Genetic programming for fiber-threading for fiber-reinforced plastics

Jonas Wilfert, Simon Stieber, Frederik Wilhelm, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Wilfert, Jonas, Simon Stieber, Frederik Wilhelm, and Wolfgang Reif. 2021. "Genetic programming for fiber-threading for fiber-reinforced plastics." In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vasteras, Sweden, 7-10 September 2021*, edited by Moris Behnam, Lucia Lo Bello, Thomas Nolte, and Marco Porta, 1–8. Piscataway, NJ: IEEE. https://doi.org/10.1109/etfa45728.2021.9613726.

Genetic Programming for Fiber-Threading for Fiber-Reinforced Plastics

Jonas Wilfert*, Simon Stieber†, Frederik Wilhelm‡ and Wolfgang Reif†
*, † Institute for Software & Systems Engineering—University of Augsburg, Germany
*jonas.wilfert@uni-a.de †{stieber; reif}@isse.de
‡ Fraunhofer Institute for Casting, Composite and Processing Technology IGCV—Augsburg, Germany
‡ frederik.wilhelm@igcv.fraunhofer.de

***Abstract*—Setting up fiber-threading for a pultrusion line is tedious, error prone and takes a long time. Between 100 and 1000 fibers have to be arranged into a two-dimensional shape, which have to be threaded between several support plates without causing crossovers. When manually planning this process based on intuition, it is hard to keep track of the complexity. This slows the process down to where it can take several hours or several days, and shortening this duration reduces the cost considerably. As planning the setup takes up a large chunk of time, we are proposing a simulation and an algorithm to automatically calculate how the fiber bundles need to be threaded from the creels through the support plates to result in the desired shape. Using a three-dimensional simulation for collision detection in conjunction with a genetic algorithm, we are able to shorten the planning of the fibers to around 10 minutes on a modern 8-core personal computer. Based on this data, further work can be done to further improve, visualize or permanently store the data in a digitized company.**

***Index Terms*—pultrusion, fiber-threading, genetic algorithm, fiber-reinforced plastic**

I. Introduction

Long set-up times delay production and increase the cost of the final product, and the pultrusion of fiber-reinforced profiles is no exception. Neither the pultrusion line nor the staff can be used profitably during this time. Before the fibers can be inserted into the heated die, the individual *rovings* (fiber bundles) must be arranged in a shape vaguely representing the final profile without intersecting. Preparing the plan manually takes from several hours up to an entire business day, preventing the employee from getting other work done. This uneconomical ratio is even worse where frequent changes of the profile type are required due to small lot sizes. A considerable improvement can be achieved by an algorithmic solution.

A. Pultrusion

The pultrusion process is an industrial manufacturing process in which fiber-reinforced plastic profiles are produced. Continuous fibers, mostly glass or carbon fibers, are used in this process. For this purpose, the prepared fiber package is impregnated with different resins, which are subsequently cured to obtain a solid composite material. Figure 1 provides an overview of a pultrusion line, the individual steps of which are explained in the following. Before the fibers are drawn through the resin bath (2), they must be arranged in the shape of the desired profile. To achieve this, the fiber bundles are threaded from the racks through several support plates (a) to (c) in the *fiber-threading* area (1). Between each pair of support plates several rovings are combined to achieve the desired density and arrangement. If the profile is to be reinforced by textile semi-finished products, such as glass fiber mats, these are also introduced here. Planning this part is the focus of our work and will be discussed more closely in the next section. Afterward, the previously arranged fiber package is pulled through the resin bath (2) to impregnate it with resin. The impregnated fiber package is then passed through a heated molding die (3) to allow the resin to cure into the final shape. Finally, the cooled profile is sawed off (5) to the desired length. The continuous movement of the fibers is ensured by the pulling unit (4). This moves the fiber composite through the line at constant speed. Due to the high degree of automation of a pultrusion line, profiles with similar mechanical properties can be produced more cost effectively than with other manufacturing processes for fiber-reinforced composites. Products from this manufacturing process are used, for example, in lightweight construction or in areas subject to high corrosion as a replacement for steel beams. [1], [2]

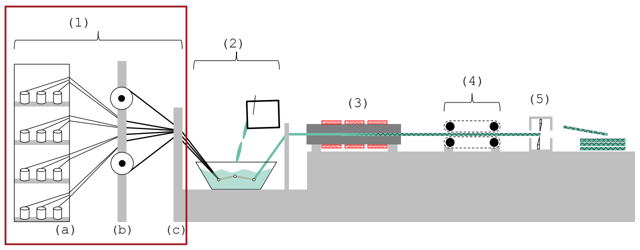


Fig. 1: Structure of a pultrusion line

B. Fiber-threading

In this work, the focus is set on the fiber routing framed in red in Fig. 1. The number of rovings required depends on the area of the profile and thus scales quadratically in the worst case when the profile is enlarged. Depending on the shape and complexity of the profile, between 100 and several 1000 rovings are usually combined. Usually, up to 10 rovings are taken from the racks and combined into one joint strand in the subsequent plate. The most import thing is that the fibers are not crossed so that they are protected from abrasion. Such defects could lead to defects in the finished product and must therefore be avoided at all costs. To make the routing process easier for the line worker, the joined spools should lie next to each other on the rack. Rovings that are combined into one strand are also referred to as a *cluster* in the following.

Often the fiber routing is planned in advance to make execution at the machine as trouble-free as possible. Currently, this is mostly done by hand using either an Excel spreadsheet or pencil and paper. This process is very tedious and error prone. Since two dimensions are insufficient for realistically representing the positions relative to each other, overlaps can only be estimated. This results in a error rate of up to 10%, which are undetectable until the fibers are being set up at the plant. Manually creating a plan can take several hours up to a several business days. In this work, we present an algorithmic solution for solving these mentioned problems and to speed up the process preparation.

C. Requirements

To be able to ensure the quality of a fiber-threading plan, the requirements have to be defined. They have to cover both the feasibility in reality as well as ensuring the comfort of the worker at the plant. The following list is given in descending priority.

(1) **Prevention of overlapping fibers** When fibers overlap, abrasion occurs, which can lead to incalculable weakening of the final product and must therefore be avoided at all costs. (2) **Prohibiting punctured clusters** Even if the individual rovings do not overlap, it is possible for another roving to puncture the cluster, like intertwined fingers. In addition to increasing the difficulty of the execution, this increases the probability of abrasion in sagging fibers and should therefore be avoided as well. (3) **Ensuring close proximity of rovings within a cluster** By placing fibers in close proximity that belong to the same cluster, they can be grabbed together and routed to the next plate, resulting in a simplified execution at the plant. (4) **Ensuring short fiber distances between support plates** Short fiber runs ensure that fibers stay in a the same area on the target plate as they did on the source plate. This means that fibers are less likely to run across other clusters, reducing the likelihood of collisions.

D. Abstraction

To simplify the pultrusion line, we represent the fiber-threading area with simple geometrical shapes. As such, we view each support plate as a two-dimensional plane, referred to as *container* in the following. The slots in the support plate are simplified to a set of points located on the plane, called the *elements*. Fibers connecting two plates can be abstracted to a line-segment in three-dimensional space. Using this abstraction, we can focus solely on the necessary data required to prepare a fiber-threading plan.

II. Related Work

No previous attempts at automatically creating a fiber-threading plan were found in the literature. However, the problem can be broken into smaller parts where significant work has been published. It boils down to finding clusters of a defined size in a set of points while the physical limitations of the rovings have to be considered. Therefore, the solution can consist of the following parts: Fiber clustering, collision detection and an algorithm to combine these two.

A. Clustering

Clustering the fibers as described in Section I-C is required to easily replicate the plan at the pultrusion line. A commonly used algorithm to combine a arbitrary set of points into clusters is k-means [3]. Using an iterative approach k-means separates the point-set into k different clusters of different sizes. To improve the performance and increase the ability to respect further factors multiple publications have been published which use a genetic algorithm to enhance the k-means algorithm [4], [5]. This can be used as a basis to the clustering part of our problem.

B. Collision Detection

Using a physics simulation as a basis for an algorithm is commonly used in actual practice. Since most ready-to-use simulation tools or game engines are generally optimized for dynamically moving objects, they cause a undesired overhead in our context [6]. Looking at a lower level there are further libraries implementing collision detection like ncollide[1] on a more basic level which can be enhanced with state of the art triangle intersection algorithms like the Möller-Trumbore-Algorithm [7] or the recently published algorithm by Baldwin and Weber [8].

C. Algorithm Selection

Selecting the algorithm is a critical step that influences the result as well as the duration the solution needs to be computed. One might think about using a clustering algorithm like k-means, but it is unable to deal with collisions in the three-dimensional space without sufficient modifications. Another failed approach consisted of using a kd-tree [9] to split the rovings into equal sectors. Choosing the closest sector with enough free rovings ensures the correct clustering of the coherent rovings. However, it is unable to respect three-dimensional collisions nor can it deal with distances e.g. between multiple rovings.

Albeit loosing the fast performance from the solutions above an iterative approach allows for more control. Optimizations can lead to a higher degree of freedom by using different fitness function to incrementally improve the solution. To assist in the selection of a fitting type of algorithm Haupt et al. [10] provides an auxiliary metric called epistasis: it looks at how interdependent the variables are, meaning how much a

[1]Sebastien Crozet, ncollide: ncollide is a 2 and 3-dimensional collision-detection library written with the rust programming language. Available: https://ncollide.org

Variable B will change when a different variable A changes. At low values, a minimum-seeking algorithm such as gradient descent is recommended. At the higher range of spectrum, random search is the only feasible solution. The middle range is where the genetic algorithm shines. Each fiber position can be viewed as an independent variable. This led us to choose a genetic algorithm.

In conclusion, we used physics simulation in conjunction with a genetic algorithm [10]–[12] inspired by biological evolution. Operations such as selection, crossover and mutation are adapted from nature to solve an optimization problem using a high number of variables. To start off with, an initial population is randomly generated. Using a fitness function, the generated chromosomes are now evaluated and assigned scalar values for comparison. During the selection stage, suitable parents for the crossover operation are selected, which are combined into a new child generation. Additionally, those new children are randomly mutated to cover a larger range of possible solution. This process is repeated until a predefined termination condition is met, e.g., no more improvements in the fitness function. Our implementation of the operations are described more closely in Section VI.

III. Data structures

As described in Section I-D the fiber routing is subdivided into the individual plates, which are called *container* (cf. Fig. 2). Each of these containers stores its position and rotation using a *global offset* and a *normal vector* pointing in the pull-off direction. The containers are arranged in such a way that the pull-off direction is in positive z-direction. A uniform axis orientation ensures consistency between different imports from CAD files. To be able to refer to each container unambiguously, each one is given an integer identification number. The containers are further divided into *elements* to represent the slots in the support plates or roving creels. Storing the position of the element in a local two-dimensional coordinate system relative to the respective container allows the global position to be determined by coordinate transformations.

When preparation an experiment, the initial setup of the fiber routing is mirrored in a digital twin. Each element stores either how many fibers are provided (*out-fibers*), consumed (*in-fibers*) or can be forwarded (*capacity*). These values only describe the initial configuration of the fiber routing and not how the fibers are ultimately routed, i.e., the capacity is only an upper boundary of fibers and does not necessarily require a fiber to pass through that element.

Since it's frequently the case that it can only be determined during the ongoing test how many fibers are required at which position, the concept of *excess fibers* was introduced. The algorithm should treat these teh same way as *in-fibers*, but are only held in reserve during system preparation when they are not yet threaded together with the other fibers. By already reserving rovings by the algorithm for specific slots, subsequent additions of excess fibers are simplified.

Rovings that run between containers are represented as a path between two elements. Here, each connection between two containers is considered a single segment. These line segments can be used to determine whether the fiber routing plan is feasible by performing collision checks. Additionally, checks for collisions with the frame of the pultrusion line could be checked with several bounding boxes, such as *Oriented Bounding Boxes (OBB)* [13]. This representation is well suited for storing the complete fiber routing line, however the genetic algorithm needs additional information to work with. Therefore, an extended high-performance data structure is presented that focuses exclusively on the interaction between two fiber guide plates. In the following, this structure will be referred to as *graph* and will be used as the chromosome of the genetic algorithm.A visual representation can be seen in Figure 2.

From the source plate, each element, i.e. each slot, is converted into a *target*. This stores the position of the element and the number of rovings that can be provided. In the case of a *source*, such as a fiber rack, this number is usually 1 as noted in the *out-fibers*. In the case of a *router*, such as a support plate, the maximum *capacity* of simultaneous rovings that can be fed through one of the slots is stored. These elements do not have to be completely filled, but must not exceed the maximum number. A *cluster* is assigned each element on the target plate. A cluster consists of any number of *nodes* and one *fixpoint*. The fixpoint stores the the coordinate of the element on the target plate to which the nodes should converge. Nodes are not static and do not have a fixed position. However, they can reserve a target and inherit its position. Only one node can be assigned per target at a time. Dynamic nodes are necessary to allow continuous changes during the calculation of the fiber routing.

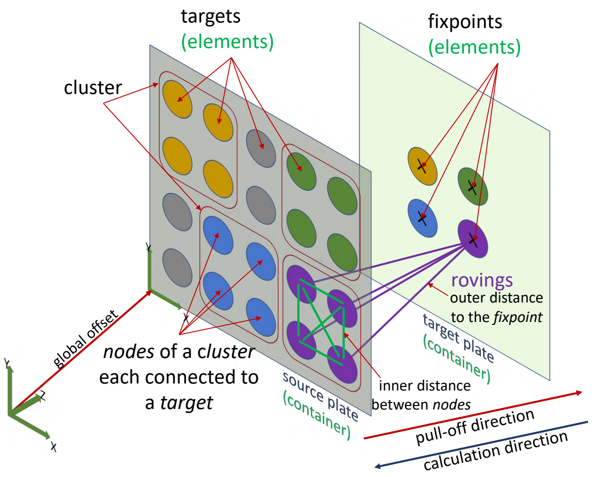


Fig. 2: Multiple cluster displayed in 3D

Like the targets, the number of nodes in a cluster depends on the type of the source plate. If it is a router, the necessary number of rovings is stored entirely in a single node, as they are threaded like a strand. In the case of a source, one node is created per consumed roving, so that different targets can be assigned to each roving. In general, this corresponds to the minimum of the sum of the *capacity* and the *out-fibers*.

More vividly, it can be explained as follows (cf. Fig. 2): The elements of the target plate (*fixpoint*) are connected with multiple rovings (*nodes*) to the source plate (*target*) and are supposed to be spatially close to each other (*cluster*). This entity forms a *graph*. The data structure presented here only considers the interaction of two containers, but the number of containers connected in series is arbitrary. To overcome this limitation, several of these graphs can be computed one after the other. Here, we start at the last target plate directly in front of the die. The distribution of the nodes on the targets of the original plate can be used as the target plate for the next container, so that the container connections are calculated in pairs, against the direction of pull-off, until finally the fiber source is reached.

IV. Degrees of Freedom

If a graph is created from two containers, they are scaled to the same size along the XY plane. It was observed empirically that only the position and angles of intersections change, but the intersections are not resolved, so that it becomes clearer how the fibers are routed. Furthermore, this allows performance optimizations using a bounding volume hierarchy on non overlapping bounding boxes [9], [14]. Allowed operations in this case are scaling and translating the fiber guide plates, but not rotating (Figure 3).

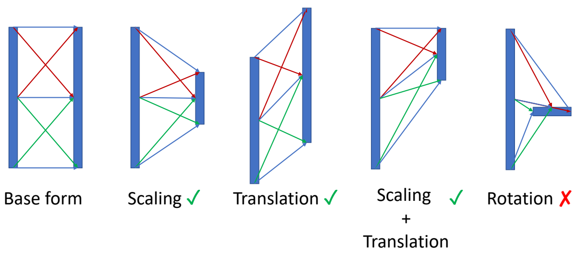


Fig. 3: Scaling and translation of a support plate in 2D

V. Fitness Functions

As described in Section II-C, a genetic algorithm uses a scalar fitness score to compare multiple different genomes. However, one fitness function is not enough to ensure that all requirements are respected in the final result. We present 4 individual cost functions, which are summed up after being multiplied by a weight parameter to form the combined fitness function. These weights represent the importance of each cost function and balance out differences in the units of measurement used. In this work, we refer to cost as the inverse of the fitness. While a higher fitness is better, the costs have to be decreased to improve the result.

A. Inner Distance

To be able to guarantee the proximity of the nodes of a cluster (requirement 3 from Section I-C), the Euclidean distances between all nodes are calculated. In Figure 2, these connecting lines are shown in green. Before summing the individual values, they are potentiated by a configurable factor to weight short distances exponentially more. The result is normalized using the *triangle number* to prevent the calculated cost from being dependent on the number of nodes. Through this number series, the number of interactions between the nodes can be determined. This number can be directly calculated by the Gaussian summation. By starting the inner sum at $i + 1$ we can avoid measuring the interaction between two nodes twice as well as skipping self interactions. The final formula for the cost of a cluster C corresponds to:

$$cost_{inner}(C) = \frac{\sum_{i=1}^{n} \sum_{k=i+1}^{n} d(N_i, N_k)^p}{\frac{n \cdot (n+1)}{2}} \tag{1}$$

Where N_i are the nodes of the cluster and p is an exponential factor.

B. Outer Distance

To keep fiber distances as short as possible and to ensure that nodes in a cluster are as close as possible to their fixpoint, the distance is also taken into account in the weighting. For this purpose, the Euclidean distances between the fixpoint and the positions of the respective nodes are measured. These connections are shown in purple in Figure 2. Like the previous section, this value is also normalized by the number of nodes. Leading to the following equation:

$$cost_{outer}(C) = \frac{\sum_{i=1}^{n} d(N_i, F)^p}{n} \tag{2}$$

Where N_i is the nodes of a cluster, F is the fixpoint, and p is the exponent.

C. 2D Angle

If no overlaps occur between two rovings, it is nevertheless disadvantageous if they cross in different directions. By projecting the fibers from three-dimensional space onto a two-dimensional plane, the angles of the now crossing fibers can be checked. Visually speaking, the rovings are viewed from a bird's eye view (XZ plane) as well as a frontal view (XY plane). The angles between the two vectors a, b in the respective planes can be calculated by the following formula:

$$\alpha_{XY}(a, b) = \arccos \frac{a_x \cdot b_x + a_y \cdot b_y}{\sqrt{a_x^2 + a_y^2} \cdot \sqrt{b_x^2 + b_y^2}} \tag{3}$$

If this angle exceeds a configurable threshold simultaneously with a collision of the vectors in the respective projection, the total cost is increased. When disregarding this condition the rovings would have to be pulled over or under already threaded fiber packages.

D. Triangle Check

Checking for triangular collisions is the most relevant metric for implementing threading at the plant. This part of the fitness function is strongly aligned with the problem formulated in Requirement 2: preventing punctured clusters. It can be represented by a mathematical cost function with triangle-line collisions, where the interior of the cluster is treated as a triangle and the rovings as line segments. With the target positions of two rovings as well as the fixpoint of the cluster a triangle can be spanned. The edges of the triangle thus form the two rovings leading to the two nodes as well as an imaginary connection between these two nodes (cf. Fig. 4). To find out between which nodes a triangle has to span, a

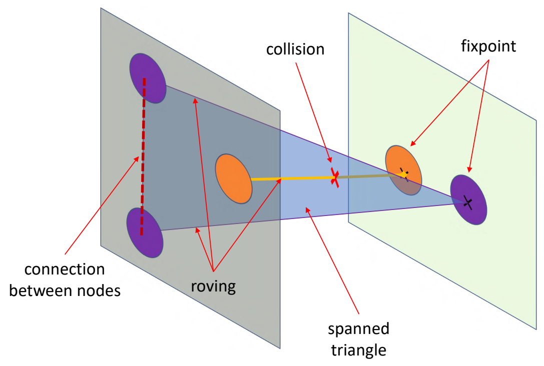


Fig. 4: Spanning of a triangle between the fixpoint and two nodes with a collision of a roving

hull is formed around the cluster. Thus, only the triangles of a simplified shape must be calculated. The Graham scan yields an algorithm that can compute the convex hull in a worst case runtime of $O(n \cdot \log n)$ by iterative angle checking over all nodes [15]. Unfortunately, the algorithm is only defined for 2-dimensional space. However, by projecting the position vectors of the nodes on to a hyperplane orthogonal to the normal vector, the the application of the algorithm is possible under the assumption of a flat support plate. Often, the simple case occurs where the support plate is parallel to the XY plane, i.e., along the standard orientation without rotation. In this case, the projection simplifies to a vector with only the x and y components. The algorithm forms the outline of the cluster along of which edges the triangles can be spanned. All collisions with edges of inner nodes excluded by the Graham scan are also covered by the connections of the contour, which means that $n+1$ triangles will have to be checked in a worst-case, instead of $\frac{n \cdot (n+1)}{2}$ if pair-wise interactions had been employed.

E. Clustering

In many cases, the *inner distance* is insufficient for determining whether a fiber packet is arranged as a cluster. The change of two directly adjacent nodes only leads to a very small change in total cost, which can be overshadowed by a simultaneous degradation of another cost function, such as the *outer distance*. To support this metric, we additionally determine how well it is clustered by counting the nearby nodes of foreign clusters. Here the opposite direction of well-known clustering algorithms as *k-means* is considered: checking clusters that have already been formed instead of finding related clusters in a set of points. In addition, restrictions to prevent collisions must be observed here, which are difficult to integrate directly into k-means. Despite the limitations, the algorithm presented here strongly resembles k-means [16] in terms of dividing the space into segments.

To quantify clustering, as shown in Fig. 5, a node $N_{central}$ is selected around which the space is divided into 4 segments, each with 90° opening angle. Now, individually for each segment, the nodes are sorted in ascending order by Euclidean distance to the central node. Additional costs are imposed if a node with a foreign or without a cluster occurs in this sorted list before all nodes of the own cluster have been processed. Said procedure is repeated for all nodes in each cluster and the costs are added up. This can be implemented efficiently using an additional variable for each segment as described in Algorithm 1. The variable acts as a memory m_i that gets committed to the final *cost* when coming across a node of the own cluster. Thus, if the total cost of this cost function reaches zero, according to this metric, all nodes have arranged themselves in clusters.

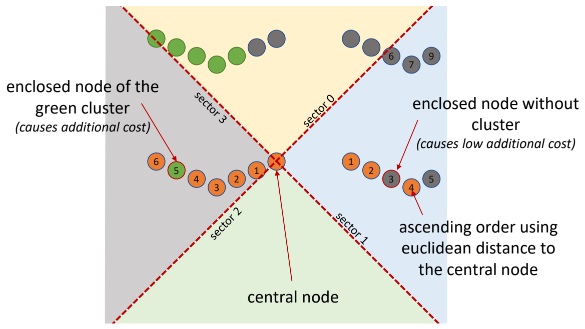

enclosed node of the green cluster
(causes additional cost)
enclosed node without cluster
(causes low additional cost)
sector 3
sector 0
sector 2
sector 1
ascending order using euclidean distance to the central node
central node
1
2
3
4
5
6
7
9

Fig. 5: Quantification of the clustering using 4 segments with each 90° opening angle

Algorithm 1: Clustering Cost-Function

```
Result: cost
1  $cost \leftarrow 0$;
2  foreach $N_{central} \in nodes$ do
3      $m_n \leftarrow 0 \; \forall n \in \{0,1,2,3\}$;
4      foreach $N_{other}$ in $nodes$ sorted by distance to $N_{central}$ do
5          $s \leftarrow$ index of sector of $N_{other}$;
6          if $cluster(N_{central}) = cluster(N_{other})$ then
7              $cost \leftarrow cost + m_s$;
8              $m_s \leftarrow 0$;
9          else
10             $m_s \leftarrow m_s + 1$;
```

VI. Genetic Algorithm

A. Selection

Based on fitness, it is possible to compare graphs pairwise. However, to choose which graphs to use for the crossover, a selection procedure is needed that selects two elements from the weighted set as parents. There are different ways to accomplish this task. E.g. *Weighted Order* selects parents weighted by position in the list sorted by fitness. Similarly *Upper Half* selects a random element in the upper half of the sorted list while ignoring the fitness score. *Weighting/Roulette*

Wheel assigns each element a probability based on the share the fitness of the element accounts of in the total sum. Comparable to a roulette wheel, the element is now selected at random based on the probability distribution. When using *Tournament Selection* n elements of the unsorted list are selected at random of which only the best element is kept. In a comparison between several selection methods by Shukla et al. [17] *Tournament Selection* turned out to be the best method, both in performance and convergence behavior.

In this work, a combination of both *Upper Half* and *Tournament Selection* methods was used. It was decided to include only the top 50% of the population in the Tournament Selection, as a single mutation can cause very large changes in fitness, leading to the possibility of getting very poor results. The calculation time of the fitness function is several orders of magnitude higher than the duration spent in selection, there the performance of the selection method is negligible.

B. Crossover

During crossover, two chromosomes, in this case *graphs*, are merged into one child that adheres to the following procedure. The number of clusters and nodes is already fixed from the beginning and cannot change during runtime. This allows both graphs to be traversed in parallel and is guaranteed to have a partner for each node.

The process begins with an additional empty graph, the *child graph*. The positions of each node are transferred to it during the crossover. For each pair of nodes that are processed, a random number is generated that is converted to a boolean value. These are used to determine from which of the two parents the position of the current nodes should be carried over. We use quasi-random number sequences instead of pseudo-random numbers to both increase the probability of a better solution as well as increasing the performance, as described by Shuhei and Matsumura [18], [19]. Now, if one of the two nodes of the parents has been selected, it must be checked whether the position on the third, the child graph, is already occupied. By different assignment of the parents nodes, it is possible that the position of a target is requested twice. In this case, the nearest possible free fiber target is selected. This guarantees that no erroneous graphs are created where targets are assigned more than once. By selecting a nearby target, a degree of clustering can still be retained. This process is repeated until all nodes of both parents have been transferred to the child.

C. Mutation

Random mutations are used to introduce certain changes with a low probability into the graphs. Mutations can either affect a node or a whole cluster. Nodes can either be swapped with an empty target nearby or swapped in place with a different node. Clusters are treated similarly and are either moved to a new free location in the graph or swapped with an existing cluster. If the clusters have a different number of nodes, the remaining ones are distributed to nearby free targets, similar to the crossover.

The cluster mutations can cause movements of entire clusters without violating requirement 3 (proximity of nodes within a cluster). In node-based operations, scattering nodes of a cluster often occurs, which could cause a temporary degradation of the overall fitness due to the cost functions *inner distance* and *clustering*. This makes it highly likely that this mutation will not be carried over to the next generation. To optimize the clusters, nodes must be preferentially swapped with nodes from clusters nearby, and a procedure similar to *Tournament Selection* is used to do this. n nodes or clusters are randomly selected, but only the closest of these are used for swapping. Thus, the probability for a resulting change is indirectly proportional to the distance.

D. Termination Condition

In order to determine when the computation of the algorithm is complete, a *primary termination condition* must be defined. As a genetic algorithm converges to some value the change in fitness between iterations can be used to determine how much the model has converged.If the difference falls below a certain threshold for an extended period of time, the algorithm is declared complete and can be terminated.

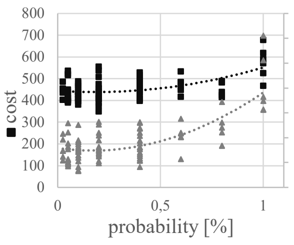
However, this is not the only termination condition present in the proposed algorithm. Since some errors are fatal for the functionality of the fiber-threading plan, parts of the fitness function can additionally be considered individually. Specifically, the cost function *triangle check* is relevant here. If errors of this type are still present in a plan, it cannot be executed reasonably in actual practice. It has been observed that errors in the triangle checks are strongly dependent on the initial population, which is why the concept of restart was introduced here. If triangle collisions are still present after a defined number of iterations, here 10000, the algorithm is restarted. In this case the complete population is deleted and randomly generated again. The number of iterations after which the restart occurs must be smaller than that of the primary termination condition. Therefore, it is impossible to complete the algorithm successfully with triangular intersections.

VII. Results and Discussion

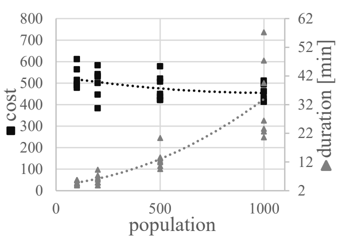
A. Hyperparameter Optimization

The developed algorithm provides a large number of parameters. These parameters can affect the runtime as well as the result of the algorithm, so they must be chosen wisely. Meaningful values can be achieved using hyperparameter optimization. A sufficiently complex tube-profile with over 500 fibers was selected for the hyperparameter optimization, as these offer the greatest challenge for the algorithm while still being computable in a reasonable amount of time (see Section VII-B).

a) Population Size: Population size is one of the two most important factors of the genetic algorithm along with the mutation chance. It determines how many chromosomes are created and evaluated in each generation. Since a genetic algorithm is based on randomness, the probability of obtaining



(a)



(b)

Fig. 6: Impact of the mutation chance (a) and population size (b) on the runtime duration and the final cost

a good mutation or crossover is higher with a higher population size. Furthermore, the algorithm starts with a higher variance in the initial population, which allows for a better result. However, execution time increases drastically as the computational effort increases proportionally to the population size. The effect of the population size on the computation time as well as on the total cost value is shown in the graph 6b.

b) Mutation chance: The second of the two most important parameters of a genetic algorithm is the mutation chance, which is the probability at which a random change is induced in the chromosome. The mutation chance is not as flexibly selectable as the population size, but ranges within a certain interval. Graph 6a shows that a lower mutation chance leads to a faster convergence with a lower final cost. Above a value of 2% it was no longer possible for the algorithm to complete successfully, resulting in termination after 100,000 iterations. Below a value of 0.025%, no result could be generated either, since the algorithm was trapped in an infinite loop due to the termination condition formulated in Section VI-D for triangle collisions. These could not be solved in time with such a low mutation chance before a restart was triggered.

The way the probability is applied explains this progression. Instead of triggering a single mutation somewhere on the whole graph with a high probability, a mutation is triggered per node with a very low probability. Thus, if the percentage is too high, multiple mutations may occur at the same time. In this case, an improvement in the fitness can be immediately overshadowed by a bad change. Unfortunately, the chance of mutation is consequently dependent on the number of fibers to be threaded. Hence, choosing a very small value is necessary for complex problems, but results in a slightly increased computation time for simple plans with few fibers.

B. Profiles

To ensure that the presented solution supports a broad spectrum of profiles, 3 different profiles are tested here. The algorithm is run with the following profiles and the result is then evaluated, with the type of profiles becoming increasingly complex.

- **Flat Profile** (50 Elements each with 10 Rovings)
- **U-Profile** (56 Elements each with 9 Rovings)
- **Tube-Profile** (100 Elements each with 5 Rovings)

Type	Duration for 500 Rovings	Stage 1	Stage 2
Flat	2:49 min	11%	89%
U	5:28 min	6%	94%
Tube	7:15 min	21%	79%

TABLE I: Durations for the different types of profiles

Creating the fiber-threading plan is divided into two steps. First, the final plate is mapped onto the support plate. Here, the fiber packages are not yet divided into the individual rovings but are threaded as a strand. The data created here is used for the second stage. There, the occupied elements of the support plate are threaded onto the racks. The fiber bundles are now divided into the individual rovings so that they can be taken individually from the racks. Thus, the second step is more computationally intensive and error prone due to the higher number of separate rovings.

C. Results

In the following, the calculations of the previous profiles are evaluated. Table I shows the duration the individual profiles took to complete both stages while running on an 8-core Ryzen 3700X. Figures 7 and 8 show selected excerpts of the fiber guidance for the tube profile. In the graphs, the Xs correspond to the source and the circles to the target. The elements connected by rovings are marked in color.

Reducing the planning time from multiple hours down to a few minutes on consumer hardware is a major improvement between one and two orders of magnitude. If the user is not satisfied with the provided result, the algorithm can simply be rerun while still maintaining a speedup to the manual approach. Figure 7a shows the left part of the first stage. Due to the low density in the center compared to the border, the fibers extend in all directions while preventing collisions. As can be seen in the projected image, the algorithm prevents overlap of fibers even on complex shapes. This is hard to replicate using a manual approach and simplifies the work with non-basic shapes. Figure 7b shows an excerpt from the second stage. Different colors represent different clusters. It shows how the algorithm attempts to place nodes of clusters in close proximity, which in this case means mostly on one row in the rack. Where possible, the cluster is placed near the fixpoint. However, if it would result in a collision between different rovings, the whole cluster is moved farther away. This ensures that the plan can be implemented without having to thread fibers between other clusters while also preventing abrasion.

However, the algorithm is only an heuristic. In rare cases, the results are suboptimal with regard to the ease of implementation. In our experiments, in 95% of the cases, the algorithm provided a sensible fiber-threading plan. This percentage fell to 90% for complex plans with several stages and many fibers. However, by inspecting the projected images after the algorithm is complete, the user can easily identify deficiencies in the fiber-threading plan. In contrast to Figure 7 which shows a good plan we can see such a defect in Figure 8. The two

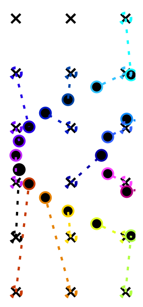
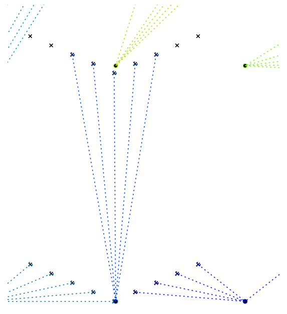


(a) (b)

Fig. 7: Excerpt from the first (a) and second stage (b) of the round profile

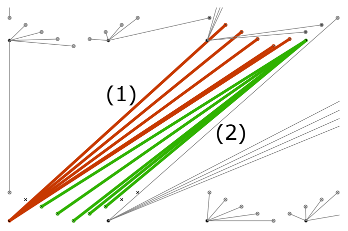


Fig. 8: Two opposed clusters (1) and (2) which should be swapped

opposed clusters (1) and (2) should be swapped for a simpler executability. This could be done by a manual postprocessing step or by rerunning the algorithm with the same input data.

D. Limitations and Future Work

One obvious limitation is the long runtime of the algorithm. For large plans, it can take up to 30 minutes to produce a meaningful result. For further acceleration, more profound measures than software optimization must be used, resulting in fewer iterations of the algorithm. By focusing on only one container and its direct successor at a time, the algorithm has fewer possibilities that need to be checked, improving its performance. A better overall result could be achieved if the algorithm processed the complete plan from the fiber rack to the die in parallel. In the current implementation, only a locally good result is targeted in each stage, but without checking whether this fiber distribution is advantageous for the next stages. A simultaneous calculation would also allow the fiber routing plan to be calculated without any pre-subdivision.

VIII. Conclusion

In this paper, we presented an algorithm that calculates with high probability a correct fiber-threading plan that can be implemented in reality from several connected fiber support plates. For this purpose, a simulation and a genetic algorithm with four different cost functions were used. The simulation provides a model that uses collision detection for checking whether the routing plan is correct. The fitness function is used to create an objective evaluation based on firmly defined requirements. With the help of these so-called fitness scores, the genetic algorithm can select, cross and mutate the chromosomes in order to iteratively obtain a better result, however the best solution cannot be guaranteed.

Through various optimization approaches, an average runtime of between 5 and 15 minutes was achieved. Since the algorithm allows many parameters to be adjusted, the most important settings were explained as well as a reasonable configuration was determined in a parameter comparison.

The intended benefit was to reduce the preparation time as well as to verify the fiber-threading plan ahead of time to prevent errors when realizing the fiber-threading at the pultrusion line. Both goals were achieved with a calculation time of less than one hour and collision detection in three-dimensional space.

References

[1] H. Ishida, "Pultrusion process for preparing composites," Patent US5 294 461A, 1989.

[2] T. F. Starr, Ed., *Pultrusion for engineers*. Boca Raton, Flor.: Abington, Angleterre and CRC Press, Woodhead, 2000.

[3] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[4] U. Maulik and S. Bandyopadhyay, "Genetic algorithm-based clustering technique," *Pattern Recognition*, vol. 33, no. 9, pp. 1455–1465, 2000.

[5] L. Qing, W. Gang, Y. Zaiyue, and W. Qiuping, "Crowding clustering genetic algorithm for multimodal function optimization," *Applied Soft Computing*, vol. 8, no. 1, pp. 88–95, 2008.

[6] M. Ryan, D. Hill, and D. McGrath, "Simulation interoperability with a commercial game engine," *Institute for Security Technology Studies*, 2005.

[7] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *Journal of Graphics Tools*, vol. 2, no. 1, pp. 21–28, 1997.

[8] D. Baldwin and M. Weber, "Fast ray-triangle intersections by coordinate transformation," *Journal of Computer Graphics Techniques (JCGT)*, vol. 5, no. 3, pp. 39–49, 2016.

[9] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[10] R. L. Haupt and S. E. Haupt, *Practical genetic algorithms*, 2nd ed. Hoboken (N.J.): Wiley-interscience, 2004.

[11] D. Shiffman, *The nature of the code*. [S.l.]: D. Shiffman, op. 2012.

[12] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[13] D. Eberly, "Geometric tools," 2018. [Online]. Available: https://www.geometrictools.com/

[14] J. T. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, 1998.

[15] R. L. Graham, "An efficient algorith for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, no. 4, pp. 132–133, 1972.

[16] K. Krishna and M. Narasimha Murty, "Genetic k-means algorithm," *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, vol. 29, no. 3, pp. 433–439, 1999.

[17] A. Shukla, H. M. Pandey, and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm," in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, 2015, pp. 515–519.

[18] M. Roberts, "The unreasonable effectiveness of quasirandom sequences," *Extreme Learning*, 2018-04-25.

[19] K. M. Shuhei Kimura, *Genetic Algorithms using Low-Discrepancy Sequences: GECCO 2005 , June 25-29, 2005 (Saturday-Wednesday) Washington, D.C., USA*. New York NY: Association for Computing Mahcinery, 2005.