Off-Part Motion Optimization for an Automated Fiber Placement Machine using Travelling Salesman Problem

Sean Doherty[1], Roudy Wehbe[2], Katie Plain[3], Ramy Harik[4] and Al Halbritter[3]

[1]University of South Carolina
[2]University of South Carolina, rwehbe@email.sc.edu
[3]The Boeing Company
[4]University of South Carolina, harik@mailbox.sc.edu

Corresponding author: Ramy Harik, harik@mailbox.sc.edu

Abstract. Automated Fiber Placement (AFP) is a common method for laying up composites for large structures in the aerospace industry. During the process, a robotic head places several carbon fiber strips (tows) over a surface in successive paths (courses) to achieve the full surface coverage. In order to improve AFP productivity, it is necessary to minimize the time when the AFP machine is not adding material to the structure especially when travelling between the end of one course/ply and the start of the next one which is also referred to as off-part motion. The off-part motion of the AFP machine creates a unique opportunity for optimization since it does not require a requalification of the process. Travelling between the plies of a composite layup is similar to travelling between cities, which led to the use of the heuristics to solve the Travelling Salesman Problem. An algorithm was created to optimize the off-part motion by first creating a precedence network consisting of sets of plies of the original part, then a path is built via a heuristic local search methodology that attempts to reduce the toolpath's off-part distance. Several test cases were developed to verify the algorithm. The test cases showed a range of 50%-80% off-part distance savings from erroneous layup strategies depending on what type of layup errors were introduced within the original toolpath.

Keywords: Automated Fiber Placement, Optimization, Traveling Salesman Problem.
DOI: https://doi.org/10.14733/cadaps.2022.220-237

1 INTRODUCTION

Several different industries including aerospace, automotive, and renewable energy are increasingly using advanced composites as a replacement of or improvement for metal structure. For example, composite materials make up 50 percent of the primary structure of the 787, including the fuselage and wing. The automotive industry is currently using composites for a small portion of the vehicles and needs further advancement in manufacturing to allow for production level quantities at an affordable cost. The renewable energy sector has the opportunity for using composites for wind turbine blades. Automated Fiber Placement technologies have enabled traditional processes that were previously inefficient and time-consuming to become more cost-efficient and to drastically reduce the amount of scrap material [1]. Although these processes have improved significantly, the potential for further optimization still exists. Machine toolpath generation for free-form surfaces has been a critical area of investigation for decades [15]. New materials and especially new computer-controlled manufacturing techniques have been formulated to produce consistent and reproducible quality structures [21]. One key area that still remains to be optimized is the path in which the robot travels to deposit composites on the (typically curved) surface. Automated Fiber Placement machines generally have a minimum of six axes and minimizing the motion of these axes can generate significant improvements in the time and hence cost-effectiveness of the process to manufacture composite structures. In this work, we aim to describe a methodology that reconstructs the off-part motion of the robot to minimize axis motion and, therefore, generate a more optimal path to be followed during manufacturing. This study uses already qualified geometries and evaluates the CNC for that part and returns a part with no restructuring of the motion that happens on the tool surface.

2 STATE OF THE ART

For this article, we will be focusing on the aerospace industry's use of carbon composite for aircraft fuselages. It is prudent to first review what the aerospace industry uses for the manufacture of large composite structures, Automated Fiber Placement (AFP). The following sections will then review several attempts at optimizing an AFP process or machine. Then, we will present the problem chosen to facilitate our optimization method and the previous works that have attempted to solve the problem.

2.1 Automated Fiber Placement (AFP) Machine Overview

The main method used for manufacturing composite aircraft fuselages is Automated Fiber Placement (AFP). For this process, a CNC driven robotic arm places strips of pre-impregnated composite on a rotating tool surface. Each strip is called a tow and there are usually 8-32 tows or bundles of fiber, laid on the tool at once (named course) using a specialized end effector [8]. Each strip is fed from a bobbin, or spool of one tow, that allows for the tows to be placed at varying speeds to match the design requirements. The design can have varying ply angles, pad ups, and curvature, and several layup strategies can be used to achieve the surface coverage [22]. Using this method is more accurate and repeatable than hand layup of composites. It is also an improvement from Automated Tape Laying (ATL), which lays a wide continuous tape on the tool which may causes defects such as wrinkling for complex curved tools [20],[24]. Temperature control in the machine head ensures that the tows are warmed to the adequate temperature achieving a proper tackiness as they are applied to the tool, while the remaining of the bobbins are kept cold to avoid aging and partial curing [18]. Many parameters must be controlled in order to optimize this process, from the properties of the fibers to the movement of the robot.

An AFP machine usually consists of a robotic arm with an end effector and a rotating mandrel. The arm has six degrees of freedom and the mandrel creates a seventh (Figure 1). In order to optimize the motion of the arm, the pose of the robot must be defined. This is accomplished by using two spaces, the task space and the configuration space [2]. The task space is used to describe the position and the orientation of the end effector. The configuration space is the set of the joint angles of the robot. Switching between the two spaces can be done using two mappings, forward and inverse kinematics. Forward kinematics calculates the end effector position and orientation from the robot joint angles. Inverse kinematics calculates the set of possible robot configurations from the end effector position and orientation [2]. Inverse kinematics are important for the optimization of the robotic arm motion; however, it is usually a computationally expensive calculation. Throughout the workspace of the robot there may be multiple to infinite solutions when mapping between the task space and configuration space.

In this article, we will focus on optimization of the robot's off-part motion rather than the inverse kinematics problem. The final algorithm is desired for production use where altering the location of the plies is not allowed and changing the on-part robot parameters would require requalification of the process, hence optimizing the off-part motion would allow for specialized optimization with a fast response for the operator.

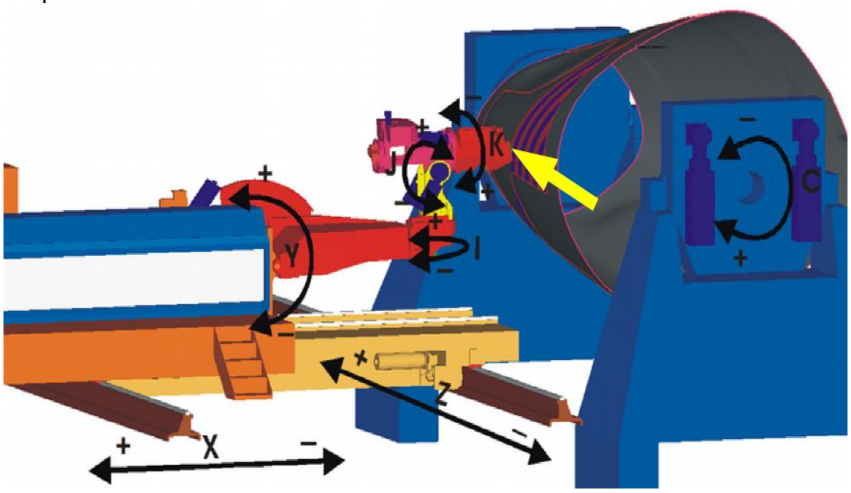

Figure 1: A 7-dof AFP machine is made up of a 6-dof robotic arm and a rotating mandrel.

2.2 Previous Work to Optimize the AFP Process

There have been several different attempts at optimization of an Automated Fiber Placement (AFP) process. Debout et al. worked on an optimization of a 7-axis tool to reduce manufacturing time by looking at the degree of redundancy and path smoothing of the machine head [9]. The use for the degree of redundancy is to decrease the kinematic loads on the control joints. The Denavit-Hartenberg (DH) notation was used to determine the allowed orientations of the joints and is used in the path smoothing of the head. It was addressed by looking at the continuities of the tool path when changing speed and direction. There are several methods to address the optimization when having task redundancy, as is outlined in [9]. In [25], a fuzzy logic approach is suggested to calculate possible solution for the inverse kinematics problem. By using the (DH) parameters once again, an algorithm generates a fuzzy model using the Jacobian matrix [25]. On the other hand, Aized and Sirinzadeh looked at different processing parameters for the composite to enable faster placement of the composite such as gas torch temperature, fiber laying head speed, and fiber compaction force [1]. Although this is a valid method of optimizing the laying of the fiber while controlling quality and improving manufacturing time, it is not applicable to our case due to the highly controlled nature of on-part movement for our task. Finally, Nik et al. compared an optimized composite design to an optimized composite manufacturing process [4]. It is possible to create an optimized design using nonconventional laminates [3], or variable stiffness laminates [23], however, recertification processes are necessary so that they can be utilized. One of the common defects that can occur in manufacturing is the gaps [11] created when dropping or adding tows while laying down a course. During manufacture, it must be assumed that there will be overlap in those areas anywhere between 0% and 100% [4]. Again, as this is an on-part optimization, it is not entirely helpful to our optimization, but it does give another example of how an AFP process can be optimized. As these works focus mainly on the on-part motion of the machine, we looked further into other manufacturing operations. The most notable is for computer numerical control (CNC) machines and the optimization of material removal. The following section outlines a common optimization problem and its application to CNC machines as well as robotics.

2.3 Traveling Salesman Problem and Existing Solution Methods

In order to complete the optimization, path planning will be used. One of the most effective methods for optimizing the path of the robot is modeling it as the Traveling Salesman Problem [2]. The traveling salesman problem (TSP) is based on a salesman visiting each city in an area in order to sell his wares. He must visit each city in his path only once and then return to the starting point. It is one of the most widely studied combinatorial optimization problems. The problem itself is to find the minimum cost cycle that includes every node in the graph exactly once. This is also referred to as the minimum weight Hamiltonian cycle. The problem can be formulated as an integer programming problem, as each variable is only an integer. Cerny claims that TSP is an NP-complete problem [7]. This means that as the number of cities increases linearly the number of solutions increases exponentially. That leads to the solution being impractically solved using brute force computation. Several researchers have been studying it using heuristic techniques to solve the TSP to avoid the long computation times but still remain close to the global optimal solution. As explained by Johnson and McGeoch, there are two main ways to address the TSP in general: using a local search or successive augmentation [13]. The method of successive augmentation builds a tour from an initial starting point and, usually, using a greedy method, begins forming a path through the cities. This is a valid way to address a robotic optimization because there is usually a home position at the start of manufacture in order for the robot to safely move without collisions.

There are variations on the TSP based on the parameters of the path to be found. First, a symmetric TSP is when the cost of going either direction between two nodes is equal. The symmetric TSP is one of the most widely studied versions. The Lin-Kernighan heuristic works well with problems with less than 1000 cities reaching within 1-2% of the optimal solution [17]. The next version is an asymmetric TSP where the cost is not equal going either direction between two nodes. There are fewer heuristics for this problem, but the Kanellakis-Papadimitrou and Zhang algorithms work well for problems with less than 1000 cities **Error! Reference source not found.**. The third version of the TSP is the generalized TSP (GTSP). For this case there are groups of cities, each of which must only be visited once. Similar approaches were developed by Lein [16]. The GTSP was converted to a larger size TSP where each group of cities was converted to one city and the optimal solution of the TSP was related back to the GTSP. The advantage to this approach is that there are several efficient methods to solve the TSP even as the problem increases. In order to define a different style of path, Durbin and Willshaw used the elastic net method [10]. Starting with a ring at the centroid of the cities, they created an algorithm where the change of one variable caused the path around the cities to alter slightly. However, the process takes several iterations to find the correct value of the variable for the lowest cost path [10]. Finally, Held and Karp provide a dynamic programming algorithm to solve the traveling salesman problem [12]. This approach assumes that the cost of a path depends only on the paths that preceded it.

One of the biggest concerns when optimizing the path of the robot is making sure that it still creates what was originally designed. For example, plies of a composite that are closest to the tool must be laid down first and any plies that overlap that first ply must come next and so on. This creates a precedence network. This is also a concern for computer numerical control (CNC) tools when machining other materials, which has been a more common study. Castelino et al. looks at optimizing off-part motion of CNC machines [6]. The difference in this case is that the tool must travel in a predefined path while cutting material and minimize the amount of time traveling between those predetermined paths. Castelino et al. formulated this problem as a generalized TSP [6]. Bockenhauer et al. proposed another take on TSP with precedence by proposing two methods: (a) finding a minimum-weight Hamiltonian path between two predetermined endpoints, and (b) the ordered TSP (k-OTSP) with k being the number of vertices given in a certain order [5]. The k-OTSP method is more relevant to our optimization as certain plies must be completed before others. There are far more heuristics available to solve the TSP in the realm of machining. Oysu has suggested the use of combining the simulated annealing heuristic, developed by Kirkpatrick et al in 1983, and the use of genetic algorithms, developed by Goldberg in 1989 [19].

Another interesting case with robots is the cost of moving certain joints of the robot during manufacturing. Khan suggests using an asymmetric TSP for moving between contours of the design [14]. This means that the cost of moving from Point 1 to 2 does not equal moving from Point 2 to 1. Finding the cost of certain joints will assist with finding the overall optimal path. We will explore this later in the article. Zhang explored using the DH parameters and the Jacobian Matrix to determine the compliance of each joint **Error! Reference source not found.**. We will explore the application of TSP to optimization of an AFP machine to optimize off-part motion keeping in mind the optimization methods used on CNC machines and specific properties of a robot with multiple joints and several degrees of freedom in the following sections.

3 AFP TOOLPATH OPTIMIZATION METHODOLOGY

The first goal of the optimization process is to partition our part into independent graphs. Each graph will represent a set of plies. The sequence of these plies within the graph has no effect on the outcome of the part, hence there is room to optimize their order. Each graph will be referred to as a precedence set while the complete hierarchical set of precedence sets will represent the precedence network (discussed further in Section 3.1). Once the precedence network has been constructed, each node of the network enters a local search algorithm that determines a starting seed, builds the most optimal Hamiltonian path (single edge reduction from a Hamiltonian cycle) it converges to, and returns this toolpath if the cost is less than the original input path. For the purposes of this work, all formulations will be derived via deterministic systems. Non-deterministic algorithms typically exploit the power of randomness (in computation) to help produce solutions that can probabilistically produce more near-optimal solutions than iterative and analytical approaches in a lesser amount of time. This approach is especially useful in applications relating to combinatorial optimization, searching, motion planning, and systems that involve real-time sensory data (i.e. online robotics) and is a quintessential aspect of most real-world algorithms built to solve these problems. In general, these formulations cannot, as a direct result of including randomness, generate repeatable results (nondeterministic algorithms) and thus, in this work, we will not rely on randomness to lead the solver to convergence.

3.1 Precedence Network

We will refer to the input to the system described in this work as the part. The part can be expressed as the set X, where:

$$\boldsymbol{X = \langle p_1, p_2, \dots, p_{n-1}, p_n \rangle}, \tag{3.1}$$

where pi is the set of courses belonging to the i-th ply of the part. Before the optimization phase of the solver, we must partition our part into disjoint sets of courses which we will refer to as a precedence set. Trivially, we could allow each pi ϵ X to constitute a set in itself as this would meet the precedence set requirements:

1. Preserving all mechanical and geometric properties of the original part
2. Producing disjoint sets of courses
3. Solving over courses of the same layer; in other words, non-intersecting and non-overlapping courses

To give the solver the chance to discover inter-ply layup patterns (an example of this will be discussed later in the results section, refer to Figure 7 for an example), we instead begin by performing an intersection test on each sequential ply pi, pi+1 ϵ X. In the case these plies do not intersect, we will join the sets to form a precedence set, L which can now be expressed as the set:

$$\boldsymbol{L = p_i \bigcup p_{i+1}} \tag{3.2}$$

We continue to evaluate sequential plies to determine if we may again join the sets until a ply is determined to intersect with one or more of the courses in the current precedence set. At this point, the current precedence set, **L_k** is finalized and a new set **L_{k+1}** is initialized containing the intersecting ply's courses. This routine continues until every ***p_i*** ϵ **X** belongs to a precedence set. A summary of this process is presented in the form of a flow diagram in Figure 2 and an example of a simple four ply part decomposed into a precedence network is illustrated in Figure 3.

In the example of Figure 3, the original part consisted of four plies in which the first two plies of the CNC program had no overlap and hence were joined. When evaluating the third ply, it was found that there existed some overlap between one of the courses of the third ply and some portion of either the first ply, the second ply, or both. This finalizes the first precedence set (precedence L1) and a new precedence set L2 is constructed containing the third ply. When the process moves to the fourth ply, it is found that there is overlap between the fourth ply and the current precedence set and thus the current precedence set is finalized, and a new precedence set L3 is constructed consisting of the fourth ply. As the fourth ply is the final ply of this part, the new precedence set is finalized, and the construction of the precedence network is completed. This methodology is the key to generate sets of courses that may be laid up in any order without affecting the part.

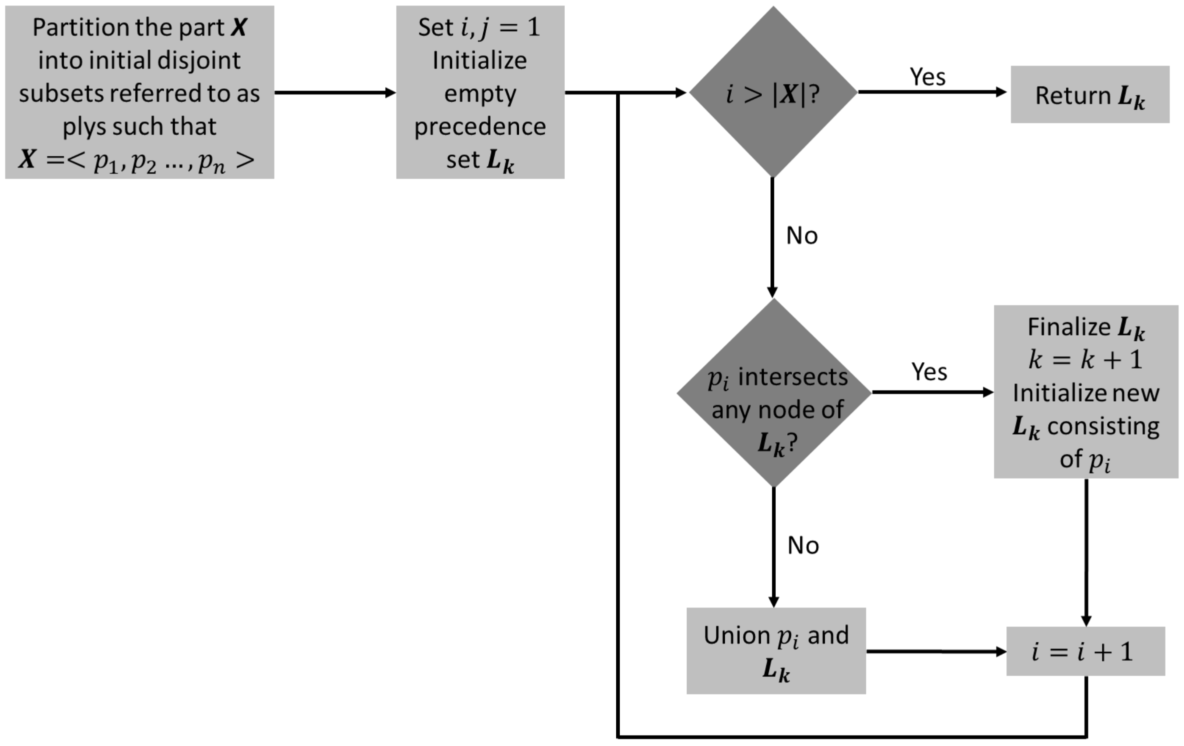
Partition the part $\boldsymbol{X}$ into initial disjoint subsets referred to as plys such that $\boldsymbol{X} =< p_1, p_2 \dots, p_n >$
Set $i, j = 1$ Initialize empty precedence set $\boldsymbol{L_k}$
$i > |\boldsymbol{X}|$?
Yes
Return $\boldsymbol{L_k}$
No
p_i intersects any node of $\boldsymbol{L_k}$?
Yes
Finalize $\boldsymbol{L_k}$ $k = k + 1$ Initialize new $\boldsymbol{L_k}$ consisting of p_i
No
Union p_i and $\boldsymbol{L_k}$
$i = i + 1$

Figure 2: Flow diagram representing the methodology of the precedence network generation.

Once all sets have been built, we formulate a complete multigraph, G which can be represented as:

$$\boldsymbol{G} = \langle \boldsymbol{L}, \boldsymbol{E} \rangle \tag{3.3}$$

where ***L*** is our finite, disjoint set of courses (or "precedence set") and ***E*** is the set of all edges connecting all distinct pairs of nodes ***l_i, l_j*** ϵ ***L***. In the special case that bidirectional layup (as opposed to unidirectional layup: only top-bottom, right-left or vice versa) has been prohibited via the process parameters, ***G*** will simply be a complete graph (opposed to a multigraph in the general case).

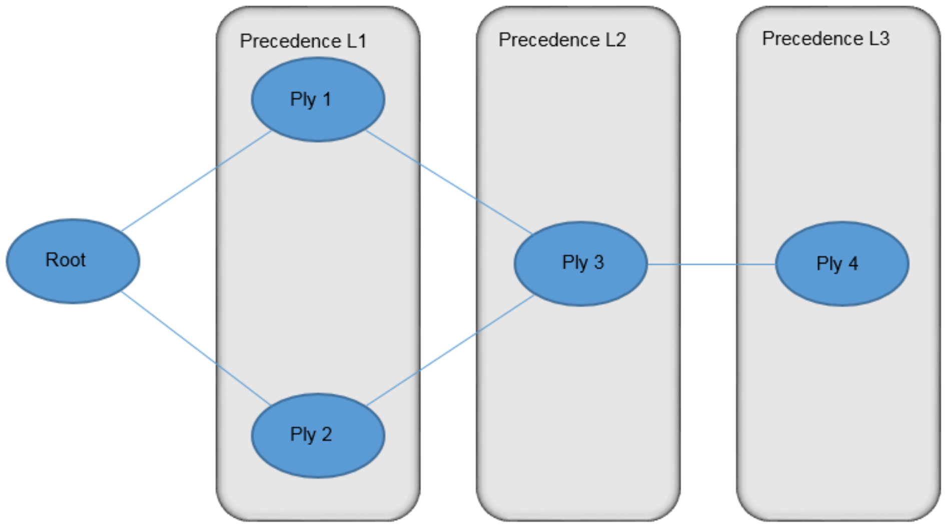

Figure 3: Illustration of a simple precedence network.

This is a result of each course having one allowable start configuration and one allowable end configuration and thus only one path can exist between two nodes of this formulation. The general case allows for each original start/end position to be treated arbitrarily as an end point producing the following edges for each distinct pair of courses within a precedence set:

- $\boldsymbol{l_i}$'s end position corresponds to $\boldsymbol{l_j}$'s start position
- $\boldsymbol{l_i}$'s start position corresponds to $\boldsymbol{l_j}$'s start position
- $\boldsymbol{l_i}$'s start position corresponds to $\boldsymbol{l_j}$'s end position
- $\boldsymbol{l_i}$'s end position corresponds to $\boldsymbol{l_j}$'s end position

Each edge ***e*** ϵ**E**, is built with a cost determined via a machine distance heuristic, ***h***, such that:

$$h(l_i, l_j) = \sum_{a}^{A} C(a) * \sqrt{\left(l_i(a) - l_j(a)\right)^2}\,, \qquad (3.4)$$

where ***A*** is the set of all controllable axes in the relevant AFP machine, ***C*** is a function to return the scalar coefficient for all ***a*** ϵ***A***, and ***l(a)*** returns the floating-point value representing the machine signal at that course end point. If ***G*** is a multigraph, this leads to $2(n)(n-1)$ edges (or four times the amount of a complete graph). It is well-known that performing an exhaustive search to formulate a Hamiltonian path (finding a minimum cost path in which every vertex is visited exactly once) has a runtime complexity of O(*n!*), where *n* is the number of vertices in ***G***, as this would determine the cost of all possible permutations (order in which we travel to each vertex in ***G***). As this branch of combinatorial problems have been classified as NP-hard, no algorithm to date can find an optimal solution in polynomial time. Hence, in this work, we will introduce a heuristic method for evaluating an original path and attempting to generate more optimal routes for the relevant AFP machine via an optimization algorithm built around key characteristics of the AFP process.

3.2 Optimization

To produce desirable results, the solver's optimization algorithm must meet several requirements:

1. Generate deterministic results
2. Create equivalent part outcome as the originally qualified part
3. Produce results in a relatively short amount of time
4. Generate a new path *p*, for which *F*(*p*) ≤ *F*(*o*) where *o* is the original path input and *F* is the objective cost function

To effectively describe the methodology, we will begin by discussing the relevant notation. The input to the algorithm can be described by a set **M** (the precedence network), such that:

$$M = \langle G_1, G_2, \dots, G_{n-1}, G_n \rangle \,, \qquad (3.5)$$

containing all precedence sets built (as described in the previous section) where *n* is the number of precedence sets determined for the relevant part. For each $G_i \in$ **M**, a path will be built beginning at the relevant ***seed*** or start point (a vertex) for the algorithm to begin its search from and moving through every vertex exactly once. For every $G_i \in$ **M**, the seed could be determined by the end position of the final course of the path generated for G_{i-1} or could be selected independently of the end position of the previous path. We have discovered experimentally that seed selection is critical when using greedy approaches in combinatorial problems as a bad seed selection can quickly lead to trapping solutions within a local minimum. For this reason, the seed selection for $G_i \in$ **M** will be dependent on only G_i and the machine motion between each precedence set will be considered negligible. Each $G_i \in$ **M** undergoes a seed selection process that performs as follows:

1. A pool of possible start vertices is generated, ***P***, that consists of all ply start (and end positions in the bidirectional cases) from the precedence set as well as the single vertex which has the maximum total distance between itself and all of vertices of G_i.
2. Each vertex $v \in P$ is used as the seed to generate a simple nearest neighbor path.
3. The vertex $v \in P$ which produces the lowest cost path is selected as the seed for ***G***.

Using this analytical approach, the solver retrieves the most promising seed for converging close to, if not exactly, at the optimal Hamiltonian path through ***G***. Once the seed has been determined, the path generation module executes as illustrated in Figure 4. The algorithm's fundamental methodology is to perform a local search from the path's most recent position. This methodology works in tandem with one of the key design characteristics of plies which is that the courses within a ply all lie within a single geometric boundary and have the same orientation. Using a local search methodology allows the solver to exploit the inherent spatial locality that exists in each graph. Initially the algorithm performs a depth-first search to determine a sufficient nearest neighbor chain length. This variable chain length is critical to prevent the solver from trapping itself at local minimum by expanding the search space and recognizing geometrical path patterns within the partitioned geometry. The key parameter to this phase of the algorithm is the ***depth*** variable. With each iteration of this phase, the depth is increased (and thus the runtime) to attempt to generate a new path with a new minimum cost. This phase of the algorithm will iterate until one of the following convergence criteria have been met:

1. Maximum iterations reached
2. Maximum depth reached
3. Minimum cost path has remained unchanged for *N* iterations

If an iteration returns a path with a cost not less than our previous minimum cost path, we increase the value of *a* by 1 and check to see if *a* is equal to *N*. Here, *N* is the maximum number of iterations. If *a* is equal to *N*, then the solver terminates the depth-first local search phase. If an iteration returns a new minimum cost path, the depth is increased and the main convergence criteria, *a*, remains unchanged from the previous iteration. This methodology results in a solver that:

1. Attempts to discover new minima by increasingly expanding the search space to discover if a lesser search space leads to a less optimal local minima.
2. Determines if the solver has already discovered the most reasonable minima (whether local or global) for producing speedy results.

One of the core solver parameters, *N*, which determines how many iterations the solver will continue to expand the search space without discovering a more optimal minima, is critical to the results of the process. When *N* is small, the solver, in general, returns results that have a greater probability of being a local minimum as opposed to the target results of the global minimum. Another consequence of *N* being small is a drastic decrease in the runtime of the algorithm. As *N* increases, the final search space of the solver grows as a result of the depth continually increasing, but while the search space grows, so does the runtime at an exponential rate. While the minimum runtime is always desirable, balancing the feedback time with the quality of the results is imperative to the overall performance of the system. The depth-first local search also terminates if the maximum depth is reached or the maximum iterations allowable is reached. These are separate parameters for the solver: the maximum iterations allowable is a static value set by the operator to ensure runtime does not surpass the maximum desired wait time of the operator; the maximum depth parameter value is a function of the course count and ply count of the relevant precedence set. The overall runtime complexity of this module of the solver is $O(n^d)$ where n is the course count of the precedence set and d is the maximum depth.

The methodology of the second phase of the solver is similar to the previous phase except that it only runs a single iteration, and search space grows in a different manner. Unlike the depth-first search which focuses on building paths of length *depth* of successive nearest vertices to determine optimality, the second phase uses a breadth-first search with a static depth equal to the final depth of returned a new minimum from phase one. This executes very similarly to the final iteration of the previous phase except where phase one builds paths of nearest neighbors, the second phase builds k paths rooting from the first to k-th nearest vertices. The profit of this phase comes from moving away from the most greedy-approach and expanding the breadth of the search to consider edges in the graph that are not considered in the greedy approach. The value of k is determined via a function of the precedence set's ply count and the layup strategy (unidirectional or bidirectional). The runtime of this phase is $O(kn^d)$ where k is the breadth, n is the course count of the precedence set, and d is the depth value following the previous phase. If the breadth-first search returns a more optimal path than the previous best path from phase one then the solver returns this new path; otherwise, the phase one path is returned. The returned path is appended to a global array to be sent back to the user once all precedence sets have been solved.

Overall, the goal of the algorithm described in this section is to build a path as near to optimal as possible while still remaining a relatively online system to generate responsive feedback to the operator. The methodology is built around exploiting some of the key characteristics of composite geometries, namely, spatial locality within what we have defined as precedence sets (machine coordinate distance and $\mathbb{R}^3$ have a strong correlation, in general). The solver attempts to determine if it has captured a path lying within one of likely many local minima by iteratively expanding its search space while pushing towards convergence when the benefits of expanding the searching space diminish. The intention to keep searching until the continued runtime is not worth the benefit. The use of local search and building an algorithm around the characteristics of the problem set for this article is to reduce the overhead compared to state-of-the-art global optimization solutions. Those require drastically larger amounts of time to return results and still commonly do not generate globally optimal results. Knowledge of the strengths and weaknesses of the relevant AFP machine can allow the operator to bias the use of axes that may be favorable and/or less-favorable to generate paths that will increase and/or decrease off-part usage of the biased axes. This bias is instigated in the machine heuristic by increasing the scalar coefficient returned by $C(a)$ to either reduce or increase the delta for a given axis, $\boldsymbol{a} \in \boldsymbol{A}$.

4 PROTOTYPE TOOL

The tool's interface is built as a web application for convenient, cross-platform access. The interface consists of several key modules: the machine configuration, program import, 3D environment, and parameters.

4.1 Machine Configuration

The machine configuration is the first module the operator will use when beginning the process. When the relevant CNC program is for a machine that has never been configured for this tool, the operator must supply a minimal model of the robot.

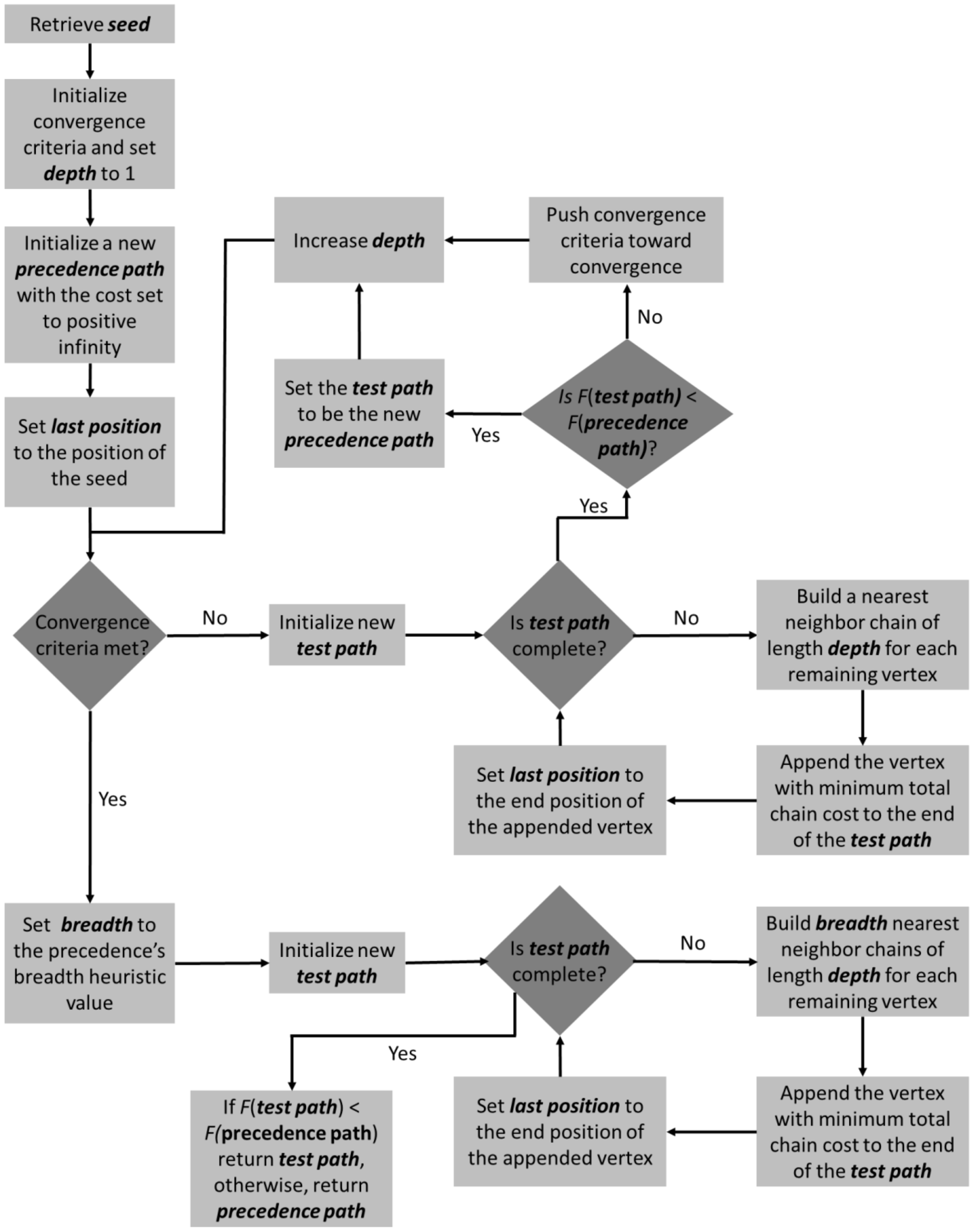
Retrieve seed
Initialize convergence criteria and set depth to 1
Initialize a new precedence path with the cost set to positive infinity
Set last position to the position of the seed
Convergence criteria met?
No
Yes
Initialize new test path
Is test path complete?
No
Yes
Build a nearest neighbor chain of length depth for each remaining vertex
Append the vertex with minimum total chain cost to the end of the test path
Set last position to the end position of the appended vertex
Is F(test path) < F(precedence path)?
Yes
No
Set the test path to be the new precedence path
Push convergence criteria toward convergence
Increase depth
Set breadth to the precedence's breadth heuristic value
Initialize new test path
Is test path complete?
No
Yes
Build breadth nearest neighbor chains of length depth for each remaining vertex
Append the vertex with minimum total chain cost to the end of the test path
Set last position to the end position of the appended vertex
If F(test path) < F(precedence path) return test path, otherwise, return precedence path

Figure 4: A diagram of the data flow and logic of the solver's optimization algorithm.

This includes providing each axis label (as they appear in the CNC program) along with the functionality of the axis whether that be prismatic, revolute (specifying whether it is continuously revolute or not), or a mandrel (if the relevant machine is a 7-axis robot). The purpose of this designation for each axis is that the machine distance heuristic includes different coefficients when dealing with prismatic or revolute motions. Once the user has entered these criteria, the machine configuration will be serialized and can be saved locally by the user for persisted use. An illustration of the machine configuration module can be seen in Figure 5.

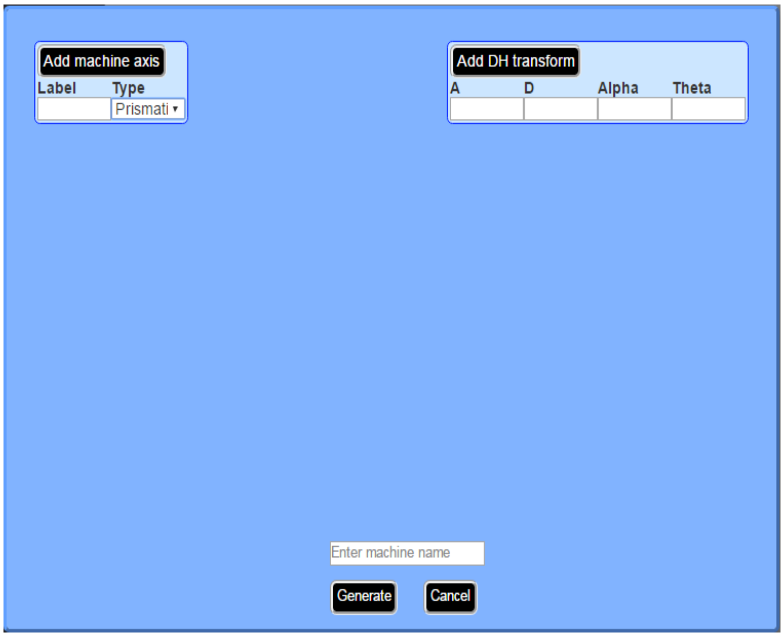

Figure 5: Snapshot of the interface for generating new machine configurations.

4.2 Program Import

Once a machine configuration has been generated or imported, the interface allows for the operator to import a CNC program. When a program is selected, the CNC program is parsed and reverse engineered so that the tool stores the program internally in an array such that each element of the array represents an individual ply. Each ply element is another array where each element represents a course represented by an object consisting of:

1. An array of approach points in the machine configuration space, and an array of approach points in $\mathbb{R}^3$
2. An array of retract points in the machine configuration space, and an array of retract points in $\mathbb{R}^3$
3. An array of layup points in the machine configuration space, and an array of layup points in $\mathbb{R}^3$.

Using this information along with the machine configuration information, the tool can compute a full replica of the machine motion for the supplied part. CNC is executed sequentially and for this reason, the internal data structures construct and store the data representing the geometries in the same manner.

4.3 3D Environment

Once the CNC program has been fully computed and the part has been stored in memory, the end-effector's path is simulated in an interactive 3D environment. This virtual representation shows the full approach, retract, and layup motion of the original program. Each ply computed can be toggled in and out of the environment giving the user the ability to scope in or out on sections of interest. Once the solver (discussed further in Section 4.4) has been called by the user and has returned results, a second set of simulated paths becomes available for the user to toggle in and out of the environment which represent the new, optimized paths generated by the solver. The environment has a perspective camera to allow for zooming, panning, and rotating the geometry. The tool's interface with a cylindrical geometry and all original plies shown in the environment are displayed in Figure 6.

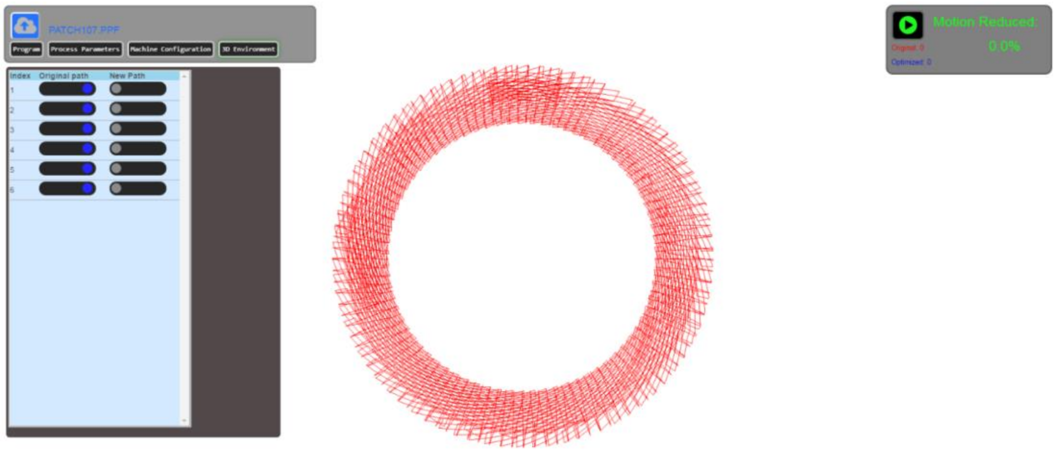

Figure 6: Snapshot of the prototype tool's interface after a six-ply cylindrical part was imported.

4.4 Implemented Module

The optimization process begins once the user has successfully imported a CNC program and selected the call solver button. Once this solver initialization is called, the precedence network is serialized and sent to the server. Once the data is received by the server, a thread on the server is started that passes the precedence network to an executable that we will refer to as the solver. The solver is the compiled program that performs the local search algorithm on each precedence set of the precedence network and generates results for the server to pass back to the client (interface) when it reaches completion. The solver has been developed as a C++ module using standard libraries under C++11 and built under the Ubuntu 16.04 Linux operating system.

4.5 Test Cases and Results

A collection of test cases was programmed by experienced process planners to incorporate common, non-optimal planning mistakes to evaluate how well our methodology could correct these errors. These process mistakes include illogical ply sequencing, illogical course sequencing, and the use of unidirectional layup when process parameters allow for a bidirectional layup strategy. The results illustrate the methodology's ability to provide optimized results that do not rely on the original layup patterns to find and exploit new optimized routes that include new inter-ply toolpaths and rectifications of process planner errors. The first section of test cases are planar geometries with a large, rectangular first ply with three non-intersecting pad-ups inefficiently placed atop the first ply. The cases explore 0^o, 45^o, and 90^o orientations to provide a robust set of results to explore.

In the first test case, we will investigate the results generated by the solver of the original geometry depicted in Figure 7 (a). The original geometry illustrates three aspects that our work is aiming to correct. Firstly, an illogical sequencing of plies that results in greater than needed inter-ply toolpaths.

Next, producing toolpaths that can greatly reduce the global toolpath by incorporating inter-ply layup patterns (producing a toolpath that lays courses of multiple plies before completing a single ply). Lastly, supposing the process parameters allow for the full potential of optimal tool pathing by allowing for bidirectional layup, the results will incorporate this as well. Figure 7 (b) shows the new toolpath results of the algorithm. Given our three error criteria, we can see that the solver has corrected each problem and the total machine motion (Euclidean distance of the end-effector) has been reduced by 38.1% while the off-part motion has been reduced by 67.9%.

In the following test case shown in Figure 8 (a) and (b), we examine a similar sequence which has the final three plies equally spaced apart. The results are quite similar except that instead of joining two plies, in this run, the solver joins all three plies. The results of the second test case has reduced the total distance by 36.7% and the off-part distance by 65%. In the third test case as shown in Figure 9 (a) and (b), we can see that the plies have been re-sequenced, and the tangent direction of every even course has been rotated 180° to incorporate a bidirectional strategy. The motion reduction of this 45° planar example has a similar distance reduction to the previous at 35.6% of total distance and 57.5% off-part. In the final planar example as presented in Figure 10 (a) and (b), we see that the same corrections are found and corrected reducing the complete part distance by 34.9% and the off-part motion distance by 56.2%.

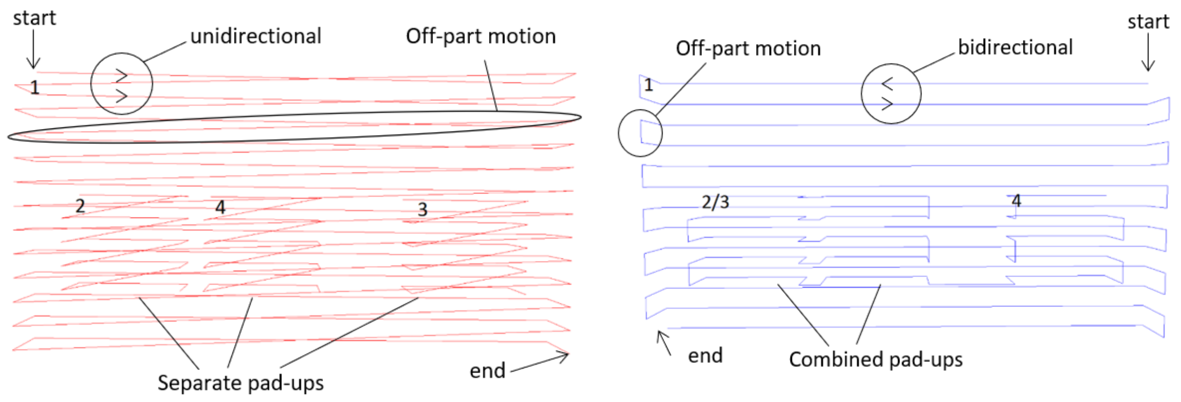

Figure 7: (a) Original planar, four ply toolpath consisting of 0° courses (red). (b) Optimized planar, four ply toolpath (blue).

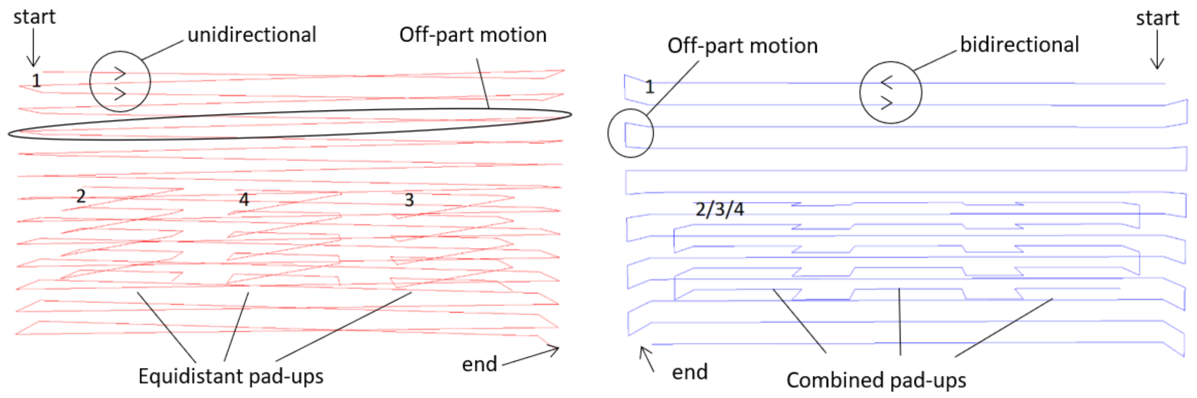

Figure 8: (a) Original planar, four ply toolpath consisting of 0° courses with the three equidistant pad-ups (red). (b) Optimized planar, four ply toolpath (blue).

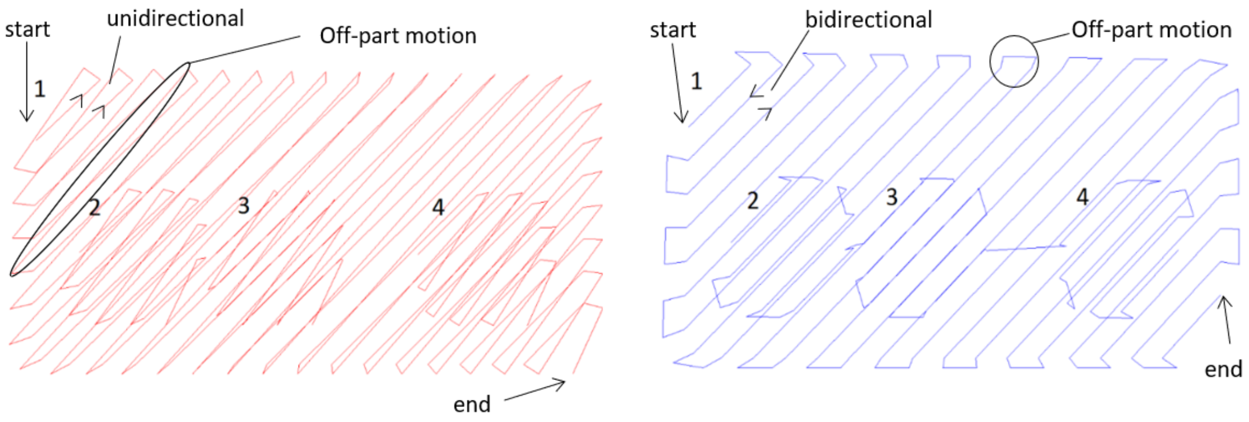

Figure 9: (a) Original planar, four ply toolpath consisting of 45° courses (red). (b) Optimized planar, four ply toolpath (blue).

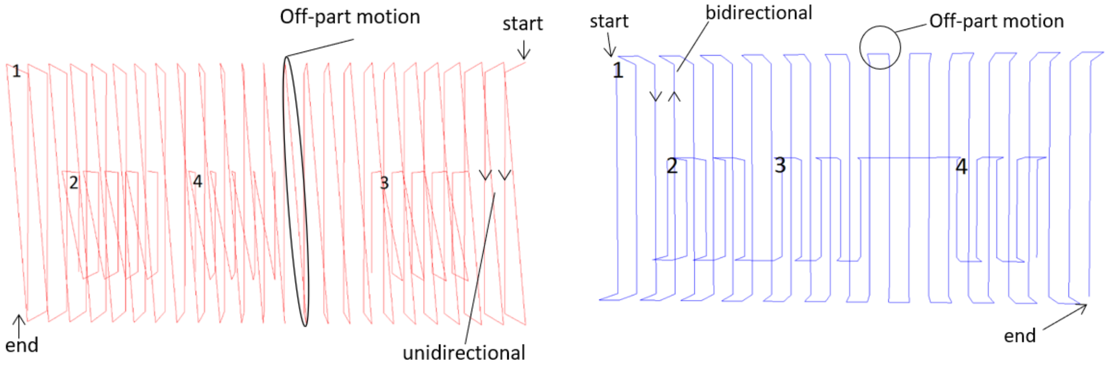

Figure 10: (a) Original planar, four ply toolpath consisting of 90° courses (red). (b) Optimized planar, four ply toolpath (blue).

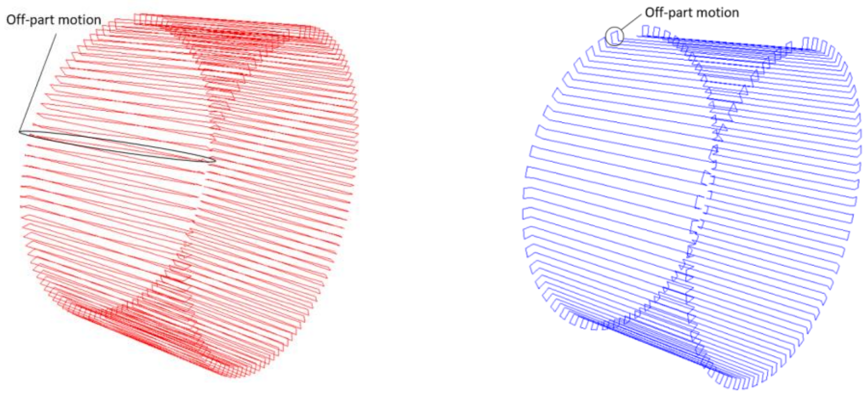

Figure 11: (a) Original cylindrical, single ply toolpath (left). (b) Optimized cylindrical, single ply toolpath (right).

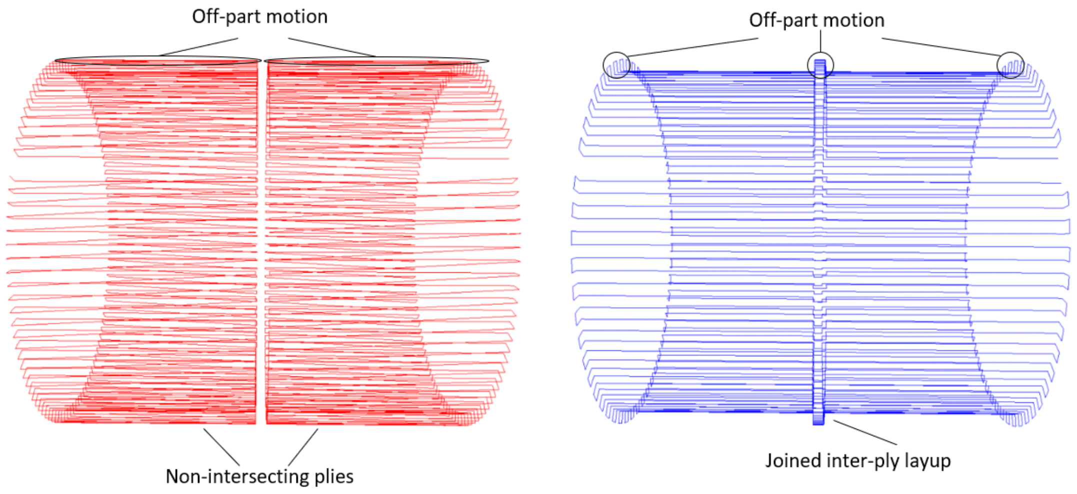

Figure 12: (a) Original cylindrical toolpath consisting of two non-intersecting plies (left). (b) Optimized cylindrical toolpath consisting of two plies joined to produce an inter-ply layup strategy (right).

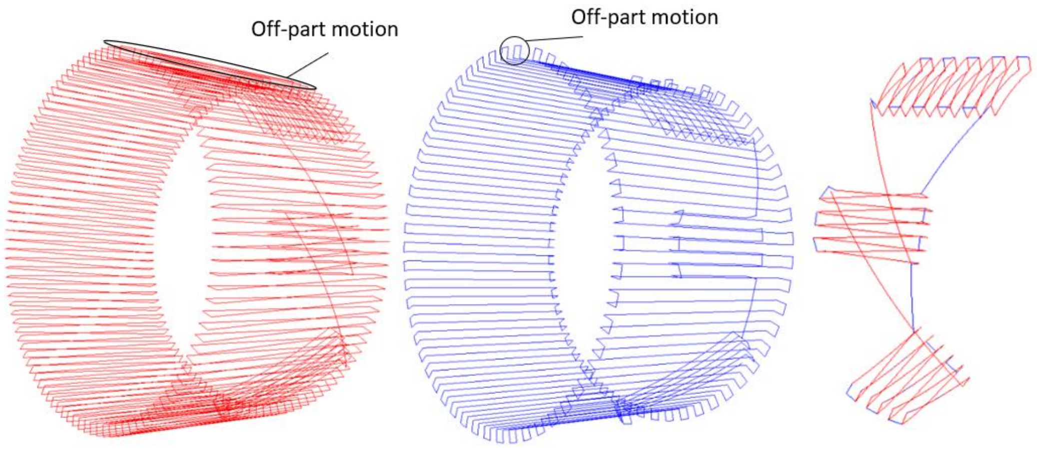

Figure 13: (a) Original toolpath of a cylindrical part consisting of a base ply along with three pad-ups placed over (top-left). (b) Optimized toolpath of a cylindrical part in which the top level, non-intersecting plys have been reordered producing a significantly less costly motion between plys (top-right). (c) Original toolpath with the optimized toolpath superimposed to illustrate the key changes constructed by the solver (right).

The next several test cases explore the usage of the solver when 7 axes are in use to produce cylindrical geometries. In the first cylindrical example (Figure 11), we see a drastic reduction in motion (by 44.6% total distance and 83.5% off-part distance) generating a bidirectional pattern for the single ply. The original toolpath for this example is shown in Figure 11 (a) while the solver's toolpath is depicted in Figure 11 (b). In the following cylindrical example (Figure 12), we investigate two non-intersecting plies and find that the solver joins them for an inter-ply layup pattern that reduces the overall distance by 45.1% and 83.3% of the off-part motion. It is clear to see from the results which of the plies are members of the same precedence set as they contain the motion between plies even when they are not joined. The final cylindrical test case contains a 0° first ply and three non-intersecting plies of different orientations atop the base ply. Illustrated in Figure 13 (c), the solver has discovered the minimum weight edges connecting the three plies that have been grouped into a single precedence set.

5 CONCLUSION

Several new ideas can be concluded from the model represented in this article along with the results. Firstly, it has been shown that, even with strict limitations, an algorithm can effectively, autonomously build paths for automated fiber placement machines using robust sets of machine and process parameters. As with most global optimization solutions, the quality of a solution is strongly correlated with the approximate execution time the system allows. The runtime of the algorithm represented in this work is easy to manipulate, as the runtime is determined by the depth parameter of the solver. This parameter determines how much the local search of the algorithm expands the search space before determining it has found either the global minimum or the most reasonable local minimum given the maximum allowed execution time. The benefit of having an autonomous system to build and/or evaluate the path of the robot is the ability to analytically avoid/remove wasted off-part motion by the robot which decreases cost and time effectiveness of the layup process. With the optimization of off-part motion reaching approximately 50-80% (when common process errors are present), we are one step closer to making the AFP process more cost effective. This would benefit all industries looking to expand their use of AFP machines.

6 FURTHER CONSIDERATIONS

There are still several challenges to overcome in the future to expand and improve on the work described in this article. Firstly, the partitioning of the part currently happens on the level of plies. As plies are simply a grouping of courses within the same geometric boundary, the intersection testing used to determine if a ply can be appended to a precedence set could be lowered to the scope of courses. Using this technique, the solver could generate results following the same initial requirement and still take partial plies (the courses of the ply that are non-intersecting), thus leaving the potential for a greater decrease in off-part motion. Next, the seeding process has proven to be crucial to the quality of the results of the algorithm. The critical influence of this selection process makes it worth investigating further and determining if other options or heuristics can be used to better influence the output. One caveat of this work, the assumption that inter-precedence graph edges are of weight 0, is a direct result of not currently having a better seeding mechanism. Thirdly, courses very commonly contain portions of on-part motion where no tows are being laid. In this scenario, there is again opportunity to reroute the robot without affecting part qualification. This technique is commonly used as a purpose of optimization but could, in fact, generate separate courses (and therefore vertices in a precedence set's graph), again leading to the potential for finding more optimal routes for an AFP machine. It is also worth investigating possible beneficial data structures for each precedence set's graph to use to reduce the overall runtime complexity of the solver. All parameters used in the algorithm of this work were constructed empirically and, hence, may not be optimal for a wider data set. Thus, tuning of these parameters could also improve the abilities of the solver. Finally, this work uses a minimalistic model of a robot, namely the Denavit-Hartenberg parameters which simply provides the kinematics of the AFP machine. With a more detailed model of the robot, more criteria could be taken into consideration in the machine distance heuristic which directly impacts the quality of the results. This information could be used to analytically weight axis motions leading to more reliable and accurate results.

Roudy Wehbe, https://orcid.org/0000-0003-0615-7634
Ramy Harik, https://orcid.org/0000-0003-1452-9653

REFERENCES

[1] Aized, T.; Shirinzadeh, B.: Robotic fiber placement process analysis and optimization using response surface method, The International Journal of Advanced Manufacturing Technology, 55(1–4), 2011, 393–404. http://doi.org/10.1007/s00170-010-3028-1

[2] Alatartsev, S.; Stellmacher, S.; Ortmeier, F.: Robotic Task Sequencing Problem: A Survey, Journal of Intelligent & Robotic Systems, 80(2), 2015, 279–298. http://doi.org/10.1007/s10846-015-0190-6

[3] Albazzan, M.A.; Harik, R.; Tatting, B.F.; Gürdal, Z.: Efficient design optimization of nonconventional laminated composites using lamination parameters: A state of the art. Composite Structures, 209, 2019, 362–374. http://doi.org/10.1016/j.compstruct.2018.10.095

[4] Arian Nik; M., Fayazbakhsh; K., Pasini; D., Lessard, L.: Optimization of variable stiffness composites with embedded defects induced by Automated Fiber Placement, Composite Structures, 107, 2014, 160–166. http://doi.org/10.1016/j.compstruct.2013.07.059

[5] Böckenhauer, H.-J.; Mömke, T.; Steinová, M: Improved approximations for TSP with simple precedence constraints, Journal of Discrete Algorithms, 21, 2013, 32–40. http://doi.org/10.1016/j.jda.2013.04.002

[6] Castelino, K.; D'Souza, R.; Wright, P.K.: Toolpath optimization for minimizing airtime during machining, Journal of Manufacturing Systems, 22(3), 2003, 173–180. http://doi.org/10.1016/S0278-6125(03)90018-5

[7] Černý, V.: Quantum computers and intractable (NP -complete) computing problems, Physical Review A, 48(1), 1993, 116–119. http://doi.org/10.1103/PhysRevA.48.116

[8] Croft, K.; Lessard; L., Pasini, D.; Hojjati, M.; Chen, J.; Yousefpour, A.: Experimental study of the effect of automated fiber placement induced defects on performance of composite laminates, Composites Part A: Applied Science and Manufacturing, 42(5), 2011, 484–491. http://doi.org/10.1016/j.compositesa.2011.01.007

[9] Debout, P.; Chanal, H.; Duc, E.: Tool path smoothing of a redundant machine: Application to Automated Fiber Placement, Computer-Aided Design, 43(2), 2011, 122–132. http://doi.org/10.1016/j.cad.2010.09.011

[10] Durbin, R.; Willshaw, D.: An analogue approach to the travelling salesman problem using an elastic net method, Nature, 326(6114), 1987, 689-691. http://doi.org/10.1038/326689a0

[11] Harik, R.; Saidy, C.; Williams, S.J.; Gurdal, Z.; Grimsley, B.: Automated Fiber Placement defect identity cards: cause, anticipation, existence, significance, and progression, In Sampe Conference Proceedings, Long Beach, California, 2018

[12] Held, M.; Karp, R.M.: A Dynamic Programming Approach to Sequencing Problems, Journal of the Society for Industrial and Applied Mathematics, 10(1), 1962, 196–210. http://doi.org/10.1137/0110015

[13] Johnson, D.; McGeoch, L.: The Traveling Salesman Problem: A Case Study in Local Optimization. Department of Mathematics and Computer Science, Amherst College, 1995.

[14] Khan, W.A.; Hayhurst, D.R.; Cannings, C.: Determination of optimal path under approach and exit constraints, European Journal of Operational Research, 117(2), 1999, 310–325. http://doi.org/10.1016/S0377-2217(98)00263-X

[15] Li, C.L.: A geometric approach to boundary-conformed toolpath generation, Computer-Aided Design, 39(11), 2007, 941–952. http://doi.org/10.1016/j.cad.2007.06.002

[16] Lien, Y.: Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem, Information Sciences, 74(1–2), 1993, 177–189. http://doi.org/10.1016/0020-0255(93)90133-7

[17] Lin, S.; Kernighan, B.W.: An Effective Heuristic Algorithm for the Traveling-Salesman Problem, Operations Research, 21(2), 1973, 498–516. http://doi.org/10.1287/opre.21.2.498

[18] Lukaszewicz, D.H.J.A.; Ward, C.; Potter, K.D: The engineering aspects of automated prepreg layup: History, present and future, Composites Part B: Engineering, 43(3), 2012 997–1009. http://doi.org/10.1016/j.compositesb.2011.12.003

[19] Oysu, C.; Bingul, Z.: Application of heuristic and hybrid-GASA algorithms to tool-path optimization problem for minimizing airtime during machining, Engineering Applications of Artificial Intelligence, 22(3), 2009, 389–396. http://doi.org/10.1016/j.engappai.2008.10.005
[20] Rajan, S.; Sutton, M.A.; Wehbe, R.; Tatting, B.; Gürdal, Z.; Kidane, A.; Harik, R.: Experimental investigation of prepreg slit tape wrinkling during automated fiber placement process using StereoDIC, Composites Part B: Engineering, 160, 2019, 546–557. http://doi.org/10.1016/j.compositesb.2018.12.017
[21] Reichert, S.; Schwinn, T.; La Magna, R.; Waimer, F.; Knippers, J.; Menges, A.: Fibrous structures: An integrative approach to design computation, simulation and fabrication for lightweight, glass and carbon fibre composite structures in architecture based on biomimetic design principles, Computer-Aided Design, 52, 2014, 27–39. http://doi.org/10.1016/j.cad.2014.02.005
[22] Rousseau, G.; Wehbe, R.; Halbritter, J.; Harik, R.: Automated Fiber Placement Path Planning: A state-of-the-art review, Computer-Aided Design and Applications, 16(2), 2018, 172–203. http://doi.org/10.14733/cadaps.2019.172-203
[23] Sabido, A.; Bahamonde, L.; Harik, R.; van Tooren, M.J.L.: Maturity assessment of the laminate variable stiffness design process, Composite Structures, 160, 2017, 804–812. http://doi.org/10.1016/j.compstruct.2016.10.081
[24] Wehbe, R.: Modeling of Tow Wrinkling in Automated Fiber Placement based on Geometrical Considerations, Master Thesis, University of South Carolina, SC, 2017.
[25] Xu, Y.; Nechyba, M.: Fuzzy inverse kinematic mapping: rule generation, efficiency, and implementation, In Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS ’93). IEEE, 1993,911–918. http://ieeexplore.ieee.org/document/583251/