Toward Optimal FDM Toolpath Planning with Monte Carlo Tree Search

Chanyeol Yoo[1], Samuel Lensgraf[2], Robert Fitch[1], Lee M. Clemon[1], and Ramgopal Mettu[3]

Abstract— The most widely used methods for toolpath planning in fused deposition 3D printing slice the input model into successive 2D layers in order to construct the toolpath. Unfortunately slicing-based methods can incur a substantial amount of wasted motion (i.e., the extruder is moving while not printing), particularly when features of the model are spatially separated. In recent years we have introduced a new paradigm that characterizes the space of feasible toolpaths using a dependency graph on the input model, along with several algorithms to search this space for toolpaths that optimize objective functions such as wasted motion or print time. A natural question that arises is, under what circumstances can we efficiently compute an optimal toolpath? In this paper, we give an algorithm for computing fused deposition modeling (FDM) toolpaths that utilizes Monte Carlo Tree Search (MCTS), a powerful general-purpose method for navigating large search spaces that is guaranteed to converge to the optimal solution. Under reasonable assumptions on printer geometry that allow us to compress the dependency graph, our MCTS-based algorithm converges to find the optimal toolpath. We validate our algorithm on a dataset of 75 models and show it performs on par with our previous best local search-based algorithm in terms of toolpath quality. In prior work we speculated that the performance of local search was near optimal, and we examine in detail the properties of the models and MCTS executions that lead to better or worse results than local search.

I. INTRODUCTION

Making 3D printing more efficient is a pressing need in the area of additive manufacturing [1, 2]. The current sluggish throughput of 3D printers keeps costs above traditional manufacturing lines and prevents adoption in medium and high volume manufacturing. Fabrication speed has long been and remains a major limiting factor for commercial adoption of additive manufacturing processes [1–3]. While adoption of 3D printing in medical and aerospace industries is driven by the low volume and high value products, consumer products and medium to high volume production depend on rapid fabrication to meet demand and keep costs competitive. Thus, tackling the speed of fabrication in 3D printing is a sector-wide challenge. Current deposition planning methods in 3D printing depend on 2D layering and complete each entire layer sequentially, resulting in very long build times and thus high cost per product. A major contributor to this additional cost is wasted motion hoping around within each layer. In this work we greatly reduce this wasted motion by reexamining the entire deposition planning order.

In prior work we have introduced a framework for optimizing toolpaths with respect to both wasted motion [4, 5] as well as overall time [6]. We introduced the notion of a dependency graph of a model to be fabricated, which is defined by the geometric properties of the extruder head. Conceptually the dependency graph constrains the order in which model features must be fabricated and thus allows us to define the space of toolpaths which would feasibly construct the input model on the given printer geometry. We also gave simple heuristic algorithms based on greedy and local search to search over the space of toolpaths. These algorithms all demonstrated that significant gains could be made in toolpath efficiency, especially with respect to wasted motion, if we were not constrained to fabricate layer-by-layer. A natural question that arises from these approaches is, what is the *optimal* toolpath that can be computed for a given input model and printer geometry? A straightforward, and discouraging, observation is that planning a single layer of a fused deposition modeling (FDM) toolpath is easily seen to reduce to TSP. Thus the scale of planning an entire toolpath, or even estimating the optimal cost, using approximation algorithms for TSP or methods such as the Held-Karp relaxation to obtain LP lower bounds is intractable. In [5] we explored LP lower bounds to characterize the performance of local search, but even then were only able to demonstrate optimality for a few, very small models.

Our main contribution in this paper is to develop the first Monte Carlo Tree Search (MCTS) approach to toolpath planning. MCTS can be seen as a stochastic branch and bound approach in which a probabilistic tree search is used to balance exploration and exploitation in the search space. MCTS is a flexible, anytime approach to optimization problems, and is guaranteed to converge to an optimal solution. To our knowledge, this is the first algorithm for toolpath planning with any guarantees on global optimality. In recent years MCTS has been shown to be effective for difficult problems in a number of application areas in robotics. Of particular note is the Dec-MCTS [7] framework, which allows MCTS to be solve optimization problems in an asynchronous, coordinated fashion. By applying the Dec-MCTS framework, the algorithm then yields an efficient approach to multi-extruder toolpath planning with guaranteed convergence, which to our knowledge does not yet exist. In order to make our algorithm efficient, we also introduce a novel clustering algorithm on the dependency graph for the input model. Using a dataset of 75 models, we show that our MCTS-based approach achieves a substantial reduction

[1]School of Mechanical and Mechatronic Engineering, University of Technology Sydney, NSW 2006, Australia {Chanyeol.Yoo, Lee.Clemon, Robert.Fitch}@uts.edu.au
[2]Department of Computer Science, Dartmouth College, Hanover, NH 03755, USA Samuel.E.Lensgraf.GR@dartmouth.edu
[3]Department of Computer Science, Tulane University, New Orleans, LA 70118, USA rmettu@tulane.edu

in wasted motion over the layer-by-layer toolpath generated by Sli3r [8]. The performance of MCTS is interestingly quite similar to that of our existing local search toolpath planner, but allows us to more carefully examine the difficulty of toolpath planning for certain types of models. We study the empirical performance of MCTS and attempt to gather insights about the optimality of our toolpaths. In particular, we have strong evidence of convergence for certain models, and we provide criteria for assessing optimality from our observations.

II. RELATED WORK

Previous research on toolpath planning and slicing of 3D printed parts as focused on examining the resolution of the build relative to the CAD model [9, 10]. This area of work led to varying layer thickness to increase fabrication speed, but achieve a specified surface resolution [9–11]. Others have focused on adjusting the orientation of the layering through transitions in multi-axis machines [12], non-planar layers [12–14], reducing support structures [15], or segmenting areas within a layer [16]. These works do not consider the segments within each layer as part of the entire build plan, but focus on each layer individually. Prior work that focuses on path planning is primarily concerned with increasing the speed of the mechanisms or material reaction rates [3, 17], or varying the thickness of each slice [18] or layering non-planar surfaces [13, 14]. In contrast to our prior work [4–6], the former approaches depend on the same sequential and ordered layering as the original 2.5D methods. Thus, the approach in this work and our related prior work represents a fundamental shift away from individual layer-planning towards a generalized concept of build-planning for additive manufacturing.

MCTS is a sequential Monte Carlo approach for efficient planning in large state spaces [19, 20]. It has been used in motion planning for robot systems in a variety of tasks such as active information gathering [21–24]. DEC-MCTS is a decentralised form of this algorithm [7, 25] that has been used for motion planning in multi-robot systems such as a pair of robot manipulators [26].

III. PRELIMINARIES AND PROBLEM FORMULATION

Given an input 3D model for fabrication, as in prior work we assume that the given model has already been decomposed into individual, printable line segments. Let this set of line segments be denoted as $L = \{\ell_1, \cdots, \ell_n\}$. Each line segment ℓ_i is an ordered pair $(\mathbf{x}_{i,0}, \mathbf{x}_{i,1})$, where $\mathbf{x}_{i,\cdot} = [x, y, z]^T \in \mathbb{R}^3$. Given L, we define the associated set of *contours* C as a partition of L into continuous series of line segments. Intuitively a contour is a series of line segments in the input model that could be printed in one continuous extrusion. Note that a contour could be either opened or closed sequence of line segments.

Let the distance to print a line segment ℓ_i be denoted as $p_i \in \mathbf{P}$, and the distance to move the extruder between a point $\mathbf{x}_j$ to $\mathbf{x}_k$ as $w_{\mathbf{x}_j, \mathbf{x}_k}$. We will write the distance required to travel between line segments ℓ_i and ℓ_j as $w_{i,j}$, where $w_{i,j} = c(\mathbf{x}_{i,1}, \mathbf{x}_{j,0})$. That is, the travel distance between two line segments is simply the distance to travel from the end of the first line segment to the beginning of the second line segment. A *toolpath* is simply a numbered ordering of contours $\pi : C \to \mathbb{N}$. The *cost* of a given toolpath is the sum of the distances to print line segments and to travel between line segments.

We note that our definition of a toolpath does not take the printer geometry into consideration; indeed a low cost toolpath may actually not be feasible (i.e., correctly produces the input model) on all printing platforms. In order to model the constraints imposed on the toolpath by the geometry of the printer, in [4] we introduced the notion of a *dependency graph* that is defined by the contours in the input model and printer geometry. We define the dependency graph $D = (C, E)$ such that the contours C are the vertices, and a directed edge $(c_i, c_j) \in E$ if c_i must be printed prior to c_j in any feasible toolpath. For an edge $(c_i, c_j) \in E$, we denote c_i as the *dependee* and c_j as *depender* for the pair. Intuitively, all dependees must be printed before a depender can be printed. In this paper, we will generate the dependency graph by considering the bounding volume of the extruder head, and defining the dependency graph edges (c_i, c_j) by whether a c_i would be unreachable after printing c_j due to the extruder geometry. We note that the dependency graph formulation can be made substantially more general if we wish to take other constraints into consideration.

With these definitions in mind, we can see that a toolpath π that feasibly prints the input model must be a sequence of contours that satisfies: i) π contains all contours, and ii) for any contour c_i and any subsequent contour c_j in π, $(c_j, c_i) \notin D$. In other words, π fabricates the input model without violating the geometric constraints of the extruder. The power of this view of the space of all toolpaths is that we can now define the *optimal toolpath*, which is simply

$$\pi^* = \arg\min_{\pi} p_i + w_{i,j}. \qquad (1)$$

This is a toolpath that minimizes movement of the extruder without printing called *extrusionless travel distance*. Since the sum of printing all line segments P is independent of toolpath, the problem can be re-written more simply as

$$\pi^* = \arg\min_{\pi} w_{i,j}. \qquad (2)$$

IV. MCTS-BASED TOOLPATH PLANNING

In this section we describe our Monte Carlo Tree Search (MCTS) toolpath planning framework. We first describe the MCTS approach, and then our adaptation of it for toolpath planning.
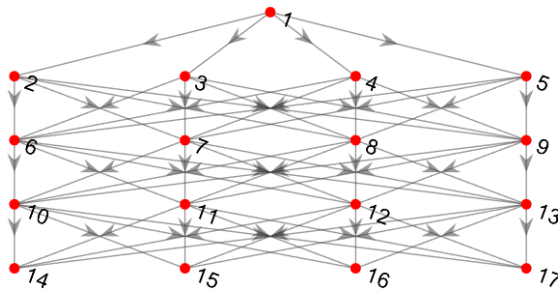
A. Monte Carlo Tree Search

Monte Carlo tree search [19] algorithm is based on biased search algorithm for finding an optimal solution asymptotically. Starting at an initial condition, a tree grows at every iteration. The algorithm finds the next best node in a tree to
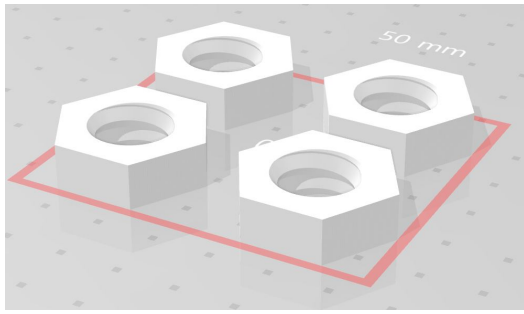
(a) STL model for 'four nuts' model
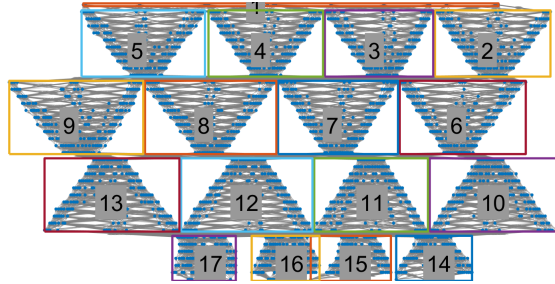
(b) Dependency graph and highly dependent subgraphs

(c) Dependency graph over highly dependent subgraphs in (b)

Fig. 1. Example model of 'four nuts' (a) image, (b) labelled dependency graph, and (c) clustered dependency graph from (b)

(a)

(b)
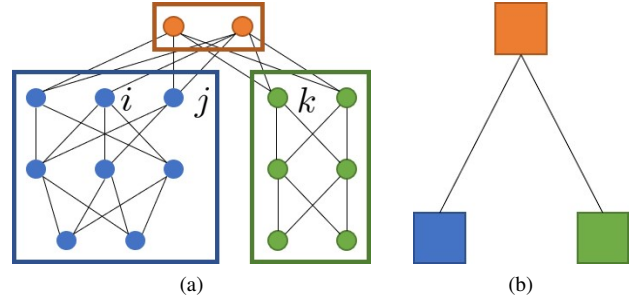
Fig. 2. An example illusting clustering algorithm in Alg. 1. (1) 16 raw contours are clustered into three highly dependent subgraphs (HDS) as shown in (b).

Algorithm 1 Dependency Clustering

Input: Dependency graph D
Output: A set of HDS H

1: $H \leftarrow \emptyset$
2: **for all** dependency depth $d \in \{0, 1, \cdots\}$ **do**
3:      $H \leftarrow \{H, \text{contour cluster}\}$ where $\gamma_{ij} \geq 0.5$
4: **end for**
5: **while** H has changed **do**
6:      **for all** adjacent subgraph pair $h_i, h_j \in H$ **do**
7:          **if** $\gamma_{ij} > \Gamma$ **then**
8:              Merge h_j into $h_i \in H$ and delete h_j from H
9:          **end if**
10:      **end for**
11: **end while**

---

expand using *upper confidence bound* (UCB), where UCB balances between exploitation and exploration. Intuitively, the node with higher likelihood of finding a better solution will be selected. Once a node is selected for expansion, one or a number of complete sequence is randomly generated from the node until reaching the end (e.g., end of time horizon). We call such complete sequence a *rollout*. Each random rollout is evaluated and the *reward* for the rollout is backprogated up to the root node.

Since the algorithm works by randomly generating at least one rollout at every iteration, MCTS is an *anytime* algorithm, where a solution exists whenever the algorithm is stopped. This is because any rollout is a valid solution to the problem. Also, the algorithm will eventually find a global optimal solution as we repeat the process due to its random nature. Unlike existing Monte Carlo methods, MCTS finds such solution faster since it biases its search using UCB.

In this paper, we implement MCTS to find a solution that solves for Problem 2 where we aim to minimise the total extrusionless travel distance of the tooltip. Given a set of contours C and dependency graph D, a valid rollout π is a sequence of contours in a form

$$\pi = \rho_1 \rho_2 \cdots \rho_{|C|} \\ \text{s.t. } R(\rho_i) \models \text{ true } \forall i \in [1, |C|], \tag{3}$$

where $\rho_i \in C$ is the i-th contour to print and $R(\rho_i)$ recursively evaluates whether ρ_i is printable given contours already printed. Before a contour ρ_i can be printed, all its dependees must be printed according to D.

Given a rollout sequence of contours, we compute the reward r in the form of extrusionless travel distance T. Since rewards in MCTS must be a value between 0 and 1, the travel distance is normalised by $r = \frac{\hat{t}-T}{\hat{t}}$, where $\hat{t}$ is the upper bound for travel distance. The upperbound is calculated by multiplying the longest extrusionless travel distance within contours C by the total number of contours. Intuitively, the reward approaches zero as the travel distance approaches the upper bound and it approaches one as the distance gets closer to zero travel distance.

B. Dependency Clustering

Although the MCTS-based method eventually finds an optimal solution, the running time grows rapidly with the number of contours. In this section, we present a clustering algorithm that enables reduction in the number of contour sets to consider, and allows us to speed up the MCTS substantially. At a high level, the clustering algorithm identifies a set of subgraphs of the dependency graph D called *highly*
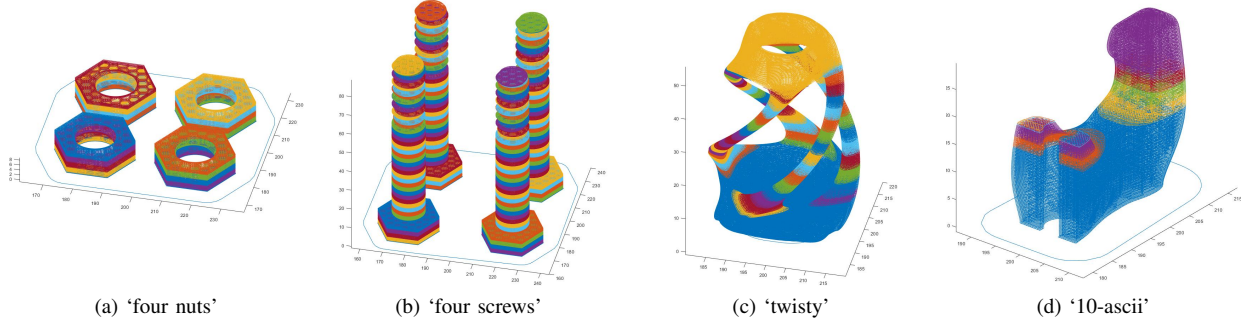
(a) 'four nuts'　　　　(b) 'four screws'　　　　(c) 'twisty'　　　　(d) '10-ascii'

Fig. 3.　Colored clusters for example parts

dependent subgraphs (HDS) H, where $H = \{h_1, h_2, \cdots\}$, $h_i \cap h_j = \emptyset$ and $\bigcup_i h_i = C$.

Suppose we have a pair of contours such that $(c_i, c_j) \notin E$ that has more than one contour c_k upon which they are dependent. Then c_k is a *dependee* and c_i, c_j are *dependers*. The set of common dependees of contours c_i and c_j is denoted as ϕ_{ij} and the sets of all dependees for contours i and j are denoted as ϕ_i and ϕ_j, respectively (i.e., $\phi_{ij} \subset \phi_i$ and $\phi_{ij} \subset \phi_j$). Likewise, ψ_{ij} and ψ_i denote the set of common dependers and set of dependers for contour i.

We can now define the *degree of connectedness* γ_{ij} as

$$\text{mean}\left(\min\left(\frac{|\phi_{ij}|}{|\phi_i|}, \frac{|\phi_{ij}|}{|\phi_j|} \right), \min\left(\frac{|\psi_{ij}|}{|\psi_i|}, \frac{|\psi_{ij}|}{|\psi_j|} \right) \right). \quad (4)$$

To gain intuition for this definition let us consider the circumstances under which $\gamma_{ij} = 1$. This happens when the number of common dependers (and dependees) and single-contour dependers (and dependees) are equal. This means that c_i and c_j share all dependers (and dependees).

Definition 1 (Highly connected pair)**.** A pair of contours i, j is *highly connected* if the degree of connectedness γ_{ij} is greater than or equal to a constant Γ.

Figure 2a illustrates an example of a dependency graph where each node represents a contour and an edge represents dependency where dependee nodes are placed above dependers. Contours i, j and k have at least one common dependees. The degree of connectedness between i and j is 0.593, where as that between i and k is only 0.25. In Sec. V we show that a choice of $\Gamma = 0.5$ works well in practice.

Definition 2 (Highly dependent subgraph (HDS))**.** A subgraph h of dependency graph D is *highly dependent* if all pairs of non-dependently adjacent contours with more than one common dependees or dependers (i.e., $|\phi_{ij}| > 0$ or $|\psi_{ij}| > 0$ for ij-pair) are highly connected.

When $\gamma_{ij} = 1$, a pair is said to be *fully connected*. A subgraph is said to be *fully dependent* when all pairs of non-dependently adjacent contours are fully connected.

The pseudocode for finding a set of HDS is shown in Alg. 1. Initially, we compute the *dependency depth* for all contours in dependency graph D. The dependency depth is the number of edge transitions from root contours. For

example, the dependency depth for contour i in Fig. 2a is 1 and 2 for all dependees. Given all contours at same depth, we enumerate each pair ij and find subgraphs H where the degree of connectivity is greater than Γ. For every adjacent subgraphs $h_i \in H$ and $h_j \in H$, we compute the degree of connectivity. If the degree is greater than Γ, then this pair is merged. We repeat this process until no further merging is possible. We refer to the resulting dependency graph as the *clustered dependency graph*. Figure 2 shows how contours are clustered into three different subgraphs, where Fig. 2b shows the final form of the clustered dependency graph. In Fig. 3, we demonstrate a various models with HDS.

The overall goal of clustering the dependency graph is to create subgraphs in which optimization of the toolpath beyond a layer-by-layer solution is not needed. Each subgraph in the clustered dependency graph is highly connected by construction. Therefore, before a contour at given depth can be printed, most of its dependees must be printed. In the extreme case of a fully dependent subgraph where all contour pairs are fully connected, all dependees must be printed. This property implies that the optimal printing sequence is such that each dependee layer (i.e., lower) must be printed in full before printing the depender layer (i.e., upper).

Such a clustering can be achieved by selecting a strict enough Γ. Once we identify a set of HDS on the original dependency graph is identified, we can then use the clustered dependency graph as input to MCTS to greatly reduce runtime. This approach is justified by the fact that the toolpath cost within an HDS cannot vary significantly and thus the reduced representation will yield a high quality toolpath. In our implementation we keep the threshold $\Gamma = 0.5$ to achieve a balance of efficiency and approximation.

V. EXPERIMENTAL STUDIES

In this section we present experimental validation of the algorithm presented in Sec. IV-A. To more closely examine the performance of MCTS against that of local search, we chose a subset of 75 models from our prior benchmark of 400+ models [5]. We implemented our MCTS algorithm in Matlab and measured the reduction in extrusionless travel relative to our existing local search algorithm and Slic3r as a baseline comparison. As shown in the boxplots in Fig. 7, the median reduction of extrusionless travel for local search
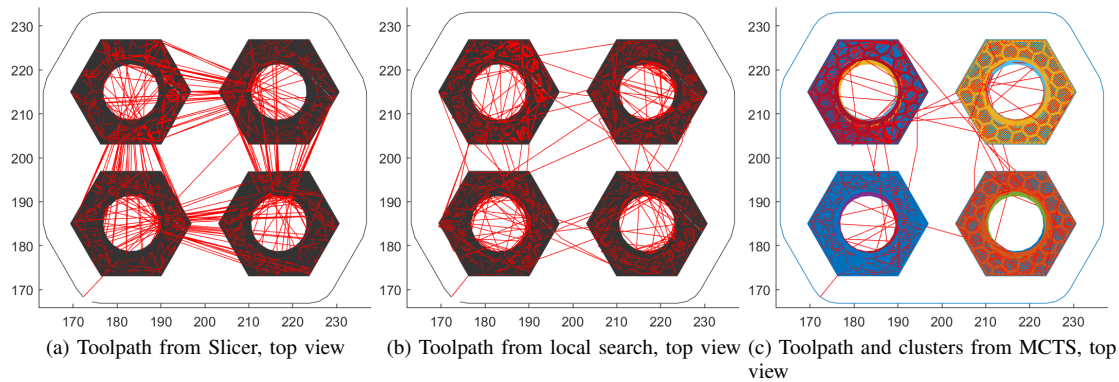
(a) Toolpath from Slicer, top view   (b) Toolpath from local search, top view   (c) Toolpath and clusters from MCTS, top view

Fig. 4. 'Four nuts' model. Toolpath for building the part by (a)(d) typical layerwise planner, (b)(e) local search from [6], and (c)(f) proposed MCTS, with red indicating non-printing motion. Solution toolpath for each method is shown in red. Extrusionless distances (in mm) are 16737, 12220 and 11057, respectively.
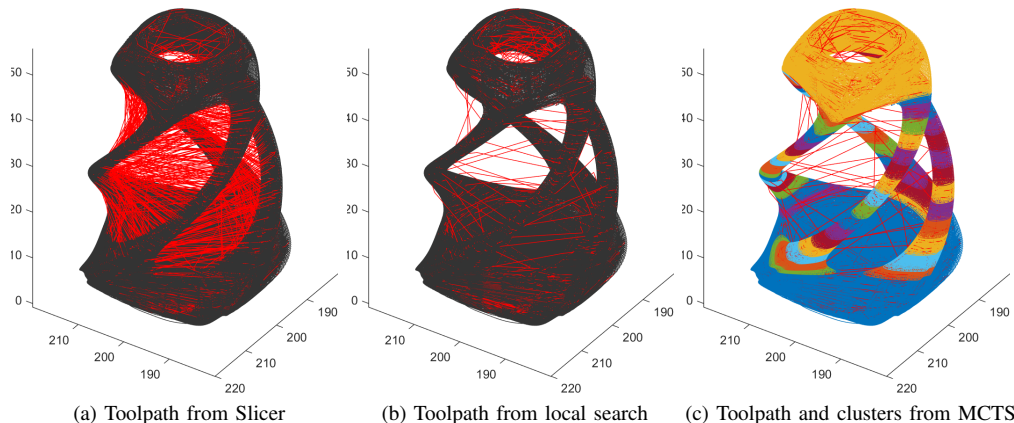

(a) Toolpath from Slicer   (b) Toolpath from local search   (c) Toolpath and clusters from MCTS

Fig. 5. 'Twisty' model. Toolpath for building the part by (a) typical layerwise planner, (b) local search from [6], and (c) proposed MCTS, with red indicating non-printing motion. Solution toolpath for each method is shown in red. Extrusionless distances (in mm) are 25021, 11423 and 11306, respectively.
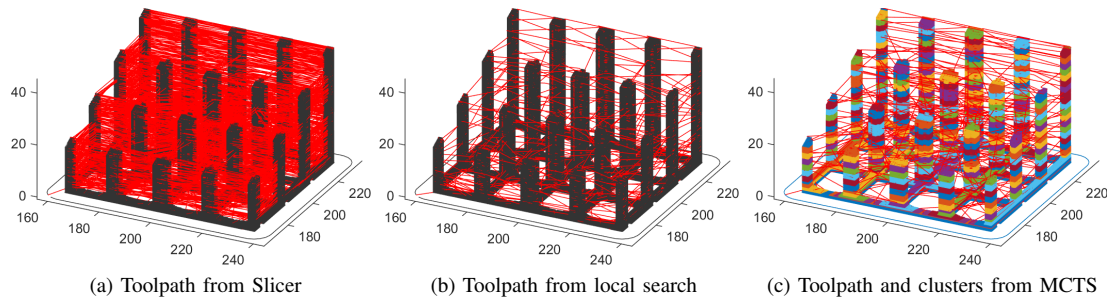

(a) Toolpath from Slicer   (b) Toolpath from local search   (c) Toolpath and clusters from MCTS

Fig. 6. 'Sponge holder' model. Toolpath for building the part by (a) Sli3r, (b) local search from [6], and (c) MCTS. Red indicates extrusionless travel, and multiple colors in (c) indicates HDS components of the dependency graph. Extrusionless distances (in mm) are 84340, 26809 and 33665, respectively.

is about 60% and 55% for MCTS. The running times of our MCTS algorithm were on the order of around 6 minutes on average; MCTS was terminated if the solution did not improve over 5 minutes of search.

While local search is guaranteed only to reach a locally optimal solution, MCTS is guaranteed to converge asymptotically to the global optimum. In our implementation, we terminated the MCTS search when there is no reduction in solution cost after 5 minutes.

As noted above, MCTS performs about as well as local search. Indeed, we noted in prior work [5] that it seemed difficult to eke any more improvement from local search – even with an ideal extruder geometry (i.e., no constraints on reachability), extrusionless travel could only be reduced an additional few percent relative to Slic3r. We were able to also show that at least for a few small models, local search achieved the LP-relaxation lower bound for a linear program set up to find an optimal toolpath in the dependency graph.

We view the MCTS results as an empirical extension of this reasoning. Since MCTS guarantees eventual convergence to optimal, we cannot say with certainty that our solutions are optimal. Indeed, some MCTS toolpaths are slightly poorer quality than those produced by local search. To study performance of MCTS more closely, we partitioned the toolpaths into three categories corresponding to whether they were better (over 10% improved), about the same (within 5%), or worse (over 10% worsen), than the local search toolpaths. To look at concrete examples of these categories, we chose
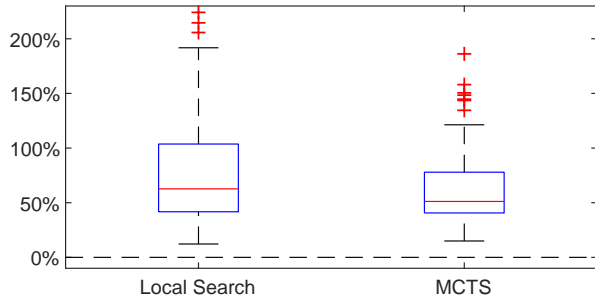
Fig. 7. Reduction in extrusionless travel of local search [5] and MCTS in comparison to Slic3r.
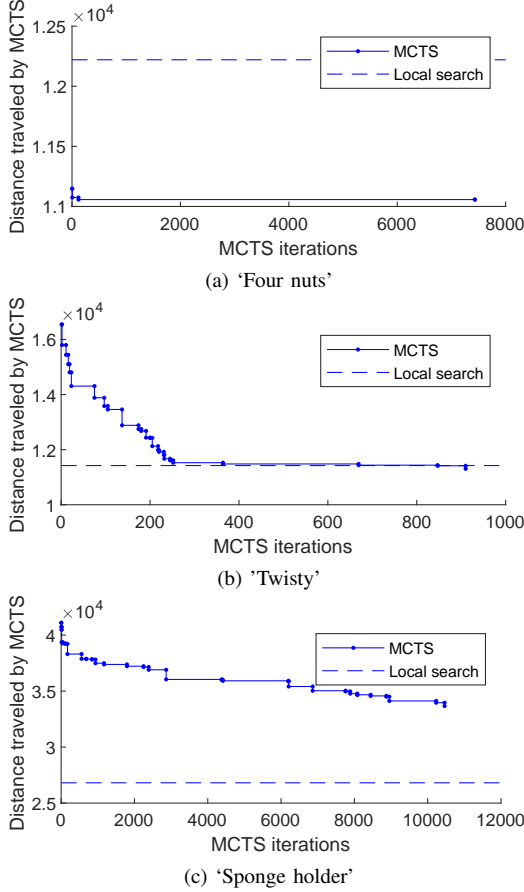

(a) 'Four nuts'

(b) 'Twisty'

(c) 'Sponge holder'

Fig. 8. extrusionless travel of the MCTS solution in each iteration for examplar models. Dashed line shows the cost of the local search solution.

exemplar models for each of these categories. Figures 4, 5 and 6 give toolpaths for the chosen model for each category, respectively, for Sli3r, local search, and MCTS. In Fig. 8, we show MCTS convergence trends for three different models in relation to the local search solution cost.

In 'four nuts' (Fig. 8a), extrusionless travel rapidly reduces in the early iterations and quickly outperforms the local search solution. In 'twisty' (Fig. 5), the MCTS solution starts off worse than the local search and surpasses the local search solution cost after about 8000 iterations. In 'sponge holder' ((Fig. 6), the distance reduced gradually but reached a point of nonimprovement and then timed out after 5 minutes.

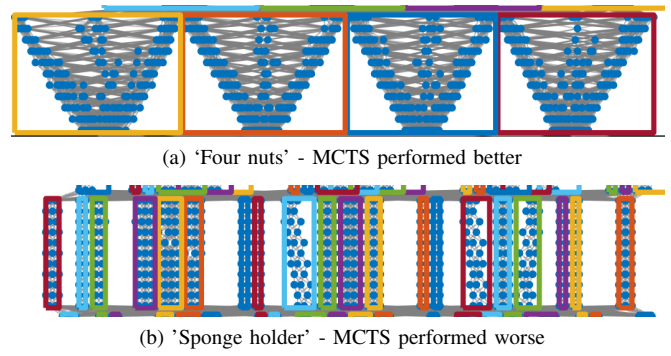To gain insight into the relative behavior of MCTS and local search, in Fig. 9 we show the parts of dependency

(a) 'Four nuts' - MCTS performed better

(b) 'Sponge holder' - MCTS performed worse

Fig. 9. Parts of dependency graphs and clusters for two exemplar models

graph and their HDS components for 'four nuts' and 'sponge holder.' In 'four nuts', the number of contours within a HDS was much higher than in 'sponge holder'. Thus the size of the clustered dependency graph given as input to MCTS is smaller and results in rapid convergence. In contrast, there exist many more HDS components in 'sponge holder.' This implies that the convergence rate for 'sponge holder' is likely much slower, with an early termination resulting in a poor solution. One way to think about the relative performance of these models is in terms of clustering threshold parameter Γ. While we set the value of $\Gamma = 0.5$ to balance solution quality and efficiency, the best value is in fact model dependent and for models such as 'sponge holder' a smaller value would have resulted in a more tractable dependency graph.

The discussion implies that the solution approaches true optimal as the number of clusters increases (i.e., running MCTS over a set of contours, not clusters). However the convergence rate would become poorer. On the other hand, we would find a converging solution fast with less clusters, but the solution could be further from true optimal if we may violate our assumption in Def. 2 by forcing high Γ. Finding the right Γ to balance remains as open question.

VI. DISCUSSION

In this paper we have developed the first provably convergent algorithm for FDM toolpath planning. In order for our Monte Carlo Tree Search approach to be feasible, we also developed a novel clustering approach to compress the dependency graph of the input model. We tested our algorithm over 75 models an observed similar overall performance on average, but empirically characterized the behavior of MCTS when it appears to converge.

A natural question is why one would use MCTS over local search for a given model. Using our empirical studies, it appears that the output of the clustering step and subsequent composition of HDS components of the dependency graph provide guidance as to whether MCTS can achieve convergence. As we saw in our empirical analysis if there enough HDS components with respect to the size of the dependency graph then it is highly likely that MCTS will converge to an optimal toolpath. If the number of HDS components is too large, or the average size is too small, then MCTS will have difficulty exploring the toolpath space and may perform worse than local search.

REFERENCES

[1] Y. Huang, M. C. Leu, J. Mazumder, and A. Donmez, "Additive Manufacturing: Current State, Future Potential, Gaps and Needs, and Recommendations," *Journal of Manufacturing Science and Engineering*, vol. 137, no. 1, p. 014001, 2015.

[2] T. D. Ngo, A. Kashani, G. Imbalzano, K. T. Q. Nguyen, and D. Hui, "Additive manufacturing (3d printing): A review of materials, methods, applications and challenges," *Composites Part B: Engineering*, vol. 143, pp. 172–196, June 2018.

[3] J.-P. Kruth, M. Leu, and T. Nakagawa, "Progress in Additive Manufacturing and Rapid Prototyping," *CIRP Annals*, vol. 47, no. 2, pp. 525–540, 1998.

[4] S. Lensgraf and R. R. Mettu, "Beyond layers: A 3D-aware toolpath algorithm for fused filament fabrication," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3625–3631.

[5] ——, "An improved toolpath generation algorithm for fused filament fabrication," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1181–1187.

[6] ——, "Incorporating kinematic properties into fused deposition toolpath optimization," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 8622–8627.

[7] G. Best, O. Cliff, T. Patten, R. R. Mettu, and R. Fitch, "Dec-MCTS: Decentralized planning for multi-robot active perception," *International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 316–337, 2019.

[8] "Slic3r - Open source 3d printing toolbox."

[9] P. M. Pandey, N. V. Reddy, and S. G. Dhande, "Real time adaptive slicing for fused deposition modelling," *International Journal of Machine Tools and Manufacture*, vol. 43, no. 1, pp. 61–71, Jan. 2003.

[10] A. Dolenc and I. Mkel, "Slicing procedures for layered manufacturing techniques," *Computer-Aided Design*, vol. 26, no. 2, pp. 119–126, Feb. 1994.

[11] P. Kulkarni and D. Dutta, "An accurate slicing procedure for layered manufacturing," *Computer-Aided Design*, vol. 28, no. 9, pp. 683–697, Sept. 1996.

[12] M. A. Isa and I. Lazoglu, "Five-axis additive manufacturing of freeform models through buildup of transition layers," *Journal of Manufacturing Systems*, vol. 50, pp. 69–80, Jan. 2019.

[13] M. K. Micali and D. Dornfeld, "Fully Three-Dimensional Toolpath Generation for Point-Based Additive Manufacturing Systems," in *Solid Freeform Fabrication 2016*, Austin, Texas, USA, 2016, pp. 36–52.

[14] D. Ahlers, "3D printing of nonplanar layers for smooth surface generation," Master's thesis, University of Hamburg, Hamburg, Germany, Oct. 2018.

[15] R. Paul and S. Anand, "Optimization of layered manufacturing process for reducing form errors with minimal support structures," *Journal of Manufacturing Systems*, vol. 36, pp. 231–243, July 2015.

[16] Y.-a. Jin, Y. He, J.-z. Fu, W.-f. Gan, and Z.-w. Lin, "Optimization of tool-path generation for material extrusion-based additive manufacturing technology," *Additive Manufacturing*, vol. 1-4, pp. 32–47, Oct. 2014.

[17] J. Go, S. N. Schiffres, A. G. Stevens, and A. J. Hart, "Rate limits of additive manufacturing by fused filament fabrication and guidelines for high-throughput system design," *Additive Manufacturing*, vol. 16, pp. 1–11, Aug. 2017.

[18] W. Ma and P. He, "An adaptive slicing and selective hatching strategy for layered manufacturing," *Journal of Materials Processing Technology*, vol. 89-90, pp. 191–197, May 1999.

[19] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[20] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. of European Conference on Machine Learning (ECML)*, 2006, pp. 282–293.

[21] T. Patten, W. Martens, and R. Fitch, "Monte Carlo planning for active object classification," *Autonomous Robots*, vol. 42, no. 2, pp. 391–421, 2018.

[22] B. Hefferan, O. M. Cliff, and R. Fitch, "Adversarial patrolling with reactive point processes," in *Proc. of ARAA Australasian Conference on Robotics and Automation (ACRA)*, 2016.

[23] B. Kartal, J. Godoy, I. Karamouzas, and S. J. Guy, "Stochastic tree search with useful cycles for patrolling problems," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1289–1294.

[24] J. Nguyen, N. Lawrance, R. Fitch, and S. Sukkarieh, "Real-time path planning for long-term information gathering with an aerial glider," *Autonomous Robots*, vol. 40, no. 6, pp. 1017–1039, 2015.

[25] G. Best, M. Forrai, R. R. Mettu, and R. Fitch, "Planning-aware communication for decentralised multi-robot coordination," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1050–1057.

[26] F. Sukkar, G. Best, C. Yoo, and R. Fitch, "Multi-robot region-of-interest reconstruction with dec-mcts," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2019.