Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 67 (2018) 410 – 415

www.elsevier.com/locate/procedia

11th CIRP Conference on Intelligent Computation in Manufacturing Engineering - CIRP ICME '17

Evolutionary optimization of the failure behavior of load introduction elements integrated during FRP sandwich structure manufacturing

Jan Schwennen[a,*], Lukas Kalbhenn[a], Jérôme Klipfel[a], Jens Pfeifle[a], Daniel Kupzik[a], Jürgen Fleischer[a]

[a]*wbk Institute of Production Science, Karlsruhe Institute of Technology (KIT), Kaiserstr. 12, 76131 Karlsruhe, Germany*

* Corresponding author. Tel.: +49 721 608-41674; fax: +49 721 608-45005. *E-mail address:* jan.schwennen@kit.edu

Abstract

Due to their high lightweight potential, fiber-reinforced-plastics (FRP) sandwich structures with foam cores are becoming increasingly important in the automotive industry. To accommodate local forces, load introduction elements called inserts are required. A new type of insert is integrated into the sandwich panel during manufacturing. To accelerate the design and development process of these inserts, an optimization framework is developed based on the *Abaqus CAE* package and the *Python* programming language. An automated parametric nonlinear finite-element method (FEM) model is used to evaluate the performance of insert geometries. This model is integrated into an optimization routine based on a genetic algorithm. The population-based approach is very scalable through parallelization and shows accurate results within reasonable processing times.

Peer-review under responsibility of the scientific committee of the 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering

Keywords: Composite; Load introduction elements; FEM; Optimization; Evolutionary algorithm

1. Introduction

Efficient design plays an increasingly large role in the automotive and aerospace industries, mostly due to rising efficiency targets. One approach to weight reduction is the use of sandwich structures with foam cores and skins out of carbon-fiber reinforced plastics (CFRP) which provide high stiffness to weight ratios and favorable energy absorption capabilities. A weakness of these sandwich structures however is the difficulty of distributing point loads into the panel. The load transfer is generally improved by utilizing load introduction elements, called inserts, to create attachment points on the panel.

An approach to reduce the high manufacturing costs and decrease cycle times while maintaining structural performance by Schwennen et al. [1] is the use of foam cores that are foamed into a 3D shape in a separate tool. Metallic parts, such as threaded inserts, can be directly integrated during the foaming process. The design and performance of this type of insert is a current area of active research. A wide range of geometries can be integrated into the foam core between the CFRP skins, but because these inserts are not as established as other load transmission methods in the industry, very few standards and analytical tools exist for these geometries, making the design and development a very time-consuming process [2]. Current design approaches involve laborious parameter studies with multiple cycles of topology optimization and FEM analyses. Furthermore, the optimization techniques currently in use do not allow the direct integration of performance criteria such as failure loads or energy absorption into the optimization process. The goal of this project is to develop a framework for the automated optimization of a specific type of insert and to evaluate the utility of an evolutionary algorithm for this application.

The insert in question is comprised of a steel bolt surrounded by a separate insert structure. The load is introduced into the threaded bolt. The structure surrounding the bolt is “through-thickness”, i.e. connected to the inner surfaces of both face sheets. This work focuses on the optimization of the insert structure surrounding the bolt.

Peer-review under responsibility of the scientific committee of the 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering
doi:10.1016/j.procir.2017.12.234

2. Parametric FEM Simulation

The baseline geometry of the insert is a rounded dumbbell based on an optimized shape by Johansson et al. [3]. A nonlinear FEM model of the baseline geometry was developed by [4]. To enable automated optimization of the shape of the insert, a parametric representation of insert geometries is needed. The *Abaqus Scripting Interface* is used to generate an FE model from a set of parameters.

2.1. Finite Element Model

The FE model of the sandwich component is implemented in *Abaqus/CAE* and solved by *Abaqus/Explicit* [4]. The overall geometry of the model in Fig. 2 is equivalent to the test pieces used by previous work in this project [1]. The test specimens are square sandwich panels with an edge length of 148 mm. The face sheets (tan, green in Fig. 2) consist of unidirectional CFRP layers with a symmetric 0°/90° layup. The insert itself consists of a steel center bolt (blue in Fig. 2) surrounded by the insert structure (red in Fig. 2). A vertical tensile force is applied to the bolt and the sandwich panel is held down by a fixture with a circular opening Ø = 125mm. Note that the steel insert is not part of the design space and is not optimized. Within this work, "insert geometry" refers only to the insert structure around the bolt.

The individual parts of the assembly are created as volume bodies. Continuum shell (SC8R) elements are used to mesh the composite face sheets and 3D continuum elements (C3D8R and C3D6) are used for the rest of the parts. *General Contact* is used for the connections between the individual parts. The bonding surface between the foam core and the insert structure are modeled by a *Cohesive Surface* with damage. In order to avoid oscillations from sudden acceleration, the load amplitude is selected as *Smooth Step*. Due to the strong deformation of the foam, hourglass and distortion control are activated. Since the panel, insert, and load are symmetric across both the X and Y planes, the calculation time is reduced by simulating a quarter of the sandwich panel and applying the appropriate boundary conditions. Mass scaling is used to reduce the number of necessary computation steps. A short time increment leads to unnecessarily long computing times, while too large values result in an artificial increase in the computational density and influence of inertial forces on the simulation result [4].

2.2. Parametric Geometry

The parametric geometry is a simplification of the original dummell geometry. It consists of two triangles and a rectangle (Fig. 1). All geometries are obtained by varying five parameters: R_0, H_T, R_T, H_B, and R_B. The 2D shapes defined by the parameters are revolved 90° around the vertical axis to form 3D parts. These are then merged into the complete insert structure. The parameter R_0 defines the thickness of the trunk of the insert structure. R_T and R_B specify the maximum radii of the insert at the top and bottom face sheets, respectively.

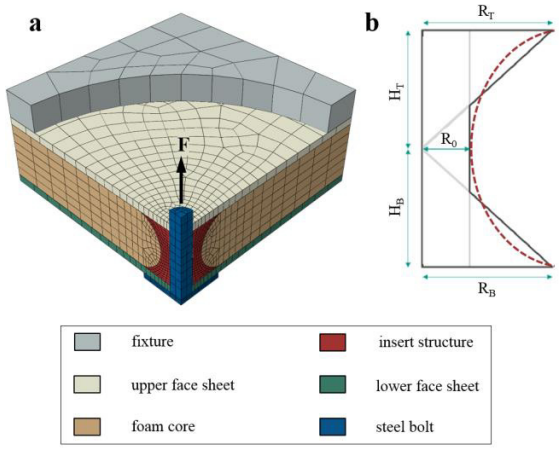

Fig. 1. (a) FEM model of test panel. (b) Parametric geometry.

H_T defines the height of the top triangle, measured vertically from the top face sheet. The height of the bottom triangle, H_B, is measured from the bottom. All parameters are completely independent, and can be varied between values of 1 and 30. The radius of the insert structure is therefore constrained to a maximum of 34 mm (radius of the steel bolt = 4 mm). The height of the insert is defined by the core thickness of 20 mm.

2.3. Model Generation

The *Abaqus Scripting Interface* allows models to be generated via *Python* scripting. The model generation script contains the same commands that a user would execute when building the model through the graphical user interface (GUI). With a script, however, the GUI does not need to be started and the model generation can be completely automated.

The script developed for this implementation reads the geometry parameters from a file in the working directory. The parameters define the 2D shapes. These are sketched and then revolved to create 3D parts. The other components of the model (see Fig. 1 (a)) are extruded. With the *Boolean Merge* function, the individual 3D part instances from the three base shapes are combined to form the insert structure (Fig. 2). *Boolean Cut* is used to cut the merged insert geometry out of the foam core. Once all part instances are created, the mesh, materials, boundary conditions, load steps, and interactions are defined. *If-then* statements handle the different cases that arise. Once the model is complete, an *Abaqus* Model Database (.cae) and an input file (.inp) are written (Fig. 3).

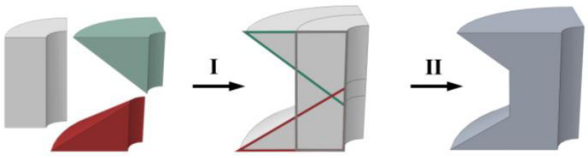

Fig. 2. Insert modelling: (I) assembly of part instances, (II) merging.

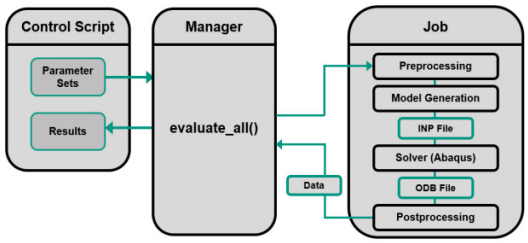

Fig. 3. Evaluation of FEM jobs through the Job and Manager modules.

3. Framework for Geometry Evaluation

For the evolutionary optimization process, a large number of individual insert geometries must be analyzed. Therefore, the entire FEM process is automated. An evaluation framework is implemented using *Python* 2.7.13 and the *Abaqus Scripting Interface*.

3.1 Structure and Functionality of the Framework

Each geometry is defined through a set of parameters as described in 0. From a parameter set, an *Abaqus* FEM job is generated as described in 0. The framework provides two main functionalities: automated evaluation of individual FEM jobs including preprocessing and evaluation of results, and a manager to handle parallelized evaluation of multiple jobs.

The evaluation framework is based on three *Python* modules, *Simulation*, *Hantel*, and *Misc*. The Simulation module contains two classes, *Job* and *Manager*. The *Job* class deals with the execution of individual FEM jobs and provides functions for preprocessing, model generation, execution, and postprocessing. A *Manager* controls sets of FEM jobs and provides a simple interface for parallelized job evaluation. The *Hantel* module is used to separate functions and definitions that are specific to the FEM model used in this case, such as analytical volume calculation. Any miscellaneous functions that are used in various places throughout the program (such as functions for cleaning up unnecessary files) are located in the *Misc* module.

Abaqus CAE provides a separate *Python* environment that includes additional modules for *Abaqus*-specific functions. Using functionality from any of the *Abaqus* modules, such as model generation or data extraction from the output database, requires a separate script that runs in the *Abaqus Python* environment. These are *generatemodel.py* (model generation from parameters), *gho.py* (extraction of history output from the output database), and *meshnap.py* (snapshots of the deformed model). The scripts are started from within the standard *Python* environment as needed.

3.2 Parallelized Evaluation of a Set of Jobs

Fig. 2 shows how an instance of the *Manager* class can be used to evaluate multiple geometries at once with a single command. A number of geometry parameter sets are passed to the manager, which creates an instance of the *Job* class for each geometry. These are stored in a job queue until the manager's *evaluate_all()* method is called. Then, the *Manager* prepares all jobs for evaluation. Through the *Jobs*' methods, working directories are created, the necessary files are copied, and the model generation script is executed.

Individual *Abaqus* processes are started to analyze the input file (INP) of each job. In contrast to the domain-level parallelization provided by *Abaqus*, no data is exchanged between the individual processes, as each process evaluates a separate job. This allows the number of parallelized job evaluations to be scaled linearly with the number of available processors, i.e. a machine with twice the number of CPU cores will be able to evaluate twice as many jobs in the same time.

After evaluation of all jobs by *Abaqus CAE*, the results of each simulation are stored in *Abaqus* output database (ODB) files in the working directories of the jobs. The manager then calls the postprocessing method of each job, which extracts the data from the ODB file by running the script *gho.py* in the *Abaqus Python* environment. The history data from the job is processed within the standard *Python* environment to determine the performance of the insert and the data is stored in the job object.

3.3 FEA Result Assessment

Functions were implemented to extract the performance data from the FEA results. The displacement and force at the reference node for each history output interval are available from the simulation. The largest force over the simulation set is defined as the maximum load. The first failure load is determined by a one percent drop in the reaction force. Any drops due to initial oscillations in the model are ignored, up to 100 N.

As mentioned in chapter 2.1, mass scaling effects can significantly influence FEA results, and must be monitored. For this quasi-static case, any kinetic energy in the simulation is mainly a product of high densities from excessive mass scaling. By visually comparing the internal (ALLIE) and kinetic energy (ALLKE) curves of the simulation, significant mass scaling effects become immediately apparent. However, due to the large number of simulations evaluated during an optimization, manual comparison of the curves is not practicable. To realize this method in the optimization, the kinetic energy history data is integrated over the model displacement. The resulting number value provides an assessment of the amount of kinetic energy over the course of the simulation. The influence was determined by evaluating an identical FE-model multiple times with various *target-time-increments*. The settings for the model used for the optimization were adjusted to minimize the influence of mass scaling in the simulations.

To decrease the runtime of the optimization, results are stored in a database across multiple runs. Non-unique parameter sets then do not need to be re-evaluated, as the results can simply be extracted from the database. This further reduces calculation times as more results are added to the archive.

4. Systematic Variation of Geometry

As a preliminary step before the optimization, the dimensions of the search space were investigated through a systematic variation of the geometry.

For this purpose, a pre-defined set of about 1500 parameter sets, spaced evenly over the search space, were generated. Because of the small number of geometries (1500 out of over 10^7 possible unique parameter sets), this method is not suited for optimization. However, it can be used to gauge the effect of different fitness functions applied to the data set. This proved useful in adjusting the weighting factors in the aggregated fitness function. Furthermore, the weight-specific failure loads of the insert geometries evaluated in the variation can be used as a benchmark to compare the results of the optimization runs.

Fig. 4 shows a scatter plot of the individuals evaluated in the systematic variation. The diagonal lines represent constant finesses with increasing values (higher failure load & lower mass) towards the lower left. The search space was covered more precisely in the area of low mass, and fewer heavier geometries were evaluated. A clear Pareto front (set of optimal geometries) is not visible, with some individual points scattered at higher fitness values outside of the main bulk. The systematic variation gives a good overview of the search space. However, a global optimization should be able to find better geometries between the scattered points on the lower left of the main bulk.

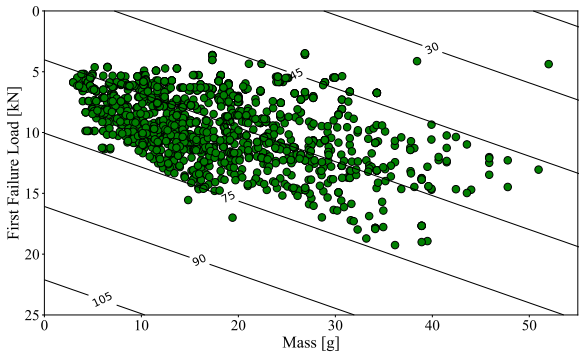

Fig. 4. Scatter plot of individuals evaluated in the systematic variation.

5. Optimization with an Evolutionary Algorithm

The goal of this work is to automatically optimize the failure behavior of the insert. The non-linear nature of the finite-element model disqualifies the use of topology-optimization techniques. Furthermore, the optimization problem is classified as a "black box" problem, as only the input (parameters) and corresponding output (failure and max. loads) are known. The global optimum is to be found. Evolutionary algorithms (EA) provide this capability, without requiring the approximation of any derivatives or prior knowledge of the problem. Another major advantage of the evolutionary method is the population-based approach, which lends itself very well to parallelized evaluation. An approach similar to that of Łodygowski et al. [5] is applied.

5.1 Implementation

To expand the simulation framework with the capability for optimization, an additional module is implemented within the *Python* package containing the framework. The module is adapted from an existing open source (GPL) *Python* package for genetic algorithms, called Galileo [6]. The module contains two classes, *Population* and *Chromosome,* which together provide the entire functionality of an evolutionary algorithm. The Population class represents a generation of individuals and implements methods to prepare the population and to execute each of the steps of the EA. A Chromosome object represents an individual in the population, i.e. the Population object contains and operates on a set of Chromosome objects.

To perform the optimization, the optimization module is imported into the control script. The evolutionary process is started by instantiating an object of the Population class, creating an initial population, and then calling methods on the population for each step of the algorithm: *Evaluate*, *Select*, *Recombine*, *Mutate*, and *Replace*. The algorithm's goal is to maximize the overall fitness of the population. Evaluation of the individuals is accomplished using a Manager object from the Simulation class (as described in 3.2). The results are translated to fitness values using an aggregated fitness function (see 5.2). The fittest genes are selected for recombination. Offspring are generated and mutated, all of the offspring replace the parents in the current generation, and the process starts again with evaluation.

5.2 Fitness Function by Aggregation

The performance of individual geometries in the FEM simulations are translated into a fitness for the evolutionary algorithm by means of the fitness function. The aggregation method was chosen to take into account the first failure load and the volume of the insert (Eq 1). The failure load of an individual, $F(x)$, is scaled to a value between 0.0 and 1.0 through division by 20.1 kN, the highest failure load observed in the systematic variation. The volume of the insert structure $V(x)$ is scaled to [0.0, 1.0] by the maximum volume allowed by the constraints on the insert geometry. The largest possible insert geometry has a volume of roughly 72 cm³.

$$f(x) = w_F * \frac{F(x)}{20.1kN} + w_V * \left(1 - \frac{V(x)}{72cm^3}\right) \quad (1)$$

The weighting factors w_F and w_V specify the relative importance of each objective over the other and are defined so that $w_F + w_V = 100$. With this fitness function, the highest possible finesses are at very high failure loads and volumes that are as close to zero as possible.

5.3 Selection Methods

In the selection step, the fittest individuals are chosen for recombination. The framework implements three selection methods: fitness-proportional selection, ranked selection, and elite-ranked selection. In fitness-proportional selection, the probability of selection is directly related to the individual's fitness in proportion to the total fitness of the population. To prevent issues with selection bias, ranked selection is also implemented, in which individuals in the population are sorted from worst to best and assigned a rank. Selection is then performed based on these ranks instead of individual fitness values. Thus, the probability distribution is uniform, and extremely fit individuals are prevented from taking over the population [7]. The third selection method implemented in the framework is called elite-ranked selection. This functions identically to ranked selection, except that only a certain number of individuals are eligible for selection. This method combines the relatively even selection probabilities of ranked selection with a strong pressure toward more fit individuals.

5.4 Recombination Methods

In the recombination step, parents selected with the selection function are combined to produce offspring. The two methods implemented in the framework, uniform and flat crossover, produce one offspring from two parents. Uniform crossover simply chooses the parameter values for the offspring randomly from either parent. Each parameter in each parent has a chance of 50% of being passed down to the offspring of the parents. Flat crossover, also called arithmetic crossover, creates an offspring that is a linear combination of the parents' features. The bias of the combination is determined by a uniform random number generated for each parameter in the chromosome.

5.5 Mutation Methods

Mutation is performed on a certain percentage of the offspring derived from the parents in order to introduce genetic diversity into the population and allow the EA to explore new areas of the search space. It also prevents inbreeding and evolutionary dead ends, which are more common in EAs due to the relatively small populations. The framework implements two types of mutation: random and uniform. Random mutation is the simplest method. After the recombination step, with a probability specified by the mutation rate, each gene of each individual is reinitialized to a random value within its allowable range. This introduces diversity, but does not allow tuning of the mutation range. In other words, a single mutation can place the mutated individual very far away from the un-mutated individual in the search space. The uniform method allows the range of the mutation to be limited by introducing a parameter for the magnitude of the mutation, called the *mutation range*.

6. Results

The following sections give an overview of the performance and results of the optimization runs that were performed over the course of this work.

6.1. Algorithm Configuration

As the evolutionary approach is based on randomly generated numbers (non-deterministic) convergence speed and results may vary between runs with identical configurations. In order to assess the effect of different algorithm setups, each parameter setting was tested a minimum number of three times.

The elite ranked selection method proved useful for the application, resulting in significantly faster convergence. The adverse effect of a decreased exploration of the search space can be compensated by carrying out multiple optimization runs. Overall the influence of elite selection is rated beneficial for the present application.

Additionally, uniform mutation (described in 5.5) with the mutation range set to 30% of the total parameter range was tested. A strong increase in convergence speed could be observed while maintaining consistently good results. However, as the effect on the exploration of the seach space could not be quantified and acceptable runtimes had already been reached, random mutation was chosen for the remaining runs. In a future use case, with more complex geometry parametrizations, uniform mutation with a specified range may have a useful effect.

6.2. Overview of Evaluated Geometries

Throughout all optimization runs, a total of more than 65,000 individual geometries were simulated. The solution space is visualized in Fig. 4, which shows the relationship between the objective function values (first failure & mass) and the fitness (diagonal contour lines). The well-defined outline of the scatter plot towards the lower left corner clearly shows the set of optimal geometries (Pareto-frontier). The evolutionary algorithm consistently returns the Pareto-optimal geometries for a set of fitness function weights.

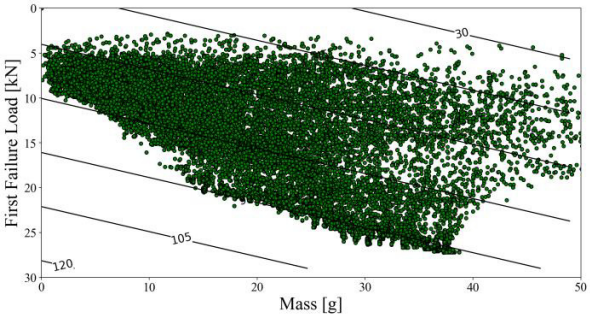

Fig. 5. Scatter plot of individuals evaluated throughout all evolutionary optimization runs.

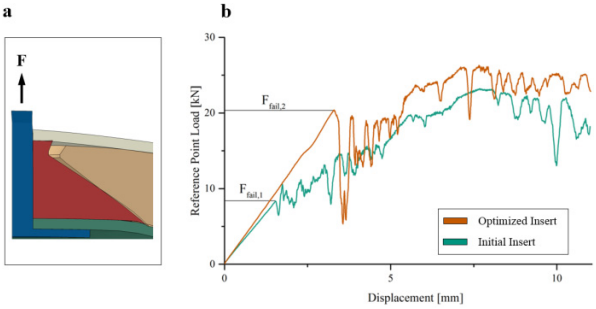

Fig. 6. (a) Exemplary optimized geometry. (b) Force-displacement plot of optimized and initial insert structures.

6.3. Resulting Geometry

An optimization of first failure load with only minimal consideration of dumbbell-mass (i.e. no penalty for heavy geometries) will result in an upward facing truncated-cone-shape with a 45-degree angle. This geometry results in advantageous stress conditions for the cohesive bond and therefore a delayed first failure. The maximum load is only slightly higher than the first failure load. An optimization of only the maximum failure delivers a dumbbell-insert with large top and bottom triangles to maximize the bonding surface between dumbbell and both upper and lower CFRP-layer. An early first failure is observed.

As both geometries are too massive for lightweight construction application, a compromise must be found using the fitness function. The cross-section of a typical optimized geometry with weighting factors of $w_F = 35$ and $w_V = 65$ is displayed in (a). Additionally the force displacement curves of the optimized geometries and the parametric geometry resembling the initial insert shape are shown in Fig. 6b.

When comparing the graphs, a drastic improvement in first failure performance can be noted. The optimized geometry shows a failure load of 20.4 kN compared to 8.32 kN of the original insert. Furthermore, the difference between first and maximum failure loads of the optimized geometries is much smaller, which demonstrates a better utilization of the load transmission potential in the FE-Model.

6.4. Mesh Verification

As the FEM-model had to be fully optimized regarding computation times, a comparatively coarse mesh, created by the A*baqus* automatic meshing algorithm, had to be used. To verify the simulation results, additional models of some of the optimized dumbbell-shapes were created manually with a fine mesh. The results show a maximum deviation of the first and maximum failure loads of ten percent between the models. This is deemed adequate, as relative evaluation is considered far more important for the purpose of an optimization than accurate failure load values.

Summary

The primary goal of this work was to enable the optimization of non-linear behavior of load-introduction elements in FRP sandwich panels. A software framework was developed in *Python* for this purpose. Automated model generation from a set up independent parameters is achieved with the *Abaqus Scripting Interface.* The evaluation of the FE models is fully automated and can be highly parallelized to greatly reduce runtimes for sets of models.

An evolutionary algorithm was applied to drive the optimization process and was found to be well-suited for this problem. Furthermore, optimization runtimes of under 12 hours were achieved, while maintaining verifiable simulation results.

Further experimental validation and adjustment of the model is necessary to more accurately model different failure modes. Additionally, greater design freedom in the form of more complex geometries and non-rotation-symmetric dumbbell shapes should be implemented. This will most likely also require a more finely-tuned approach to the generation of the mesh of the insert structure and foam core.

Acknowledgements

At this point, we wish to extend out special gratitude to the Federal Ministry of Education and Research (BMBF) which has funded the presented work under Grant No. 03X3041S.

SPONSORED BY THE
Federal Ministry of Education and Research

References

[1] Schwennen, J., Sessner, V. & Fleischer, J., A New Approach on Integrating Joining Inserts for Composite Sandwich Structures with Foam Cores. 6th CIRP Conference on Assembly Technologies and Systems (CATS), 2016; Vol. 44, pp. 310-315.

[2] Fliegener, S., Johe, J., Design and Optimization of Load Application Elements for Thermoplastic Sandwich Structures in Automotive Applications. 11th International Conference on Sandwich Structures (ICSS-11), 2016.

[3] Johansson, H., The optimized post-fitted foam sandwich insert. 13th. European Conference on Composite Materials, 2008; pp. 1–7.

[4] Schwennen, J., Kupzik, D., Fleischer, J., Development of a Calculation Model for Load Introduction Elements Integrated During FRP Sandwich Structure Manufacturing. Accepted for SAMPE Seattle 2017.

[5] Lodygowski, T., Szajek, K., Wierszycki, M., Optimization of dental implant using genetic algorithm. Journal of Theoretical and Applied Mechanics, 2009; Vol. 47, pp. 573-598.

[6] Goodman-Wilson, D., Galileo: a Distributed Genetic Algorithm, https://github.com/DEGoodmanWilson/Galileo Accessed: 2017-02-19.

[7] Whitley D., The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. 3rd International Conference on Genetic Algorithms, San Fransisco, USA, 1989; pp. 116-123.