Path Optimization Along Lattices in Additive Manufacturing Using the Chinese Postman Problem

Gregory Dreifus,[1] Kyle Goodrick,[1] Scott Giles,[2] Milan Patel,[2] Reed Matthew Foster,[2]
Cody Williams,[2] John Lindahl,[1] Brian Post,[1] Alex Roschli,[1] Lonnie Love,[1] and Vlastimil Kunc[1]

Abstract

We develop a method for programming minimal time tool paths for single-bead-wide extrusions in additive manufacturing (AM) along an arbitrary lattice. We present a graphical model of the three-dimensional (3D) printing process and use the solution to the Chinese Postman Problem (CPP) to optimize the motion of an extruder on a given mesh. We present some graph theory background and explain how to solve the CPP. We then present experimental results, in which we demonstrate the implementation of the CPP in 3D printing. Last, we explain how our graphical understanding of AM can be further utilized to achieve greater optimization in additive research.

Keywords: additive manufacturing, 3D printing, graph theory, Chinese Postman Problem, software, lattices

Introduction

One of the challenges in designing machining software is developing the optimal tool path for the robotic hardware to follow when completing a task. The demand for optimal tool paths is especially acute in the field of additive manufacturing (AM) (more commonly known as three-dimensional [3D] printing), which gives the user great control in determining the task of the machine. Specifically, 3D printing technology lets a user design a computer aided design (CAD) image with the expectation that the machine will be adaptive enough to print the desired output. This requires that the technology have as little error as possible, demanding that the processes of the machine be as optimal as possible. The algorithms we developed for this article focused on fused deposition modeling (FDM) in which a mechanical printer head deposits a polymer material layer by layer to complete a print. Our method is most likely applicable to additive metal manufacturing as well, but we did not implement our research in that area.

Suboptimal tool path planning can increase the time it takes to print a part, giving extruded material too much time to cool and therefore resulting in increased warping on larger scale printing processes. However, research on tool path optimization in AM is rare. Rather, work on path planning has been done on machining software in general, most notably to try to minimize airtime.[1] An optimal tool path algorithm for a 3D printer must not only minimize air time but must also traverse the geometry of the printed part in minimal time as well.

There is a more substantial body of research for tool path planning in computer numerical control (CNC) machining than in AM. The theoretical underpinnings of the so-called milling problem, which algorithmically determines how to cover a given region with a mill without lifting the milling machine, are described in.[2,3] However, this approach allows the milling machine to traverse previously visited regions, which is impossible in FDM. There have been studies of optimized CNC machining with minimal turning and minimal airtime.[1,4] Optimization approaches have also been studied for surface finish, speed, and feed rate parameters.[5,6]

In this article, we make use of a graph theoretic approach to design a minimal time tool path algorithm for arbitrary meshes in 3D printed parts. Specifically, we utilize a solution to the Chinese Postman Problem (CPP) to optimize tool paths along in arbitrary lattice in a 3D printed part. By implementing the solution to the CPP on mesh grids, we were able to save a significant amount of time on prints and in fact empowered 3D printers to print geometries that could not be processed through convention AM software.

Current iterations of AM software cannot generate arbitrary infill patterns, often relying on paradigms in CNC machining. They rely on a finite number of possible geometric patterns: raster, zigzag, contour, spiral, fractal, and hybrid. One of the

[1]Manufacturing Demonstration Facility, Oak Ridge National Laboratory, Knoxville, Tennessee.
[2]Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, Tennessee.

most common infill path generation methodologies is a cross-hatching scheme, a back-and-forth straight line movement as seen in Dunlavey.[7] A zigzag pattern was deployed in Park and Choi[8] and Rajan *et al.*[9] Contour patterns, which travel along the inset of a part's boundary have been discussed in Farouki *et al.*,[10] Yang *et al.*,[11] and Li *et al.*[12] Spiral patterns were developed in Wang *et al.*[13] and Ren *et al.*[14] and fractal infills were developed in Kulkarni *et al.*[15] and Bertoldi *et al.*[16] Hybrid methodologies of these schemes can be found in Zhang *et al.*,[17] Dwivedi and Kovacevic,[18] and Ding *et al.*[19]

This may seem like a large amount, but they do not provide algorithms for arbitrary infill meshes. Furthermore, the particularities of the algorithms developed do not explicitly optimize the tool path. Tool path optimization techniques were discussed in Jin *et al.*,[20] Xiaomao *et al.*,[21] and Choi and Cheung,[22] but these techniques are not easily generalized. Our methodology enables printers to output any such infill, in addition to any of the infills listed above, so long as there exists a mesh representative of that geometry. By employing a graphical model, we can generalize our optimization results, which can then by tailored for a specific case.

The capacity to print arbitrary lattices in optimal time has a number of industrial applications. The Tennessee-based company Branch Technology prints lattices for interior support structures in the construction industry.* In another application, the company WirePrint is printing frames based on the STL facets of a given geometry.† Printing core meshes also have applications in the aerospace industry, like via Northrop Grumman's Astro Mesh.‡ Utilizing our approach, the applications can potentially achieve faster and more flexible 3D printing.

The format of the article is as follows: we present some background about AM, explain our graphical interpretation of the AM process and the CPP, and then give our experimental results. For our experimental results, we printed a variety of tessellation patterns, including triangular and hexagonal meshes. We then measured the time it took to print before and after implementing the CPP solution.

Materials and Methods

AM is the field of 3D printing. AM allows a mostly untrained user to design a CAD model, which the machine then prints as output with as much precision as possible. FDM systems make use of an extruder that deposits polymer onto a printer bed in the geometry of a two-dimensional (2D) intersection of the CAD design. In its current form, AM software converts the image into a triangulated approximation and then culls out 2D projections of the approximated image. Ultimately, the software converts the original CAD image into lists of data points that approximate each layer of the print. These data points are then fed into a G-Code parser. G-code is an industry standard machining language that tells the machine how to move.

For our prints, we ran experiments on Oak Ridge National Laboratory's (ORNL) Big Area Additive Manufacturing (BAAM) system. The BAAM system was developed by ORNL and is 6 m long, 2.4 m wide, and 1.8 m tall. It runs a single-screw extruder with a feedstock of carbon fiber reinforced acrylonitrile butadiene styrene (ABS).[23] The feedstock pellet size is 3.45 mm in diameter and 3.75 mm in length. By reinforcing ABS with carbon fiber, we greatly improve the mechanical stress properties of a part. Our method allows for the printing of single-bead-wide mesh grids, and so greater stress properties in a print are imperative.[24] We also ran prints on a ShopBot Desktop Max, a CNC system setup with an air pump system. We attached on a nozzle filled with epoxy base resin and printed with this as well for experiments on thermoset printing.

Research has been done to streamline the software flow for AM, thereby completely revolutionizing the nature of the typical designing process. It is now possible to design based on what the known output will become and iterate at a rapid pace. This design flow is called Direct Digital Manufacturing.[25]

Graphical models have been used before to optimize machining tool paths.[18,26,27] In formal terms, a graph $G = \{V, E\}$ is a pairing of a vertex set $V = \{v_1, v_2, \ldots, v_n\}$, for some integer n and an edge set $E = \{v_i v_j | i \neq j\}$, where $v_i v_j$ is an edge connecting vertices v_i and v_j, and $1 \leq i \leq n$ and $1 \leq j \leq n$. In our representation, we let our vertices $v_i = (x_i, y_i)$ be the points along which the printer head will run and output by the parsing software. We then weigh each edge $v_i v_j$ by the amount of time it takes the printer head to traverse the distance between vertex v_i and v_{i+1}. The final necessary definition is the degree of a vertex, which is the number of edges to which a given vertex is connected by edges. For example, if vertex v_1 is connected to vertex v_{15} by an edge and v_2 by an edge, we say the vertex has degree two. If the vertex has odd degree, we call it an odd vertex, and similarly for even vertices. Last, a list of the degrees of all the vertices of a graph is called a degree sequence.

The CPP asks the following question: having begun at a given vertex on a graph, what is the minimal distance required to traverse all the edges of the graph? CPP is a well-known mathematical problem with a well-known solution. Its implementation in AM is novel. The idea behind CPP's solution is simple: first, one checks to see if the graph is Eulerian. An Eulerian graph is one in which it is possible to trace every edge of the graph without picking up one's pen. If the graph is Eulerian, then the minimal distance to the next neighbor is followed. If the graph is not Eulerian, then the minimal number of requisite edges is added to the graph to make it Eulerian. In physical terms, this means that the printer head must lift into the air and cross the printer bed to the next consecutive point. Once the graph is made Eulerian, the so-called Eulerian trail is followed. The last exceptional circumstance occurs with a semi-Eulerian graph, a graph with only two odd vertices. In the case of a semi-Eulerian graph, it is possible to take a Eulerian trail that begins at one odd vertex and ends at the other. It is worth noting a well-known theorem that a graph contains an Eulerian path if and only if all its vertices are even, a fact crucial to solving the CPP.

The nature of AM and the current state of its software make the solution to the CPP a natural fit for tool path optimization. The points that define the G-Code path correspond naturally to the vertices of a graph. Also, the extra edges added to the graph by the CPP algorithm correspond smoothly to what we denote ''airtime edges,'' printer head motion with the extruder turned off, while edges in the original graph correspond to so-called ''extrusion edges,'' printer head motion with the extruder turned on.

*www.branch.technology
†http://stefaniemueller.org/wireprint
‡www.northropgrumman.com/businessventures/astroaerospace/products/pages/astromesh.aspx

So, the algorithm is as follows: develop a degree sequence. If the vertices are all even, proceed along the path such that the sum of the weights is smallest. If there are odd numbers in the degree sequence, determine a list of all odd vertices. Then, list all possible pairings of odd vertices and calculate the weights between each possible pairing. Find the set of pairing such that the sum of the weights is minimized. Add the edges to the graph corresponding to those minimal weight pairings. All vertices on the graph are now even; so proceed along the path such that the sum of the weights is smallest. If the graph is semi-Eulerian, start the path at one of the odd vertices and follow the nearest neighbor, while avoiding the other odd vertex if possible.[28]

Ultimately, the solution to the CPP we employ entails the use of several well-known graph theoretic algorithms: First, we implement Hierholzer's algorithm to check to see if our graph is Eulerian. Hierholzer's algorithm follows a system of loops, in which one begins at an arbitrary vertex and follows an arbitrary path until one returns to the original vertex. If the tour (the looped path) is completed, begin a new looped path without traversing an edge already visited. For a graph $G=(V,E)$, Hierholzer's algorithm is $\Theta(|E|)$.[27] If an Eulerian graph is not found, we then proceed with an optimal matching algorithm known as the Blossom algorithm. The Blossom algorithm relies on the fact that, if a vertex is matched uniquely with another vertex, the resultant edge is completely disconnected from other edges in the graph. As such, the Blossom algorithm finds a matching in which the path traversed along the alternating edges of the matching begins and ends at different vertices. If such a path begins and ends at vertices isolated from the matching, the matching is not maximal, and the symmetric difference of the path and the matching is taken, given an optimal matching. If no such path ending at isolated vertices exists, it is called maximal. The Blossom algorithm runs in $O(|V|^2|E|)$.[29,30] We run the Blossom algorithm on the set of odd vertices in the graph, and we put the edges formed by the maximal matching into the edge set. The newly formed graph is now Eulerian, and we run Hierholzer's algorithm on it. The pseudocode for the algorithm ends up being extremely straightforward:

```
if Hierholzer is not successful Blossom
else Hierholzer
```

We outline pseudocode for Hierholzer and Blossom at the end of this section.

In the classical CPP solution, odd vertex pairings are weighted by backtracking along the weighted edges that created paths between odd pairs. Because our vertices represent actual coordinates in $\mathbb{R}^2$, as opposed to ambient space, we are able to measure weighted pairs by the Euclidean distances between them. However, our approach is easily transferable to the classical case by using a graph metric. This enabled us to speed up our matching using the Blossom algorithm. Inspired by network systems control theory, we took advantage of the mechanical realities of our problem and developed an algorithm that looks at the local nodes around each odd vertex. We created subgraphs by searching those odd vertices that were in a distance $<\epsilon$ around a given vertex, and we searched odd vertices among those subgraphs. Our enhanced odd pairings search method is easily extended to the graph metric as well by searching open balls with radius $<\epsilon$ in accordance with whatever metric one decides upon.

While a brute force approach toward odd pairings is of the order $O(n!)$, our search is $O(n^2+(n-k)!)$, where k is the minimum number of vertices removed to form the largest subgraph over all the vertices. This is always faster than $O(n!)$.

Pseudocode for the Hierholzer algorithm is as follows[31]:

```
tour = $\emptyset$
$v_0$ = graph[0][0]
tour.append($v_0 t$)
while len(graph) > 0 do
    for edge in graph do
        if $v_0 \in$ edge then
            if edge[0] == $v_0$ then
                $v_0$ = edge[1]
            else:
                $v_0$ = edge[0]
            graph.remove(edge)
            tour.append($v_0$)
            break
        else: return false
    return tour
```

We take our pseudocode for the Blossom algorithm directly from[32] the following:

Let M be a matching and Q a queue, holding a single unmatched vertex r_1 in graph G. Label r_1 EVEN. An augmenting path is an alternating path that starts and ends at two distinct vertices a and b, such that, for any $e \in M$, e is not incident to a or b.

```
MAIN ROUTINE Û
If Q $\neq \emptyset$ then
    take vertex $v$ off the head of the queue
    if $v$ is labeled EVEN then
        using breadth-first search, move along all of the unmatched edges emanating from $v$. Call the set of vertices on the opposite end of such edges $W$
        add the vertices in $W$ to the queue
        for $w \in W$ do
            if $w$ is not in an edge in $M$ then
                set $AP_B$ equal to the path from $w$ to $r_1$.
                expand the odd length cycle that alternates on the current matching (denoted a blossom) in the reverse order in which they were shrunk, propagating the augmenting path as follows:
                restore the edges within the blossom.
                reattach edges currently attached to $v_B$ back to the vertices in the blossom to which they were originally attached before shrinking.
                extend AP through expanded blossom.
                recurse out of MAIN ROUTINE, propagating $AP_B$ through each expanded blossom.
                upon arriving at G, and concurrently $A$P, flip the matching of $A$P. Go to Û.
                restart entire routine.
            if $w$ is in an edge in $M$ and is labeled EVEN then
                shrink the blossom, whose vertices are the symmetric difference of $P_1$, the path from $v$ to $r_i$, the initial $M$-uncovered vertex, and $P_2$, the path from $w$ to $r_i$. To shrink the blossom:
                remove all edges in the blossom.
                reattach edges originally attached to the blossom vertices to $v_B$
            recurse into MAIN ROUTINE, starting at Û.
          if $w$ is in an edge in $M$ and is unlabeled then
            label $w$ ODD
        rake $v$ off of the queue.
      if $v$ is labeled ODD then
        move along matched edge to vertex $h$
        if $h$ is labeled ODD then
          shrink the blossom, whose vertices are the symmetric difference of $P_1$, the path from $v$ to $r_i$, the initial $M$-uncovered vertex, and $P_2$, the path from $h$ to $r_i$. To shrink the blossom:
          remove all edges in the blossom.
          reattach edges originally attached to the blossom vertices to $v_B$
          recurse into MAIN ROUTINE, starting at Û.
        if h is unlabeled then
          label $h$ EVEN.
        add $h$ to the queue.
      if $Q \neq \emptyset$ add a vertex $r_2$ to $Q$ which is unlabeled and not in an edge in $M$, if it exists. Go to Û.
    else
    end
```

Results

We utilized several pieces of open source software that each implemented Hierholzer's algorithm and the Blossom algorithm and connected them to a less efficient open source CPP solver. We tailored these pieces of software to inputs and outputs that suited our 3D printers. To test our algorithm, we developed a rudimentary hexagonal grid generator that could produce the coordinates of a grid of hexagons of multiple sizes (Fig. 1). We also tailored our code to read meshes generated in Mathematica. In this way, we were able to produce complex geometric mesh structures and showed that we could print these in an optimal way.

Our software produced output in G-Code that we read into small- and large-scale 3D printers. We ran our algorithm on a multisize hexagonal grid and printed it in epoxy on a thermoset printer we had connected to a Shopbot. We ran two different geometries on our large-scale ''Blue Gantry'' printer, developed at the Manufacturing Demonstration Facility. One geometry was a wheel with a spiral mesh; the other was a triangulated mesh contained in a rectangle. We chose these geometries because they give a clear demonstration of the utility of our software and the manner in which our software handles various geometries (Table 1).

The multisize hexagonal grid is currently impossible to print with existing tool path planning AM software. Three-dimensional printers extrude a given material into a part in what we call ''beads.'' The present state of AM software cannot handle parts that are a single-bead wide. If one were to design a hexagonal grid using CAD software and export it to an STL file that is converted to G-Code using slicing software, such as MDF's ORNL Slicer, the printer would print each hexagon separately, making each hexagon two beads wide, and the print would ultimately be several adjacent hexagons, rather than a single-bead-width grid structure as intended. Our algorithm printed our hexagonal grid with only a single bead. We compared print times between the print in Figure 1 and the same part designed and printed by Slic3r software. The latter print time was 80 s for a single layer, and our methodology took 64 s, saving 20% of the total print time (Table 2).

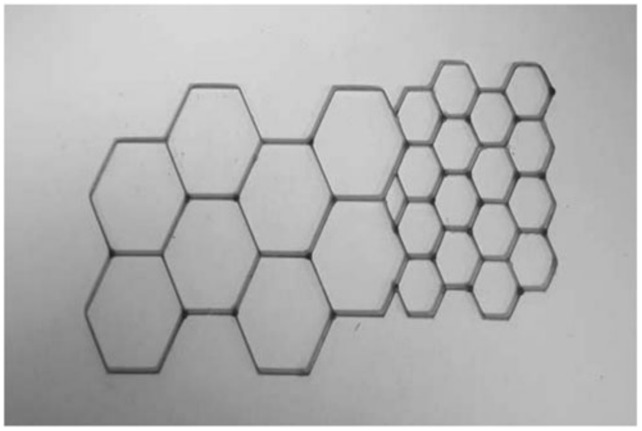
FIG. 1. Samples of thermoset-printed, multisize hexagonal grids.

We also ran experiments printing grids divided into various numbers of chambers: 2×2, 3×3, and so on up until 9×9. For each grid, we ran an implementation of the print time using the CPP software against analogous grids input into Slic3r and the ORNL Slicer. Prints utilizing Slic3r were run on the ShopBot with printer speed set to 960 PWMs. They were 1×1 ft in size. Prints utilizing the ORNL Slicer were run on the ORNL BAAM system with printer speed set to 2.008 in/s. Since the slicing software cannot process single-bead-width paths, all output from the slicers ran on double-bead-width paths. Even taking this discrepancy into account, all our print times ran less than half the print times of the Slic3r tool paths and less than three-quarter the print time on the ORNL Slicer tool paths.

The wheel with a spiral mesh was printed because, from a graphical perspective, it was designed to have no odd vertices and so be an Eulerian graph (Fig. 2). As such, the printer head never needed to lift to print an entire layer of the wheel. The wheel therefore gives clear evidence that the solution to the CPP gives a minimal time path for any single-bead-wide path of a given part. The triangular mesh was printed to show that the algorithm does exactly what it should for its airtime moves. That is to say, airtime moves in a print are precisely

TABLE 1. PRINTS OF 3×3 FT GRIDS WERE RUN BY MEANS OF THE CHINESE POSTMAN PROBLEM SOLUTION VERSUS ORNL SLICER SOFTWARE AND PRINT TIMES RECORDED

Print times—BAAM		
Grid size	*CPP tool path*	*ORNL slicer tool path*
9×9	7 m 16 s	12 m 13 s
8×8	6 m 49 s	11 m 9 s
7×7	6 m 17 s	9 m 51 s
6×6	5 m 26 s	8 m 44 s
5×5	4 m 46 s	7 m 22 s
4×4	4 m 5 s	6 m 11 s
3×3	3 m 17 s	4 m 56 s
2×2	2 m 39 s	3 m 41 s

BAAM, Big Area Additive Manufacturing; ORNL, Oak Ridge National Laboratory.

Table 2. Prints of 1×1 ft Grids Were Run by Means of the Chinese Postman Problem Solution Versus Slic3r Software and Print Times Recorded

Print times—shop bot		
Grid size	*CPP tool path*	*Slic3r tool path*
9×9	5 m 26 s	16 m 9 s
8×8	4 m 52 s	14 m 22 s
7×7	4 m 14 s	12 m 59 s
6×6	3 m 42 s	11 m 10 s
5×5	3 m 8 s	9 m 44 s
4×4	2 m 40 s	7 m 56 s
3×3	2 m 4 s	6 m 11 s
2×2	1 m 33 s	4 m 35 s

CPP, Chinese Postman Problem.

those edges added between the odd vertices of the graph. Furthermore, it shows that this algorithm can handle complicated infill patterns that the current stage of AM could not.

The hexagonal grid and triangular mesh also show that airtime moves are necessary in certain print geometries, a helpful fact that is important to designers looking to print complicated infill patters with minimal error. A designer should try and minimize the amount of odd edges in an infill design to minimize lifts, minimizing airtime and therefore total time of a print. Airtime moves also result in tiering effects of the extrusion. In larger prints, these tiers can dry and the error can propagate in a part.

For our experiments, we wrote elementary G-Code generating software. So, our printer commands did not implement some of the technical tricks state-of-the-art printers use to combat commonplace error in 3D printing. Specifically, the Blue Gantry system employs a ''tip wipe'' move to prevent tiering. Furthermore, the size of our prints was too large for our printer beds. It was necessary to make our prints as large as they were, so that they could maintain the detail of some of the finer parts of a mesh, but by virtue of their size, our prints extended off the vacuum sealed beds of our printers, causing lamination problems. Also, without a tip wipe move, our printer head ran over nodes multiple times, quickly causing detrimental problems in our print. On the ShopBot, our prints of hexagonal grids were tested as part of experimentation on thermoset printing. So we experienced problems with accumulation of the thermoset material on our printer head. For these reasons, and because tip wipe techniques are a well-understood technique, we only ran one layer thick on the Blue Gantry and no more than 5–10 layers on the ShopBot because the effectiveness of the CPP solution was still demonstrated (Fig. 3).

Discussion

Our software gives a clear temporally optimal method for path planning in 3D printing. It can be used to print the previously unprintable single-bead-wide paths and facilitate the printing of arbitrary mesh infill patterns. This means that infills can be of various sizes within a part to control stress properties.

It is common practice to print double-bead-width walls for the perimeter of a given part. For proper implementation, our software must be incorporated into slicing software. This can be done by including the vertices of the inner bead wall in the vertex set of the graph defining a given infill pattern. For future work, this is how we intend to implement our software into the ORNL Slicer. Once our software is integrated into slicing software, issues such as tip wiping and tiering effects will be resolved.

Our algorithm can also be extended to optimally oriented parts output from a given CAD file. If a user intends to print multiple objects at once, our software can be retrofitted to minimize airtime travel as the printer head moves between the objects.

Our methodology searches for a path that ensures the printer head will visit all edges of our graph under minimal weight. At present, our weights are determined solely by the time it takes to travel an edge. However, many mechanical and physical processes are significant in determining the outcome in 3D printing. Factors like temperature and stress are also important. A graphical representation of AM gives a simple means of incorporating such factors into tool path planning. One simply needs to set weights to some cost function of temperature, stress, and time. This is a ripe area for future research.

One other improvement that could be made to our Chinese Postman solution is that Hierholzer's algorithm could be made more intelligent. If a vertex is visited more than once, it would be ideal if the printer did not pass entirely over an already printed edge. Rather, the printer head should choose the next edge to visit the edge that forms the smallest angle with the previously visited edge, if possible. This improvement should also have the effect of keeping printer head motion as localized as possible, empirically preventing too much cooling in a part as a print proceeds. Last, the Blossom algorithm is perhaps the most efficient methodology in the literature for optimal matching algorithms. Further research

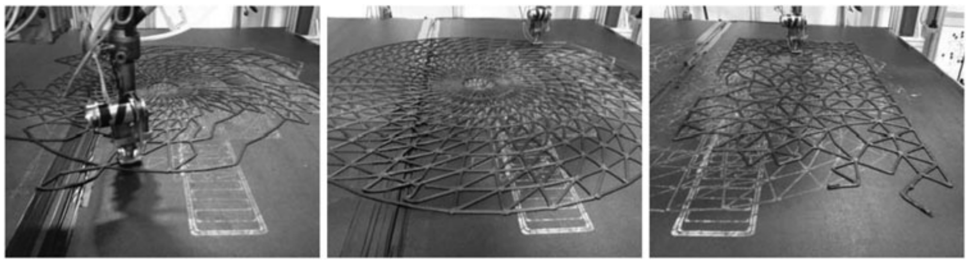

FIG. 2. Demonstrations of prints on the Blue Gantry system. The *left figures* show the meshed wheel, and the *right figure* is the triangulated mesh. One can see the progression of each print and the algorithm that determines them is able to complete the printed layer.

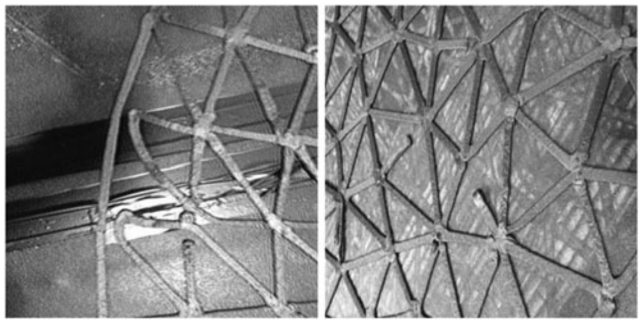

FIG. 3. Errors caused by lamination. One can also see how an implementation of a tip wipe is necessary.

in the area of maximal matchings would be a boon to the AM software community.

Conclusion

We developed software that gives a graphical representation of the AM process. We then ran the resultant graph into the solution to the CPP, which determines the minimally weighted path along all the edges of an undirected weighted graph. This representation gives a minimal time path for an arbitrary single-bead-wide print. It also enables the printing of single-bead walls in general, allowing novel possibilities in 3D printer motion. It enables one to print arbitrary lattice structures in optimal time. This significantly saves on printing time, which is crucial for better AM.

Acknowledgments

We would like to acknowledge the Manufacturing Demonstration Facility, Oak Ridge National Laboratory, the Department of Energy, UT-Batelle, Oak Ridge Associated Universities, and the DOE's Advanced Manufacturing Office for their support in this research.

Author Disclosure Statement

This article has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725, with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

References

1. Castelino K, Wright PK. Tool-path optimization for minimizing airtime during machining. J Manuf Syst 2014;22:173–180.
2. Arkin EM, Fekete SP, Mitchell JSB. Approximation algorithms for lawn mowing and milling. Comput Geometry 2000;17:25–50.
3. Arkin EM, Bender MA, Demaine ED, Fekete SP, Mitchell JSB, Sethia S. Optimal covering tours with turn costs. arxiv.org, June 21, 2005.
4. de Assis IR, de Souza CC. Experimental evaluation of algorithms for the orthogonal milling problem with turn costs? In: Experimental Algorithms Pardalos PM, Rebennack S. 10th International Symposium Proceedings, SEA 2011, Kolimpari, Chania, Crete, Greece, May 2011, pp. 304–315.
5. Kumar N, Chhabra KK. An overview of optimization techniques for CNC milling machine. Int J Eng Manage Sci 2014;1:13–16.
6. Sencer B, Altintas Y, Croft E. Feed optimization for five-axis CNC machine tools with drive constraints. Int J Mach Tools Manuf 2008;48:733–745.
7. Dunlavey MR. Efficient polygon-filling algorithms for raster displays. ACM Trans Graph 1983;2:264–273.
8. Park SC, Choi BK. Tool-path planning for direction-parallel area milling. Comput Aid Des 2000;32:17–25.
9. Rajan V, *et al.* The optimal zigzag direction for filling a two-dimensional region. Rapid Prototyp J 2001;7: 231–241.
10. Farouki R, *et al.* Path planning with offset curves for layered fabrication processes. J Manuf Syst 1995;14:355–368.
11. Yang Y, *et al.* Equidistant path generation for improving scanning efficiency in layered manufacturing. Rapid Prototyp J 2002;8:30–37.
12. Li H, *et al.* Optimal toolpath pattern identification for single island, sculptured part rough machining using fuzzy pattern analysis. Comput Aid Des 1994;26:787–795.
13. Wang H, *et al.* A metric-based approach to two-dimensional (2D) tool-path optimization for high-speed machining. Trans Am Soc Mech Eng J Manuf Sci Eng 2005;127:33.
14. Ren F, *et al.* Combined reparameterization-based spiral toolpath generation for five-axis sculptured surface machining. Int J Adv Manuf Technol 2009;40:760–768.
15. Kulkarni P, *et al.* A review of process planning techniques in layered manufacturing. Rapid Prototyp J 2000; 6:18–35.
16. Bertoldi M, *et al.* Domain decomposition and space filling curves in toolpath planning and generation. Proceedings of the 1998 Solid Freeform Fabrication Symposium, The University of Texas at Austin, Austin, TX, 1998, pp. 267–274.
17. Zhang Y, *et al.* Weld deposition-based rapid prototyping: A preliminary study. J Mater Proc Technol 2003;135:347–357.
18. Dwivedi R, Kovacevic R. Automated torch path planning using polygon subdivision for solid freeform fabrication based on welding. J Manuf Syst 2004;23:278–291.
19. Ding D, Pan Z, Cuiuri D, Li H. A tool-path generation strategy for wire and arc additive manufacturing. Int J Adv Manuf Technol 2014;73:173–183.
20. Jin Y-A, He Y, Fu J-Z, Gan W-F, Lin Z-W. Optimization of tool-path generation for material extrusion-based additive manufacturing technology. Additive Manuf 2014;1: 32–47.
21. Xiaomao H, Chunsheng Y, Yongjun H. Tool path planning based on endpoint build-in optimization in rapid prototyping. Proceedings of the Institution of Mechanical Engineers (Part C) 2011;225:2919–2926.
22. Choi SH, Cheung HH. A topological hierarchy-based approach to toolpath planning for multi-material layered manufacturing. Comput Aid Des 2006;38:143–156.
23. Duty C, Kunc V, Compton B, Post B, Erdman D, Smith R, *et al*. Structure and mechanical behavior of big area additive manufacturing (BAAM) materials. Rapid Prototyp J 2017;23:181–189.

24. Love L, Kunc V, Rios O, Duty C, Elliot A, Post B, *et al.* The importance of carbon fiber to polymer additive manufacturing. J Mater Res 2014;29:1893–1898.
25. Gibson I, Rosen DW, *et al.* Direct Digital Manufacturing Additive Manufacturing Technologies. New York: Springer, 2010, pp. 363–384.
26. Bollweg P, Hasanbegovic D, Müller H, Stöneberg M. Surface-adaptive and collision-avoiding path planning for five-axis milling January 2006. Forschungsbericht No. 808/2006.
27. Jungnickel D. Graphs, networks and algorithms. In: Algorithms and Computation in Mathematics, Chapter 2. Secaucus, NJ: Springer-Verlag New York, Inc., 2014.
28. Eiselt HA, Gendreau M, Laporte G. Arc routing problems, part 1: the Chinese postman problem. Oper Res 1995;43:231–242.
29. Edmonds J. Maximum matching and a polyhedron with 0,1 vertices. J Res Natl Bureau Stand 1965;69B:125–130.
30. Michel X. Goemans, lecture notes on non-bipartite matching February 27, 2007. Handout 5. Available at: http://math.mit.edu/goemans/18433S07/edmonds.pdf (last accessed January 30, 2017).
31. Hierholzer C. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. Math Ann 1873;6:30–32.
32. Kusner M. Edmond's blossom algorithm: A maximum matching algorithm for any graph. Available at http://matthewkusner.come/MatthewKusner_BlossomAlgorithmReport.pdf (last accessed January 30, 2017).

Address correspondence to:
Gregory Dreifus
Manufacturing Demonstration Facility
Oak Ridge National Laboratory
Bldg NRTC-2, 2360 Cherahala Boulevard
Knoxville, TN 37932

E-mail: dreifusgd@ornl.gov