Evolutionary and GPU Computing for Topology Optimization of Structures

Laxman Ram and Deepak Sharma*

Department of Mechanical Engineering, Indian Institute of Technology Guwahati, Assam, India, PIN-781039

Abstract

Although Structural topology Optimization, as a discrete optimization problem, has been successfully solved several times in the literature using evolutionary algorithms (EAs), the two key difficulties lie in generating geometrically feasible structures and handling a high computation time. These two challenges are addressed in this paper by adopting triangular representation for two-dimensional continuum structures, related crossover and mutation operators, and by performing computations in parallel on the graphics processing unit (GPU). Two case studies are solved on the GPU that show 5× of speedup over CPU implementation. The parametric study on the population size of EA shows that the approximate Pareto-optimal solutions can be evolved using a small population with the proposed EA operators.

Keywords: Topology Optimization, Structure Representation, Crossover, Mutation, GPU Computing, Evolutionary Algorithm

1. Introduction

Structural topology optimization is a discrete optimization problem that aims to find the optimal layout of a structure by distributing the material inside the design domain [1]. Topology optimization usually determines the number, location and shape of holes, and the material connectivity inside the design domain of a particular structure. A design domain is generally represented by a Cartesian mesh in which the material is assigned to a few elements of the mesh. The assignment of material is decided with the help of existing methods such as solid isotropic material with penalization (SIMP) [2], level set method [3], genetic algorithms (GAs) [4] etc. Except for GA, the numerical optimization techniques are used to find the optimal layout. It has been reported that SIMP, level set method etc., are computationally effective, but their convergence to the global optimum solution is not guaranteed [5].

*Corresponding author

Email address: dsharma@iitg.ernet.in (Laxman Ram and Deepak Sharma)

GA on other hand can precisely handle discrete optimization. It is also insensitive to the design variables, and can handle multiple objectives simultaneously [6, 7]. Major drawback of population based algorithms including EAs and swarm algorithms that they are computationally expensive as compared to SIMP, level set based methods etc. However, these population based algorithms found a niche for the global search and optimization in the field of structural optimization. Based on the recent survey [8], GA, Artificial Immune Algorithms, Ant Colonies, Particle Swarms, Simulated Annealing, Harmony Search, Differential Evolution Schemes, Bacterial Foraging have been used for various structural optimization problems. New algorithms such as evolution strategy [9], Firefly algorithm [10], Symbiotic Organisms Search [11] have been used for solving various engineering design and truss optimization problems.

Many multi-objective EAs have been developed for structural optimization which are reported in the recent survey [12]. These algorithms generates many Pareto-optimal solutions by showing trade-off among the different objectives. It was concluded in the paper that non-dominated sorting GA (NSGA-II) is the most widely used algorithm followed by Strength Pareto EA (SPEA2). Variants of multi-objective particle swarm optimization have also been used.

Using EAs and multi-objective EAs, various structural optimization problems have been targeted including truss optimization [9], compliant mechanisms [13], civil structure optimization [14], airfoil design optimization [15] to name a few.

But GAs including other EAs often fail to generate geometrically feasible structures, where the regions of applied load and imposed boundary conditions must be connected throughout the material [13, 14]. Another major difficulty with EAs is higher computation time [16], which is mainly due to finite element (FE) analysis of a continuum structure. It has been observed that the solver for FE state equations consumes most of the time of analysis [17]. The solver usually involves a large amount of sparse matrix-vector multiplications to find a solution for the system of linear equations [18]. These two challenges are addressed in this paper by representing structures using triangles, and performing FE simulations in parallel on the GPU.

The main contributions of this paper are as follows:

- A triangular topology representation from [19] is adopted to represent a geometrically feasible continuum structure. Kawamura et al. [19] used triangle topology representation using frame/beam element, whereas it is extended to two-dimensional linear continuum structure in this paper.

- This representation is coupled with non-dominated sorting genetic algorithm (NSGA-II) [20]. The crossover and mutation operators are designed for the triangular topology to preserve geometrical feasibility of a generated structure.

- The second challenge is tackled by performing FE simulations on the GPU in parallel to reduce computation time. Mainly, the solver for the linear

finite element state equations is designed using matrix-free conjugate gradient (CG) method.

Two examples are solved by simultaneously minimizing weight and strain energy stored in the structures to generate multiple diverse topologies.

The organization of this paper is as follows. Section 2 overviews the relevant literature for representing the structure and high performance computing in structural optimization. Section 3 presents the bi-objective optimization problem formulation for two different case studies. In section 4, evolutionary and high performance computing for topology optimization are discussed in detail. In section 5, various results of two case studies are presented, and speedup is shown. The paper is concluded in section 6 with a note on future work.

2. Overview

In this section, an overview of related studies targeting structure representation and operators for EA is presented. The literature survey is also extended to high performance computing used in structure optimization.

2.1. Overview of Structure Representation Using EA

In the literature, well-connected structures are created by adopting special representation for EAs. A bit-array representation is the most obvious choice using a binary string. Bit value '1' represents material in the grid, otherwise it is a void [4]. This representation, however, has the drawbacks of generating (i) disconnected topology at the regions of applied and boundary conditions, (ii) floating elements which were not connected to a topology of a structure, (iii) checkerboard topology, and (iv) topology with point connectivity between two grids of material [13].

A morphological structure representation was developed by Tai and Chee [21], where the Bézeir parametric curves with varying thickness were used to connect the regions of applied and boundary conditions. This morphological representation method was able to generate well-connected structures without any checkerboard problem. It has been further improved by using the graph theory by Wang and Tai [22] in which vertices were connected with the piecewise cubic Bézier curve. This representation overcame the drawbacks of higher-order Bézier curve. These two methods were further improved by skeleton and flash morphology technique [23]. Recently, pairs of curves have been used to generate topologies of compliant mechanisms [14]. In this implementation, the areas bounded by a pair of curves define the material distribution between them.

The linear piece-wise segment connectivity was proposed by Sharma et al. [13] to reduce complexity of Bézier curve. In this method, the regions of applied and boundary conditions were connected through straight lines via intermediate points inside the design domain. Such representation was able to generate well-connected structures, and eliminated the floating element problem. However, the material connectivity was destroyed by a two-dimensional crossover operator

suggested in [16]. A repair operator was then devised to generate well-connected structures.

The topologies generated by EAs or other numerical techniques are not smooth in general [24, 25, 26]. A different approach has been adopted in the literature to smoothen the boundaries and holes of structural topology by using feature based shape recognition techniques [27] employing artificial neural network. The smooth structures were then further optimized for their shape using various hybrid algorithms like GA with Taguchi method [28, 29], immune algorithm with Taguchi method [30], differential evolution algorithm with Taguchi method [31] and particle swarm optimization with Taguchi method [32]. In the hybrid algorithms, refinement of the population space was introduced by Taguchi method in which variable bounds were refined, and the shape optimization problem was solved using different EAs. In multicomponent structural assembly, the feasibility of structures was maintained by designing various constraints and repairing rules in which the design domain was divided into square structural elements, thin strip elements and small square diagonal joint elements [7].

A well-connected truss topology was developed by using triangular topology members [19]. In this method, a triplet of three truss members, which are making a triangle, was added to the topology till all the regions of applied and boundary conditions were connected. Although this method was developed for truss topology, it can be extended for a continuum structure. In this work, we adopted the triangle topology representation to generate well-connected structures, and to reduce associated problems discussed earlier.

2.2. Overview of High Performance Computing in Structure Optimization

Structural optimization is computationally expensive in general because of the involvement of FE analysis at each step. We can reduce this computation time by performing FE simulations in parallel. The massage passing interface (MPI) was used in the literature in which each structure of EA got evaluated on different computers [6]. However, such clusters of computers require high installation cost, power consumption, and large floor areas.

From last few years, many researchers have been attracted toward the GPUs, which are high-performance multi-thread many-core processors with tremendous computational power and very high memory bandwidth [33]. Moreover, it is affordable, and can easily be fixed with a computer. In structural optimization, GPUs were used to perform FE simulations in parallel [34, 35, 36] using gradient based optimization techniques. Iterative CG method is generally used for solving the finite element state equations.

Two approaches have been described in the literature for the solver of FE state equations. In one approach, the global stiffness matrix was constructed in the CPU and stored in a specialized format on the GPU to save the device memory [34, 36]. The solution is then sought by minimizing the residual for FE state equations. In another approach, the residual for FE state equations was calculated element- or node-wise to reduce the device memory access [35]. The

matrix-vector products for the residuals needed in the CG iteration were generated procedurally without storing the global stiffness matrix. We are exploring the GPU computing with EA for topology optimization in this paper using the matrix-free CG method.

3. Problem Description

Topology optimization problem has been solved in the literature using single and multiple objectives for aerospace, civil and mechanical structures [1], where weight minimization is the primary objective. Weight minimization generates flexible structure which may not be rigid enough to bear external loads [37]. In such situation, another objective such as minimization of strain energy or total potential energy of a structure etc., can be used with the primary objective. This bi-objective optimization can generate multiple structures with varying flexibility and rigidity. This idea has been explored earlier by incorporating a helper or secondary objective to the primary objective for topology optimization [13, 16].

In this paper, we generate structures using a bi-objective optimization formulation which is given as,

$$\begin{array}{ll} \min & W, \\ \min & U, \\ \text{subject to} & \vartheta \leq 50\% V, \end{array} \tag{1}$$

where W is the weight of a structure, U is the stored strain energy inside a structure, ϑ is the volume of a structure, and V is the total volume of a design domain of a structure.

These two objectives are conflicting in nature as weight minimization introduces flexibility to a structure which otherwise increases strain energy stored in it. We use the continuum mechanics approach to calculate both objectives. In this approach, a given design domain is divided into finite grids or Cartesian mesh. Material is then assigned to few grids to generate a structure. An example is shown in Fig. 1, in which a rectangular domain is represented by finite grids. Few of these grids are filled with material (black in color), and rest of them are void. The material in each finite grid is represented by '1' and void finite grids get '0'. The binary representation creates a binary string of length equivalent to the number of finite grids. We, then, calculate weight, W, of a structure by counting number of '1's in the binary string.

Another objective is minimization of strain energy, U, stored in the structure, which is given by,

$$U = \frac{1}{2}\{u\}^T[K]\{u\}, \tag{2}$$

where $\{u\}$ is the nodal displacement vector, and $[K]$ is the global stiffness matrix of a structure. It is worth noting that the vector and matrix are generated from assembly of elemental nodal displacement vector and elemental stiffness matrix, respectively. Details of FE analysis for linear elasticity can be found in any

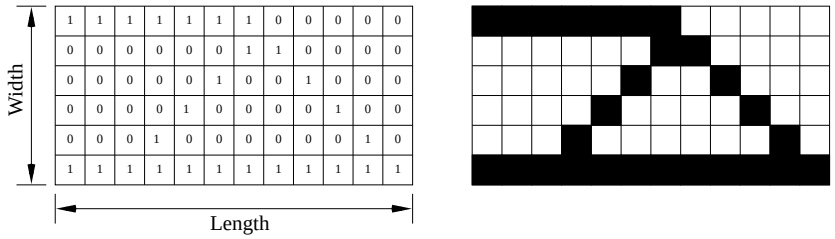


Figure 1: A rectangular design domain is represented by finite grids wherein few elements are filled with material, and rest of them are void.

standard finite element book like [38]. We then calculate U by performing finite element simulations on the graphics card. It is worth mentioning that the matrix $[K]$ becomes singular when material is assigned to some of the elements, and the rest of them are left as void. Therefore, we assign minimum density of 0.3 to voids and 1 to the elements filled with material. We consider non-dimensional unit of material density so that we can use our finite element analysis for any type of material.

We consider one constraint on the volume of the structure which is directly proportional to the weight of the structure, W. We, thus, count the number of '1's in the binary representation of a structure, which should be less than a particular fraction (50%, for the present implementation) of the binary string length.

It is to be noted that the Boolean representation $(0-1)$ for each grid shown in Fig. 1 is the decision variable for the given problem. Number of decision variables is equal to number of grids.

We can observe from the above discussion that topology optimization is a discrete optimization problem. The primary objective depends on the material distribution, and the second objective is based on finite element analysis of a structure generated from the same material distribution.

4. Evolutionary and High Performance Computing

In this section, we present evolutionary algorithm (EA) to solve a discrete structural topology optimization problem. EA is used as global search and optimization technique because EA can offer various advantages as mentioned in section 1. In the following sections components of EA and GPU computing for finite element simulations are described.

4.1. NSGA-II

Elitist non-dominated sorting genetic algorithm (NSGA-II) is one of the benchmark algorithms for two- and three-objective optimization [39]. Thus, we choose NSGA-II for solving a discrete bi-objective topology optimization problem of elastic structure. A detailed description of NSGA-II can be found in [20]. A generalized flow chart of EA describing the steps in NSGA-II is

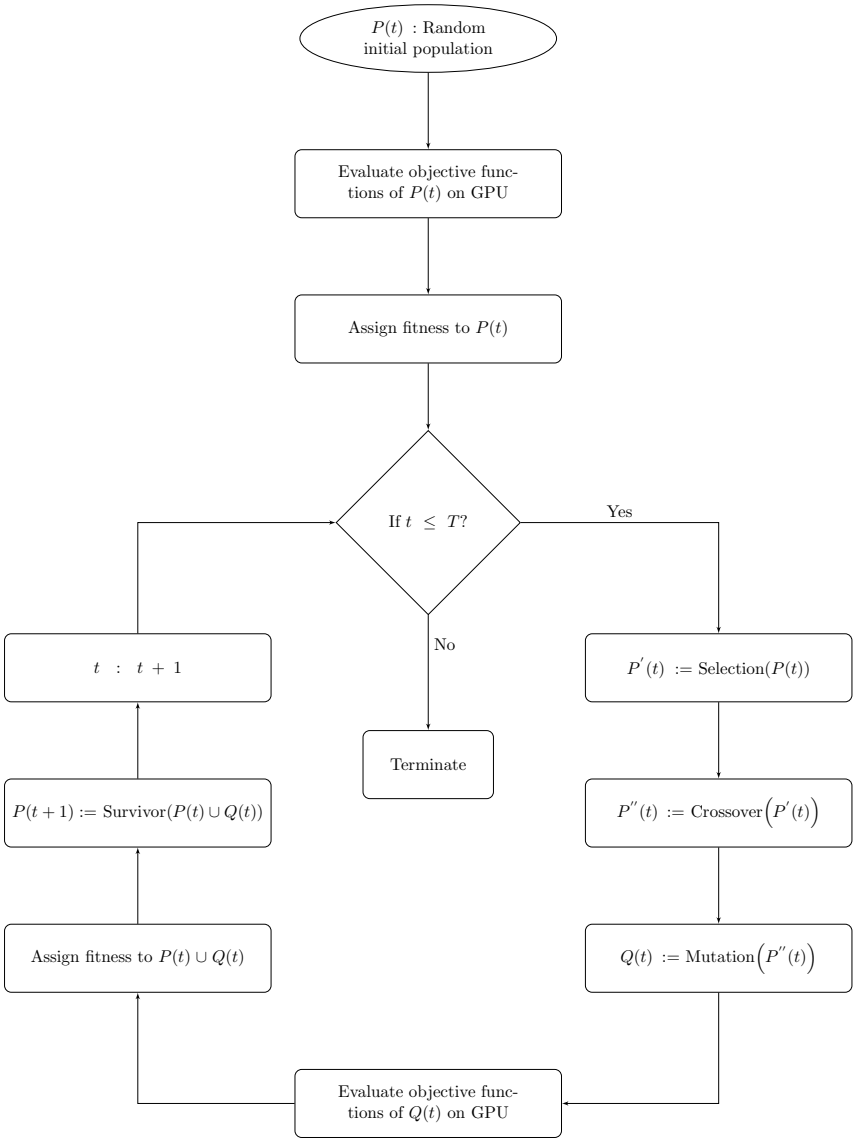


Figure 2: Flow chart of EA for topology optimization.

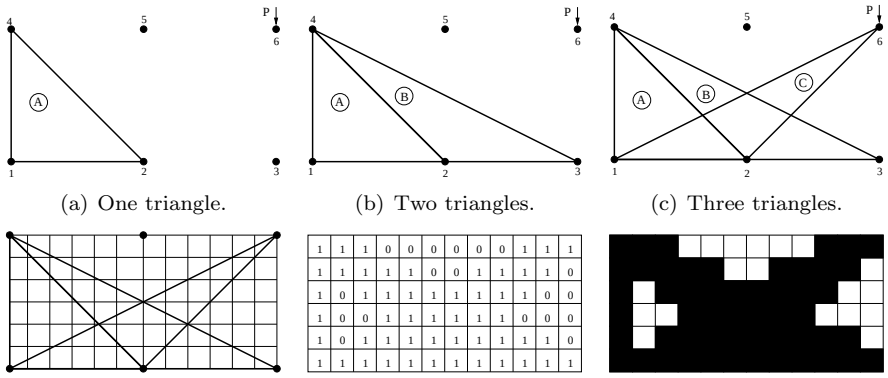
shown in Fig. 2. The algorithm starts by generating random initial population, which is then evaluated by calculating the values of the objective functions and constraint. NSGA-II assigns fitness to its solutions by using mathematical partial-ordering principle. It emphasizes (i) non-dominated population members by non-dominated sorting operator, and (ii) isolated population members by using a crowding distance operator in every iteration. In a standard loop of EA, NSGA-II uses crowded comparison binary tournament selection operator in which a solution with better non-dominated rank and larger crowding distance wins the tournament. The selected solutions are copied to the mating pool. The crossover and mutation operators are then performed on the mating pool solutions to generate new population, which is referred as child or offspring population. NSGA-II uses elite preservation procedure at the survival stage in which parent and offspring populations are combined together. The fitness is assigned to the combined population by using non-dominated sorting operator and crowding distance operator. The solutions with better fitness are selected for the next generation.

From the above description of EA, it can be observed that a random binary string representation can generate a geometrically infeasible structure which has no practical meaning [6]. Moreover, the standard crossover and mutation operators for binary strings can further distort the topology of structures [40]. It is observed from the literature [13, 41] that performance of EA can be enhanced by incorporating domain knowledge. It indicates that representation of solution, crossover and mutation operators for topology optimization require structural changes for NSGA-II [13, 16].

4.2. Triangular Representation for Structures

We adopted the triangular representation from [19], in which truss topology was represented by a combination of triangles. We chose this representation because it is simple to implement for various topology optimization problems. We extend this representation for topology optimization of 2D continuum structure so that a monolithic and joint-less structure can be generated [1].

The triangular representation is shown in Fig. 3 wherein a set of six nodes is used to draw triangles within the rectangular design domain. We consider three nodes in the horizontal direction and two nodes in the vertical direction. If the boundary conditions are applied, say, at nodes 1, 4 and 6, it is necessary for these three nodes to be connected through some combination of triangles to make a geometrically feasible structure. We start the representation by joining nodes $1-2-4$ at random as shown in Fig. 3(a). It is to be noted that any randomly selected triplet of nodes must not be collinear. At this stage node 6 is not connected. We then draw another triangle by randomly selecting two nodes from previous triangle, i.e., triangle **A** and one non-collinear node from rest of the nodes, i.e., 3, 5 and 6. Suppose, nodes 2 and 4 are selected at random from triangle **A**, and node 3 is selected to draw the second triangle **B** that is shown in Fig. 3(b). Still, node 6 is not connected to nodes 1 and 4 by any combination of triangles. We now draw a third triangle by selecting any two nodes from previous triangles, i.e., triangles **A** and **B**. Suppose, nodes 1 and 2



(a) One triangle. (b) Two triangles. (c) Three triangles.

(d) Triangles in a design domain.

1	1	1	0	0	0	0	0	0	1	1	1
1	1	1	1	1	0	0	1	1	1	1	0
1	0	1	1	1	1	1	1	1	1	0	0
1	0	0	1	1	1	1	1	1	0	0	0
1	0	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1

(e) Binary representation.

(f) Material representation.

Figure 3: Triangular representation for structure.

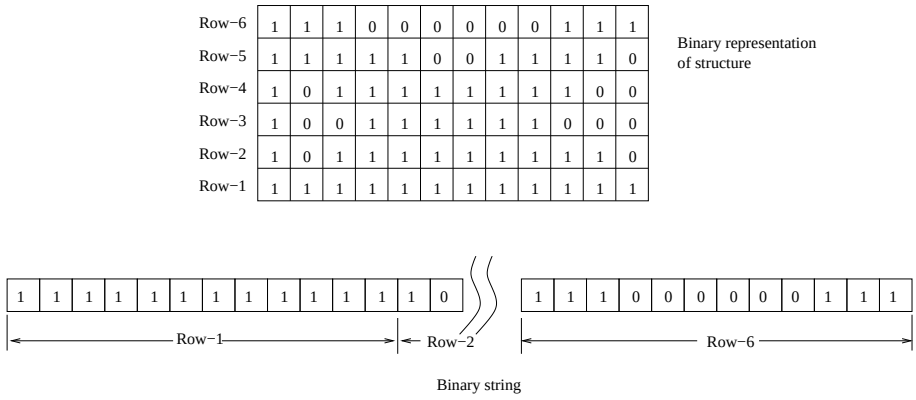


Figure 4: A binary string representation of a discrete structure for NSGA-II.

are selected so that they can be connected to either node 5 or node 6. Assuming node 6 is selected, triangle **C** is drawn as shown in Fig. 3(c). This completes the triangular representation for a structure in which three triangles are required to connect nodes at the applied and boundary conditions. It is worth mentioning that nodes $1-4-6$ at the applied and boundary conditions can be connected in one triangle. In this case the triangular representation for a structure gets completed.

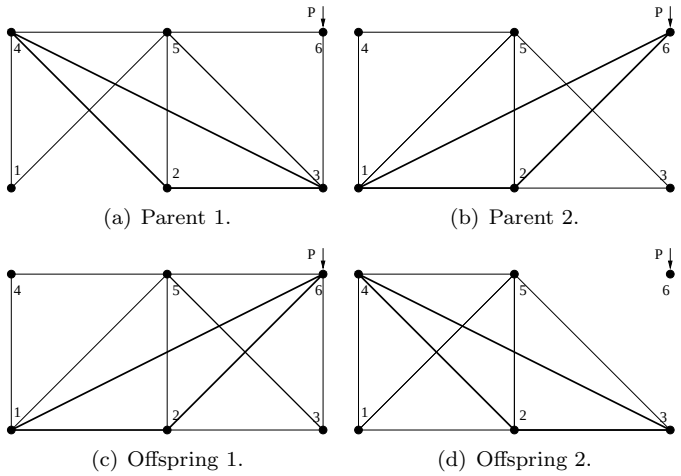
The material is now distributed in the design domain according to the triangles. It can be seen in Fig. 3(d) that the design domain is divided into finite grids. A grid is assigned with '1', if any edge of the triangles passes through it as shown in Fig. 3(e). Otherwise, '0' is assigned to it. The material is then assigned to the grids having a value '1' as shown in Fig. 3(f). Later, these grids represent four node quadrilateral finite elements for FE analysis.

The binary representation shown in Fig. 3(e) is used to generate a binary string for each individual of NSGA-II population. The rows are copied one by one into a single binary string a shown in Fig. 4. Here, the number of decision variables is equal to number of grids that can take either '0' or '1' value to represent a discrete structure.

The procedure for generating triangles and distributing material in the design domain is repeated to generate a random initial population ($P(t)$) of geometrically feasible structures.

4.3. Triangular Crossover Operator

Crossover is the primary EA operator that is responsible for exploring search space by creating new structures. Typically, single-point crossover operator, multi-point crossover operator etc., are used for binary strings to generate new solutions. However, we observe from the literature that such crossover operations are not efficient for topology optimization [41]. Therefore, other crossover operators like block exchange [13], domain specific [16] etc., are devised to improve performance of EA.



(a) Parent 1. (b) Parent 2.

(c) Offspring 1. (d) Offspring 2.

Figure 5: Triangular crossover operator.

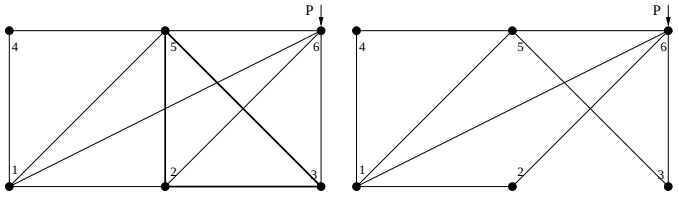
Triangular crossover operator is thus developed that can support our triangular representation. We describe triangular crossover operator by an example shown in Fig. 5. One triangle from both parent 1 and parent 2 is chosen at random as shown in Figs. 5(a) and 5(b). The triangle with the thicker edges joining nodes $2-3-4$ from parent 1 and the triangle joining nodes $1-2-6$ from parent 2 are swapped to create two new structures as shown in Figs. 5(c) and 5(d). The thicker edges in these figures show addition of new triangle.

4.4. Mutation Operators

We propose two mutation operators to use with the triangular representation. The first mutation operator is responsible for deleting a triangle at

random. Another operator termed as the repairing operator creates triangle(s), if any structure becomes geometrically infeasible due to triangular crossover and triangular mutation.

We describe triangular mutation operator by an example shown in Fig. 6(a) in which nodes $2-3-5$ of a triangle is chosen at random. The mutated structure is shown in Fig. 6(b) in which the chosen triangle is deleted.



(a) Structure before mutation. (b) Structure after mutation.

Figure 6: Triangular mutation operator.

The repairing mutation operator is performed when a structure becomes geometrically infeasible after the application of crossover and mutation. It means that the nodes at the applied and boundary conditions are not connected through any combination of triangles. One such example can be seen in Fig. 5(d) where the node 6 is not connected. Another type of infeasibility can be seen in Fig. 7 where, although the nodes 1, 4 and 6 are connected by triangles, they are not connected to each other.

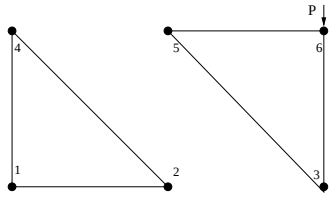


Figure 7: Infeasible structure.

We make such structures feasible by choosing two nodes from previous triangles at random, and connect them to the node where the applied and boundary conditions are applied. The procedure is similar to the triangular representation which was discussed in section 4.2.

4.5. GPU Computing

GPU computing is coupled with EA to reduce its computation time. The objective functions and constraint are calculated using (1) in which finite element linear elastic simulations are being performed on the graphics card. The weight of the structure is calculated by counting the number of '1's of the binary string of a structure. The strain energy is calculated by using Galerkin finite element method [38]. A set of partial differential equations is thus evolved

which takes a form of linear state equations, $[K]\{u\} = \{f\}$, after considering all finite elements. Here, $[K]$ is the global stiffness matrix, $\{u\}$ is the nodal displacement vector, and $\{f\}$ is the force vector of internal and external forces. It has been observed from the literature that the solver for FE state equations is the most time consuming step [18]. The conjugate gradient (CG) method is generally used for solving a system of linear equations, iteratively. However, the CG method requires a lot of sparse matrix-vector multiplications, which renders the process computationally expensive. This calls for the use of the GPU.

Two procedures are followed in the literature for solving FE state equations in which the solution, $\{u\}$, is calculated by minimizing the residual $|R|$, where $\{R\} = \{f\} - [K]\{u\}$. In first procedure, matrix $[K]$ is assembled on the CPU, and later copied to the GPU memory [42]. This procedure requires more global memory on the GPU, and requires many costly CPU-GPU transactions through the PCI bus. For solving large scale problems, a domain decomposition approach is used [43], but it is difficult to find the correct partition of a domain that can result in a higher speedup. Other approach is to use the matrix-free CG procedure [35], in which $[K]$ is not stored explicitly, and nodal computation is used for the finite element matrix-vector multiplications to find the residual. It is observed that some extra computations are always performed for sharing nodes, and padding of zeros may be required, if the size of Cartesian mesh is not multiple of a block-size of the GPU. However, this procedure can use shared memory of the GPU efficiently, fulfills the necessities for memory coalescing, and avoids shared memory bank conflict.

We adopted Schmidt and Schulz [35] procedure in this work, where the matrix-free CG method is used as an iterative solver for FE state equations. We consider one thread per node, and use the built-in data type 'float4' or 'double4' to store displacement and density of each node. This provides better GPU memory alignment. Each thread loads data into shared memory such that the block of threads can access data for neighboring nodes. This implementation can minimize global memory access. It is worth mentioning that the nodal finite element matrix-vector computation requires data of neighboring node for linear finite element test function. However, some threads, specially on the edge or corner of the design domain, must load halo values (defined later). If the number of halo values is not multiple of half-warp size, loading halo value cannot be done entirely on coalesced way.

The implementation is shown in algorithm 1 and their steps are described in subsequent paragraphs.

4.5.1. Algorithm 1, step 1

In this step, some constant parameters like iteration counter and maximum allowed iterations are initialized.

4.5.2. Algorithm 1, step 2

The residual $\{R\}$ is calculated on the GPU. Here, one thread is assigned to each node of the finite element mesh. The implementation for calculating $\{R\}$ is shown in algorithm 2.

Algorithm 1 Matrix-Free Conjugate Gradient Method for FE Simulations

1: Iteration counter $t = 0$ and maximum allowed iterations T.
2: Calculate residual $\{R_t\}$ for each node on the graphics card using algorithm 2.
3: $\{p_t\} = \{R_t\}$
4: **while** $(t < T \ || \ |R_t| < \epsilon)$ **do**
5: $\quad \alpha_t = \dfrac{R_t^T R_t}{p_t^T K p_t}$
6: $\quad u_{t+1} = u_t + \alpha_t p_t$
7: $\quad R_{t+1} = R_t - \alpha_t K p_t$
8: $\quad \beta_t = \dfrac{R_{t+1}^T R_{t+1}}{R_t^T R_t}$
9: $\quad p_{t+1} = R_{t+1} + \beta_t p_t$
10: $\quad t = t + 1$
11: $\quad$ Calculate residual $\{R_t\}$ for each node on the graphics card using algorithm 2.
12: **end while**

Algorithm 2 Residual calculation by a compute thread for node (i, j) of finite element.

1: Copy the coordinates of node (i, j) and halo values, if any, to the shared memory.
2: Assign $|R_t| = 0$.
3: Update $\{R_t\}$ if Neumann or Dirichlet boundary conditions are applied at node (i, j).
4: Identify a set of elements E for which node (i, j) is a common vertex.
5: Copy the density of all elements belong to E to the shared memory.
6: **for** each element $e \in E$ **do**
7: $\quad$ Fetch density (ρ^e) of element e from the shared memory.
8: $\quad$ Identify a set of all local nodes L of element e.
9: $\quad$ **for** each local node $k \in L$ **do**
10: $\quad\quad$ Fetch the global coordinates for local node k.
11: $\quad\quad$ Calculate residual using (3).
12: $\quad$ **end for**
13: **end for**
14: Synchronize thread.

Algorithm 2, step 1. We first copy coordinates of a finite element node into the shared memory of the GPU. We can see from Fig. 8 that node (i, j) is one of the vertices for four-node quadrilateral finite element. We assign a thread to node (i, j) which we refer as a compute thread (i, j). The rectangle represented by dashed line in the same figure shows that all the nodes inside it are being computed by one block of threads which has a common shared memory. The coordinates of nodes inside the dashed rectangle are copied to the shared

memory to minimize access to the device memory. It is noted that coordinates of neighboring nodes are also required for FE analysis of the particular block. Therefore, some threads on the boundary of the thread block will also store coordinates of neighboring nodes in the shared memory [35]. These extra coordinates are known as halo values in the literature. Similarly, threads of other blocks will store the required coordinates into the shared memory.

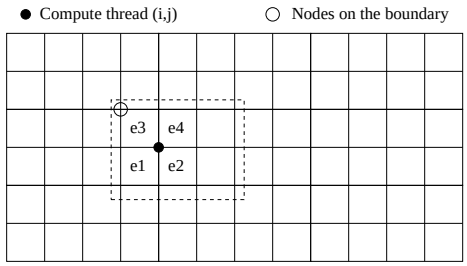


Figure 8: Schematic for GPU computing.

Algorithm 2, steps 2–3. Once the required data is copied into the shared memory, the compute thread (i, j) initializes zero residual for node (i, j). The residual gets updated, if Neumann or Dirichlet boundary conditions are applied at node (i, j).

Algorithm 2, steps 4–5. It can be seen from Fig. 8 that node (i, j) is the common vertex to few elements. In this case, residual $\{R\}$ for node (i, j) is calculated with respect to the elements sharing the common vertex (i, j), that is, elements $E = \{e1, e2, e3, e4\}$. This implies that $\{R\}$ of node (i, j) is calculated with respect to all nodes belonging to the set E. When the set E is identified by the compute thread (i, j), density of each element $e \in E$ is copied into the shared memory.

Algorithm 2, steps 6–8. The residual for node (i, j) is now calculated wherein density of element $e \in E$ is fetched from the shared memory. A set of local nodes of element e is identified as $L = \{1, 2, 3, 4\}$. It can be seen in Fig. 9 that common node (i, j) is represented as node 4 for element e_1, node 3 for element e_2, node 2 for element e_3, and node 1 for element e_4. This local node numbering of elements helps to decide correct trial or test finite element function.

Algorithm 2, steps 6–13. For each local node, the global coordinates are fetched from the shared memory by the compute thread (i, j), and the residual is calculated, which is given by,

$$\{R\} = \{R\} - \rho^e \times ([K]^e \{u\}^e), \tag{3}$$

where ρ^e is the density of element, $[K]^e$ is the elemental stiffness matrix, and $\{u\}^e$ is the elemental displacement vector. This completes residual calculation

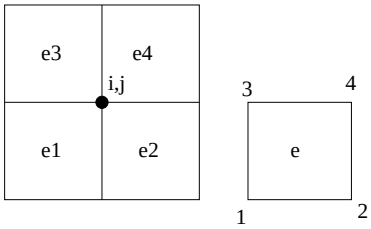


Figure 9: Elements having common vertex (i, j) and local numbering of nodes for any element e.

by the compute thread (i, j) on the GPU. At the end of residual computation, all compute threads of the GPU are synchronized. Now, the computations return to algorithm 1.

4.5.3. Algorithm 1, steps 3−10

Residual $\{R_t\}$ is now copied to the host wherein residual values are copied to another vector $\{p_t\}$ for performing CG iterations. The termination condition is then checked at step 4 in which CG method gets terminated when it reaches to the maximum allowed iterations T or the residual becomes less than some ϵ value. Till step 10, the standard steps of CG method are followed.

4.5.4. Algorithm 1, step 11

Residual $\{R_t\}$ is again calculated on the GPU using algorithm 2. However, only Dirichlet boundary conditions are applied at step 3 of algorithm 2.

When CG method terminates, the solution $\{u\}$ is utilized to calculate strain energy of a structure. This step involves multiplication of vectors and matrix which are also performed on the GPU. The standard steps of matrix−vector multiplication on the GPU are followed which can be found in [44].

5. Results and Discussion

We consider two case studies for topology optimization from [1]. We choose the same set of EA parameters and finite element simulation parameters for both case studies as given in Table 1. Runs are taken for different population sizes including 4, 8 and 12. The number of generation is kept fixed after observing progress of the approximate Pareto-optimal solutions depicted in Fig. 11. We kept higher value of crossover probability and lower mutation probability which is generally recommended in the EA literature. ϵ value is kept small for termination CG iterations. We run EA for 10 times with different initial population to perform statistical analysis.

The statistical analysis is performed to measure the strengths and weaknesses of algorithm based on the quality of the evolved solution such as proximity to the reference set, spread and evenness of the non-dominated solutions in the objective space. Several indicators are available to indicate the performance of multi-objective EAs which independently explore different features

Table 1: EA and FE parameters.

EA parameters		Finite element parameters	
Population	12	Number of elements in x-direction	200
Generation	50	Number of elements in y-direction	100
Crossover probability	0.9	ϵ for CG termination	10^{-3}
mutation probability	0.01		

of the algorithm [45, 46, 47]. In this paper, we choose hypervolume indicator and attainment surface plots for the assessment. It is worth to mention that the attainment plot is a qualitative indicator and the hypervolume indicator is quantitative. Both indicators can assess performance of the algorithm on convergence, spread and evenness of the non-dominated solutions simultaneously.

Hypervolume indicator ($\mathbf{I}_{\bar{H}}$): [48]

The hypervolume indicator I_H measures the hypervolume of that portion of the objective space that is weakly dominated by an approximate set A. This indicator gives the idea of spread quality and has to be maximized. As recommended in the study [47], the difference in values of hypervolume indicator between the approximate set A and the reference set R is calculated in this paper, that is, $I_{\bar{H}} = I_H(R) - I_H(A)$. The smaller value suggests good spread [46, 47].

Attainment surface: [49, 50]

An approximate set A is called the $k\% - approximate\ set$ of the empirical attainment function (EAD) $\alpha_r(z)$, iff it weakly dominates exactly those objective vectors that have been attained in at least k percentage of the r runs. Formally, $\forall \in Z : \alpha_r(z) >= k/100 \leftrightarrow A \preceq \{z\}$ where $\alpha_r(z) = \frac{1}{r}\sum_{i=1}^{r} I(A^i \preceq \{z\})$. A^i is the ith approximation set (run) of the optimizer and $I(.)$ is the indicator function, which evaluates to one if its argument is true and zero if its argument is false.

An attainment surface of a given approximate set A is the union of all tightest goals that are known to be attainable as a result of A. Formally, this is the set $\{z \in \Re^n : A \preceq z \wedge \neg A \prec\prec z\}$ [47].

The runs are taken on a workstation with Intel Xeon processor E5-1650 (12M cache, 3.20GHz) equipped with 12GB DDR3-1600 RAM, 12GB NVIDIA Tesla K40c GPU card with 2880 cores, and 1GB NVIDIA Quadro K600 GPU card with 192 cores. Further details of the GPU cards can be found on NVIDIA website[1].

5.1. Case Study 1: Cantilever Plate Design

We consider a rectangular design domain for cantilever plate design that is shown in Fig. 10. One end of this plate is fixed, and a concentrated load is

[1] www.nvidia.com

applied at one corner. For our analysis we consider nodes 1, 3, and 4 where a material connectivity is sought.

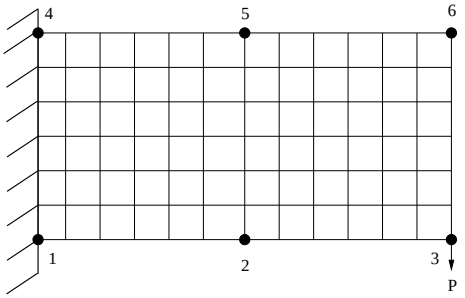


Figure 10: A design domain for cantilever plate.

The approximate Pareto-optimal solutions of a typical EA run is shown in Fig. 11. It can be seen in the figure that nine solutions are evolved by showing trade-off between two objectives. Solution 1 gets evolved as the lightest solution. A simple topology of solution 1 can be seen in Fig. 12(a) in which three nodes at the applied and boundary conditions are connected by one triangle.

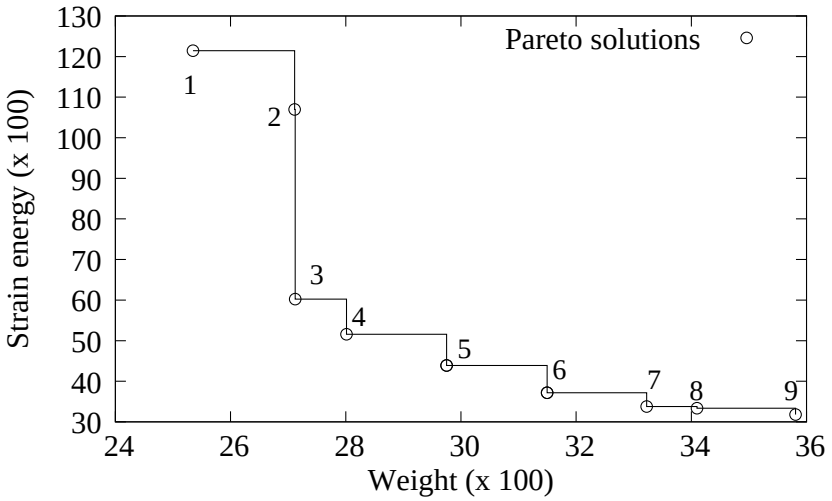


Figure 11: The approximate Pareto-optimal solutions for cantilever plate design.

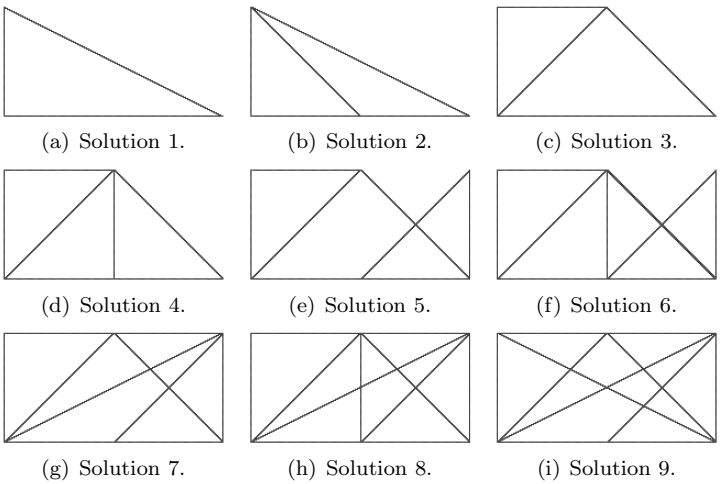
Solutions 1 and 2 in Figs. 12(a) and 12(b) have common triangle. But, solution 2 stores less strain energy due to the presence of a stiffener between nodes 2 and 4. Similarly, different material connectivity between solutions 2 and 3 results in drastic reduction in strain energy. But at the same time, weight of the structures increases.

Solutions 3 and 4 of Fig. 11 have different topologies as shown in Figs. 12(c) and 12(d) with additional stiffener of material between nodes 2 and 5. It can

be observed that inclusion of stiffener increases weight of structure but reduces strain energy stored in the structure.

Solution 5 in Fig. 11 is topologically different and little heavier than solutions 3 and 4 due to additional triangle joining nodes 2, 3 and 6 as shown in Fig. 12(e). The strain energy stored in solution 5 is less due to presence of material at node 3 where an input force is applied.

Solution 6 is stiffer than solution 5 in Fig. 11 due to the presence of a stiffener joining nodes 2 and 4 as shown in Fig. 12(f). Also, solution 7 in Fig. 12(g) has one more triangle joining nodes 1, 5 and 6 than solution 5 which makes it stiffener but heavier. Again a stiffener at nodes 2 and 5 is extra in solution 8 as shown Fig. 12(h) which makes it more stiffer than solution 7. In solution 9, a triangle of solution 1 gets added to the topology of solution 7 which makes it the most stiffer structures among the Pareto solutions which stores minimum strain energy.



(a) Solution 1. (b) Solution 2. (c) Solution 3.

(d) Solution 4. (e) Solution 5. (f) Solution 6.

(g) Solution 7. (h) Solution 8. (i) Solution 9.

Figure 12: Topology of Pareto-optimal solutions for cantilever plate design.

It can be observed from the evolved topologies of Fig. 12 that these approximate Pareto-optimal solutions are evolved due to the presence of different stiffener or triangle. It can also be observed from the same figure that triangles of solution 3 is common among all solutions except solutions 1 and 2 to provide stiffness to structure. Other triangles keep on adding to make structures more stiffer.

Progress of non-dominated front of the same typical run is shown in Fig. 13. It can be seen that EA is converged to the approximate Pareto-optimal solutions in only 40 generations with a small population size of 12. It is due to the triangular representation that helped EA to evolve topologically different solutions quickly.

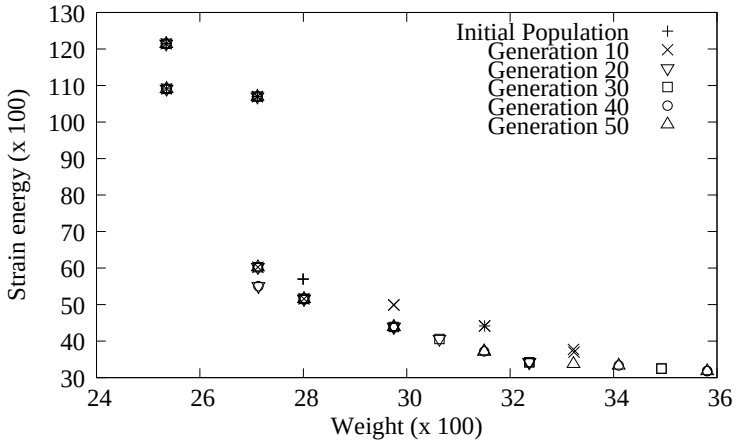


Figure 13: A progress of non-dominated solutions for 50 generations of EA.

5.1.1. Effect of Population Size

The cantilever plate design problem is also solved using population sizes of 4 and 8. Rest of EA parameters are kept same as given in Table 1. EA is then run for 10 times, and statistical analysis is done using attainment plot and hypervolumne indicator [46]. Fig. 14 shows 0% attainment plots for three population sizes. The plot suggests a good spread and proximity of approximate Pareto-optimal solutions among themselves. Statistical hypervolumne indicator values are given in Table 2. The best hypervolume values suggest that small population sizes are able to generate a good set of the approximate Pareto-optimal solutions. Other statistical parameters are also small and close to zero value. It can be concluded that EA with the triangular representation shows equivalent performance with small population sizes that can further reduce computation time.

Table 2: Statistical values of hypervolume indicator for different population sizes. Indicator value -1 is best, and $+1$ is worst.

Population	Mean	Median	Standard deviation	Best	Worst
4	0.2409	0.2385	0.1576	0.0657	0.6407
8	0.0992	0.1054	0.0469	0.0530	0.1910
12	0.0418	0.0442	0.0162	0.0263	0.0798

5.1.2. Computation Time

Finite element simulations using the matrix free CG method are performed both on the CPU and the GPU. Table 3 shows computation time and speedup for a typical run of CG for 2000 iterations. We can achieve a speedup of $5\times$ by using Tesla K40c card over the CPU. However, we could achieve a speedup of $2\times$ only using Quadro K600 GPU card over the CPU. The reason is that more

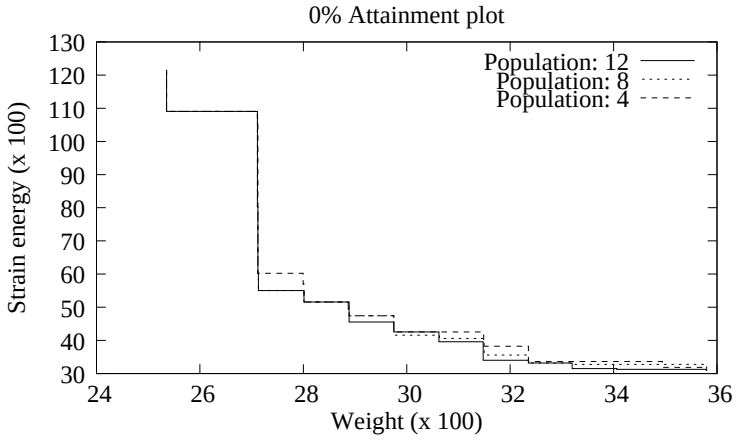


Figure 14: 0% attainment plots for different population.

number of computing cores and larger memory are provided by Tesla K40c than Quadro K600. A speedup of 2.58× is observed between two GPUs for CG iterations.

Table 3: Computation time in seconds and speedup for 2000 CG iterations.

CPU	98.46
Tesla K40c	19.14
Quadro K600	49.31
Speedup K40c vs CPU	5.14
Speedup Quadro K600 vs CPU	2.00
Speedup K40c vs Quadro K600	2.58

Average computation time of EA for 10 runs with finite element simulations on the graphics cards are shown in Table 4. Same speedup is obtained as reported in the last table. However, a significant computation time can be saved with Tesla K40c GPU. Moreover, a marginal difference can be seen in total computation time of optimization and FE simulations. This suggests that EA operators consume negligible time as compared to FE simulations. The higher computation time results from function evaluations where finite element simulations are required to calculate strain energy stored in the structures.

Table 4: Computation time in seconds for overall optimization process including FE computations on the GPUs.

Population	4		8		12	
GPUs	T. K40c	Q. K600	T. K40c	Q. K600	T. K40c	Q. K600
FEM time	3888.05	9374.06	7762.38	18795.80	11657.54	28217.22
Total time	3889.13	9375.13	7763.54	18796.93	11658.77	28218.41

5.2. Case Study 2: Plate Supported from Both Ends

We consider another case study of a plate supported from both ends, and the load is applied at the middle of the top edge. The design domain is shown in Fig. 15 in which we sought material connectivity among nodes 1, 3 and 5.

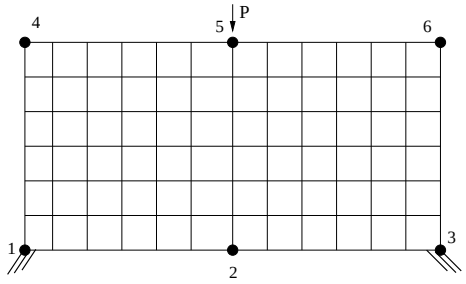


Figure 15: A design domain for plate supported at both ends.

The approximate Pareto-optimal solutions evolved by a typical run of EA are shown in Fig. 16, and their respective topologies are shown in Fig. 17.

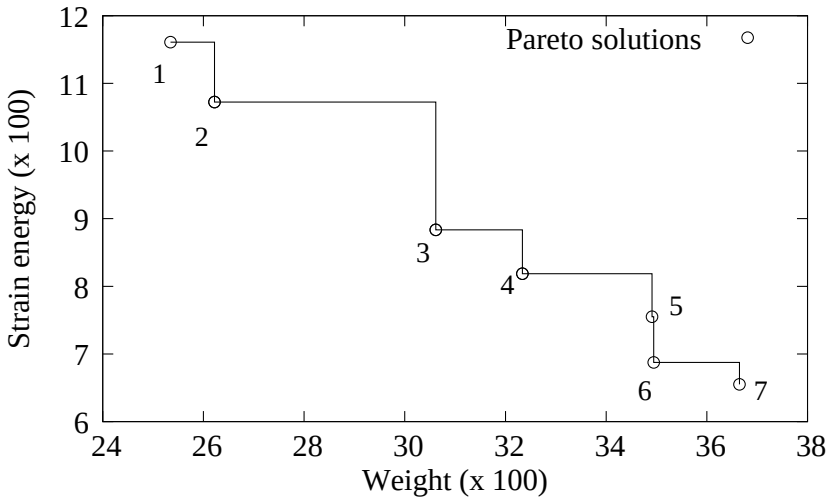
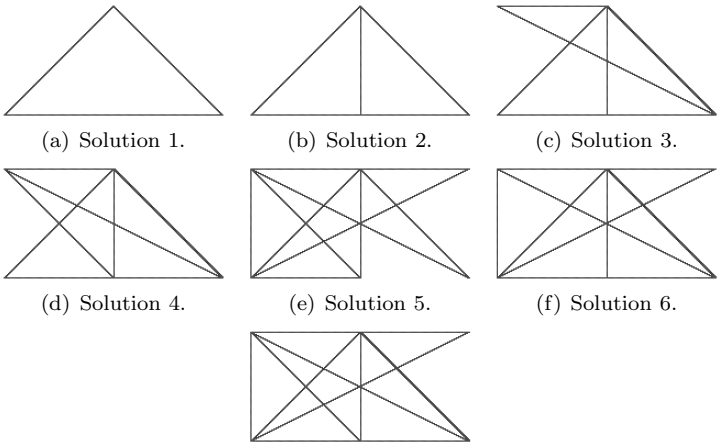


Figure 16: The approximate Pareto-optimal solutions for a plate design supported from both ends.

It can be seen from the figure that solution 1 is the simplest and lightest solution which has one triangle joining the nodes 1, 3, and 5. Solution 2 becomes stiffer due to a stiffener between nodes 2 and 4, but makes it heavier than solution 1. For other solutions, similar observations can be drawn here that inclusion of triangles and stiffeners make the structures stiffer which store less strain energy but increase their weight. In this case, study topology of solution 1 shown in Fig. 17(a) exists in all solutions, except solution 5. Apart from



(a) Solution 1. (b) Solution 2. (c) Solution 3.

(d) Solution 4. (e) Solution 5. (f) Solution 6.

(g) Solution 7.

Figure 17: Topology of Pareto-optimal solutions for a plate design which is supported at both ends.

solution 1, two triangles of solution 2 of Fig. 17(b) are also present in all solutions, except solution 5.

Progress of non-dominated solutions is shown in Fig. 18. For this case study also EA converged to the approximate Pareto-optimal solution in 40 generations only.

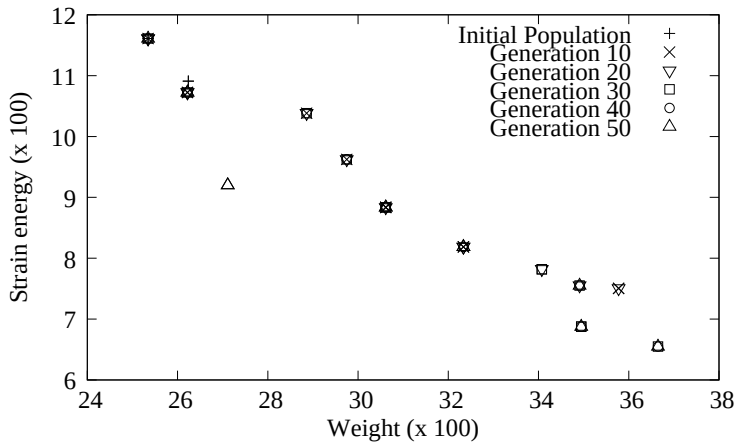


Figure 18: A progress of non-dominated solutions for 50 generations of EA for designing a plate supported at both ends.

Effect of different size of population for the present case study is shown in Fig. 19 for 0% attainment plot, and statistical values of hypervolume indicator are given in Table 5. The plot shows a good spread and proximity among the

approximate Pareto-optimal solutions of different populations sizes. Also, the best hypervolume values for different populations sizes are quite small and close to zero value. The same conclusion can be drawn here that a small population with triangular representation is able to show equivalent performance on the quality of solutions.

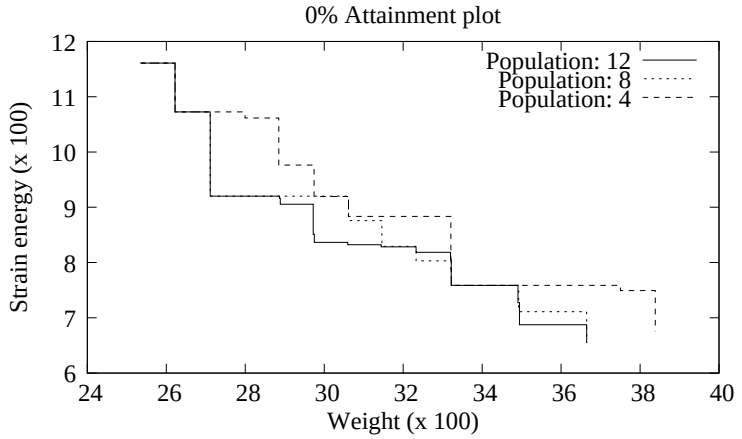


Figure 19: 0% attainment plots for different population sizes for second case study.

Table 5: Statistical values of hypervolume indicator for different population sizes for second case study. Indicator value -1 is best, and $+1$ is worst.

Population	Mean	Median	Standard deviation	Best	Worst
4	0.1843	0.2017	0.1514	0.0360	0.4638
8	0.0713	0.0712	0.0570	0.0102	0.2076
12	0.0360	0.0357	0.0224	0.0068	0.0850

The computation time of a typical run of matrix free CG method for finite element simulations is same as reported in the last case study. It means that the speedup of $5\times$ is achieved in this case study because number of elements in x- and y-directions is same for both case studies. The overall computation time and time taken by FE analysis are shown in Table 6. The computation time is also similar because most of the optimization time gets consumed in performing finite element simulations, while EA operators consume negligible time.

Table 6: Computation time in seconds for overall optimization process including FE computations on the GPUs for second case study.

Population	4		8		12	
GPUs	T. K40c	Q. K600	T. K40c	Q. K600	T. K40c	Q. K600
FEM time	3874.20	9329.09	7749.24	18914.18	11566.10	28383.44
Total time	3875.28	9430.17	7750.40	18915.32	11567.34	28384.64

6. Conclusion

In this paper, the triangular representation, crossover and mutation operators were proposed and coupled with NSGA-II for generating geometrically feasible structures. It can be concluded from the study that the triangular representation and EA operators proposed in this paper can generate the approximate Pareto-optimal solutions with a small population, and also in less generations. The approximate Pareto-optimal solutions showed trade-off between the two objectives by adding or deleting triangles and stiffeners for evolving diverse topologies of structures. The GPU computing was also performed to reduce computation time of the algorithm by performing FE computations on the GPU. It was observed that a large fraction of the total time was consumed by the FE simulations, whereas the time consumed by EA operators was negligible in comparison. Approximately 5× of speedup was achieved by using the GPU computing over the CPU. For future work this study can be extended to generate topologies for three dimensional structures.

Acknowledgment

We acknowledge the support from SERB, Department of Science and Technology (DST), India (grant # SB/FTP/ETA-28/2013) and IIT Guwahati under start-up grant scheme (SG/ME/DS/P/01). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40c GPU used for this research.

[1] M. P. Bendsoe, O. Sigmund, Topology Optimization: Theory, Methods and Applications, Springer, ISBN 9783540429920, 2004.

[2] M. P. Bendsøe, Optimal Shape Design as a Material Distribution Problem, Structural and Multidisciplinary Optimization 1 (4) (1989) 193–202.

[3] M. Y. Wang, X. Wang, D. Guo, A Level Set Method for Structural Topology Optimization, Computer Methods in Applied Mechanics and Engineering 192 (1–2) (2003) 227–246.

[4] C. D. Chapman, K. Saitou, M. J. Jakiela, Genetic Algorithms as an Approach to Configuration and Topology Design, ASME, Journal of Mechanical Design 116 (4) (1994) 1005–1012.

[5] G. I. N. Rozvany, Aims, Scope, Methods, History and Unified Terminology of Computer-Aided Topology Optimization in Structural Mechanics, Structural and Multidisciplinary Optimization 21 (2) (2001) 90–108.

[6] D. Sharma, K. Deb, N. Kishore, Towards Generating Diverse Topologies of Path Tracing Compliant Mechanisms using a Local Search Based Multi-Objective Genetic Algorithm Procedure, in: Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on, IEEE, 2004 –2011, 2008.

[7] A. R. Yıldız, K. Saitou, Topology Synthesis of Multicomponent Structural Assemblies in Continuum Domains, ASME Journal of Mechanical Design 133 (1) (2011) 011008–011008–9, doi:10.1115/1.4003038.

[8] D. J. Munk, G. A. Vio, G. P. Steven, Topology and shape optimization methods using evolutionary algorithms: a review, Structural and Multidisciplinary Optimization 52 (3) (2015) 613–631, ISSN 1615-1488, doi:10.1007/s00158-015-1261-9, URL http://dx.doi.org/10.1007/s00158-015-1261-9.

[9] A. Ahrari, A. A. Atai, K. Deb, Simultaneous topology, shape and size optimization of truss structures by fully stressed design based on evolution strategy, Engineering Optimization 47 (8) (2015) 1063–1084, doi:10.1080/0305215X.2014.947972, URL http://dx.doi.org/10.1080/0305215X.2014.947972.

[10] A. H. Gandomi, X.-S. Yang, A. H. Alavi, Mixed variable structural optimization using Firefly Algorithm, Computers & Structures 89 (23–24) (2011) 2325–2336, ISSN 0045-7949, doi:http://dx.doi.org/10.1016/j.compstruc.2011.08.002, URL http://www.sciencedirect.com/science/article/pii/S0045794911002185.

[11] M.-Y. Cheng, D. Prayogo, Symbiotic Organisms Search: A new metaheuristic optimization algorithm, Computers & Structures 139 (2014) 98–112, ISSN 0045-7949, doi:http://dx.doi.org/10.1016/j.compstruc.2014.03.007, URL http://www.sciencedirect.com/science/article/pii/S0045794914000881.

[12] G. R. Zavala, A. J. Nebro, F. Luna, C. A. Coello Coello, A survey of multi-objective metaheuristics applied to structural optimization, Structural and Multidisciplinary Optimization 49 (4) (2014) 537–558, ISSN 1615-1488, doi:10.1007/s00158-013-0996-4, URL http://dx.doi.org/10.1007/s00158-013-0996-4.

[13] D. Sharma, K. Deb, N. N. Kishore, Domain-Specific Initial Population Strategy for Compliant Mechanisms using Customized Genetic Algorithm, Structural and Multidisciplinary Optimization 43 (4) (2011) 541–554, ISSN 1615-147X.

[14] N. Wang, X. Zhang, Topology optimization of compliant mechanisms using pairs of curves, Engineering Optimization 47 (11) (2015) 1497–1522, doi:10.1080/0305215X.2014.977274, URL http://dx.doi.org/10.1080/0305215X.2014.977274.

[15] M. Emmerich, B. Naujoks, Metamodel Assisted Multiobjective Optimisation Strategies and their Application in Airfoil Design, in: I. Parmee (Ed.), Adaptive Computing in Design and Manufacture VI, Springer, London, 249–260, 2004.

[16] D. Sharma, K. Deb, N. N. Kishore, Customized Evolutionary Optimization Procedure for Generating Minimum Weight Compliant Mechanisms, Engineering Optimization 46 (1) (2014) 39–60.

[17] T. Zegard, G. H. Paulino, Toward GPU accelerated topology optimization on unstructured meshes, Structural and Multidisciplinary Optimization 48 (3) (2013) 473–485, ISSN 1615-1488, doi:10.1007/s00158-013-0920-y, URL http://dx.doi.org/10.1007/s00158-013-0920-y.

[18] M. Dehnavi, D. Fernandez, D. Giannacopoulos, Finite-Element Sparse Matrix Vector Multiplication on Graphic Processing Units, Magnetics, IEEE Transactions on 46 (8) (2010) 2982–2985, ISSN 0018-9464, doi: 10.1109/TMAG.2010.2043511.

[19] H. Kawamura, H. Ohmori, N. Kito, Truss Topology Optimization by a Modified Genetic Algorithm, Structural and Multidisciplinary Optimization 23 (6) (2002) 467–473, doi:10.1007/s00158-002-0208-0.

[20] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, Evolutionary Computation, IEEE Transactions on 6 (2) (2002) 182 –197.

[21] K. Tai, T. H. Chee, Design of Structures and Compliant Mechanisms by Evolutionary Optimization of Morphological Representations of Topology, ASME, Journal of Mechanical Design 122 (2000) 560–566.

[22] S. Y. Wang, K. Tai, Graph Representation for Structural Topology Optimization Using Genetic Algorithms, Computers & Structures 82 (2004) 1609–1622.

[23] K. Tai, S. Akhtar, Structural Topology Optimization Using A Genetic Algorithm with A Morphological Geometric Representation Scheme, Structural and Multidisciplinary Optimization 30 (2005) 113–127.

[24] A. Yildiz, N. Kaya, N. Öztürk, O. Alankuş, Optimal design of vehicle components using topology design and optimisation, International Journal of Vehicle Design 34 (4) (2004) 387–398, doi: http://dx.doi.org/10.1504/IJVD.2004.004064.

[25] A. R. Yıldız, Optimal Structural Design of Vehicle Components Using Topology Design and Optimization, Materials Testing 50 (4) (2008) 224–228, doi:10.3139/120.100880.

[26] I. Durgun, A. R. Yıldız, Structural Design Optimization of Vehicle Components Using Cuckoo Search Algorithm, Materials Testing 54 (3) (2012) 185–188, doi:10.3139/120.110317.

[27] A. Yildiz, N. Öztürk, N. Kaya, F. Öztürk, Integrated optimal topology design and shape optimization using neural networks, Structural and Multidisciplinary Optimization 25 (4) (2003)

251–260, ISSN 1615-1488, doi:10.1007/s00158-003-0300-0, URL http://dx.doi.org/10.1007/s00158-003-0300-0.

[28] N. Öztürk, A. Yildiz, N. Kaya, F. Öztürk, Neuro-Genetic Design Optimization Framework to Support the Integrated Robust Design Optimization Process in CE, Concurrent Engineering 4 (1) (2006) 5–16, doi:10.1007/s00158-003-0300-0.

[29] A. R. Yıldız, N. Öztürk, N. Kaya, F. Öztürk, Hybrid multi-objective shape design optimization using Taguchi's method and genetic algorithm, Structural and Multidisciplinary Optimization 34 (4) (2006) 317–332, ISSN 1615-1488, doi:10.1007/s00158-006-0079-x, URL http://dx.doi.org/10.1007/s00158-006-0079-x.

[30] A. R. Yıldız, A new design optimization framework based on immune algorithm and Taguchi's method, Computers in Industry 60 (8) (2009) 613 – 620, ISSN 0166-3615, doi:http://dx.doi.org/10.1016/j.compind.2009.05.016, URL http://www.sciencedirect.com/science/article/pii/S0166361509001353, computer Aided Innovation.

[31] A. R. Yıldız, Comparison of Evolutionary-based Optimization Algorithms for Structural Design Optimization, Engineering Applications of Artificial Intelligence 26 (8) (2013) 327–333, doi:http://dx.doi.org/10.1016/j.engappai.2012.05.014.

[32] A. R. Yıldız, A new hybrid particle swarm optimization approach for structural design optimization in the automotive industry, Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering 226 (10) (2012) 1340–1351, doi:10.1177/0954407012443636.

[33] nVidia, nVidia CUDA^{TM} Pragramming Guide Version 7.0, 2015.

[34] E. Wadbro, M. Berggren, Megapixel Topology Optimization on a Graphics Processing Unit, SIAM Review 51 (4) (2009) 707–721, doi:10.1137/070699822.

[35] S. Schmidt, V. Schulz, A 2589 line topology optimization code written for the graphics card, Computing and Visualization in Science 14 (6) (2011) 249–256, ISSN 1432-9360, doi:10.1007/s00791-012-0180-1.

[36] V. Challis, A. Roberts, J. Grotowski, High resolution topology optimization using graphics processing units (GPUs), Structural and Multidisciplinary Optimization 49 (2) (2014) 315–325, ISSN 1615-147X, doi:10.1007/s00158-013-0980-z.

[37] M. I. Frecker, G. K. Ananthasuresh, S. Nishiwaki, N. Kikuchi, S. Kota, Topological Synthesis of Compliant Mechanisms Using Multi-Criteria Optimization, ASME, Journal of Mechanical Design 119 (2) (1997) 238–245.

[38] J. N. Reddy, An Introduction to the Finite Element Method, Mc Graw Hill Education, New York, USA, 3rd edn., ISBN 978-0-07-060741-5, 2005.

[39] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, Chichester, UK: Wiley, first edn., 2001.

[40] D. Sharma, K. Deb, N. N. Kishore, A Domain-Specific Crossover and a Helper Objective for Generating Minimum Weight Compliant Mechanisms, in: Proceedings of the 10th annual conference on Genetic and evolutionary computation, GECCO '08, ACM, New York, NY, USA, ISBN 978-1-60558-130-9, 1723–1724, 2008.

[41] S. Y. Wang, K. Tai, M. Y. Wang, An Enhanced Genetic Algorithm for Structural Topology Optimization, International Journal for Numerical Methods in Engineering 65 (1) (2006) 18–44.

[42] A. Dziekonski, P. Sypek, A. Lamecki, M. Mrozowski, Accuracy, Memory, and Speed Strategies in GPU-Based Finite-Element Matrix-Generation, IEEE Antennas and Wireless Propagation Letters 11 (2012) 1346–1349, ISSN 1536-1225, doi:10.1109/LAWP.2012.2227449.

[43] C. Cecka, A. J. Lew, E. Darve, Assembly of finite element methods on graphics processors, International Journal for Numerical Methods in Engineering 85 (5) (2011) 640–669, ISSN 1097-0207, doi:10.1002/nme.2989, URL `http://dx.doi.org/10.1002/nme.2989`.

[44] NVIDIA, CUDA C Programming Guide, 5.5, Tech. Rep., NVIDIA Corporation, August 2014.

[45] C. M. Fonseca, P. J. Fleming, On the performance assessment and comparison of stochastic multiobjective optimizers, in: Proceedings of Parallel Problem Solving from Nature IV (PPSN-IV), 584–593, 1996.

[46] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, V. Grunert da Fonseca, Performance Assessment of Multiobjective Optimizers: An Analysis and Review, IEEE Transactions on Evolutionary Computation 7 (2) (2003) 117–132.

[47] J. Knowles, L. Thiele, E. Zitzler, A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers, TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 2006.

[48] E. Zitzler, L. Thiele, Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach, Evolutionary Computation, IEEE Transactions on 3 (4) (1999) 257–271.

[49] V. G. Fonseca, C. M. Fonseca, A. O. Hall, Inferential performance assessment of stochastic optimizers and the attainment function, in: Proceedings of the First Evolutionary Multi-Criterion Optimization (EMO-01) Conference, 213–225, 2001.

[50] C. Fonseca, V. Fonseca, L. Paquete, Exploring the Performance of Stochastic Multiobjective Optimisers with the Second-Order Attainment Function, in: C. Coello Coello, A. Hernndez Aguirre, E. Zitzler (Eds.), Evolutionary Multi-Criterion Optimization, vol. 3410 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, ISBN 978-3-540-24983-2, 250–264, 2005.