Sequencing of Tissue-Imitating 3D Printed Toolpaths for Time-Optimal Printing Under Physical Printability Constraints

Spencer Bertram

December 2022

1 Abstract

In this paper, a problem regarding sequencing of 3D printer toolpaths is explored. In particular, the goal of this paper is to develop an algorithm for sequencing toolpaths in order to minimize the distance of non-extruding movements, so as to reduce the time of printing. The toolpath sets explored in this paper are generated to mimic fiber-orientation variance in living tissue. As such, the time-optimal ordering of these toolpaths has implications for 3D bioprinting and similar fields. This paper also considers the physical constraints of 3D printing via a concept known as 'printability', which is used in order to reduce the search space of toolpath permutations. This paper explores various approaches to both modeling and searching the toolpath generation problem. In the end, a locally-optimal hill climbing search algorithm is used to generate a time-optimal toolpath sequence that adheres to physical printability constraints.

2 Introduction

3D printing is a fabrication method wherein successive layers of material (with varying cross-sectional shapes) are deposited on top of one another to form a 3D object with a desired geometry. Traditionally, these layers are planar and stacked vertically, parallel to one another. The 'slicing' method for converting 3D object files into individual printable lines (tool paths) in this planar method is relatively straightforward. It is the foundation of the slicing algorithms in almost all commercial slicer software. In a sense, this planar slicing method renders 3D printing a misnomer: repeated 2D printing is more accurate. The planar slicing approach has many faults, including surface tessellation errors and the need for support structures. These drawbacks can be easily mitigated using a non-planar slicing approach, wherein a printer can deposit lines of material that aren't confined to a particular plane: rather, they can travel throughout the entirety of 3D space.

Many such algorithms have been developed - such as smooth surface generation [1], decomposition-based curved slicing [10], and helical (”vase mode”) slicing [8]. Because non-planar slicing is a relatively new area of research, many of these algorithms impose heavy restrictions on the geometry of the input model. Their toolpath generation processes are generally designed to be optimal for a specific ”type” of input model possessing specific features that allow the model to be sliced in a non-planar fashion.

In this paper, a 2D tool path generation problem will be explored. This problem is a sub-problem that arises in non-planar slicing algorithms, especially in the slicing of biological models for 3D bio-printing.

Consider a 2-dimensional rectangular sheet, divided into i by j tiles. Each tile will be assigned a random ”tilt angle” value within a specified range (between $0°$ and $-45°$ from the $+x$ axis). The exact method of this random generation is somewhat arbitrary - it is meant to mimic random fiber orientation deviations in biological tissue, and will be discussed in the *Toolpath Generation Approach* section of this paper.

The goal of the problem is to generate a minimum travel time tool path sequence over the entire i by j grid, where all tool paths in a tile have the same ”tilt angle” that was assigned to the tile. The tool path solution must have ”printability”. That is, for every tool path line l_a, at every point $(x, l_a(x))$ on that line, every line l_b which is defined at x and for which $l_b(x) < l_a(x)$ must have already been traveled through. In other words, once a tool path is traversed, a tool path cannot be traversed below it.

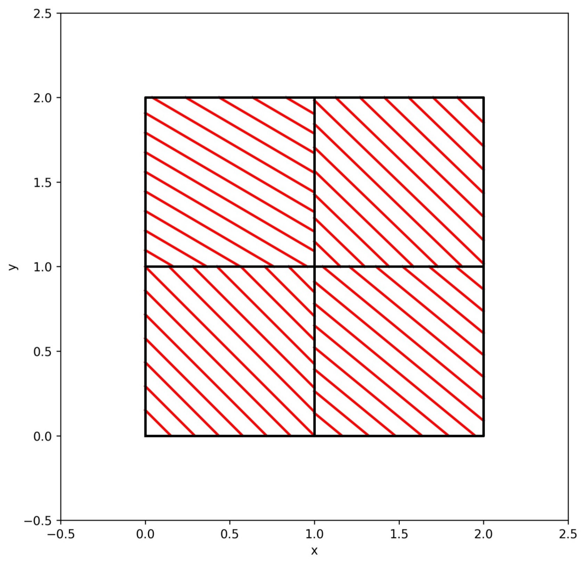
This requirement stems from the origin of the problem: this is a 2D simplification of a 3D problem, where in this case the y-direction would be upwards, and in order to actually print an object, the layers must be deposited from bottom to top. In order to ensure that this requirement is feasible, the starting tile $(i, j) = (0, 0)$ will be assigned a negative tilt angle such that the tool path generation can begin from the bottom left corner and expand in the $+x$ and $+y$ directions. A sample series of tool paths is shown in Figure 1.

It should be noted that the lines in each tile are not connected to any lines in adjacent tiles - in other words, no endpoint is shared by any toolpath lines. It should also be noted that all travel times between nodes - even (and especially) when not laying down material - are considered. This is actually the main hurdle in this problem: the toolpath traversals must be ordered such that the total distance traveled between points while not extruding is minimized.

3 Related Work

This paper explores the problem of toolpath ordering in a plane that is perpendicular to the build plate. Namely, in Figure 1, the $+y$ axis points upward, and the $+x$ axis is parallel to the build plate. However, almost all existing literature that explores toolpath ordering does so in the context of 2D layers that are used in traditional planar slicing (layers that are parallel to the build plate). Both problems can be modeled in essentially the same fashion, besides one key characteristic: printability. In a 2D layer of toolpaths intended to be printed parallel to the build plate, there is no restriction on toolpath sequencing. In the problem

Figure 1: Sample toolpaths with semi-random tile angles generated across a 2x2 grid



explored in this paper, however, toolpaths must be traversed from the "bottom-up", such that an ordering of toolpaths is printable. This greatly reduces the number of possible print line permutations - a fact which will be further explored in the *Toolpath Generation Approach* section of this paper. All literature reviewed in this section lies in the context of traditional, build plate-parallel layers.

3.1 Problem Modeling

The toolpaths in a given layer can be represented by a set of n continuous toolpaths $C = \{c_1, c_2, ..., c_n\}$, where each c_n is an ordering of points which must be traversed. In the problem posed in this paper, each c_i has exactly two points, but in typical slicing, this is not necessarily the case. In most cases, the travel speed of the printer nozzle is constant for a given layer, and at the very least it is proportional to the distance traveled for a given toolpath. It is often convenient to use the distance $d(c_i, c_j)$ as the property of the toolpath sequence to be minimized since it can be easily calculated as the euclidean distance between two points. In some cases, the kinematics of toolpath movement are considered [6] (in particular, the time required for turning corners can be determined). However, this approach rarely affects the output toolpath sequence, since almost all of the travel time is not spent turning. Note that $d(c_i, c_j)$ is the distance between the end point of c_i and the start point of c_j. The goal of minimizing the total travel time then comes down to minimizing the distance function J for a given layer, where [4]

$$J(C) = \sum_{i=0}^{n-1} d(c_i, c_{i+1})$$

Note that the actual travel distance for each path c_i is not considered in this function. If $TD(c_i)$ is the travel distance of toolpath c_i, then the sum of the travel distances for all toolpaths in C:

$$\sum_{i=0}^{n} TD(c_i)$$

is always the same, regardless of the orderings of the toolpaths. The problem then effectively reduces to minimizing the non-extruding toolpath distances between each path c_i - in particular, the goal is to find a permutation of C such that $J(C)$ is minimized.

The problem can also be modeled as a graph where each vertex is a point to visit, and each edge corresponds to either an extruding or non-extruding toolpath. After converting the graph to a Euclidean one using algorithmic approaches, the problem then reduces to the traveling postman problem[3, 4], which is a well-known problem.

3.2 Algorithms Used to Find Optimal Toolpath Ordering

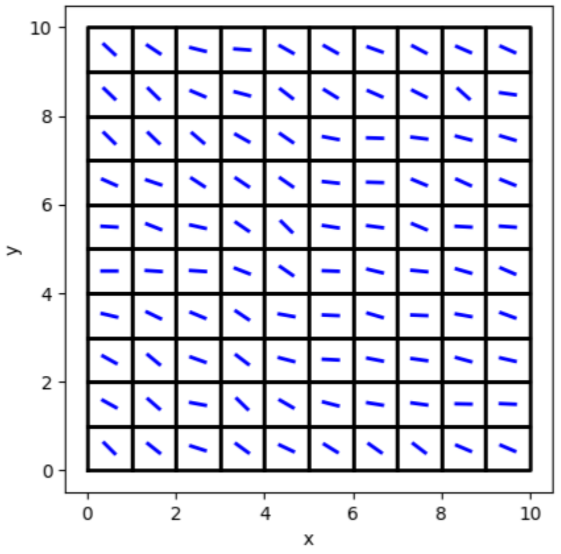
In the case where toolpath ordering is unrestricted (ie. does not adhere to printability), the size of the search space is $n!$, where $n = |C|$. The number of toolpaths in a given set can vary greatly depending on the width of each extruded line and the size of the object being printed, but in general, a rough estimate of 100 toolpaths is reasonable. This would result in a search space size of $100! \approx 10^{157}$. It is obviously not feasible to search this entire space, and so in general non-complete search algorithms must be implemented. Most commonly, locally optimal search algorithms are used [2, 4, 7]. In particular, it is common to run greedy best-first search n times, where each iteration changes the initial starting toolpath c_1 [4]. The evaluation function of this search is d, such that every non-extruding travel from the endpoint of c_i to the start point of c_{i+1} is minimized. The toolpath permutation chosen from the n iterations is the one with the lowest J value. The solution state for this problem is always at the same depth (n), so the criteria for finding a solution state with the greedy best-first search is simply being at a depth of n.

While locally optimal searches like the one described above are the current standard in most research and most slicer software, there have been recent interesting developments in using non-local algorithms like Monte Carlo Tree Search (MCTS)[9] and Ant Colony Optimization (ACO)[5]. It was demonstrated that using MCTS with a relatively short time constraint to order a toolpath set led to a total print time similar to what typical locally optimal searches produce, but when the time constraint was expanded, the total print time was reduced by approximately 15% when compared to typical locally optimal searches.

4 Toolpath Generation Approach

The tilt angle of each tile will be determined via a random-walk process, where a starting tile ($(i, j) = (0, 0)$ - the bottom left corner) will be seeded with an initial value ($-45°$). The angle of each subsequent tile will be determined by taking the average of the tiles to the left

Figure 2: Sample tile angles generated between 0° and $-45°$, with a random deviation range of $-22.5°$ to $22.5°$



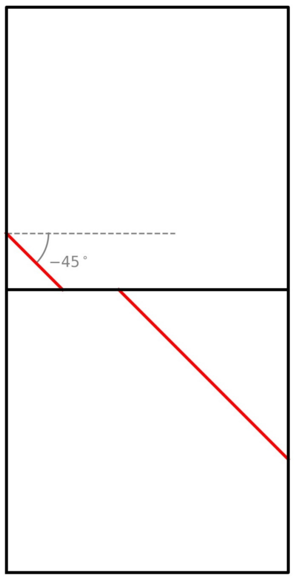
and below it, and then deviating according to a uniform random distribution, with bounds of $-22.5°$ and $22.5°$ for each deviation step. If a value is out of range, it is "rebounded" back into range by the amount it went out of range. A sample 10x10 grid of angles is shown in Figure 2.

As has been done in the previous literature, this problem will be modeled as an ordering problem of the n toolpaths $C = \{c_1, ..., c_n\}$, where the goal is to find an ordering of C such that the total distance traveled between toolpaths is minimized. This distance between two toolpaths is denoted by $d(c_i, c_{i+1})$, which is the euclidean distance between the endpoint of toolpath c_i and the starting point of toolpath c_{i+1}. Overall, the goal is to find an ordering of C that minimizes the function $J(C) = \sum_{i=1}^{n-1} d(c_i, c_{i+1})$.

A search agent of some kind will be used to find this optimal ordering, so it is useful to define valid states and actions for this problem. A state is a set of toolpaths, where each toolpath has two endpoints and has either been traversed or has not been traversed, with the current endpoint and the distance of the last non-extruding travel (the purpose of this will be later explained) indicated. An action is the selection of the next printable toolpath to traverse and the direction in which to traverse it. This reduces to selecting an endpoint from all endpoints corresponding to non-traversed, printable toolpaths.

Not accounting for the printability requirement, the solution space for this problem is absolutely massive, as noted in the *Related Work* section of this paper. In particular, with no restrictions on the toolpath ordering, this problem has a search space of size $n!$. However, printability greatly reduces the search space. Assume we have a $w \times w$ grid containing n lines. Because the steepest possible angle for a line is $-45°$, there can be at most 2 printable

Figure 3: Sample two-tile column with two toolpaths of maximum steepness, indicating that the maximum number of printable toolpaths per column is two



lines in a given column. Figure 3 may provide some intuition for this. With w columns, this means from any toolpath endpoint, there can be a maximum of $2w$ non-traversed, printable toolpaths, each with 2 endpoints. This gives a maximum branching factor of $2 \times 2w = 4w$. By definition, a solution has a depth of n in the search tree. Therefore, accounting for printability reduces the search space size from $n!$ to n^{4w}. Technically speaking, this reduces the time complexity of a complete search algorithm from $O(n!)$ to $O([n^w]^4)$. For grids with a small number of tiles, this search space may be reasonable enough to execute a complete search algorithm.

For example, consider the toolpath set in Figure 1. This set has a total of $n = 52$ toolpaths, with $w = 2$ columns. The search space for this toolpath set is then of size $(52^2)^4)$, which is on the order of 10^{13}. This may be feasible to search with a complete algorithm, given enough time and computational resources. However, this tiling is of a rather impractical size. Any object printed with reasonable resolution will have at least $w = 10$ - assuming 10 toolpaths per tile, this would result in a search space of size $((10 \times 10 \times 10)^{10})^4$, which is on the order of 10^{120}, which is completely infeasible to search with a complete algorithm.

Therefore, even accounting for printability, a reasonably-sized toolpath grid cannot be feasibly searched in a finite amount of time using a complete search algorithm. Therefore, incomplete search algorithms will be explored in this paper. In particular, hill-climbing search will be used, where the purpose of each action is to select the nearest point that lies on a printable line. This will be done by implementing the following value function, V:

$$V(state) = mws\sqrt{2} - d_{prior}$$

Where m is the number of lines in the grid that have already been traversed, w is the number of columns/rows in the grid, s is the side length of each tile, and d_{prior} is the length of the last non-extruding movement (ie, the cost function d between the toolpath currently position on and the last toolpath). This function may seem overly complicated. However, it's purpose becomes clear after considering its derivation. Consider three states: S_0, $S_{1_{short}}$, and $S_{1_{long}}$, and suppose that $S_{1_{short}}$ and $S_{1_{long}}$ were generated by taking an action from state S_0. Furthermore, assume that $d_{short} < d_{long}$ such that the distance function from the second to last to the last toolpath is smaller from S_0 to $S_{1_{short}}$ than it is from S_0 to $S_{1_{long}}$. In the case of hill climb search, these three states would be compared, and the highest valued state would be selected.

First, consider the states $S_{1_{short}}$ and $S_{1_{long}}$, where:

$$d_{short} < d_{long}$$

$$-d_{long} < -d_{short}$$
$$mws\sqrt{2} - d_{long} < mws\sqrt{2} - d_{short}$$
$$V(S_{1_{long}}) < V(S_{1_{short}})$$

Therefore, states that result in shorter travel distances will always be chosen, assuming they have the same m value. But what if the m values are not the same? When S_0 and, say, $S_{1_{long}}$ are compared in hill climb search, $S_{1_{long}}$ should always be selected (if S_0 was selected, the algorithm would stop). In other words, states that are generated by the initial state's actions should always be selected: states with higher m values should be selected over states with lower m values, regardless of the d value of either. The value function V, defined above, ensures this. Consider S_0 and S_1 (either long or short, it's inconsequential):

$$V(S_1) > V(S_0)$$

$$m_1 ws\sqrt{2} - d_1 > m_0 ws\sqrt{2} - d_0$$
$$(m_0 + 1)ws\sqrt{2} - d_1 > m_0 ws\sqrt{2} - d_0$$
$$m_0 ws\sqrt{2} + ws\sqrt{2} - m_0 ws\sqrt{2} > d_1 - d_0$$
$$ws\sqrt{2} > d_1 - d_0$$

Where the greatest possible value of $d_1 - d_0$ is $ws\sqrt{2}$, so $ws\sqrt{2} > d_1 - d_0$, and S_1 will always be chosen over S_0. Overall, states will be selected by the hill climbing search algorithm such that states with more traversed toolpaths are chosen (so as to eventually reach a solution), and states with shorter in-between toolpath distances (d values) will be chosen.

In effect, hill climb search with the value function V, defined above, results in starting at a toolpath endpoint, selecting the nearest printable toolpath endpoint, traversing the toolpath, and repeating until all toolpaths have been traversed. In this algorithm, a solution sits at depth n in the search tree by definition. Because hill climb search is a locally optimal algorithm, only one node will be selected from each solution level. Therefore, the time complexity of this implementation of hill climb search is $O(n)$, where n is the number of toolpaths in the grid.

5 Methods

All code for the experiments in this paper were written in Python. Search algorithms were implemented using the *aima-python* GitHub repository, the companion code for the textbook: *Artificial Intelligence: A Modern Approach.* Custom code was written for modeling the specifics of the toolpath search problem outlined in this paper - this code inherited framework code from *aima-python*, which allowed several testing procedures within the *aima-python* framework to be utilized for evaluation.

In particular, the *aima-python* codebase enables the following information to be gathered after a search algorithm has been run: the number of successors reached (the number of nodes visited), the number of goal states checked (which is irrelevant for local searches), and the number of states generated. These metrics allow for the general performance (particularly the time and space complexity) of an algorithm to be analyzed after it has been executed.

In this paper, locally optimal hill climb search was evaluated on two toolpath sets. Both toolpath sets were generated using the semi-random walk method described in the *Toolpath Generation Approach* section of this paper. For both, the side lengh of each tile was 1.0 units, and the orthogonal distance between toolpaths was 0.1 units. The random angle of each tile was bounded between $0°$ and $-45°$, as is also described in *Toolpath Generation Approach* section of this paper. Figure 3 also depicts the exact angle this metric refers to (the angle with the $+x$ axis). The only way in which the two sample toolpath sets differed was in their size: one was a 2 by 2 tiled grid, and one was a 4 by 4 tiled grid. For both sets, the number of successors visited, the number of states created, and total $J(C)$ value (sum of all non-extruding distances, discussed in the *Toolpath Generation Approach* section) were recorded.

All visualization was done using the Python library Matplotlib.

6 Results

The quantitative results of the experiment are summarized in Table 1. Furthermore, visualizations of both outputs are shown in Figures 4 and 5 - in each, red lines represent extruding movements, and blue lines represent non-extruding movements. As has been previously stated in the *Introduction*, the first toolpath is the toolpath in the bottom-leftmost corner of the grid.

Table 1: Results of hill climbing search executed on one 2x2 and one 4x4 grid of toolpath tiles

	Nodes Visited	States Created	$J(C)$
2x2 Grid	52	224	8.340
4x4 Grid	200	1206	33.644

Figure 4: 2 by 2 toolpath sequence after executing hill climbing search

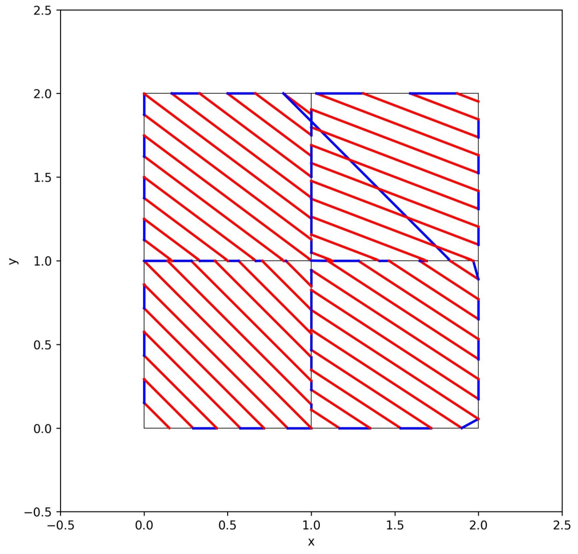
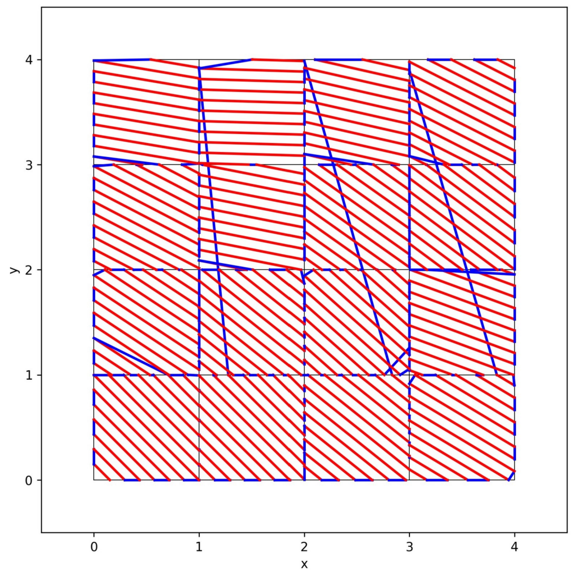


Figure 5: 4 by 4 toolpath sequence after executing hill climbing search



As expected, the algorithm ran in exactly $O(n)$ for both toolpath sequences - the 2 by 2 toolpath sequence visited 52 successor nodes (Table 1), and the 4 by 4 grid visited 200 successor nodes (Table 1). The sequences contained 52 and 200 toolpaths, respectively.

7 Conclusions

Overall, greedy search (hill climbing search) was a highly effective method for generating toolpath sequences from toolpath sets, while adhering to printability. In analyzing Figures 4 and 5, it is clear that much of the $J(C)$ function for each toolpath set C can be attributed to a few large non-extruding lines - in the case of the 2 by 2 grid, this was actually just one line. Because of the similar toolpath orientation angle in adjacent tiles (resulting from the random walk process used to generate the toolpath orientations) and the requirement to adhere to printability, the toolpath sequencing was a rather intuitive process, and the results of the experiment look similar to what a person might come up with manually. The lines in adjacent tiles are almost always close to parallel, so the optimal strategy is to "zig-zag" between adjacent lines, which is the process that the agent employed in this paper mainly used. Overall, for toolpath generation of nearly-parallel toolpath constituent lines, locally optimal search algorithms like hill climbing search are very effective in generating time-minimizing toolpath sequences.

References

[1] Daniel Ahlers, Florens Wasserfall, Norman Hendrich, and Jianwei Zhang. 3d printing of nonplanar layers for smooth surface generation. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1737–1743, 2019.

[2] Yu an Jin, Yong He, Jian zhong Fu, Wen feng Gan, and Zhi wei Lin. Optimization of tool-path generation for material extrusion-based additive manufacturing technology. *Additive Manufacturing*, 1-4:32–47, 2014. Inaugural Issue.

[3] Gregory Dreifus, Kyle Goodrick, Scott Giles, Milan Patel, Reed Matthew Foster, Cody Williams, John Lindahl, Brian Post, Alex Roschli, Lonnie Love, and Vlastimil Kunc. Path optimization along lattices in additive manufacturing using the chinese postman problem. *3D Printing and Additive Manufacturing*, 4, 2017.

[4] Chloë Fleming, Stephanie Walker, Callie Branyan, Austin Nicolai, Geoffrey Hollinger, and Yigit Mengüç. Toolpath planning for continuous extrusion additive manufacturing. In Mario Botsch, Yongjie Jessica Zhang, and Stefanie Hahmann, editors, *Proceeding of SPM 2017 Symposium*. Elsevier, June 2017.

[5] Kai-Yin Fok, Chi-Tsun Cheng, Nuwan Ganganath, Herbert Ho-Ching Iu, and Chi K. Tse. An aco-based tool-path optimizer for 3-d printing applications. *IEEE Transactions on Industrial Informatics*, 15(4):2277–2287, 2019.

[6] Prashant Gupta, Yiran Guo, Narasimha Boddeti, and Bala Krishnamoorthy. Sfcdecomp: Multicriteria optimized tool path planning in 3d printing using space-filling curve based domain decomposition. *International Journal of Computational Geometry & Applications*, 31(04):193–220, 2021.

[7] Samuel Lensgraf and Ramgopal R. Mettu. An improved toolpath generation algorithm for fused filament fabrication. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1181–1187, 2017.

[8] Ismail Enes Yigit and I. Lazoglu. Helical slicing method for material extrusion-based robotic additive manufacturing. *Progress in Additive Manufacturing*, 4(3):225–232, Sep 2019.

[9] Chanyeol Yoo, Samuel Lensgraf, Robert Fitch, Lee M. Clemon, and Ramgopal Mettu. Toward optimal fdm toolpath planning with monte carlo tree search. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4037–4043, 2020.

[10] Gang Zhao, Guocai Ma, Jiangwei Feng, and Wenlei Xiao. Nonplanar slicing and path generation methods for robotic additive manufacturing. *The International Journal of Advanced Manufacturing Technology*, 96(9):3149–3159, Jun 2018.