Bieri 1995). A Nef polyhedron is a polygon or polyhedron that is generated by Boolean operations on half-spaces, which may be open or closed; i.e. they may or may not be inclusive of the defining boundary of the half-space. Nef polyhedra may then have

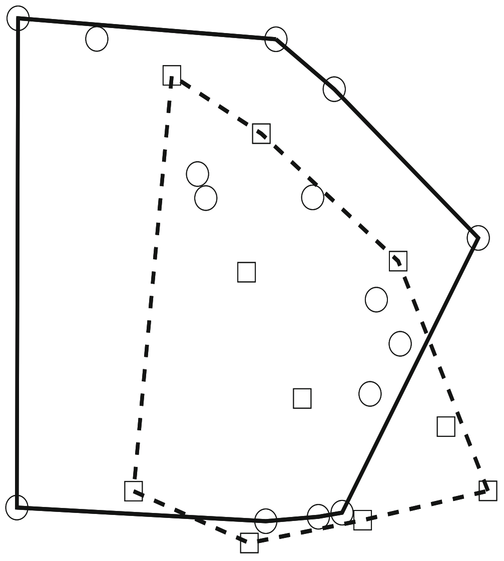


**Fig. 3** Two randomly generated sets of points and their convex hulls

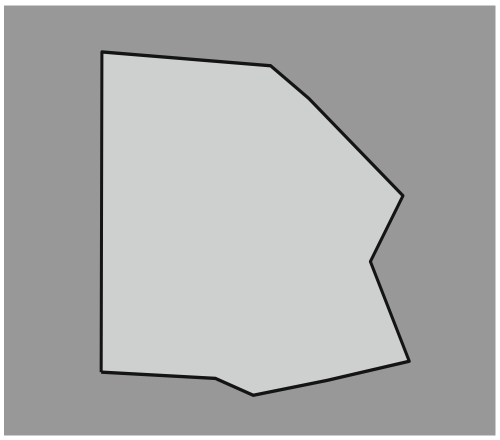


**Fig. 4** A concave polygon result generated by applying the tree operation in Fig. 2 to the convex polygons in Fig. 3

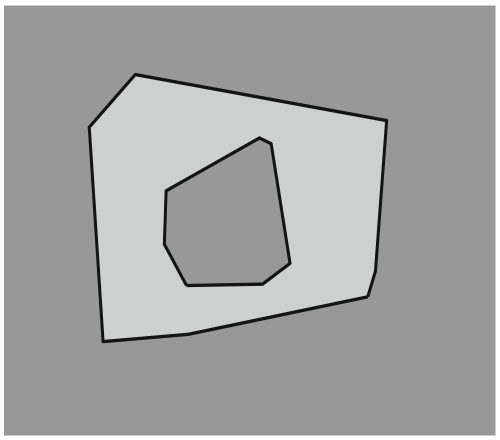


**Fig. 6** A hole topology result generated by applying the tree operation in Fig. 2 to the convex polygons in Fig. 5

infinite extents and be open or closed on their boundaries. While there is no specific "overlap" function in CGAL, we can generate one using Boolean operations. Consider a binary overlap function, with Nef polyhedra as operands in general. The first step in computing an overlap function is to compute a boundary inclusive "union mask" of the right operand. This union mask is simply the union of all underlying operands, ignoring material properties and including the defining boundaries of the topology. Next, we subtract the union mask from the left operand with its boundary excluded. Finally, the union of the result of the subtraction operation and the right operand (with boundary excluded) is computed to generate the final topology. Essentially, our binary overlap operator, must include the boundary inclusive union mask as a third parameter, which is updated at each node and passed up the tree during computation.

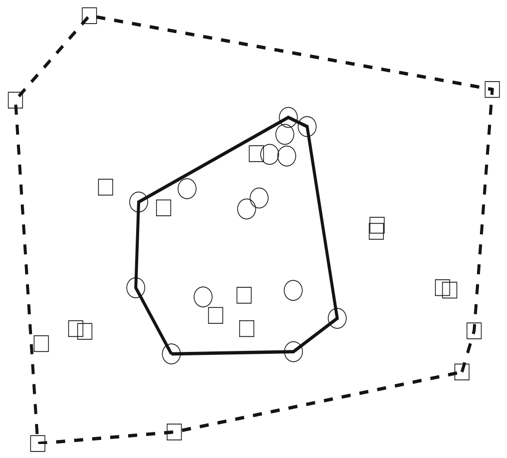


**Fig. 5** Two randomly generated sets of points and their convex hulls

## 2.2 Genetic programming

Genetic programming (Koza 1999) is a variation of a GA, whereby the chromosome structure is tree-shaped, rather than linear with fixed length (Goldberg 1987). It is functionally equivalent to a standard GA because it uses the same three genetic operators—selection, crossover, and mutation—applied iteratively to a population of chromosomes. Because the chromosome is tree shaped, crossover is the same in spirit, though issues such as tree bloat (Banzhaf et al. 1999) preclude a naïve implementation. Due to the flexibility in the chromosome, mutation has several more options than in standard GA. The following subsections discuss the implementation of our topology scheme in GP in more detail.

### *2.2.1 Chromosome structure*

The chromosome in a GA is a data structure that encodes a potential solution to the optimization problem. In a standard GA with a fixed-length, linear chromosome, the position on the chromosome of each "gene" or value in the chromosome has a specific meaning. In other words, if our optimization problem were to find the optimal rectangular solid for a given forward problem, we could parameterize the shape as shown in Table 1, where $h$ is the height of the unrotated rectangle, $w$ is its width, $\theta$ is a rotation angle, and $x_0$ and $y_0$ are its center location.

**Table 1** A chromosome with five genes representing a parameterization of a rectangle

| $g_1$ | $g_2$ | $g_3$ | $g_4$ | $g_5$ |
|---|---|---|---|---|
| $h$ | $w$ | $\theta$ | $x_0$ | $y_0$ |

Each gene has a specific meaning pertaining to some aspect of the rectangle and of course hybridizing $g_1$, the height with $g_3$, the rotation angle, would be meaningless.

In GP, chromosomes are more flexible, and as such, specific data or functions may have to be labeled. For example, if we use a more general binary tree that includes different operations, we would have to label each function node with its function: union, subtraction, etc. As with a standard GA, hybridizing genes with different meanings may not aid in optimization. In order to keep crossover meaningful, we attempt to simplify the data structure as much as possible, so that each function node is identical (i.e. overlap the shape on the left), and each terminal node contains a list of points and material properties. Material properties can either be arbitrary with a given range, or chosen from a database. A complete chromosome is then described by two data structures, a tree giving the constructive geometry, and point lists and material properties for each terminal node of the tree. Table 2 gives an example of the data structure of two terminal nodes that might correspond to the tree shown in Fig. 2.

### 2.2.2 Selection

GAs have many options for selection. Typical single objective methods include tournament selection, roulette wheel selection, or truncation selection (Goldberg 1987). In this work, we will use selection appropriate for multiobjective problems. GAs are amenable to multiobjective problems because they use a population of potential solutions. Rather than a single solution, we seek a set of solutions corresponding to the optimal tradeoffs between two conflicting objectives. Consequently, to convert a standard GA to a multiobjective GA, the only change necessary is to the selection operator. The most popular multiobjective selection method is known as non-dominated sorting (Srinivas and Deb 1994; Deb et al. 2002), in which the population is ranked based on its performance relative to the current Pareto front.

Non-dominated sorting proceeds as follows: First, the Pareto front is determined by finding all non-dominated chromosomes. The set of dominated chromosomes (for minimization) $Y$ can be defined as

$$Y = \left\{ \mathbf{y} | \exists \mathbf{y}^* \wedge \forall i,\ y_i^* \leq y_i \wedge \exists j,\ y_j^* < y_j \right\}, \tag{1}$$

where $\mathbf{y} = [y_1, y_2, ...]$ is a goal vector or vector of objective function values. The non-dominated set is then just the complement of the dominated set. In other words, the dominated set is simply the set of chromosomes with objective function values that are clearly worse than at least one other chromosome in the population; for each dominated chromosome, there exists another with all objectives less than or equal, and at least one objective strictly less. The chromosomes in the non-dominated set cannot be compared within their set as each will have at least one better performing goal.

With the definition of non-domination in hand, we can now assign a single objective function value to each chromosome, beginning by ranking each in terms of relative distance to the Pareto front. The first non-dominated set of chromosomes is given a rank of one, and temporarily removed from the population. The non-dominated chromosomes of the remaining set are determined and assigned of rank two. After removal, this process is repeated until each chromosome in the population has a rank. An example of this procedure is shown in Fig. 7, with two goals being minimized and four total ranks.

A chromosome's rank gives a measure of a chromosome's relative distance to the Pareto front, however, it can be insufficient as an objective function value for selection as it will not encourage movement along the front. After all, we want a set of designs that well-represents the Pareto front. To encourage a diverse front, we add a sharing value that penalizes chromosomes that are close to each other in goal

**Table 2** Example data structure of a chromosome with two terminal nodes

| $C_1$ | | $C_2$ | |
|---|---|---|---|
| Mat ID | | Mat ID | |
| $k$ | | $l$ | |
| $x$ | $y$ | $x$ | $y$ |
| $x_0$ | $y_0$ | $u_0$ | $v_0$ |
| $x_1$ | $y_1$ | $u_1$ | $v_1$ |
| $x_2$ | $y_2$ | $u_2$ | $v_2$ |
| $x_3$ | $y_3$ | $u_3$ | $v_3$ |
| | | $u_4$ | $v_4$ |
| | | $u_5$ | $v_5$ |

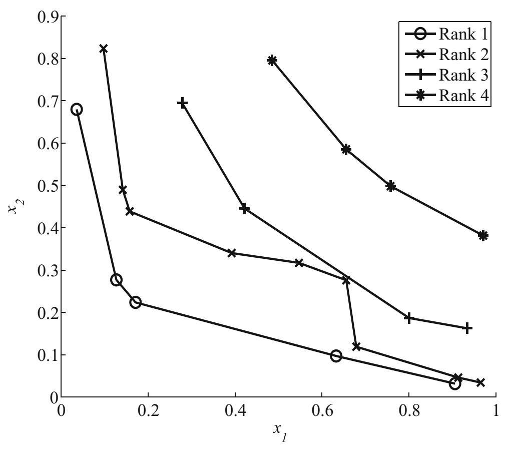


**Fig. 7** An example of goal vectors assigned with Pareto ranks

space. Each chromosome $i$ with goal vector $\mathbf{x}_i$ is assigned a sharing value by first finding all other chromosomes within a radius $r$ in goal space. The sharing value for chromosome $i$ is then given by

$$s_i = N_i - \sum_{j=1}^{N_i} \frac{\left\| \mathbf{x}_i - \mathbf{x}_j \right\|}{r}, \tag{2}$$

where $\mathbf{x}_j$ is the goal vector of chromosome $j$ within sharing radius $r$ of chromosome $i$ and $N_i$ is the number of chromosomes within the sharing radius. This approach is described in more detail in Weile et al. (1996).

Finally, a single objective function value that will be maximized can be assigned to each chromosome using its Pareto rank and sharing value. First, all rank one chromosomes are given an objective function value of $f_i^1 = 1/s_i$. Next, the minimum objective function value of the rank one chromosomes is used as the starting value for the rank two chromosomes, i.e. now, for rank two, $f_i^2 = \phi_1/s_i$, where $\phi_1 = \min_i f_i^1$. This process is repeated for rank three and so on. After all chromosomes are assigned an objective function value, standard roulette wheel selection is used.

### 2.2.3 Crossover

After selection, a new population is generated with an average fitness better than the previous generation. Crossover is then used to combine traits of those surviving chromosomes to hopefully generate even better performing chromosomes. In a standard GA, crossover is typically performed by first randomly choosing two chromosomes from the new population. A gene, say $g_3$ in Table 1, and hybridized in some way. The genes following ($g_4$ and $g_5$) are then swapped between the two chromosomes.

In GP, crossover is performed in a similar way: Two nodes in a tree are chosen at random and hybridized if applicable, and their subtrees are swapped. Given that our function nodes are identical, there is no hybridization to be done, however, if two terminal nodes are chosen, their point lists and material properties can be hybridized. As the point lists are of arbitrary length, they can grow without bound in some situations. This is related to the issue of tree bloat (Banzhaf et al. 1999), in which tree sizes grow without bound as protection against harmful mutations and crossover. Larger trees (or point lists) can contain more redundant information, so that crossovers and mutations with the potential to harm performance are minimized. This can lead to population stagnation, so it is undesirable. For point lists, we then use a two point crossover, ensuring that the number of points exchanged between two terminal nodes is equal so that each point list remains the same length. An example of this process is shown in Table 3: The double-horizontal lines and text in bold in the point lists show the two crossover points, and the coordinates $x_i$, $y_i$ and $u_i$, $v_i$ will be swapped between nodes $C$ and $D$. The crossover points are chosen at random. Additionally, the points at the crossover locations are hybridized, i.e. mixed together by choosing a number $t$ from a uniform random distribution between 0 and 1 as

$$\begin{aligned} \mathbf{p}^* &= t\mathbf{p} + (1-t)\,\mathbf{q} \\ \mathbf{q}^* &= t\mathbf{q} + (1-t)\,\mathbf{p}, \end{aligned} \tag{3}$$

where $\mathbf{p}$ and $\mathbf{q}$ are the points at the crossover locations. Finally, because we are using a database of material properties, we do not hybridize materials, although this is not a restriction, as randomly generated constitutive parameters may benefit from hybridization.

**Table 3** Example of crossover between two terminal nodes

| *C* | | *D* | |
|---|---|---|---|
| Mat ID | | Mat ID | |
| *k* | | *l* | |
| *x* | *y* | *x* | *y* |
| $x_0$ | $y_0$ | $u_0$ | $v_0$ |
| $\boldsymbol{u_1}$ | $\boldsymbol{v_1}$ | $\boldsymbol{x_1}$ | $\boldsymbol{y_1}$ |
| $\boldsymbol{u_2}$ | $\boldsymbol{v_2}$ | $\boldsymbol{x_2}$ | $\boldsymbol{y_2}$ |
| $\boldsymbol{u_3}$ | $\boldsymbol{v_3}$ | $\boldsymbol{x_3}$ | $\boldsymbol{y_3}$ |
| $x_4$ | $y_4$ | $u_4$ | $v_4$ |
| | | $u_5$ | $v_5$ |

To combat tree bloat, we use a crossover probability based on geometric similarity. Typically, the crossover probability is a constant around 80 %, and subtrees are chosen at random between two random chromosomes. Our implementation sets a crossover probability by first selecting a chromosome for crossover. A mate is chosen out of a pool of random chromosomes, the size of which is an input parameter, typically chosen as ten. Each chromosome in the mating pool is assigned a probability by randomly choosing a given number of subtrees (again, usually chosen to be at most ten) in each and comparing their decoded geometries as

$$p_c = \left[ \frac{\Delta\left(C_1 \bigcap C_2\right)}{\Delta\left(C_1 \bigcup C_2\right)} \right]^s, \tag{4}$$

where $\Delta\left(C_1\right)$ indicates the area of the topology generated from subtree $C_1$ and $s$ is a biasing exponent typically chosen as 2.5. Consequently, if $C_1$ does not overlap $C_2$, then $p_c$ is zero, and if $C_1 = C_2$ then $p_c$ is one. Now, $p_c$ is computed for a given number of randomly chosen subtrees of a mating pair, and the maximum is saved. After the maximum $p_c$ is found for each mate in the pool, a mate is chosen at random using a weighted roulette wheel with the weights given as the crossover probabilities. In other words, we choose a random mate from the pool, favoring mates with higher crossover probabilities. Crossover is then performed using the chosen mating pair and subtrees with the maximum $p_c$.

While this scheme is somewhat convoluted, it is designed to match similar pairs of subtrees, but with some randomness. Matching similar subtrees helps ensure population convergence and randomness aids diversity. Striking a balance between the two is important in the design of a GA, so that it can cover a large search space and resolve fine details simultaneously. These issues are not present with a fixed length chromosome representation as we know *a priori* the function of each gene in a chromosome, thus ensuring that crossover is meaningful.

### *2.2.4 Mutation*

The final genetic operator, mutation, is used to inject new genetic information into a population, preventing premature convergence or stagnation. In our implementation, the flexibility in the chromosome allows for a large number of options for mutation. We can, on one hand, manipulate solely the encoded chromosome (tree structure with point lists at the terminal nodes) and on the other hand, adjust the decoded topology. The mutation rates for the examples in Section 4 are constant throughout, so they are listed here.

First, single points from terminal node point lists are deleted with a given probability. This rate can be set somewhat high (for mutation rates) as terminal nodes tend to grow in length for reasons discussed earlier. Here we use a point deletion rate of 2 % per point. There is a minimum length for terminal nodes of three, as three points are required for a 2D simplex. We also duplicate points in a terminal node with a 0.5 % rate per point. New points are first duplicated then shifted by a random amount according to a Gaussian distribution with standard deviation of 10 % of the largest dimension of the design bounds (20 mm in all examples from Section 4).

Points are also added by splitting existing convex hull segments. First, a terminal node is decoded by taking its convex hull. Next, for each line segment in the hull, we randomly insert a new point at a random location between 10 % and 90 % of its length. This mutation is performed with a rate of 0.5 % per segment.

In a mutation most analogous to a standard GA, points are shifted by a random amount. The shift amount is chosen from a Gaussian distribution with zero mean and standard deviation of 10 % of the region size (as above for point duplication), and the mutation rate is 1 % per point.

One issue with the above point mutation method is that a non-coding point (a point not on a convex hull) can be moved to another non-coding position, meaning that ultimately the mutation had no effect on the decoded topology. A convex hull aware point mutation was designed to remedy this situation. Given a point $\mathbf{p}$ inside a convex hull, the closest segment on the hull is found. Two vectors are formed pointing from the point inside the hull, to the two points $\mathbf{q}_1$ and $\mathbf{q}_2$ defining the closest segment. The point inside the hull is then moved inside the parallelogram formed by the addition of these vectors as

$$\mathbf{p}^* = r_1 (\mathbf{q}_1 - \mathbf{p}) + r_2 (\mathbf{q}_2 - \mathbf{p}) + \mathbf{p}, \tag{5}$$

where $r_1$ and $r_2$ are two random values from a uniform distribution between 0 and 1. This mutation then has a 50 % probability of altering the convex hull and is used with an overall rate of 1 % per point.

Points are also deleted by pruning non-coding points that are inside a point set's convex hull. As an extreme example, a point list may contain hundreds of points, but only three actually on the convex hull that defines its decoded shape. These redundant points can be harmful to a population's progress, and so they can be pruned. Points to be pruned are chosen by first computing the convex hull of the point list and scaling it by a random amount between 0.4 and 0.8. All points within this scaled convex polygon are removed from the point list. This mutation is performed at a rate of 5 % per terminal node.

We can also apply affine transformations to the terminal node point lists, effectively altering the overall geometry. Here, we separately apply scaling, rotation, or translation mutations with a rate of 1 % per terminal node. If a node is chosen for affine transformation, scaling, rotation, or translation is chosen at random with equal probability. Each transformation has its defining parameter chosen from a Gaussian distribution: Scaling uses a mean value of 1 and a standard deviation of 0.1, rotation uses a mean of 0 and standard deviation of $\pi/10$, and translation uses a mean of zero and standard deviation of 10 % of the region size for both coordinates.

Material properties can also be mutated. Here we use a database of materials and so, with a given probability of 1 % per terminal node, we simply choose a new material at random from the database.

Finally, we can alter the tree structure of a chromosome. The first type of tree mutation is standard in GP, subtree deletion and regrowth. In this mutation, a subtree is deleted and replaced with a randomly generated subtree with a given probability. This type of mutation can be destructive so it is applied with a low probability of 0.5 % to each node in a tree. Another way of altering the tree structure is to split terminal nodes into a function node with two new terminal nodes. There are a few ways of accomplishing this, first, we can simply split a terminal node's point list at an arbitrary point and separate it into two new nodes. Next, we can separate points internal to a list's convex hull, much like the pruning operation described above. Here, we again scale down the convex hull of a point list, and separate points

**Table 4** Summary of mutation rates

| Type | Rate | Unit |
|---|---|---|
| Point deletion | 2 % | Point |
| Point duplication | 0.5 % | Point |
| Segment splitting | 0.5 % | Segment |
| Point translation | 1 % | Point |
| Convex hull mutation | 1 % | Point |
| Point pruning | 5 % | Terminal node |
| Affine transformation | 1 % | Terminal node |
| Material mutation | 1 % | Terminal node |
| Delete node | 0.5 % | Node |
| Split list | 1 % | Terminal node |
| Split hole | 1 % | Terminal node |
| Split line | 1 % | Terminal node |
| Aggregate lists | 6 % | Function node |

inside the scaled hull and those outside into two new terminal nodes. Finally, we can split a set of points by placing a line through the polygon and separating all points (geometrically) left of the line into one new list and points right into another. The line is chosen by passing an infinite line through two randomly generated points within the list's convex hull. The intersection points of this line and the convex hull are computed and inserted into each new terminal node to ensure that the decoded subtree is similar to the original. Each of these splitting mutations are performed with a rate of 1 % per terminal node. As these mutations can lead to large tree sizes, subtree point list aggregation is also performed in opposition. In this mutation, a function node is chosen and all point lists in the terminal nodes within the subtree are aggregated into a single terminal node. This is performed at a rate of 6 % per function node.

The types and rates of mutation are summarized in Table 4. The number and complexity of the mutation operators is due to the flexibility in the topology representation, and some work remains in determining the effectiveness and usefulness of each individual operator.

## 3 Multiscale forward problem

The goal of this work is to optimize a material's local-scale structure. One approach would be to construct a model of a material by repeating a given RVE over a finite region and use a very fine mesh; however, this may lead to overly long execution times due to the fine mesh. Instead, we will use the concurrent multiscale finite element method Multi-Mech, which uses RVEs to represent a material at a local scale (Souza et al. 2008; Souza and Allen 2010, 2013). We assume that our global scale structure is statistically homogeneous, but has inhomogeneous local structure, which will be optimized. We will consider static loading problems to address structural stability along with dynamic problems for energy dissipation.

### 3.1 Methodology

Typically, a finite element problem will have the material properties of each element as an input. In a multiscale FEM problem, the global scale only assumes that the material properties over an element are statistically homogeneous, and they are then derived from local scale representative volume elements (RVEs). Each integration node in the global mesh has an associated RVE, each of which is itself a finite element problem, with loads derived from the global mesh. These RVE problems are quasi-static with spatially homogenous (linear displacement in this case) boundary conditions. Material properties are then garnered from the RVE problems and used to solve the global problem. The purpose of this work is to then optimize the topology of the material at the RVE scale, assuming a homogenous set of RVEs, though the model RVE itself is not homogeneous. Figure 8 gives an example of a multiscale problem: A global mesh is shown with points representing integration nodes of a one-point rule. Each integration node then has an RVE attached, the geometry of which is shown in the inset. The global geometry is assumed to have the same RVE geometry at each element, and the RVE itself is a bi-material system, with one material shown in black and one in white in this example; however, this methodology is easily generalized by allowing RVEs to vary from point to point for solution of a globally inhomogeneous optimization problem. The RVE is shown to scale; in this case, it is one tenth the size of the global mesh.

Each RVE is an independent finite element problem and so they can be solved simultaneously. MultiMech can take

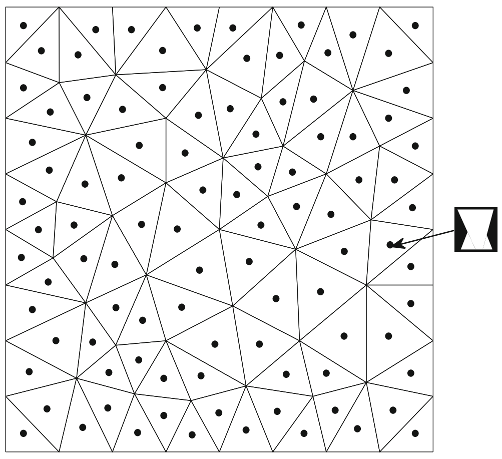


**Fig. 8** An example of a global mesh with embedded RVEs

advantage of this and parallelize the computation of the individual RVE FEM problems. We can also parallelize the GA, leading to a parallel problem that requires a large number of compute nodes. Consider a GA running 150 compute nodes in parallel, if we then want to use 16 nodes for each multiscale FEM forward problem, we need 2,400 compute nodes for this problem.

#### *3.1.1 Damage modeling*

Damage is modeled in this FEM code using cohesive zones (Xu and Needleman 1994, 1996) at the local scale and XFEM at the gobal scale. At the local scale, cohesive zones are automatically inserted between elements after the traction has exceeded a given value. If a number of cohesive zones coalesce within the RVE, the damage is propagated up to the global scale. In this case, extended finite elements (Chessa et al. 2002) are inserted in the global scale finite element mesh. Currently in MultiMech, XFEM together with multiscale RVEs are only available for static loading, so they are not used in the dynamic problem under consideration. One unfortunate side effect of this approach is that inserting cohesive zones adds elements to the mesh, increasing the number of unknowns. A design that is badly damaged can lead to long run times, so it is important to parallelize both the GA and the forward problem. Finally, triangular elements are used at the global scale throughout, rather than the more standard quadrilateral elements, because the XFEM algorithm in MultiMech is implemented for triangular elements.

#### *3.1.2 Meshing*

FEM necessitates the use of a mesh for each design. The global scale can be meshed once and used for each forward evaluation, but each new chromosome must be meshed before it can be used as an RVE. As described in Subsection 2.1.2, the topology is decoded using Boolean operations on Nef polyhedra using CGAL (CGAL 2007), which must be performed with exact arithmetic using rational numbers. We use the meshing algorithm from CGAL, though some care must be taken: CGAL's meshing utilities use inexact arithmetic with floating point numbers, so there can be slight errors on the order of machine precision in converting from exact to inexact numbers. CGAL's meshing algorithm uses a constrained Delaunay triangulation, requiring the input of constraints. After decoding a topology, constraints are simply the edges and vertices of the resulting polygons; however, if a vertex should lie on an edge in a "T"-like junction in the exact representation, but ends up slightly off due to roundoff error during the floating point conversion, the meshing algorithm may crash. Figure 9 gives an example of this issue. The vertex in the

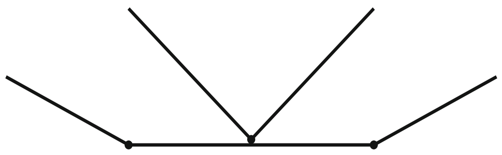


**Fig. 9** An example of a difficult meshing case

center belongs to one polygon and the flat edge on the bottom belongs to a second polygon. The meshing algorithm will attempt to place very small triangles in this area, in an attempt to maintain triangles that are close to equilateral. As the vertex approaches the edge, the number of triangles grows, and at some point the meshing algorithm will crash.

This situation is most easily handled on the exact side through some pre-processing. We iterate over all vertices and all edges and check if a vertex is "close" to an edge, but is not one of its defining vertices (i.e. a T-junction). "Close" is defined here with an input parameter, which is compared to the length of the vector connecting the vertex and its orthogonal projection onto the edge. If this length is less that the input parameter (used here as 0.01 mm), then the vertex is replaced with its orthogonal projection. In this representation the vertex is now exactly lying on the edge, and so the segment will be replaced with two segments connecting at that vertex during decoding.

As we are using a stochastic optimization method, difficult to mesh designs frequently crop up. There are situations that may lead to meshes with excessively large numbers of triangles, and so these are best discarded as designs that cannot be evaluated to save execution time.

Figure 10 gives a flow chart that summarizes the overall algorithm. The main GA loop is shown in the center, consisting of evaluation, selection, crossover, and mutation. Each stage has an exploded view on the right or left describing those steps in detail.

### 3.2 Problem setup

We will optimize for two types of loading in a multiobjective setting: Quasi-static and dynamic. The goal of the quasi-static loading problem is to develop a material for structural stability, so we will minimize displacement under a given vertical load. The goal of the dynamic loading problem is to design a material that will absorb energy, so we will maximize strain energy. The two types of problems are described in the subsequent subsections.

#### *3.2.1 Static loading*

The quasi-static problem is configured as shown in Fig. 11. We apply a vertical displacement to a 200 mm-by-200 mm block of material that has fixed displacement in both the $x$

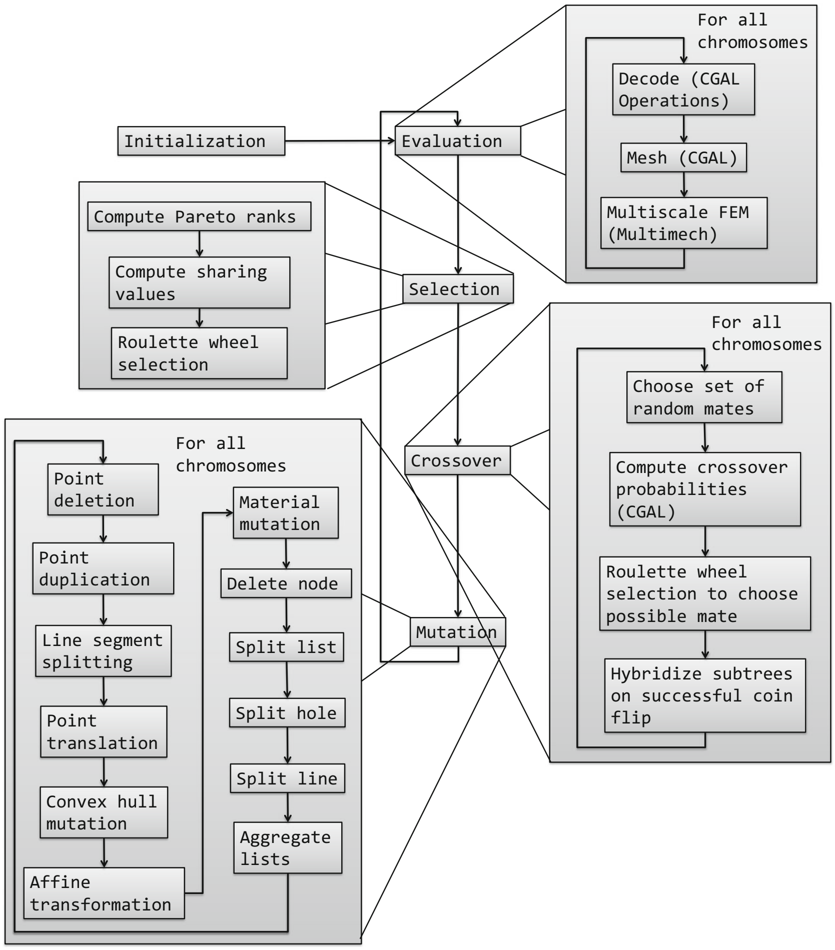


**Fig. 10** Algorithm flow chart

and $y$ directions along the $y = 0$ edge. The size of the RVE is 20 mm-by-20 mm and the problem is plane stress. We apply a load in the $-\hat{\mathbf{y}}$ direction with a value of 2 Pa along the top edge of the structure. The goal of the optimization problem is to minimize the displacement along the loading area, so the objective function is simply the sum of the displacement along the top edge:

$$\mathbf{D} = \int_{\Gamma} \mathbf{u}(\mathbf{x}, T)\, d\mathbf{x}, \tag{6}$$

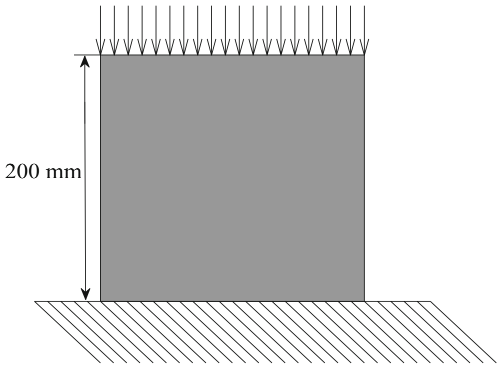


**Fig. 11** Static loading problem

where $\mathbf{u}$ is the displacement, $T$ is the time at the final time step, and $\Gamma$ is the boundary where loading is applied at the global scale. Without the inclusion of damage, the problem is linear so the magnitude of the load is not important because it will be normalized in the objective function. With damage, we use total time of 0.1 s with 200 steps in a quasi-static solver. Another goal used in the static loading case will be total mass of a local scale RVE:

$$m = \int_{\Omega_l} \rho(\mathbf{x})\, dA, \tag{7}$$

where $\Omega_l$ represents the local scale region, and $\rho$ is the density.

### 3.2.2 Dynamic loading

The setup of the dynamic problem (shown in Fig. 12) is similar to that of the static problem, though we apply a horizontal load along the $x = 0$ and $x = 200$mm edges as well and with a pressure-time history shown in Fig. 13. Here, we use a time step size of 1 $\mu$s over a total of 300 $\mu$s. The

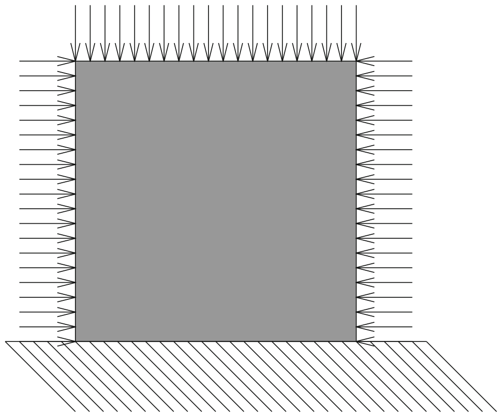


**Fig. 12** Dynamic loading problem

objective function is the strain energy at the global scale, summed over the entire run time, as

$$U = \frac{1}{2} \int_0^T \int_{\Omega_g} \sigma_{ij} (\mathbf{x}, t) \epsilon_{ij} (\mathbf{x}, t) \, dA dt, \qquad (8)$$

where $\sigma$ is the stress tensor, $\epsilon$ is the strain tensor, $\Omega_g$ is the global scale region, and Einstein summation is assumed. This objective will be maximized as we seek a design that absorbs energy. It represents a conflicting goal with the static loading problem, and so we must solve in a Pareto-optimal sense.

## 4 Results

In this section, results of the optimization are presented. Several design cases are presented, each with the same material system, (a dense, stiff linear elastic material and void), though with varying damage models. To summarize, first, in Subsection 4.1, a baseline linear elastic model at the local scale with no damage modeling is given. Next, in Subsection 4.2, we consider the same linear elastic material,

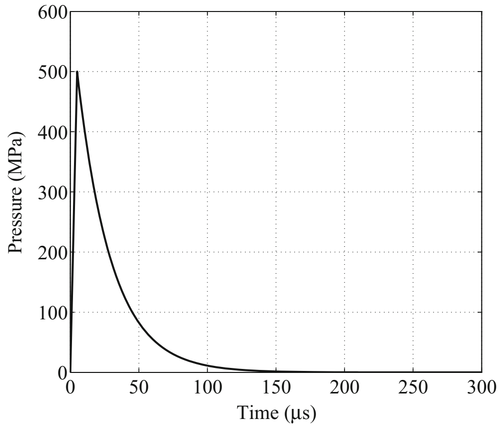


**Fig. 13** Pressure load for the dynamic problem

but allow damage in the form of cohesive zone insertion at the local scale only. Finally, in Subsection 4.3, the same local scale model (linear elastic with cohesive zone insertion) is used, but global scale damage is considered using XFEM and a multiscale localization method described below. In each case, we will use two objectives, forming a generic goal vector as:

$$\mathbf{f} = [f_1, f_2], \qquad (9)$$

with $f_1$ and $f_2$ being one of the objectives defined above, $U$, $\mathbf{D}$, or $m$.

In addition, each example was run twice (with different seeds for the random number generator) and compared to demonstrate stability of the stochastic method. As this is a multiobjective optimization that does not iteratively step through one design goal as a constraint, any comparison between optimization results must consider the curve of the approximate Pareto fronts. To that end, we define the difference in results as the integral of the difference in approximate Pareto fronts, using a linear interpolation between rank-one chromosomes, given by the piece-wise parameterization

$$\mathbf{F}(t) = (n - t + 1)\, \mathbf{f}_n + (t - n)\, \mathbf{f}_{n+1}, \; n \le t \le n + 1 \qquad (10)$$

where $n = 1, \ldots, N_1$ and $N_1$ is the number of rank-one designs. While the results from two runs may have different values of $N_1$ and different spacings between all $f_1$, (10) can nonetheless be converted from a parametric form as, by definition of Pareto optimality, it can be defined as a function. Given a suitable mapping to a function $g(x)$, a relative error measure can then be defined as

$$\text{Error}_{a,b} := \frac{\int_1^{f_{1,\max}} |g_a(x) - g_b(x)| \, dx}{\sqrt{\int_1^{f_{1,\max}} |g_a(x)| \, dx \int_1^{f_{1,\max}} |g_b(x)| \, dx}}, \qquad (11)$$

where $f_{1,\max} = \min \left( \max f_{a,1}, \max f_{b,1} \right)$ is the minimum of the maximum values of goal $f_1$ of both runs, $f_{a,1}$ is the first goal of one run and corresponds to function $g_a$, and $f_{b,1}$ and $g_b$ represent a second run. In practice, the two functions are simply discretized on an even, fine grid and the integral is computed numerically.

### 4.1 Linear elastic

The first results use a linear elastic material in the RVE, with Young's modulus 65 GPa, Poisson's ratio 0.2, and density 2235 kg/m$^3$. The remaining material was assumed to be void. The global scale geometry is given in Figs. 11 and 12, with loads and boundary conditions as discussed in Subsections 3.2.2 and 3.2.1. Here, the goal vector is given as

$$\mathbf{f} = \left[ \left| \mathbf{D} \cdot \hat{\mathbf{y}} \right|, U \right], \qquad (12)$$

and we wish to minimize $f_1$ and maximize $f_2$.

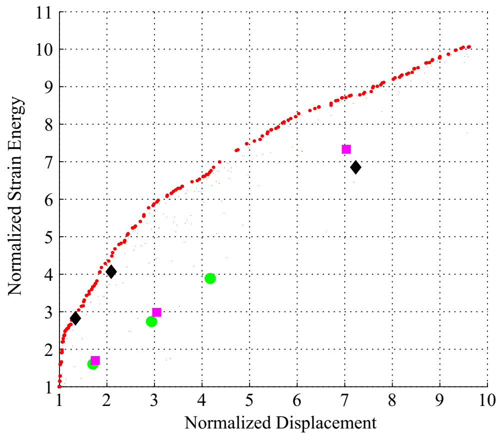


**Fig. 14** Population for the linear elastic example after 350 generations

The algorithm was run for 350 generations with a population size of 500, resulting in the population shown in Fig. 14, plotted in goal space. In this figure, the red circles indicate the approximately Pareto optimal members of the set (the rank 1 members), while the set of blue $\times$ show the suboptimal population members. The static displacement is given on the $x$-axis with the strain energy on the $y$-axis and both were normalized by the value given by a design consisting fully of the material given above (i.e. no voids). The approximate Pareto front appears to be devoid of many discontinuities and only contains a small area of large slope on the left, indicating that any choice on the front would be satisfactory if it fit a problem's constraints. Several designs from the front are shown in Fig. 15, in which the elastic material is shown as black and void is shown as white. The first is the best performing design in terms of strain energy shown in Fig. 15a. The outer frame is deliberately inserted by the algorithm, so that closed cell solutions are generated. The next design is from the center of the front (Fig. 15b), and has a normalized displacement of approximately 5 and a normalized strain energy of approximately 7.5 (in other words, the displacement is 5 times higher and the strain energy is 7.5 times higher than those of a fully dense design). It has an interesting design, with a pillar through the middle of the cell. The final two designs (the first with a normalized displacement 2.3 and a normalized strain energy 4.8 shown in Fig. 15c, and the second with displacement of nearly 1 and strain energy of 1.9 shown in Fig. 15d) show designs with two voids. Designs with normalized values of 1 are fully dense with material, so the final design shows some diminishing returns in terms of displacement, i.e. we have to give up a large amount of strain energy to get only a small amount of displacement. This type of analysis is difficult when simply applying constraints to generate a single design.

To test the significance of the results, a few canonical designs were tested and compared against the designs returned by the optimization algorithm. First, a local scale

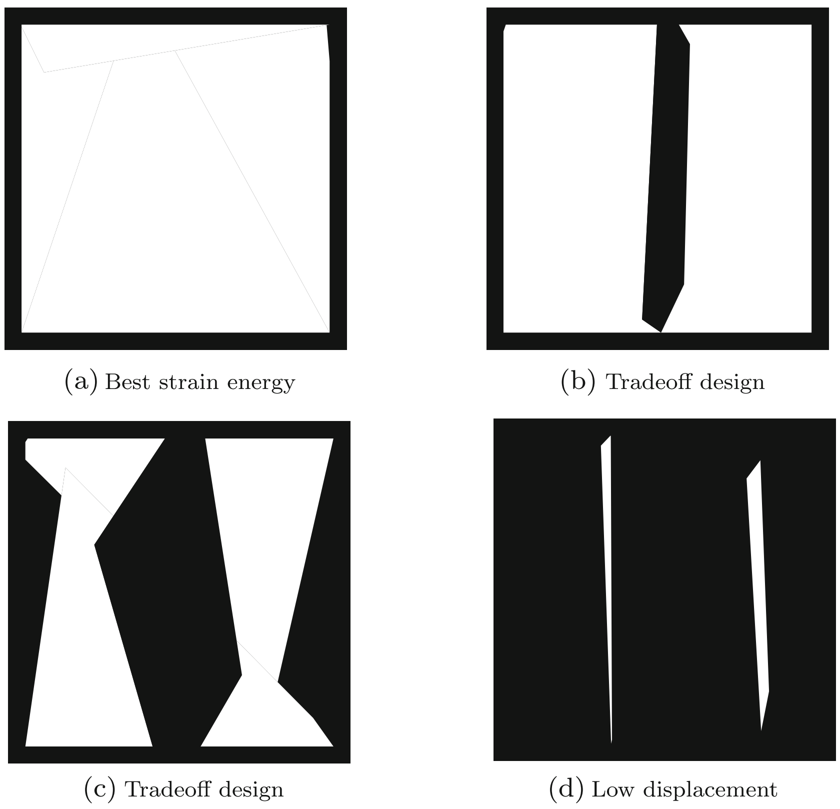


**Fig. 15** Selected designs from the approximate Pareto front

design with a centered, circular inclusion was tested for three different volume fractions, 36 %, 50 %, and 75 %, and are shown in Fig. 14 as the green circles, with the 75 % volume fraction design having the lowest relative displacement. As can be seen from the plot, the designs with circular inclusions do not approach the approximate Pareto front. Next, centered, square inclusions were tested, again with volume fractions of 25 %, 50 %, and 75 % and shown in Fig. 14 as the magenta squares. Again, these designs are not competing with the designs returned by the optimization algorithm, though the optimal design at the minimal displacement and maximal strain energy is a square inclusion due to the constraints placed on the designs. Finally, the SIMP algorithm was run on a 50-by-50 grid constrained at the same volume fractions, with the outer frame forced as solid material and the boundary condition given by the static displacement boundary condition used at the global scale. These designs were then used as the local scale RVEs and their objective function values computed. (Their contours were generated using the built-in Sobel edge detection algorithm from Matlab and the 50 % volume fraction design is shown in Fig. 16 as used in MultiMech before automatic meshing.) The SIMP designs are plotted in Fig. 14 as the black diamonds. The lowest displacement design (75 % volume fraction) lies with the rank-1 GP-generated designs, while the 50 % and 25 % designs are suboptimal in the Pareto sense. That the lower displacement design lies on the approximate Pareto front is not surprising; because low-displacement designs are mostly solid material, small changes in geometry would not produce large changes in strain energy.

To further study the results, local optimization was performed on a design from the approximate Pareto front. As we are interested in two goals, the local optimization will attempt to improve one goal, while holding the second as a constraint using a penalty in the form:

$$f = \left|\mathbf{D} \cdot \hat{\mathbf{y}}\right| + p \left|U_0 - U\right|^2 , \tag{13}$$

where $p$ is a weighting factor set to 10 and $U_0$ is the strain energy of the initial design before local optimization. The CSG description of the solution is converted to a set of linear splines to remove any redundant points, thus reducing the dimensionality of the search space. A gradient-based local search algorithm, BFGS, is used on the components of the points in the linear spline. The initial design is shown in Fig. 17 in red, with the final result shown in blue. After optimization, the normalized displacement of the design was improved to 2.34, over 2.43 of the original. Interestingly, while a small improvement was realizable, the resulting design is no more symmetric than the original, indicating a possible insensitivity to symmetry in the optimization problem.

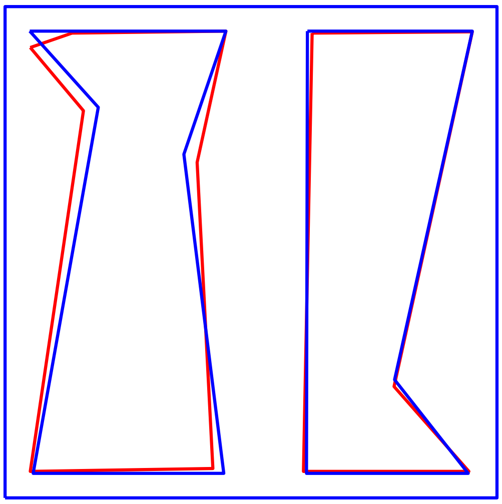


**Fig. 17** Result of local optimization

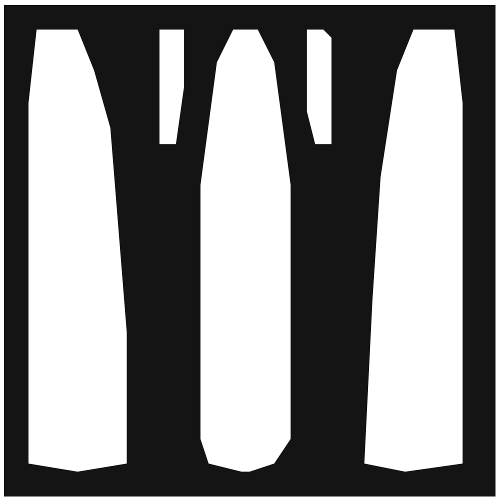


**Fig. 16** A SIMP-generated design with 50 % volume fraction

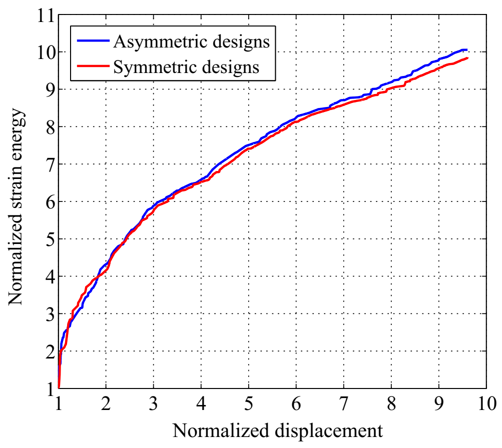


**Fig. 18** Approximate Pareto fronts for forced symmetric designs vs. asymmetric designs

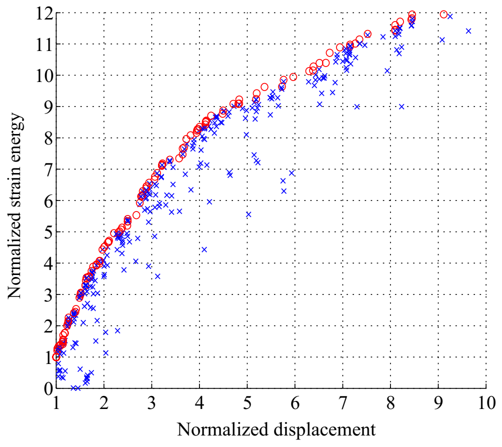


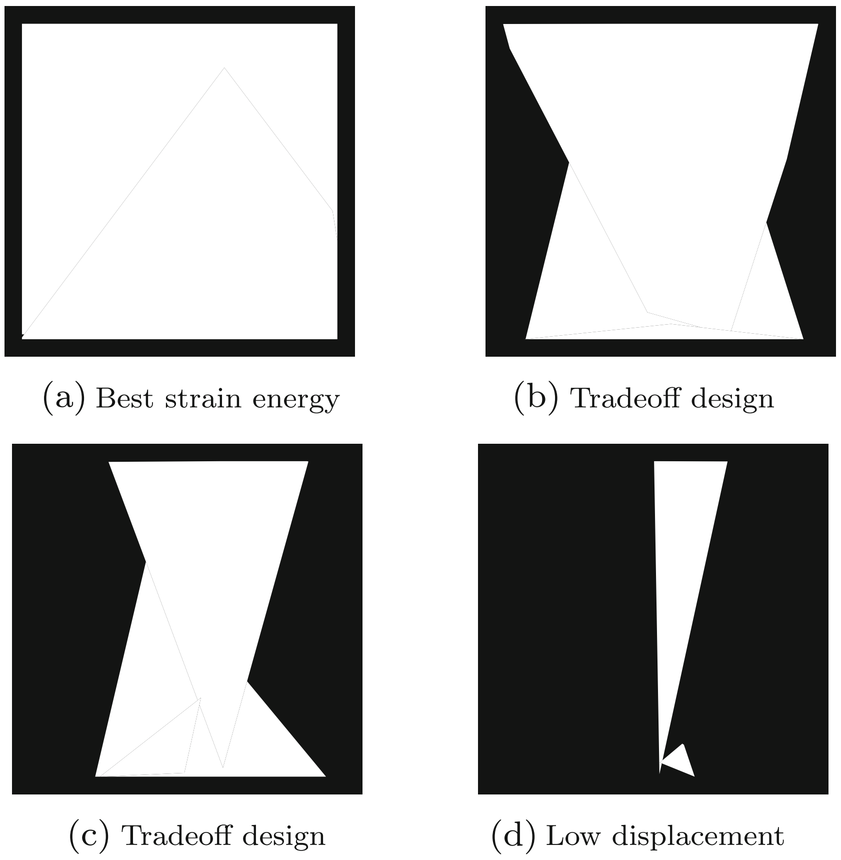
**Fig. 19** Population for the example with local scale damage after 250 generations

Finally, as the algorithm tends to generate asymmetrical designs, it was tested with symmetry enforced on the local scale RVE designs. The symmetry plane is $x = 0$ so that the left plane is simply copied and reversed to the right half plane. All dimensions remain the same, though the design domain is now of course restricted to half the original domain. The symmetric designs resulted in an approximate Pareto curve with a difference relative to the asymmetric designs of $1.9 \times 10^{-1}$. Overall, the asymmetric designs outperformed the symmetric designs, except for an area below $f_1 < 2$; the two interpolated fronts are plotted in Fig. 18.

## 4.2 Linear elastic with damage

The problem set up is identical to that of Subsection 4.1, though here we allow insertion of cohesive zone elements at the local scale. Otherwise, all parameters and goals are identical to those in Subsection 4.1, with the cohesive zone model given by the viscoelastic formulation introduced in (Allen and Searcy 2001). The critical opening distances were set to 1 mm, the maximum stresses were set to 10 MPa, and the elastic properties for the model had a Young's modulus of 1 KPa, a Poisson's ratio of 0.2, and a density of 1000 kg/m$^3$. The approximate Pareto front after 250 generations is shown in Fig. 19, with selected designs shown in Fig. 20. The design in Fig. 20a has a normalized strain energy of 11.9 and a normalized displacement of 9.1, the design in Fig. 20b has a normalized strain energy of 8.3 and a normalized displacement of 4.0, the design in Fig. 20c has a normalized strain energy of 4.5 and a normalized displacement of 2.0, and the design in

**Fig. 20** Selected designs from the approximate Pareto front shown in Fig. 19



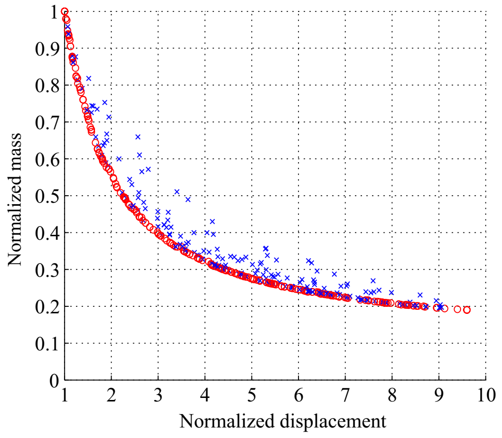


**Fig. 21** Population for the example with local scale damage and a mass goal after 500 generations

Fig. 20d has a normalized strain energy of 1.7 and a normalized displacement of 1.14. The addition of damage to the optimization problem seems to have removed designs with a thin, central pillar, rather using an hourglass-like design.

Next, the dynamic loading goal (minimization of strain energy) was changed to minimization of mass, to observe the difference in designs resulting from these two goals. The static displacement goal remains the same, again with the insertion of cohesive zones. The population size was again 500, with the Pareto optimal designs shown in Fig. 21 after 500 generations. The resulting designs are similar to those above in that they do not have any thin segments, though they lack the hour-glass shape seen in designs incorporating strain energy. For example, a design with a similar displacement to that shown in Fig. 20b is shown in Fig. 22, and has a normalized displacement of 1.72 and a normalized mass of 0.63. A second run was performed, and similar results were obtained, with an error as defined by (11) of $4.6 \times 10^{-3}$.

## 4.3 XFEM results

Finally, an example was run using XFEM at the global scale. The criterion for XFEM element insertion was that a cohesive element is transitioned to the macroscale if the acoustic tensor of the homogenized material tangent degrades to 90 % of its original value, according to

$$\det\left[Q^0_{ij}(t)\right] \leq \chi_c \det\left[Q^0_{ij}(t=0)\right], \tag{14}$$

where $Q^0_{ij}$ is the acoustic tensor at the macroscale at each time step and $\chi_c = 0.9$ is the multiscale localization criterion (Souza and Allen 2011; Nguyen et al. 2011). Here, XFEM element insertion is only available for quasi-static problems, so we use static displacement as one goal and total mass as a second, as

$$\mathbf{f} = \left[\left|\mathbf{D}\cdot\hat{\mathbf{y}}\right|, m\right]. \tag{15}$$

In this example, normal loading (as shown in Fig. 11), a population size of 500, and a total of 280 generations were used. The approximate Pareto front at the final generation is shown in Fig. 23, along with a tradeoff design from the center of the curve (with a value of 1.7 for normalized displacement, and 0.66 for normalized mass) in Fig. 24. This design can be compared with that of Fig. 22, as they have similar objective function values. As expected, for the same displacement, the design using XFEM requires a slightly higher mass, because damage at the global scale weakens the response.

The normal stress (in the $y$-direction) of a randomly chosen design (with its local scale geometry shown in Fig. 25b) is shown in Fig. 25a along with the inserted XFEM zones (as black lines) at the final time step. A second run resulted in similar results, with an error of $9.8 \times 10^{-3}$.

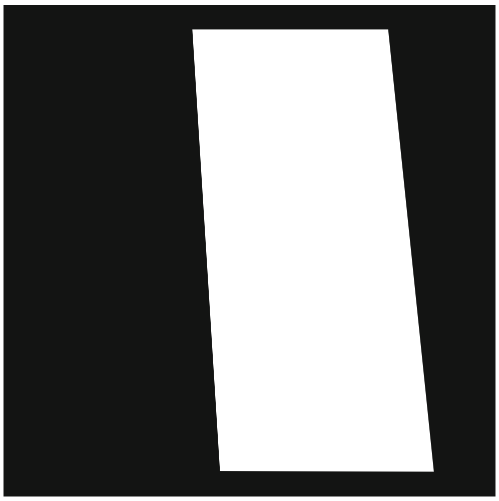


**Fig. 22** An example of a tradeoff design from Fig. 21

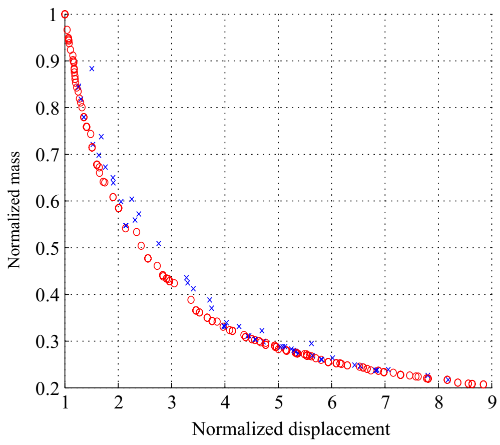


**Fig. 23** Population for the example with XFEM 280 generations

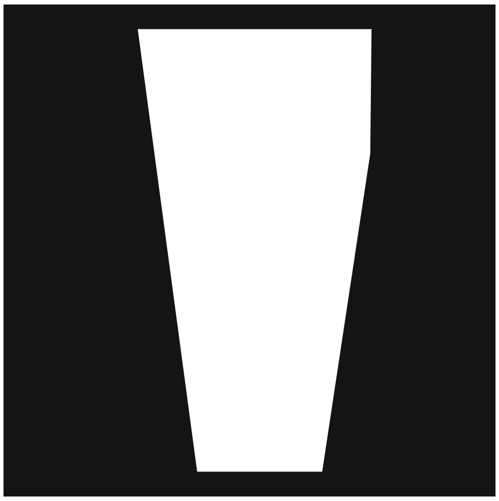


**Fig. 24** Tradeoff design

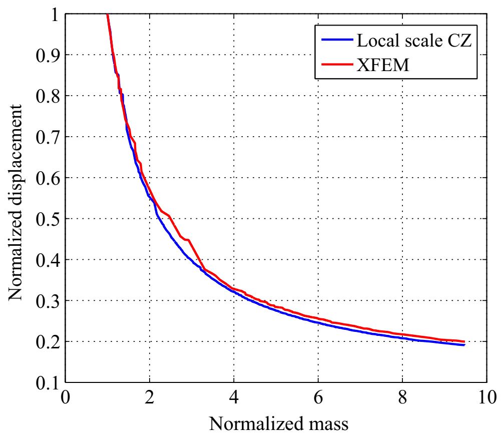


**Fig. 26** Comparison between local scale damage model and XFEM

## 4.4 Discussion

Several design cases are presented above, each producing slightly different results. The design goals always included minimization (at the global scale) of static deflection and either maximization of elastic strain energy or minimization of mass. The use of different damage models (none, local scale CZ, and global scale XFEM) was to establish a baseline design (i.e. no damage model) and compare two damage models, one more accurate but with a higher computational cost (XFEM). The purpose of replacing energy maximization with mass minimization is that they are non-conflicting goals and so it may be possible to replace energy maximization with the more computationally efficient mass minimization.

The results comprise a linear elastic case with no damage, one with damage modeled solely at the local scale in the form of cohesive zone insertion, and XFEM with damage localization at the global scale. The main difference seen between the models that include damage and the baseline is the inclusion of thin supporting members through the center of the RVE. This is sensible as any thin members may incur damage and not contribute to the improvement of static loading, while contributing to the detriment of the strain energy. The main difference seen between the cases including energy maximization or mass minimization as the second goals is the hour-glass like shape of the inclusions. A comparison of Figs. 24 and 20b or 20c reveals that designs accounting for energy maximization tend to form hour-glass structures, while those using total mass result in a more standard structure. Finally, Fig. 26 shows the difference between the Pareto curves generated by the local scale damage model and XFEM. The curves are fairly similar, with a difference of $4 \times 10^{-2}$, and, as would be expected, show that the local scale damage model gives results that are consistently better in displacement for the same mass. This figure demonstrates

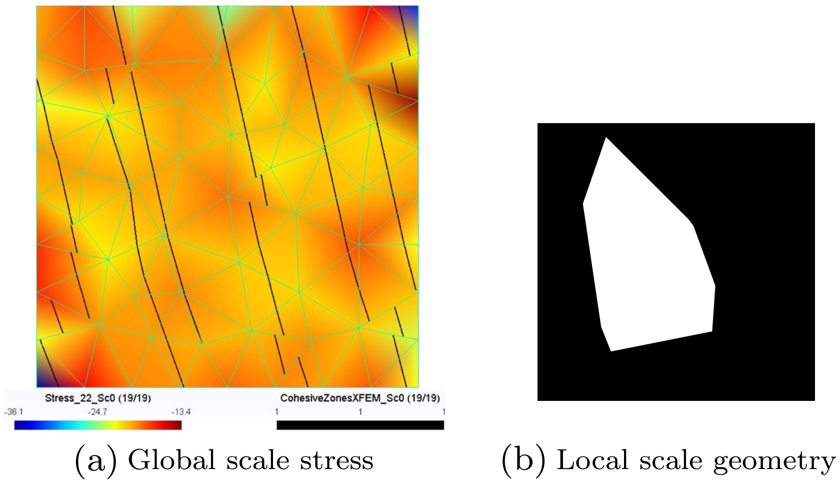


**Fig. 25** An example of the global scale stress and inserted cohesive zones

that it may be possible to forgo the use of XFEM for this design study.

## 5 Conclusions

A method for the multiobjective design of energy absorbing materials was presented that uses a GP-based topology optimization method and a multiscale FEM code. Topology was represented using a CSG-like approach, where convex polygons defined by the convex hulls of lists of points were used as primitives and combined with an overlapping function. This approach is flexible in that convex polygons with any number of sides can be combined together generating complicated topologies. Material properties are embedded in the convex polygons as well. A commercially available multiscale FEM code was used that models damage using automatic insertion of cohesive zone elements at the local scale and XFEM at the global scale. Pareto optimal designs were generated for several different cases, including quasi-static and dynamic loadings, with and without local scale cohesive zone insertion, and with and without global scale XFEM insertion.

Future work includes extending the algorithm to 3D designs. The method is easily implemented in 3D as all concepts–convex hulls and CSG–are equally valid in 3D. We also plan to incorporate a hierarchical design scheme, where the global structure is optimized along with the local structure. This can be done again with a homogenous assumption of global scale material properties, or we could enable multiple local scale designs that can be used at the global scale.

**Acknowledgments** This research was supported by grants of computer time and resources on HPC systems under the Department of Defense High Performance Computing Modernization Program (HPCMP) Challenge Project ARLAP02331C5V.

## References

Ahmed F, Bhattacharya B, Deb K (2013) Constructive solid geometry based topology optimization using evolutionary algorithm. In: Bansal JC, et al. (eds) Proceedings of seventh international conference on bio-inspired computing: theories and applications (BIC-TA 2012), Volume 201 of advances in intelligent systems and computing, Berlin, Germany, pp 227–238. Springer-Verlag, Berlin

Allaire G, Kohn R (1993) Optimal-design for minimum weight and compliance in plane-stress using extremal microstructures. Eur J Mech A-Solids 12(6):839–878

Allen D, Searcy C (2001) A micromechanical model for a viscoelastic cohesive zone. Int J Fract 107(2):159–176

Anghileri M, Chirwa E, Lanzi L, Mentuccia F (2005) An inverse approach to identify the constitutive model parameters for crashworthiness modelling of composite structures. Compos Struct 68(1):65–74

Banzhaf W, Nordin P, Keller RE, Francone FD (1999) Genetic programming: an introduction. Morgan Kauffman, Inc., San Francisco

Bendsoe M, Sigmund O (2003) Topology optimization: theory, methods and applications. Engineering Online Library. Springer

Bieri H (1995) Nef polyhedra: A brief introduction. In: Hagen H, Farin G, Noltemeier H (eds) Geometric modelling, Volume 10 of Computing Supplement, pp 43–60. Springer, Vienna

CGAL (2007) Computational Geometry Algorithms Library. http://www.cgal.org

Qi C, Yang S, Yang L-J, Wei Z-Y, Lu Z-H (2013) Blast resistance and multi-objective optimization of aluminum foam-cored sandwich panels. Compos Struct 105:45–57

Chessa J, Smolinski P, Belytschko T (2002) The extended finite element method (XFEM) for solidification problems. Int J Numer Methods Eng 53(8):1959–1977

Chiandussi G, Avalle M (2002) Maximisation of the crushing performance of a tubular device by shape optimisation. Comput Struct 80(27–30):2425–2432

Cohon J (1978) Multiobjective programming and planning. Academic, New York

Cohon JL, Marks DH (1975) A review and evaluation of multiobjective programming techniques. Water Resour Res 11(2):208–219

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

Duysinx P, Bendsoe M (1998) Topology optimization of continuum structures with local stress constraints. Int J Numer Methods Eng 43(8):1453–1478

Goldberg DE (1987) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading

Hamza K, Saitou K (2004) Optimization of constructive solid geometry via a tree-based multi-objective genetic algorithm. In: Deb K, et al. (eds) Genetic and Evolutionary Computation Gecco 2004, Pt 2, Proceedings, Volume 3103 of lecture notes in computer science, Berlin, Germany, pp 981–992. Springer-Verlag, Berlin

Huang X, Xie YM, Lu G (2007) Topology optimization of energy-absorbing structures. Int J Crashworthiness 12(6):663–675

Huang X, Zhou SW, Xie YM, Li Q (2013) Topology optimization of microstructures of cellular materials and composites for macrostructures. Comput Mater Sci 67:397–407

Koza JR (1999) Genetic programming III: Darwinian invention and problem solving. Morgan Kauffman, Inc, San Francisco

Guo L, Tovar A, Penninger C, Renaud J (2011) Strain-based topology optimisation for crashworthiness using hybrid cellular automata. Int J Crashworthiness 16:239–252

Main JA, Gazonas GA (2008) Uniaxial crushing of sandwich plates under air blast: influence of mass distribution. Int J Solids Struct 45(7–8):2297–2321

Mei Y, Wang X (2004) A level set method for microstructure design of composite materials. Acta Mechanica Solida Sinica 17(3):239–250

Mozumder C, Renaud JE, Tovar A (2012) Topometry optimisation for crashworthiness design using hybrid cellular automata. Int J Veh Des 60(1–2):100–120

MultiMech Research & Development (2012) MultiMech. http://www.multimechrd.com/

Nguyen VP, Stroeven M, Sluys LJ (2011) Multiscale continuous and discontinuous modeling of heterogeneous materials: a review on recent developments. J Multiscale Model 3(4):1–42

Qiao P, Yang M, Bobaru F (2008) Impact mechanics and high-energy absorbing materials: review. J Aerosp Eng 21(4):235–248

Requicha A, Voelker HB (1977) Constructive solid geometry, TM-25. Technical memorandum. Production Automation Project, University of Rochester

Rueda MAF, Cui L, Gilchrist MD (2009) Optimisation of energy absorbing liner for equestrian helmets. Part I: layered foam liner. Mater Des 30(9):3405–3413

Sethian J, Wiegmann A (2000) Structural boundary design via level set and immersed interface methods. J Comput Phys 163(2):489–528

Sigmund O (1994) Materials with prescribed constitutive parameters - an inverse homogenization problem. Int J Solids Struct 31(17):2313–2329

Sigmund O (1995) Tailoring materials with prescribed elastic properties. Mech Mater 20(4):351–368

Sigmund O (2000) A new class of extremal composites. J Mech Phys Solids 48(2):397–428

Srinivas N, Deb K (1994) Multiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2(3):221–248

Souza FV, Allen DH (2010) Modeling failure of heterogeneous viscoelastic solids under dynamic/impact loading due to multiple evolving cracks using a two-way coupled multiscale model. Mech Time-Dependent Mater 14(2):125–151

Souza FV, Allen DH (2011) Modeling the transition of microcracks into macrocracks in heterogeneous viscoelastic media using a two-way coupled multiscale model. Int J Solids Struct 48(22–23):3160–3175

Souza FV, Allen DH (2013) Verification of a two-way coupled multiscale finite element code for dynamic/impact problems. Mech Adv Mater Struct 20(1):74–90

Souza FV, Allen DH, Kim YR (2008) Multiscale model for predicting damage evolution in composites due to impact loading. Compos Sci Technol 68(13, SI):2624–2634

Steuer R (1989) Multiple criteria optimization: theory, computation, application. Krieger, Malabar

Weile D, Michielssen E, Goldberg D (1996) Genetic algorithm design of Pareto optimal broadband microwave absorbers. IEEE Trans Electromagn Compat 38(3):518–525

Wildman RA, Gazonas GA (2009) Gravitational and magnetic anomaly inversion using a tree-based geometry representation. Geophysics 74(3):I23–I35

Wildman RA, Gazonas GA (2011) Genetic Programming-based Phononic Bandgap Structure Design. Technical Report ARL-TR-5733, US Army Research Laboratory

Wildman RA, Weile DS (2007) Geometry reconstruction of conducting cylinders using genetic programming. IEEE Trans Antennas Propag 55(3):629–636

Wildman RA, Weile DS (2008) Greedy search and a hybrid local optimization/genetic algorithm for tree-based inverse scattering. Microw Opt Technol Lett 50(3):822–825

Wildman RA, Weile DS (2010) Inverse scattering of dielectric cylindrical targets using genetic programming. Electromagnetics 30(1–2, Sp. Iss. SI):222–236

Xie Y, Steven G (1993) A simple evolutionary procedure for structural optimization. Comput Struct 49(5):885–896

Xie Y, Steven G (1997) Evolutionary structural optimization. Springer

Xu X, Needleman A (1994) Numerical simulations of fast crack-growth in brittle solids. J Mech Phys Solids 42(9):1397–&

Xu X, Needleman A (1996) Numerical simulations of dynamic crack growth along an interface. Int J Fract 74(4):289–324

Yamagiwa M, Uehara M, Murakami M, Yoneyama M (2010) A reconstruction method for ultrasonic deterioration image by the combination of constructive solid geometry and strongly typed genetic programming. Comput Syst Sci Eng 25(2, SI):161–170

Zohdi T (2002) On the tailoring of microstructures for prescribed effective properties. Int J Fract 118(4):L89–L94