UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ELIS CASSIANA NAKONETCHNEI DOS SANTOS

HOTFILL: UM ALGORITMO PARA PLANEJAMENTO DE PREENCHIMENTO POR RASTER COM RESTRIÇÃO DE TEMPO DE RESFRIAMENTO PARA IMPRESSÃO 3D POR EXTRUSÃO

CURITIBA

2022

ELIS CASSIANA NAKONETCHNEI DOS SANTOS

HOTFILL: UM ALGORITMO PARA PLANEJAMENTO DE PREENCHIMENTO POR RASTER COM RESTRIÇÃO DE TEMPO DE RESFRIAMENTO PARA IMPRESSÃO 3D POR EXTRUSÃO

HotFill: A Cooling Time Constrained Raster-Fill Planning Algorithm for Extrusion 3D Printing

Dissertação apresentada como requisito para obtenção do título de Mestre em Ciências do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Rodrigo Minetto

Coorientador: Prof. Dr. Neri Volpato

CURITIBA

2022

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Curitiba

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ELIS CASSIANA NAKONETCHNEI DOS SANTOS

HOTFILL: UM ALGORITMO PARA PLANEJAMENTO DE PREENCHIMENTO POR RASTER COM RESTRIÇÃO DE TEMPO DE RESFRIAMENTO PARA IMPRESSÃO 3D POR EXTRUSÃO

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestra Em Ciências da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Computação.

Data de aprovação: 26 de Maio de 2022

Dr. Rodrigo Minetto, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Jorge Vicente Lopes Da Silva, Doutorado - Centro de Tecnologia da Informação Renato Archer

Dr. Ricardo Luders, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 26/05/2022.

Para minha família, amigos e professores, que sempre me apoiaram e encorajaram a seguir em frente.

ACKNOWLEDGEMENTS

Primeiramente, agradeço à minha família e meus amigos, em especial minha mãe, por estarem ao meu lado durante essa etapa tão importante da minha vida, sempre me oferecendo suporte através de palavras cheias de motivação e sempre sendo compreensivos quando precisei estar ausente.

Ao meu orientador, Prof. Dr. Rodrigo Minetto, meu coorientador, Prof. Dr. Neri Volpato, e aos Prof. Dr Jorge Stolfi e Prof. Dr. Ricardo Dutra, que me acompanharam nessa incrível jornada, me apoiando e me ensinando valiosas lições para a vida.

À UTFPR, professores, coordenação, administação e manutanção, que me proveram um ambiente apropriado para continuar aprendendo coisas novas todos os dias.

Gostaria também de agradecer todos que tiveram participado, direta ou indiretamente desse trajeto, pois cada pequeno gesto foi essencial para que esse trabalho fosse finalizado.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Let's think the unthinkable, let's do the undoable. Let us prepare to grapple with the ineffable itself, and see if we may not eff it after all. (ADAMS, Douglas; Dirk Gently's Holistic Detective Agency, 1987).

RESUMO

A flexibilidade oferecida pela manufatura aditiva durante a fabricação de peças complexas, comparada aos métodos tradicionais, a torna uma ferramenta atrativa para diversas aplicações. Porém, as decisões tomadas na etapa de planejamento de processo influenciam nas propriedades e custo da peça final. No caso de tecnologias baseadas na técnica de extrusão de material termoplásticos, a definição da rota a ser seguida durante a construção de uma camada tem impacto direto na qualidade da peça, pois influencia na adesão entre os filamentos adjacentes de uma mesma camada, um dos fatores responsáveis pela resistência mecânica de objetos fabricados por impressoras 3D. Por sua vez, essa adesão depende da temperatura do material previamente depositado, que é diretamente afetada pelo tempo entre a deposição dos filamentos adjacentes. Este trabalho descreve HotFill, um algoritmo que busca um trajeto para preenchimentos tipo zigue-zague que mantenha o tempo de resfriamento entre as deposições de filamentos adjacentes abaixo dos limites especificados pelo usuário. O algoritmo, baseado na técnica de programação dinâmica, foi implementado em Python e testado em várias camadas representativas de modelos 3D. Os resultados mostram que o HotFill é rápido, produz trajetos que respeitam os limites de resfriamento impostos com tempo de fabricação compatível com estas restrições.

Palavras-chave: extrusão com restrição de tempo; planejamento de trajeto; impressão 3d; preenchimento tipo raster; programação dinâmica.

ABSTRACT

The flexibility offered by additive manufacturing when fabricating complex parts, compared to traditional methods, makes it an attractive tool for many applications. However, the decisions made in the process planning stage influence the properties and cost of the final part. In the case of additive manufacturing technologies based on thermoplastic extrusion, the definition of the route to be followed during the construction of a layer has a direct impact on the quality of the part. The route created directly influences the adhesion between adjacent filaments of the same layer, one of the factors responsible for the mechanical strength of objects manufactured by thermoplastic extrusion 3D printers. This, in turn, depends on the time lapsed between their deposition. In this work we describe HotFill, an algorithm that finds a tool-path for solid raster infill that keeps the cooling time intervals between depositions below user-specified limits. The algorithm, based on the dynamic programming technique, was implemented in Python and tested on several representative slices of 3D models. The results show that HotFill is fast and produces tool-paths that have low fabrication time while respecting the cooling time constraints.

Keywords: time constrained extrusion; tool-path planning; 3d printing; raster filling; dynamic programming.

LIST OF ALGORITHMS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Notations

m	move
$\overleftarrow{m}$	reversal of a move
p_{INI}	initial position of a move or a tool-path
p_{FIN}	final position of a move or a tool-path
P	a tool-path
$\overleftarrow{P}$	reversal of a tool-path
$\#P$	number of moves in a tool-path
$\langle\rangle$	a empty tool-path
Λ	a invalid tool-path
σ	thickness of the slice
λ	average cross-section area divided by σ
λ_{FILL}	trace width
T_{INI}	time when the nozzle will start the move
T_{FIN}	time when the nozzle will finish the move
T_{FAB}	manufacturing or fabrication time
T_{COV}	cover time
T_{COOL}	cooling time
$T_{\mathsf{COOL}}^{\mathsf{MAX}}$	maximum cooling time
$T_{\mathsf{COOL}}^{\mathsf{PRED}}$	precompute the value
T_{RAST}	total extrusion time
R_{COOL}	cooling time ratio
$R_{\mathsf{COOL}}^{\mathsf{MAX}}$	relative cooling time
τ	cooling time limit
Δ	cooling time limits
sides	pair of traces that surround the contact
$\mathcal{C}$	set of contacts
$\mathrm{contacts}$	elements of $\mathcal{C}$ that have at least one side that is a trace P or its reversal
r	raster line element
$\mathcal{R}$	set of raster elements
$\mathcal{L}$	set of raster link
μ	lines limit in a bandpath
$\mathbb{P}$	potential tool-paths
ℓ	cut-lines
F	fullpath

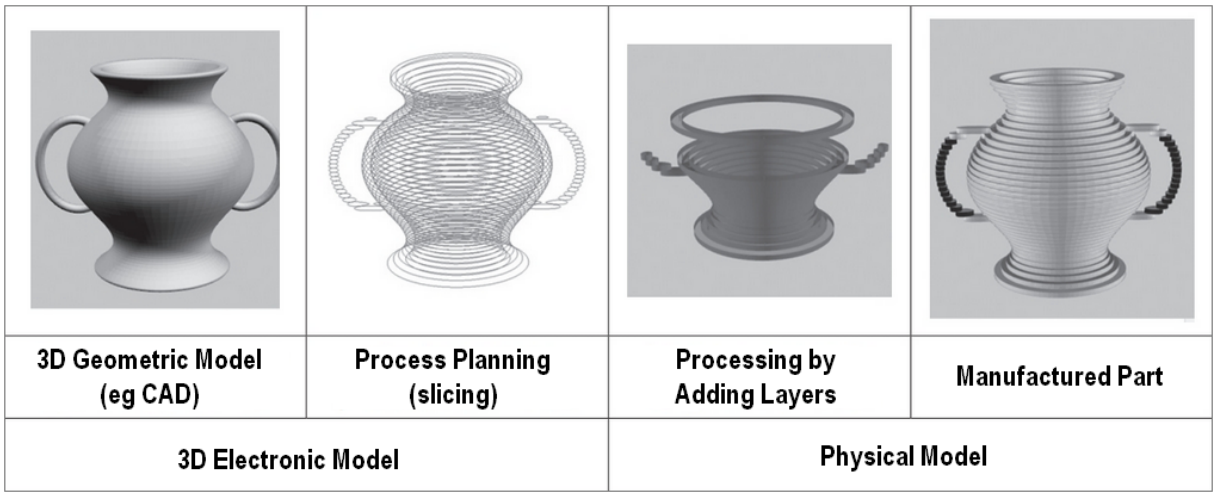
B	bandpath
T_{CONN}	connection time
$T_{\text{CONN}}^{\text{HF}}$	HotFill connection time
$T_{\text{CONN}}^{\text{REF}}$	reference connection time
$T_{\text{FAB}}^{\text{HF}}$	HotFill fabrication time
$T_{\text{FAB}}^{\text{REF}}$	reference fabrication time
$R_{\text{CONN}}^{\text{HF}}$	connection time ratio

CONTENTS

1 INTRODUCTION

Additive manufacturing, also known as 3D printing, is a process that allows transforming three-dimensional computer models into physical objects by laying down successive layers of materials, as illustrated in Figure 1. 3D printers are similar to traditional laser or inkjet printers (BERMAN, 2012), however, instead of create just one layer, a 3D printer creates successive layers that will eventually make up the physical object.

Figure 1 – Additive Manufacturing Process



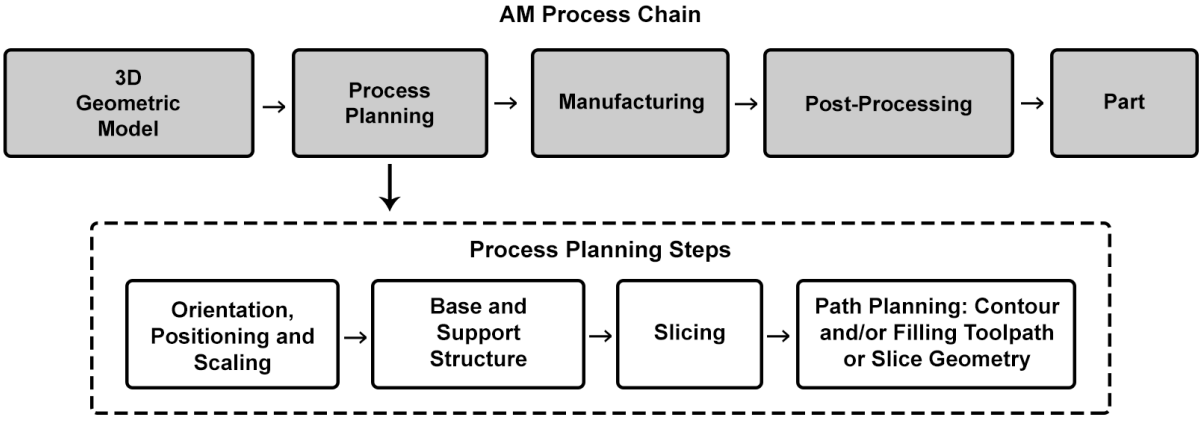
Source: (VOLPATO, 2017).

The variety of adding principles and technologies, and its flexibility, make it possible to manufacture parts in a relatively economical way, in addition to allowing the construction of complex models that could hardly be obtained with other forms of manufacture (BIKAS; STAVROPOULOS; CHRYSSOLOURIS, 2016; NGO *et al.*, 2018). This characteristic has led additive manufacturing to be adopted in different industries (GUO; LEU, 2013), such as biomedicine (SINGH; RAMAKRISHNA, 2017), automotive (LEAL *et al.*, 2017), aerospace (Schiller, 2015) and construction (LIM *et al.*, 2012).

After modeling the desired object and before sending to 3D printer, the geometric model must undergo to the *process planning*, which means a sequence of tasks that includes: orienting and positioning the object in the printer's workspace, adding support structures (if required), cutting the geometric model into layers, and finally planning the printer's tool-path (MINETTO *et al.*, 2017), as shown in Figure 2. In the end of the path planning step, it is generated the final file containing machine-specific instructions, which can be written in a proprietary, or in a more generic, format, such as the G-code.

Material extrusion additive manufacturing technologies based on thermoplastic polymer, the most common used in 3D printers (KULKARNI; MARSAN; DUTTA, 2000; GIBSON; ROSEN; STUCKER, 2015; MINETTO *et al.*, 2017), builds the target object one layer (*slice*) at a time,

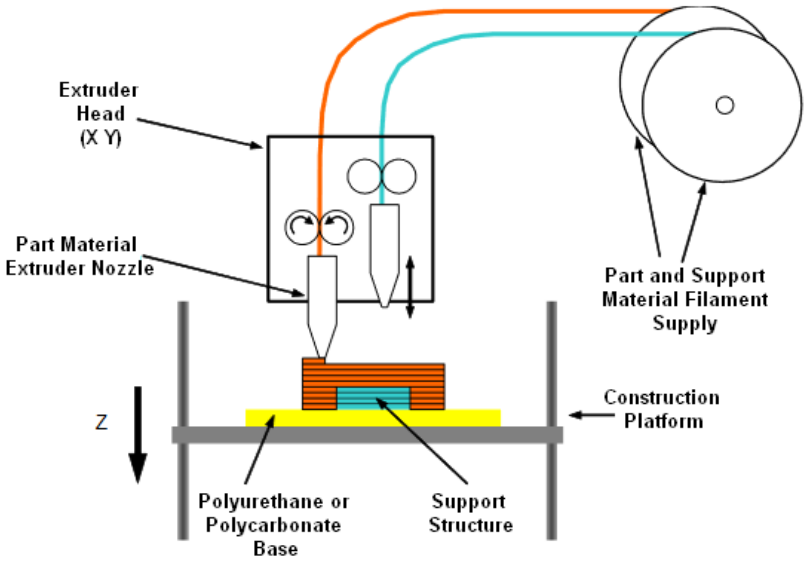
Figure 2 – Additive Manufacturing Process



Source: (VOLPATO, 2017).

using a heated nozzle to deposit a thin filament of molten material following a suitable *tool-path*, as illustrated in Figure 3.

Figure 3 – Thermoplastic Polymer Material Extrusion Process Working Principle



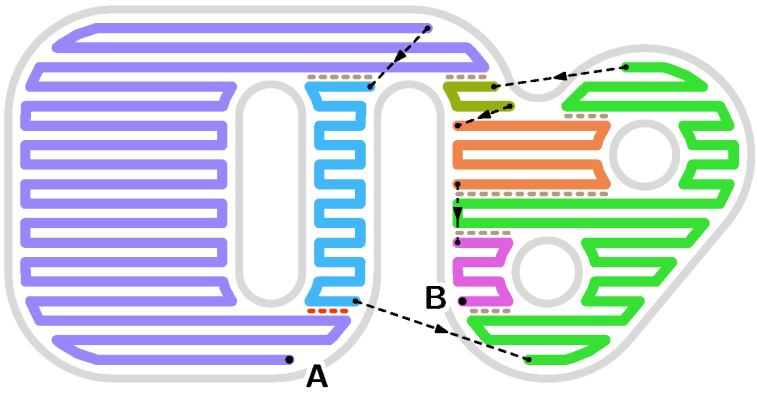
Source: (VOLPATO *et al.*, 2007).

Typically, the tool-path consists of a *contour*, a set of poly-lines following the perimeter of the slice; a *filling* or *infill* of its interior; and, if necessary, some temporary *support* structures to hold up overhanging parts during fabrication. The filling may use many different strategies (Hodgson; Ranellucci; Moe, 2021), such as concentric poly-lines, honeycomb patterns, or Hilbert curves. A common strategy for solid infill (100% density, air gap 0) uses a set of parallel closely spaced *raster lines* (GIBSON; ROSEN; STUCKER, 2015; Volpato *et al.*, 2019), also known as zig-zag and rectilinear.

Figure 4 shows a possible tool-path for the solid filling of a slice of a hypothetical part, as might be produced by process planning programs like Slic3r (Hodgson; Ranellucci; Moe, 2021)

or RP3 (Volpato, 2021). Depending on the shape of the region to be filled, the tool-path may have to be split into two or more *continuous extrusion sections* (CES), shown by different colors in Figure 4; in which case the nozzle will have to jump between these sections without extruding any material (AGARWALA *et al.*, 1996).

Figure 4 – A possible tool-path A–B for the solid filling of a slice of a hypothetical part, whose extruded contour is shown in gray. Each color identifies a separate continuous extrusion section of the path. The black dashed lines denote nozzle movements without extrusion. The part measures $39mm \times 20mm$, and the raster lines are $1mm$ apart and $1mm$ wide



Source: (NAKONETCHNEI *et al.*, 2022).

Algorithms for tool-path generation typically try to minimize the total fabrication time for each slice (Jin *et al.*, 2014; Volpato *et al.*, 2019; WAH *et al.*, 2002; YANG, 2009), which consists of the *extrusion time* plus the non-productive *air time* spent during the jumps (Volpato *et al.*, 2019; LIU *et al.*, 2020). Minimizing the number of jumps may be desirable also to improve the quality of the part's surface, by reducing the occurrence of "strings" (spurious thin filaments of molten material that leak out of the nozzle during jumps). Energy efficiency in computer numerical control machines is another issue that is affected by idle movements (LI; LI; HE, 2018). There is still no algorithm that will find the absolutely best tool-path in a practical amount of computer time; therefore, path-planning programs generally use *heuristics*: algorithms that produce "good enough" tool-paths, but not necessarily optimal ones.

There are several *specialized heuristics* that try to exploit the specific features of the tool-path optimization problem. For example, in (GANGANATH *et al.*, 2016) and (Fok *et al.*, 2016) observed that it is similar to the much-studied traveling salesman problem (TSP), and adapted some heuristics previously developed for the latter to the former. The main heuristics they compared were: nearest neighbor selection (NN), Christofides's algorithm, and the k-opt algorithm (CROES, 1958; FLOOD, 1956; LIN; KERNIGHAN, 1973; ROSENKRANTZ; STEARNS; LEWIS II, 1977). In (Volpato *et al.*, 2019) compared the NN heuristic to the nearest insertion heuristic (NI).

There is an enormous literature on *generic heuristics*, general-purpose methods for approximate optimization of arbitrary goal functions. These include branch-and-bound (MORRISON *et al.*, 2016), simulated annealing (MIAO; TIAN, 2013), tabu search (LI; ALIDAEE, 2016),

randomized greedy search (RESENDE; RIBEIRO, 2010), and many more (PEARL, 1984; ROTHLAUF, 2011). These methods basically probe the search space (the set of all potential solutions) with different strategies, hoping to exploit some weak continuity or non-randomness that is usually present in such functions, or some extra information such as quickly-computable domain bounds. There is even a vast array of *meta-heuristics*, which are strategies to search for effective combinations of problem-specific heuristics treated as black boxes. These include genetic algorithms (GA), which was applied to the tool-path optimization problem by (YANG, 2009), ant colony optimization (ACO), considered by (Fok *et al.*, 2019), and many more (CHEN *et al.*, 2020; LI *et al.*, 2021; LIANG; HE; ZENG, 2020; LIANG *et al.*, 2022). Unfortunately, the large size of the search space and the irregular nature of the goal function, which has many widely separated local minima, conspire to make generic heuristics and meta-heuristics too slow for practical use in this application.

However, fabrication time is not the only criterion that matters when planning the tool-path. Several authors have studied how the orientations of extruded lines influence the tensile and flexural strength of an infill layer. For instance, it was observed that fractal-like fillings, with extruded lines in multiple orientations, generally improve these mechanical properties (Yang *et al.*, 2003; ZHAO; GUO, 2020); and that the orientation of filling rasters affects the quality of raster corners (Jin *et al.*, 2014; KOCH; Van Hulle; RUDOLPH, 2017; XIA; LIN; MA, 2020).

The order of extrusion also affects the strength of the infill because the bonding between two adjacent raster lines is stronger if the second one is deposited while the first is still hot (AGARWALA *et al.*, 1996; SUN *et al.*, 2008; GURRALA; REGALLA, 2014; FAES; FERRARIS; MOENS, 2016; VOLPATO; ZANOTTO, 2019). This effect was noted by several authors, starting with (SUN *et al.*, 2008; COSTA; DUARTE; COVAS, 2017; AKHOUNDI; BEHRAVESH, 2018; FERRARIS; ZHANG; Van Hooreweder, 2019). Studies of this effect using numerical models and experimental methods were published by (AKHOUNDI; BEHRAVESH, 2018) and by (VOLPATO; ZANOTTO, 2019), respectively.

In the example of Figure 4, the red and brown horizontal dashed lines indicate contacts between adjacent raster lines with significant cooling times. The contact highlighted in red has the largest time, 7.26 seconds. For context, the predicted fabrication time for the solid fill alone is $13.51\,\mathrm{s}$ including $12.59\,\mathrm{s}$ of extrusion and $0.91\,\mathrm{s}$ of air time. (The times in this and following examples assume the printer parameters specified in Chapter 4.) Even if one followed the common practice of rotating the direction of the filling rasters by 90 degrees between successive layers (GIBSON; ROSEN; STUCKER, 2015), the part may be much weaker at those bonds than elsewhere.

Adhesion between adjacent filaments can be generally improved by fabricating the part inside a heated chamber. However this facility is not available in many desktop printers. Anyway, the chamber temperature that would be needed to ensure really good adhesion, even between beads deposited a minute or more apart, is likely to be so high that the deposited filament will not properly solidify and the part may deform under its own weight (Lepoivre *et al.*, 2020).

In (VOLPATO; ZANOTTO, 2019), the authors performed experiments to verify the impact of the time interval between the deposition of neighboring raster lines. In one of their tests, all 90° angled layers had a pause of 17 or 65 seconds in the middle of their build, while 0° angled layers were built normally. When performing some tension tests, it was possible to observe that all the specimens broke in the middle, where the programmed pause occurred, as shown in the image on the right. At the end, the authors suggested that a tool-path optimization algorithm should take into account specified limits on the elapsed time between the deposition of adjacent filaments. We call this the *raster-filling hot tool-path problem* (RF-HTPP).

To the best of our knowledge, there is no published algorithm that attempts to solve the RF-HTPP. The problem is similar to the traveling salesman problem with time windows (TSP-TW), in which each node must be visited in a specific time interval (DUMAS *et al.*, 1995). The TSP-TW problem was proved to be NP-complete by (SAVELSBERGH, 1985); a property that is believed to imply that there is no practical algorithm to solve it exactly. Approximate solutions for the TSP-TW were described by (Cheng; Mao, 2007), using an ant colony meta-heuristic; by (LóPEZ-IBáñEZ; BLUM, 2010), using a combination of ant colony and beam search; and by (MLADENOVIĆ; TODOSIJEVIĆ; UROŠEVIĆ, 2013), using a two-stage variable neighborhood search. However, none of these TSP-TW solutions were applied to the RF-HTPP. More importantly, the constraints of the latter are different from those of standard TSP-TW, since we wish to impose a maximum *difference* between the times of arrival at certain node pairs, rather than a specified time interval for reaching each node.

In this context, this work has the goal to answer the following research question (work hypothesis): is it possible to develop an algorithm, of practical use, for the hot tool-path problem for solid raster filling?

To answer this question, we define the following objectives: define an algorithm that finds a solution to the RF-HTPP problem within an acceptable time; implement the algorithm using Python; perform tests and compare their results to solutions found without applying an optimization.

1.1 Contributions

In this work, as our main contribution, we describe HotFill, an algorithm that uses dynamic programming to tackle the hot tool-path problem for the common case of a solid raster filling (RF-HTPP) . As another contribution, we formalize the general RF-HTPP problem, including a precise definition of the tool-path model, of the cooling time constraints, the nominal cooling time and fabrication time formulas, and the expected input and output of the problem.

We observe that solving the RF-HTPP by brute-force enumeration of tool-paths is impractical, except for very small instances. The HotFill algorithm gets around this problem by limiting the search to a proper subset of possible tool-paths, that is expected to include valid tool-paths with low enough fabrication times. We show that HotFill is a generalization of two straightforward

scan-line heuristics to solve the RF-HTPP problem. The algorithm was implemented in Python and is available as open source (Nakonetchnei, 2021).

Through computer simulations, we also compare the tool-paths produced by HotFill with those created by path planning programs like Slic3r and RP3 and by the scan-line heuristics above. We conclude that HotFill is fast enough for practical use, yet produces tool-paths that have low fabrication time while satisfying the specified cooling time constraints. We note that these constraints often force the tool-path to have many more jumps than the paths that are produced by standard process planning programs, independently of the algorithm used to build it.

1.2 Structure of the Document

The remainder of the document is organized as follows. In Chapter 2 we formally define the hot tool-path problem for raster filling, we also define some basic concepts and the notation used in the rest of this document, as well as, some general considerations about this problem. Chapter 3 describes the HotFill algorithm and discusses some algorithm improvements and programming techniques. In Chapter 4 we report tests of the algorithm on some typical object slices. Our conclusions are in Chapter 5. In the appendix (Chapter A), we detail the assumed printer dynamics model.

2 THE RASTER-FILLING HOT TOOL-PATH PROBLEM

In this chapter, we formalize the raster-filling hot tool-path problem (RF-HTPP). We begin by describing the tool-path model and the basic notation used throughout this work. Then, we define the cooling time constraints, and the input and output of this problem. We conclude by showing some basic solutions and other general considerations about the RF-HTPP.

2.1 Tool-Path Model

Moves: In this work, we assume that the printer builds each layer of the target object by executing a series of *moves*, each move being a straight-line motion of the printer's nozzle. The initial and final position of the nozzle as it executes a move m will be denoted by $p_{\mathsf{INI}}(m)$ and $p_{\mathsf{FIN}}(m)$. The *reversal* of a move m is a move $\overleftarrow{m}$ such that $p_{\mathsf{INI}}(\overleftarrow{m}) = p_{\mathsf{FIN}}(m)$ and $p_{\mathsf{FIN}}(\overleftarrow{m}) = p_{\mathsf{INI}}(m)$. A move and its reversal are said to differ in their *orientation*.

Traces and Jumps: A move may be either a *trace* or a *jump*, depending on whether material is to be extruded or not during the motion.

In the first case, we assume that deposited material has a constant cross-section area through most of the trace's length, except at the very ends. We define the (nominal) *width* $\lambda(m)$ of a trace m as being its average cross-section area divided by the thickness σ of the slice. Then a solid fill of some region can be obtained by depositing a set of parallel raster traces, with the mid-lines spaced λ_{FILL} apart and with same width $\lambda(m) = \lambda_{\mathsf{FILL}}$.

For path-planning purposes we further assume that the projected area occupied by the material deposited during each trace is a *thick segment*, a rectangle of width $\lambda(m)$ with round caps centered at each endpoint of the move. In illustrations, we will reduce the width of this thick segment for clarity. See Figure 5.

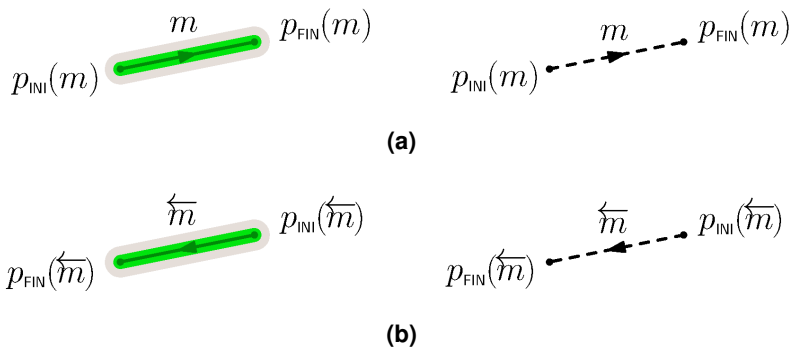
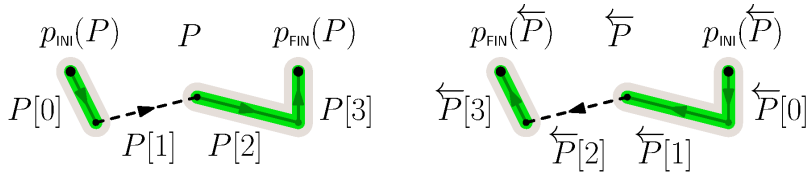


Figure 5 – Graphical representation of a trace (left) and a jump (right), showing the effect of reversal on the attributes p_{INI} and p_{FIN}. The dark thin line in the trace is the nozzle's trajectory. The light gray area is the approximate extent of the material deposited by the nozzle. The green "thick segment" is the conventional representation of the trace used in this work

Source: Own Authorship.

Tool-Paths: For this work, we define a *tool-path*, or *path* for short, as a sequence P of moves that are meant to be executed consecutively, in a specific order. We will denote the moves as $P[k]$ for $k = 0,1,\ldots,n-1$; where n is the number of moves, denoted by $\#P$. Therefore, each move must begin where the previous one ends; that is, $p_{\mathsf{INI}}(P[k]) = p_{\mathsf{FIN}}(P[k-1])$ whenever the two indices are valid. The initial and final points of the path are then $p_{\mathsf{INI}}(P) = p_{\mathsf{INI}}(P[0])$ and $p_{\mathsf{FIN}}(P) = p_{\mathsf{FIN}}(P[n-1])$. See Figure 6 (left).

Figure 6 – A path P with three traces and a jump (left) and its reversal (right).



Source: Own Authorship.

The *reversal* of a path P is the path $\overleftarrow{P}$ with the same moves, reversed and in the reverse order; that is, such that $\overleftarrow{P}[k] = \overleftarrow{P[k']}$ for $k = 0,1,\ldots,n-1$, where $k' = n-1-k$ and $n = \#P = \#(\overleftarrow{P})$. See Figure 6 (right).

Trivial Paths and Zero-Length Moves: It is convenient in the algorithms to allow *trivial paths*, with zero moves. By convention, for such a path $\#P = 0$, and $p_{\mathsf{INI}}(P) = p_{\mathsf{FIN}}(P)$ is some arbitrary point. A move m too may have zero length (that is, its midline may be a single point $p_{\mathsf{INI}}(m) = p_{\mathsf{FIN}}(m)$. However, a zero-length jump in a path makes no sense, and a zero-length trace in a path makes sense only if it is not preceded or followed by another trace, so that execution produces an isolated roundish drop of extruded material.

Manufacturing Time: We assume that there is a function $T_{\mathsf{FAB}}(m)$, the *manufacturing* or *fabrication time*, or *fab-time* for short, that gives the estimated time it will take for the printer to execute a move m. If m is a trace, $T_{\mathsf{FAB}}(m)$ includes the time needed to extrude the material. For a jump, it includes the time needed to displace the nozzle from point $p_{\mathsf{INI}}(m)$ to point $p_{\mathsf{FIN}}(m)$ without extruding; and also the time needed to raise and lower the nozzle, and/or to suck back and re-feed the filament, if required by the printer. In both cases, $T_{\mathsf{FAB}}(m)$ in our model should include the time needed to accelerate and decelerate the carriage, assuming that the nozzle is stationary or has to change direction at each end. The formulas we use for T_{FAB} are detailed in Appendix A.

For simplicity, we assume that the *total fabrication time* $T_{\mathsf{FAB}}(P)$ needed to perform all traces and jumps of a tool-path P is just the sum of the individual move times; that is, $\sum_{k=0}^{n-1} T_{\mathsf{FAB}}(P[k])$ where $n = \#P$.

With this assumption, we ignore the possibility that, on some printers, the carriage may not need to fully decelerate and accelerate between traces that are consecutive in P. This discrepancy should not have a significant effect on the *relative* fabrication times of different tool-paths. Therefore, a path that has optimum T_{FAB} is likely to be close to optimal in practice too.

With this assumption, the *initial time* $T_{\mathsf{INI}}(P,k)$ and the *final time* $T_{\mathsf{FIN}}(P,k)$ when the nozzle will start or finish the move $P[k]$, relative to the starting time of P, are given by Equations (1) and (2).

$$T_{\mathsf{INI}}(P,k) = \sum_{i=0}^{k-1} T_{\mathsf{FAB}}(P[i]) \tag{1}$$

$$T_{\mathsf{FIN}}(P,k) = \sum_{i=0}^{k} T_{\mathsf{FAB}}(P[i]) = T_{\mathsf{INI}}(P,k) + T_{\mathsf{FAB}}(P[k]) \tag{2}$$

It is convenient also to define $T_{\mathsf{INI}}(P,n) = T_{\mathsf{FAB}}(P)$ and $T_{\mathsf{FIN}}(P,-1) = 0$; then $T_{\mathsf{FIN}}(P,k-1) = T_{\mathsf{INI}}(P,k)$ for every move $P[k]$.

Empty and Invalid Path: In some operations it is also useful to have the *empty* path $\langle\rangle$, a special object that, when concatenated with any path P, results in P itself. In this aspect, $\langle\rangle$ differs from a trivial path T, whose concatenation with P would require a jump from $p_{\mathsf{FIN}}(T)$ to $p_{\mathsf{INI}}(P)$ if the points do not coincide.

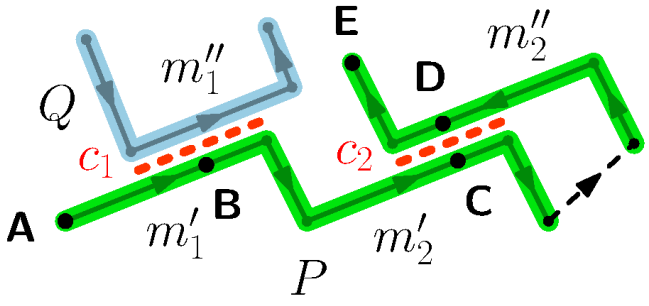
It is also sometimes convenient to have an *invalid* path Λ, a special code that signifies the absence of a path. The concatenation of Λ with any path is undefined, and may be assumed to be Λ.

The endpoint functions p_{INI} and p_{FIN} are not defined for either $\langle\rangle$ or Λ. It is convenient to define $T_{\mathsf{FAB}}(\langle\rangle) = 0$ and $T_{\mathsf{FAB}}(\Lambda) = +\infty$.

2.2 Cooling Time Constraints

Contacts: We define a *contact* as a straight line segment between two adjacent traces that should be welded together. See Figure 7. For example, when completely filling an area with horizontal traces, there will usually be a contact between any two traces whenever the Y coordinates of their mid-lines differ by the common trace width λ_{FILL}, and their X ranges overlap.

Figure 7 – Graphical representation of two contacts c_1 and c_2 (red dashed lines) between traces m'_1, m''_1 and m'_2, m''_2, of two paths, P (green) and Q (blue).



Source: Own Authorship.

We will denote by $\mathrm{sides}(c)$ the pair of traces that surround the contact c. In Figure 7, $\mathrm{sides}(c_1)$ would be the pair of traces $\{m'_1,m''_1\}$. For any path P and any set $\mathcal{C}$ of contacts, we will denote by $\mathrm{contacts}(P,\mathcal{C})$ the elements of $\mathcal{C}$ that have at least one side that is a trace P or its reversal.

Cover Times: We define $T_{\mathsf{COV}}(m,u)$ as a function that gives the time interval between the start of the extrusion of a trace m and the moment when the nozzle is at the point u' on the trace's mid-line that is closest to a given point u of the plane. See Appendix A for how this time can be estimated.

Moreover, if m is one of the side traces of a contact c, and P is a path such that $P[k]$ is m or $\overleftarrow{m}$, we define the *cover time* $T_{\mathsf{COV}}(P,c,m)$ *of* that side *by* P as $T_{\mathsf{INI}}(P,k) + T_{\mathsf{COV}}(P[k],u)$, where u is the midpoint of c. Note that the second term takes into account the orientation of m in P. In Figure 7, $T_{\mathsf{COV}}(P,c_1,m'_1)$ is the fab-time of the path P from **A** to **B**, while $T_{\mathsf{COV}}(\overleftarrow{P},c_1,m'_1)$ is the fab-time of $\overleftarrow{P}$ from **E** to **B**.

Contact Cooling Time: If a path P contains both sides m' and m'' of a contact c, we define the *cooling time* $T_{\mathsf{COOL}}(P,c)$ *of* c *in* P as the absolute difference between the two cover times, namely $|T_{\mathsf{COV}}(P,c,m') - T_{\mathsf{COV}}(P,c,m'')|$. In Figure 7, $T_{\mathsf{COOL}}(P,c_2)$ is the fab-time of P from **C** to **D**. It is an estimate of the interval of time elapsed between the deposition of material on the two sides of the contact; which is assumed to determine the strength of the corresponding physical weld.

Strictly speaking, if two traces are extruded in opposite directions, or if the speed of the nozzle is not uniform while covering the contact, the actual cooling time will vary from point to point along the contact. Thus the value computed for the midpoint is only an estimate of the contact's average cooling time; but it should be good enough for practical purposes.

Cooling Time Limit: We assume that each contact c also has a *cooling time limit* $\tau(c)$, a user-specified maximum allowed value for $T_{\mathsf{COOL}}(P,c)$ in the final tool-path P. See Section 2.3. The *cooling time ratio* $R_{\mathsf{COOL}}(P,c)$ *of* c *in* any path P that closes it is the quotient $T_{\mathsf{COOL}}(P,c)/\,\tau(c)$; it is greater than 1 if and only if that limit would be violated if P were to be chosen as the tool-path. If there is no constraint on the cooling time limit, we may let $\tau(c)$ be $+\infty$ (so that $R_{\mathsf{COOL}}(P,c)$ will always be zero).

The *maximum cooling ratio* $R^{\mathsf{MAX}}_{\mathsf{COOL}}(P,\mathcal{C})$ of a tool-path P and a set $\mathcal{C}$ of contacts is the maximum value of $R_{\mathsf{COOL}}(P,c)$ among all contacts c in $\mathcal{C}$ which have both sides in P.

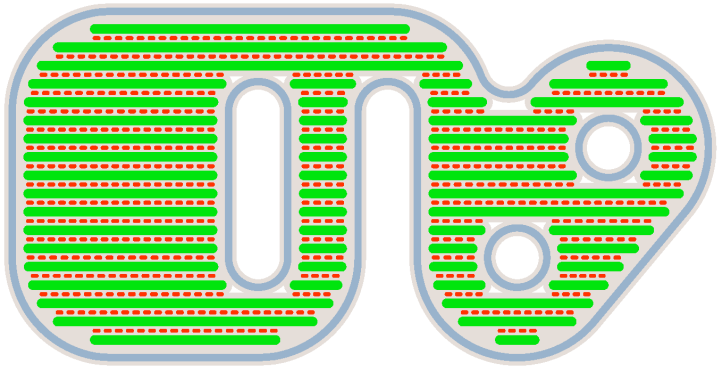
2.3 Statement of the Problem

2.3.1 Input

Formally, the input for the RF-HTPP consists of a set of n *raster elements* $\mathcal{R}$, a set $\mathcal{C}$ of *relevant contacts* between those elements.

Each raster element is a single trace. For simplicity of exposition, we assume that every trace is horizontal and oriented from left to right. The rasters are supposed to lie on a set of s horizontal *scan-lines* with uniform spacing λ_{FILL}, equal to the width λ of all raster traces. For example, in Figure 8 there are 18 scan-lines. A relevant contact is a contact between distinct rasters that has a finite cooling time limit.

Figure 8 – A possible input data set for the RF-HTPP problem, suitable for planning the solid raster filling of the slice shown in Figure 4. There are 56 raster elements (green thick segments) on 18 scan-lines. The total fab-time for those rasters, not counting the time to move between them, is 10.10 seconds. The red lines are the relevant contacts $\mathcal{C}$. The blue-gray lines are the contour of the slice (shown for reference only, not part of the input data).

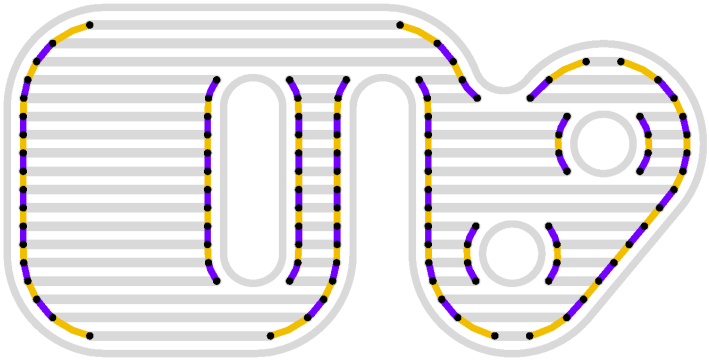


Source: Own Authorship.

Each raster element r in $\mathcal{R}$ also has a set $\mathcal{L}(r)$ of associated *links*. A link is a short path that is to be included in the tool-path, in place of a jump, to connect other rasters (or their reversals) to r. Thus $p_{\text{FIN}}(S) = p_{\text{INI}}(S)$ for every link S in $\mathcal{L}(r)$. There is a separate set $\mathcal{L}(\overleftarrow{r})$ of links that can be used to connect other rasters to $\overleftarrow{r}$. For every such link S, $p_{\text{FIN}}(S) = p_{\text{INI}}(\overleftarrow{r}) = p_{\text{FIN}}(r)$. See Figure 9. Normally $\mathcal{L}(r)$ and $\mathcal{L}(\overleftarrow{r})$ may have 0, 1, or 2 paths each. (The provision of multiple link choices as part of the input data, that can be selected and combined as needed to assemble the tool-path, seems to be an original feature of our approach. It decouples the geometric processing of the outline from the path planning phase, and plays an important role in our algorithm.)

More generally, if P is any path that begins with a raster line r, in the given or reversed orientation, we define $\mathcal{L}(P)$ to be the same as $\mathcal{L}(r)$. Similarly, if P ends with a raster r, we define $\mathcal{L}(\overleftarrow{P})$ to be the same as $\mathcal{L}(\overleftarrow{r})$.

Figure 9 – A possible collection of link paths for the input elements of Figure 8.



Source: Own Authorship.

There must not be two links in the input data with the same pair of endpoints. Typically, each link path will run parallel to the boundary of the slice, between the endpoints of adjacent raster elements.

The contour paths in Figure 8 are shown only to indicate the shape of the slice, but are not part of the input data set; we assume that they will be fabricated before or after the whole raster filling. We also assume that the contacts between the filling and the contour have no cooling time constraints, and therefore are omitted from $\mathcal{C}$.

2.3.2 Output

The desired output of the RF-HTPP problem is a tool-path H that uses every raster of $\mathcal{R}$ exactly once, in any of its two orientations; and is *valid*, meaning that it satisfies the *cooling time constraint* $R^{\text{MAX}}_{\text{COOL}}(H,\mathcal{C}) \leq 1$.

Any two raster elements that are consecutive in H must be connected by a link if available, or by a jump if there is no such link. If the rasters are r' and r'' (in their original or reversed orientations), the link, if it exists, will be the only one that connects $p_{\text{FIN}}(r')$ to $p_{\text{INI}}(r'')$; that is, the only s such that $\overleftarrow{s} \in \mathcal{L}(\overleftarrow{r'})$ and $s \in \mathcal{L}(r'')$.

There may be no valid tool-path for the given input data. In this case, by convention, the output should be the invalid path Λ. This outcome means that it is impossible to fabricate the slice, using the given set of rasters and links, without the cooling time at some contact c exceeding its cooling time limit $\tau(c)$.

Ideally, among all the valid tool-paths, the solution should be the *best* one, meaning the one with smallest fabrication time. However, no efficient algorithm is known that will find this path in any case. Therefore, one can only expect the resulting path to be "good enough" for fabrication. We assume that the time T_{RAST} spent depositing the rasters is the same in any tool-path built from them (see Appendix A); thus minimizing the total fab-time means minimizing the *connection time* $T_{\text{CONN}}(H)$, the time spent extruding links and executing jumps.

Using links instead of jumps will normally reduce the path's fabrication time, and may also improve the adhesion between the filling and the contour by filling some gaps between the two that would not be filled by the raster elements alone.

2.4 Special Solutions

There are three simple heuristics that can be applied to the RF-HTPP problem, which may be sufficient in practice and are important for the presentation and testing of our algorithm. They are the greedy heuristics, and two scan-line-order heuristics.

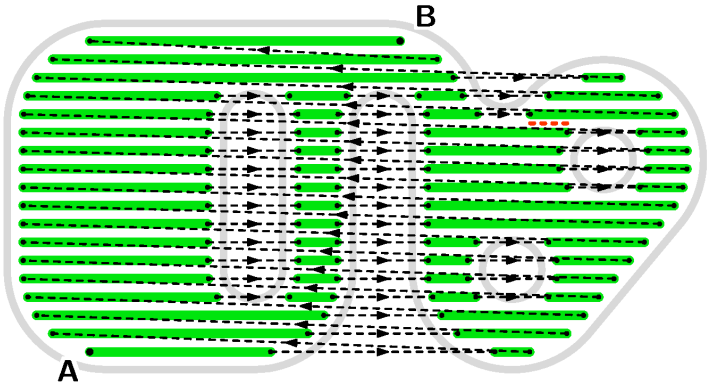
2.4.1 Greedy Solution

A simple way to construct a tool-path H from the given raster elements is to use a *greedy* heuristic. Namely, starting with an empty path H, one repeatedly selects one raster r from $\mathcal{R}$, and appends either r or $\overleftarrow{r}$ to H, filling any gap with a link or jump, as appropriate; where r is chosen so as to minimize the fab-time of that link or jump. If the set $\mathcal{R}$ has only one raster per scan-line, this process may yield a tool-path that has optimal or near-optimal fabrication time. On the other hand, if some scan-lines have two or more raster elements, this greedy tool-path may end up with very large contact cooling times, such as that of Figure 4.

2.4.2 Scan-line Solutions

Another simple possible solution to the RF-HTPP is a *non-alternating scan-line-order* (SCN) tool-path, which fabricates the scan-lines in order of increasing Y coordinate, with the rasters each scan-line extruded in the same order and direction (all left-to-right, or all right-to-left). See Figure 10.

Figure 10 – A non-alternating scan-line-order tool-path for the solid raster fill data of Figures 8 and 9. Its fab-time is $23.71\,\mathrm{s}$, including $10.10\,\mathrm{s}$ of extrusion and $13.61\,\mathrm{s}$ of air time. The red dashed line is the contact with the largest cooling time, $1.67\,\mathrm{s}$.



Source: Own Authorship.

Such a tool-path is likely to be optimal or near-optimal with regards to contact cooling, in the sense of having the minimum possible $R^{\text{MAX}}_{\text{COOL}}$. However, its fabrication time may be much larger than that of the optimum valid path.

Alternating scan-line solution: A slightly less trivial possible solution for the RF-HTPP is an *alternating scan-line-order* (SCA) path, which also processes the scan-lines in order of increasing Y coordinate, but reverses the order of rasters and the direction of extrusion from one scan-line to the next. See Figure 11.

Figure 11 – An alternating scan-line-order tool-path for the solid raster fill data of Figures 8 and 9. Its fab-time is $17.95\,\text{s}$, including $11.02\,\text{s}$ of extrusion, and $6.93\,\text{s}$ of air time. The red dashed line at upper right is the contact with largest cooling time, $2.25\,\text{s}$.

Source: Own Authorship.

An alternating scan-line solution usually has much smaller fab-time than the non-alternating version, because the jumps between scan-lines are replaced by shorter jumps or by links. However, it is more likely to violate cooling constraints, if some cooling time limits are comparable to time required to extrude one scan-line. Note that, from Figure 10 to Figure 11, the maximum contact cooling time increased from 1.67 to $2.25\,\text{s}$.

2.5 General considerations

2.5.1 Non-Connected Infills

If the input raster set $\mathcal{R}$ can be partitioned into two or more subsets $\mathcal{R}_1$, $\mathcal{R}_2$, ..., with no contacts between different subsets, it may be sufficient to run the algorithm separately on each of those subsets, and then concatenate the resulting tool-paths H_1, H_2, ..., with intervening jumps or links, into a single path H. Splitting the problem this way will not affect the existence of a valid path.

The fabrication time of the concatenated path H will depend on the order of the components H_i and on their orientations. (Note that, in our model, if H_i is valid, then $\overleftarrow{H_i}$ is also valid, and

has the same fab-time.) Finding the best order and orientation for the H_i would be a separate problem, that could be addressed by the heuristics mentioned in Chapter 1.

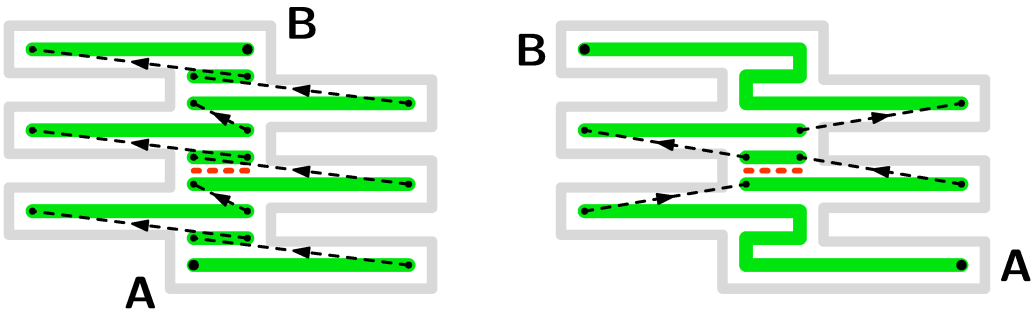
2.5.2 Monotonic Infill Regions

The RF-HTPP is usually trivial if the input set $\mathcal{R}$ has a single raster line on each scan-line. We call such a raster set *monotonic*, because it arises, in particular, if the infill region D is Y-monotonic in the geometric sense — meaning that every horizontal line intersects it in at most one line segment (PREPARATA; SHAMOS, 1985).

Therefore, if the raster set $\mathcal{R}$ is monotonic, it is almost never necessary to use the HotFill algorithm described in Chapter 3. In that case, one can try an SCA (alternating scan-line) tool-path first, which will probably have fab-time close to the minimum, and it will be valid as long as the cooling time limits $\tau(c)$ are not too small (say, not less than twice the fab-time of the longest scan-lines). If the SCA tool-path turns out to exceed some cooling limits, then one can try an SCN (non-alternating) tool-path; which, as observed above, is generally likely to be optimal with respect to cooling. If the SCN path also fails, then it is likely that the problem has no solution with the specified limits.

To be precise, there are instances of the RF-HTPP, even with only one raster per scan-line, for which the SCN tool-path is not optimal in the sense of cooling. That is the case of the instance shown in Figure 12: if the cooling time limits $\tau(c)$ of all contacts were set to $0.41\,\mathrm{s}$, the SCN solution (at left) would not be valid, but the SCA solution (at right) would be.

Figure 12 – A monotonic instance of the RF-HTPP for which the maximum contact cooling time of the SCN solution at left ($0.42\,\mathrm{s}$) is higher than that of the SCA solution at right ($0.40\,\mathrm{s}$).

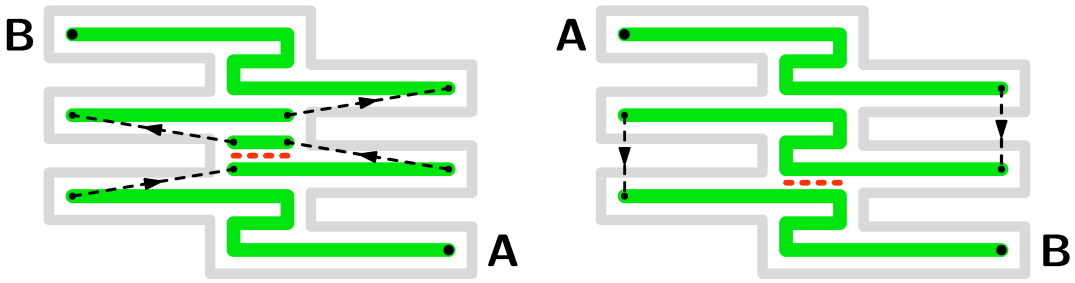


Source: Own Authorship.

There are also monotonic instances of the RF-HTPP such that the valid solution with minimum fab-time is much better than the SCA tool-path, and may even use the rasters out of scan-line order. That is the case of the instance shown in Figure 13.

However, these anomalous situations generally arise only when the infill region D has substantially irregular borders, leading to large offsets between adjacent rasters. In practice, those situations are probably too rare to worry about. Thus, for monotonic datasets, a scan-line tool-path will probably be adequate.

Figure 13 – A monotonic instance of the RF-HTPP for which the fabrication time of the SCA solution at left ($2.38\,\mathrm{s}$) is significantly higher than that of the tool-path at right ($2.03\,\mathrm{s}$), which does not follow the scan-line order. The maximum contact cooling times are $0.40\,\mathrm{s}$ and $0.73\,\mathrm{s}$, respectively.



Source: Own Authorship.

2.5.3 Problem Decomposition at Monotonic Sections

More generally, it is possible to split the RF-HTPP into smaller independent sub-problems if the input raster set contains monotonic sections. Specifically, suppose that there are one or more consecutive scan-lines that have a single raster each. Let r' and r'' be the lowest and highest of those rasters, respectively. Then the problem can be split into three independent problems, with raster sets $\mathcal{R}'$, $\mathcal{R}^*$, and $\mathcal{R}''$; where $\mathcal{R}'$ consists of r' and all rasters below it, $\mathcal{R}''$ is r'' and all the rasters above it, and $\mathcal{R}^*$ is all the rasters between r' and r'', including both. Let P' and P'' be RF-HTPP solutions for $\mathcal{R}'$ and $\mathcal{R}''$, respectively, and P^* be the greedy solution for $\mathcal{R}^*$ starting at r'. If these partial solutions exist, a solution for the original input set $\mathcal{R}$ may be obtained by concatenating P', P^*, and P'', or their reversals, removing the duplicate rasters r' and r''.

This decomposition assumes that the path P' ends with raster r', and that P'' will start with r''. This will always be the case if the paths are computed with the algorithm of Chapter 3 (procedure HotFill). It also assumes that the greedy solution P^* starts with r' and ends with r''; which will be the case if the bi-directional greedy algorithm used in that section (procedure BandPath) is used to build it.

2.5.4 Exhaustive Enumeration

In theory, the RF-HTPP could be solved by "brute force", that is, exhaustive enumeration of all possible tool-paths. However, that solution would not be practical.

Let's define a *potential path* for an instance of the RF-HTPP as being a tool-path that includes exactly one orientation of each raster of $\mathcal{R}$; with any gap filled by the appropriate link if it exists, or by a jump otherwise. We denote by $\mathbb{P}(\mathcal{R})$ the set of all those potential tool-paths.

The RF-HTPP then can be redefined as to find a tool-path H in the set $\mathbb{P}(\mathcal{R})$ that is valid, that is, whose maximum relative cooling time $R^{\mathrm{MAX}}_{\mathrm{COOL}}(H,\mathcal{C})$ does not exceed 1. We will denote the set of valid candidate paths by $\mathbb{V}(\mathcal{R},\mathcal{C})$.

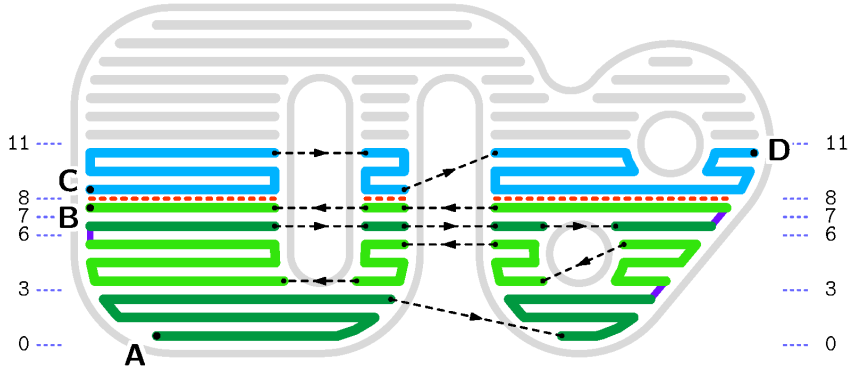
A potential path is completely determined by an ordering and orientation of the raster elements in it. Therefore, $\#\mathbb{P}(\mathcal{R}) = 2^n n!$, where $n = \#\mathcal{R}$ is the number of raster elements. Clearly, the brute-force enumeration of all the paths in $\mathbb{P}(\mathcal{R})$ is practically impossible, except for very small instances (with a couple dozen rasters at most).

3 HOTFILL: FINDING A VALID TOOL-PATH

In this chapter we present the HotFill, an algorithm specialized to find a valid tool-path for solid raster filling. As observed in Section 2.4, the set $\mathbb{P}$ of potential tool-paths is usually too large to be enumerated exhaustively. The HotFill algorithm gets around that problem by limiting the search to a proper subset of $\mathbb{P}$, that is expected to include tool-paths that are valid and have low enough fabrication times, using the cut-line and bands concepts.

We define a *cut-line* as a horizontal line that runs between successive scan-lines. If the rasters of $\mathcal{R}$ lie on s successive scan-lines, there are $s+1$ relevant cut-lines, from ℓ_0 (just below the lowest raster) to ℓ_m (just above the highest one). (See the blue dotted lines in Figure 14.) For any i,j with $0 \le i < j \le s$, we define the (i,j)*-band* $\mathcal{R}_{i,j}$ as the set of rasters between cut-lines ℓ_i and ℓ_j.

Figure 14 – Illustration of the partial state at some point during the execution of the HotFill procedure (Algorithm 1) for the rasters and links of Figure 8 and 9. The colored paths comprise an $(8,11)$-fullpath, specifically $F_0[8,11]$, consisting of a $(7,8)$-fullpath $F_1[7,8]$ = A–B (green and purple) and an $(8,11)$-bandpath $B_0[8,11]$ = C–D (cyan). The fullpath $F_1[7,8]$ consists of bandpaths $B_0[0,3]$, $B_1[3,6]$, $B_0[6,7]$, and $B_1[7,8]$ (light and dark green) and three links between them (purple). The red dashed lines are the contacts between $F_1[7,8]$ and $B_0[8,11]$ that must be checked when concatenating them to obtain an $F_0[8,11]$. The horizontal dotted blue lines are the cut-lines that delimit those bandpaths.



Source: Own Authorship.

The HotFill algorithm builds its solution by concatenating (i,j)*-bandpaths*. These paths will be specified later. For now, it suffices to know that an (i,j)-bandpath includes all the raster elements in the band $\mathcal{R}_{i,j}$ – exactly once each, in either orientation – plus any applicable links; and satisfies the cooling constraints of all contacts that are internal to the band.

The result of HotFill is built incrementally by concatenating one or more of those bandpaths, starting with some $(0,j)$-bandpath; where the top of each bandpath is the same cut-line as the bottom of the next bandpath. We define an (i,j)*-fullpath* as being such a concatenation that ends with a (i,j)-bandpath. See Figure 14. A fullpath is valid if and only if all its bandpaths are valid, and it satisfies the cooling time constraints of all contacts between its successive bands.

The solution returned by HotFill can be viewed as an elaboration of the scan-line solutions, using bandpaths instead of individual scan-lines, in order to reduce the air time where

possible. Indeed, a scan-line solution (alternating or non-alternating) is a fullpath composed from bandpaths that span a single scan-line each.

3.1 Dynamic programming

The HotFill algorithm considers at most two valid bandpaths for each (i,j)-band, which are stored in the array elements $B_0[i,j]$ and $B_1[i,j]$. Either or both of these elements may be set to Λ to signify that the bandpath was not created for some reason. It also constructs at most two fullpaths $F_0[i,j]$ and $F_1[i,j]$ for each (i,j)-band; where $F_0[i,j]$ is the best valid (i,j)-fullpath that ends with $B_0[i,j]$, $F_1[i,j]$ is the best valid (i,j)-fullpath that ends with $B_1[i,j]$, and "best" means "with minimum fab-time". Either element is Λ if there is no such valid fullpath. For this algorithm, it is convenient to let $T_{\mathsf{FAB}}(\langle\rangle)$ be zero and $T_{\mathsf{FAB}}(\Lambda)$ be $+\infty$.

The HotFill heuristic (Algorithm 1) uses the *dynamic programming* approach (BELLMAN, 1957) to find these optimal fullpaths. It enumerates all the (i,j)-bands from bottom to top, determining the optimal (i,j)-fullpaths $F_0[i,j]$ and $F_1[i,j]$ for each one. At the end, the desired solution will be one of the fullpaths $F_0[i,s]$ or $F_1[i,s]$, for all i in $\{0,1,\ldots,s-1\}$ — whichever has the least fab-time.

For efficiency reasons, the algorithm only considers bandpaths that span at most a specified number μ of scan-lines. That is, it ensures and assumes that $B_z[i,j]$ is Λ if $j - i > \mu$. Therefore, each execution of the loops on i (in Algorithm 1) and k (in Algorithm 2) will perform at most μ iterations, instead of s.

Algorithm 1 – HotFill

Input: A set $\mathcal{R}$ of horizontal rasters on s distinct scan-lines, with the associated links; a set $\mathcal{C}$ of relevant contacts between them; and a band height limit μ.
Output: A valid fullpath H that uses all the rasters $\mathcal{R}$ and any applicable links; or Λ if it cannot find such a path.

```
 1: for j in 1, 2, ..., s do
 2:     for i in 0, 1, ..., j − 1 with j − i ≤ μ do
 3:         for z in {0, 1} do
 4:             B_z[i,j] ← BandPath(R,C,i,j,z)
 5:             F_z[i,j] ← MinFullPath(R,C,μ,i,j,F_0,F_1,B_z[i,j])
 6:         end for
 7:     end for
 8: end for
 9: H ← Λ
10: for i in 0,1,...,s − 1 with s − i ≤ μ do
11:     for z in {0, 1} do
12:         if T_FAB(F_z[i,s]) < T_FAB(H) then
13:             H ← F_z[i,s]
14:         end if
15:     end for
16: end for
17: return H
```

Source: Own Authorship.

The key observations that make the dynamic programming approach possible are:

1. if $i = 0$, the only (i,j)-fullpaths are the $(0,j)$-bandpaths $B_0[0,j]$ and/or $B_1[0,j]$, if they exist.

2. if $i > 0$, an (i,j)-fullpath P is a (k,i)-fullpath P' concatenated with an (i,j)-bandpath B, either $B_0[i,j]$ or $B_1[i,j]$.

3. the path P above, if it exists, is valid if and only if P' and B are valid, and the contacts between them (on the cut-line i) have their cooling constraints satisfied.

4. the cooling times of these contacts depend only on the final (k,i)-bandpath B of F'_0 (either $B_0[k,i]$ or $B_1[k,i]$) and the (i,j)-bandpath B.

It follows that the (i,j)-fullpath with minimum fab-time that ends with $B_0[i,j]$, if it exists, must be a (k,i)-fullpath S with minimum fab-time, either $F_0[k,i]$ or $F_1[k,i]$, concatenated with the (i,j)-bandpath $B_0[i,j]$, for some k in $0,1,\ldots,i-1$; provided that the cooling constraints of the contacts between S and $B_0[i,j]$ are satisfied by the concatenation of those two paths. The analogous conclusion holds for the minimum (i,j)-fullpath that ends with $B_1[i,j]$.

3.2 The MinFullPath procedure

The inner loop of the dynamic programming logic is implemented in the auxiliary procedure MinFullPath (Algorithm 2). The procedure is called when the entries $F_0[k,i]$ and $F_1[k,i]$ have been defined for all k in $0,1,\ldots,i-1$, and is given an (i,j)-bandpath B (either $B_0[i,j]$ or $B_1[i,j]$). It then computes the (i,j)-fullpath F (possibly Λ) that has minimum fab-time among all such valid fullpaths that end with B. In this computation, it assumes that $F_u[k,i]$ is Λ whenever $i-k$ exceeds μ. The path F is then stored by HotFill into $F_0[i,j]$ or $F_1[i,j]$, depending on which bandpath was given as B.

The number of non-Λ entries in each of the tables F_0 and F_1 is at most $\mu[(s-\mu)+(\mu+1)/2]$, which is less than $s\mu$. Figure 15 shows the typical situation of the array F_0 before a call to MinFullPath with $\mu = 5$, $i = 8$, and $j = 11$, for the input data of Figures 8 and 9. Light gray is used for entries $F_0[i',j']$ that will never be set nor used because $i' \geq j'$ or $j' - i' > \mu$. Entries in red were set to Λ because all potential (i',j')-fullpaths ending with the bandpath $B_1[i',j']$ were found to violate internal cooling constraints. Entries in black have yet to be computed. The remaining green entries $F_0[i',j']$ hold already computed valid (i',j')-fullpaths. The array F_1 has a similar appearance.

The entry $F_0[i,j]$ to be computed by MinFullPath is marked with a blue-green dot. The white dots indicate all the entries $F_0[k,i]$ (and $F_1[k,i]$) that need to be examined to compute $F_0[i,j]$. At the end of HotFill, the result will be one of the black entries in column 18.

Algorithm 2 – MinFullPath

Input: A set $\mathcal{R}$ of horizontal rasters on s distinct scan-lines, with the associated link paths; a set $\mathcal{C}$ of relevant contacts between them; a maximum band height μ; a pair of cut-line indices i,j with $0 \le i < j \le s$; two $(s+1) \times (s+1)$ arrays F_0,F_1 whose elements are fullpaths or Λ; and an (i,j)-bandpath B or Λ.
Output: A (valid) (i,j)-fullpath F that uses all the rasters $\mathcal{R}$ up to cut-line ℓ_j, ends with the bandpath B, and has minimum fab-time among such fullpaths; or Λ if there is no such valid fullpath.

```
 1: F_* ← Λ
 2: if B ≠ Λ then
 3:    if i = 0 then
 4:       F ← B
 5:    else
 6:       for k in 0,1,...,i − 1 with i − k ≤ μ do
 7:          for z in {0,1} do
 8:             if F_z[k,i] ≠ Λ then
 9:                T ← Concat(F_z[k,i],B)
10:                if ValidPath(T,C) then
11:                   if T_FAB(T) < T_FAB(F_*) then
12:                      F_* ← T
13:                   end if
14:                end if
15:             end if
16:          end for
17:       end for
18:    end if
19: end if
20: return F_*
```

Source: Own Authorship.

The MinFullPath algorithm also uses the sub-routine ValidPath$(P,\mathcal{C})$ that checks whether the cooling constraints of the contacts $\mathcal{C}$ that are closed by the tentative fullpath P are satisfied.

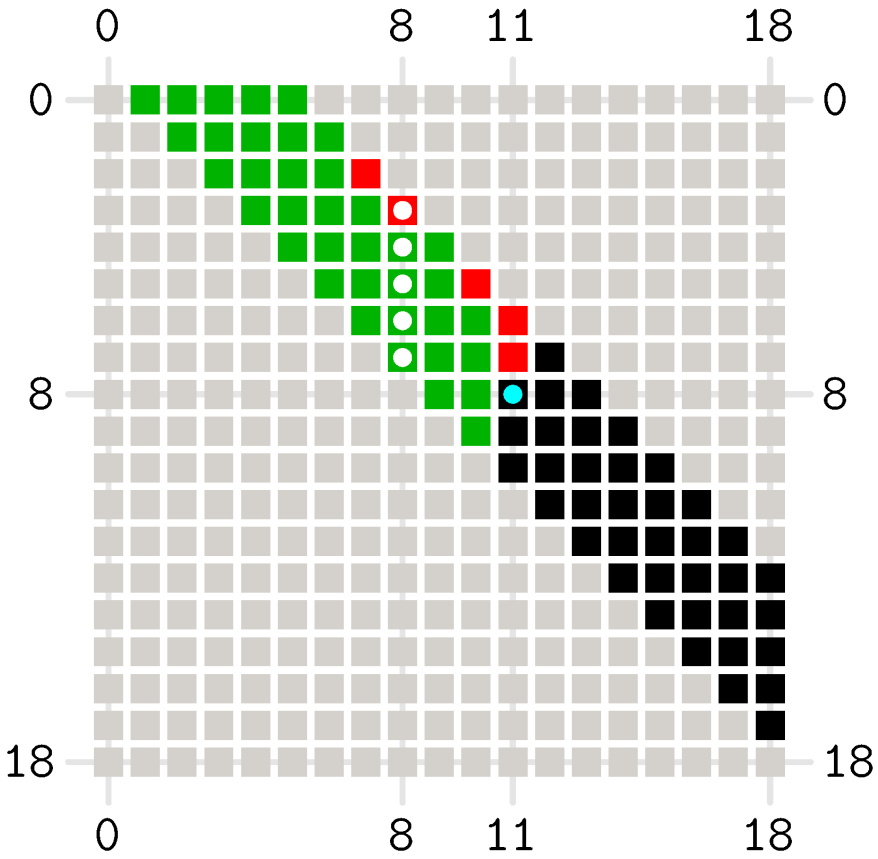
The function Concat(P',P'') is supposed to construct the concatenation of two given paths P' and P''. If either P' or P'' is the empty path $\langle\rangle$, it should return the other path. If either is the invalid path Λ, it should return Λ. If $p_{\mathsf{FIN}}(P') \neq p_{\mathsf{INI}}(P'')$, the gap should be bridged with a link path with those two endpoints, or with a jump if there is no such link path. The link, if it exists, will be the only element S such that $\overleftarrow{S} \in \mathcal{L}(\overleftarrow{P'})$ and $S \in \mathcal{L}(P'')$.

3.3 Choosing the Bandpaths

Each of the two (i,j)-bandpaths used by HotFill, $B_z[i,j]$ for $z = 0$ and $z = 1$, is defined by sub-routine BandPath$(\mathcal{R},\mathcal{C},i,j,z)$. Its goal is to assemble the band rasters $\mathcal{R}_{i,j}$ into a tool-path that satisfies the cooling constraints of the contacts between those rasters, and has small enough fab-time.

Finding the absolutely best (i,j)-bandpath is too hard; in fact, computing the best possible valid $(0,s)$-bandpath is the same as computing the best valid tool-path for the input rasters, which, as we have noted, is still a difficult problem to solve. Therefore, BandPath must be some

Figure 15 – Typical status of the array F_0 upon entry to MinFullPath with parameters $\mu = 5$, row index $i = 8$, and column index $j = 11$, with the input data of Figures 8 and 9. For the color codes, see the text.



Source: Own Authorship.

heuristic that considers only some small subset of all possible paths that can be built from the rasters $\mathcal{R}_{i,j}$. The path it returns must satisfy the cooling constraints of all internal contacts (between rasters of $\mathcal{R}_{i,j}$). The procedure may return Λ if it cannot find such a path among the paths it considers. However, if the band has a single scan-line, the procedure must return the path consisting of all the rasters in that scan-line, sorted and oriented from left to right (if $z = 0$) or right to left (if $z = 1$). This ensures that, if the scan-line order paths SCN and/or SCA are valid, HotFill will consider them among all the possible fullpaths; and therefore it will never return a solution that has greater fab-time than those two.

Apart from the two requirements above, the BandPath procedure can be quite variable. Ideally it should return a path that is likely to have a good fab-time, hopefully much smaller than the fab-times of the scan-line solutions. Moreover, the path should begin and end in such a way that it can be concatenated with the bandpaths below and above it, respectively, without excessively long jumps.

The bidirectional greedy bandpath: For this work, we have chosen the following heuristic for the BandPath procedure. When $z = 0$, the result of BandPath (that will become $B_0[i,j]$) starts with the leftmost raster on scan-line i (at the bottom of the band), and ends with the right-

most raster on scan-line $j - 1$, both oriented from left to right. When $z = 1$, the result (that will be $B_1[i,j]$) starts with the *rightmost* raster on scan-line i, and ends with the *leftmost* raster on scan-line $j - 1$, both oriented *from right to left*.

In either case, BandPath builds the path incrementally, by the greedy method, starting at both ends and "meeting in the middle". Namely, it creates two paths P' and P'', initialized with the first and last rasters, respectively, chosen as above. At each iteration it selects a rasters r' among the still unused rasters in $\mathcal{R}_{i,j}$ or their reversals, so that $p_{\mathsf{INI}}(r')$ is closest to $p_{\mathsf{FIN}}(P')$. It then selects a raster r'', distinct from r', such that $p_{\mathsf{FIN}}(r'')$ is closest to $p_{\mathsf{INI}}(P'')$. Here "closest" means that the fab-time of the connector (link path or jump) between the two elements is minimized. Then r' is appended to P', and r'' is prefixed to P''.

If there is only one left-over raster r, either it or its reversal is appended to P', depending on which one will give the smallest total connection time. In any case, the final bandpath is the concatenation of P' and P'', with an intervening link path or jump, as appropriate.

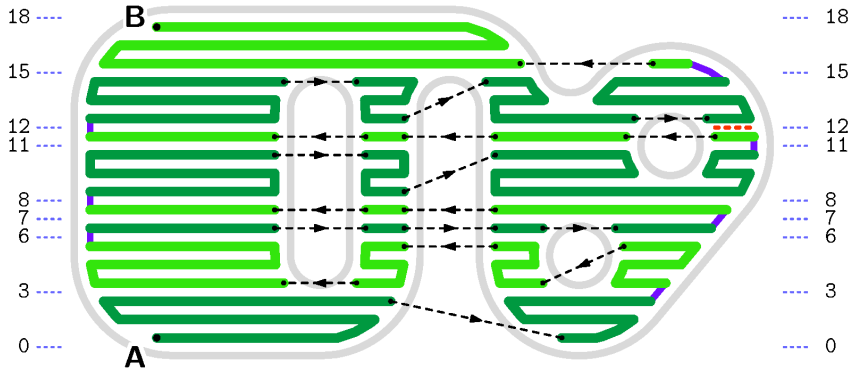
If the band has two or more scan-lines, the result of this BandPath heuristic will typically consist of one or more continuous extrusion sections, each moving up or down, with rasters in alternating directions; and these sections will be ordered from left to right (when $z = 0$) or from right to left (when $z = 1$). These paths will typically be better than the scan-line tool-paths for the rasters $\mathcal{R}_{i,j}$, because many of the jumps will be replaced by link paths or shorter jumps. Often, each CES will span the whole width of the band, and all internal contacts will be between successive rasters of the same CES. In these cases, the internal cooling constraints are almost certain to be satisfied, if the input data admits a valid solution at all.

In particular, when the band has a single scan-line ($j - i = 1$), the bandpath $B_0[i,j]$ will be the rasters on that scan-line oriented and concatenated from left to right; and $B_1[i,j]$ will be the reverse of that path.

3.4 Sample Solutions

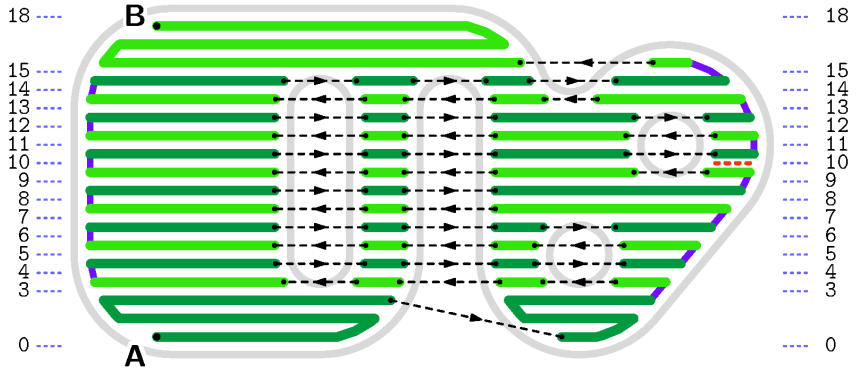
For illustration purposes, we show below the outputs of HotFill for the input data of Figure 8 and 9, with the cooling time limits set to different values. More realistic examples are given in Section 4. Figures 16, 17, and 18 show the HotFill tool-paths with the cooling time limits $\tau(c)$ of every contact c set to $3.5\,\mathrm{s}$, $2.3\,\mathrm{s}$, and $1.7\,\mathrm{s}$, respectively. The red dashed lines show the contacts with maximum cooling time and the blue horizontal dotted lines are the cut-lines that separate the bandpaths that comprise the solution.

Figure 16 – The HotFill solution with $\mu = 5$ and all cooling time limits set to $3.5\,\mathrm{s}$. The largest contact cooling time (red dashed line) is $3.36\,\mathrm{s}$. The total fab-time is $15.26\,\mathrm{s}$ including $11.97\,\mathrm{s}$ of extrusion and $3.29\,\mathrm{s}$ of air time.



Source: Own Authorship.

Figure 17 – The HotFill solution with $\mu = 5$ and all cooling time limits set to $2.3\,\mathrm{s}$. The largest contact cooling time (red dashed line) is $2.25\,\mathrm{s}$. The total fab-time is $17.68\,\mathrm{s}$ including $11.17\,\mathrm{s}$ of extrusion and $6.51\,\mathrm{s}$ of air time.



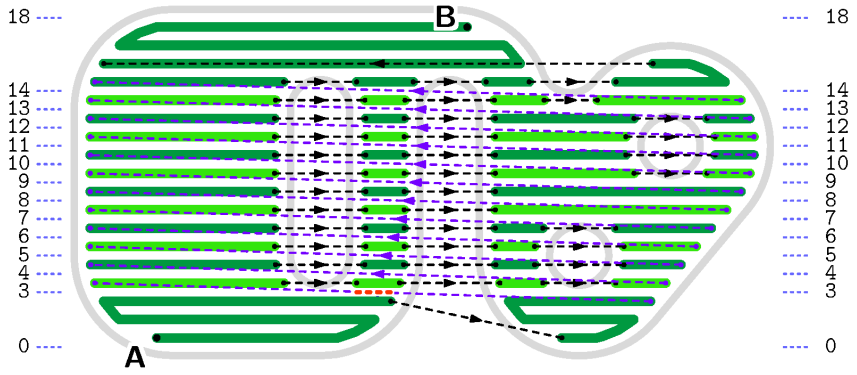
Source: Own Authorship.

The last two solutions should be compared to those of the scan-line solutions, alternating (Figure 11) and non-alternating (Figure 10). The HotFill solutions have the same maximum cooling times, but slightly lower fab-times ($17.68\,\mathrm{s}$ vs. $17.95\,\mathrm{s}$ and $22.24\,\mathrm{s}$ vs. $23.71\,\mathrm{s}$, respectively).

Figure 19 shows the HotFill output when μ is set to 50 (which is the same as not having any limit on band height) and all cooling time limits are set to $50\,\mathrm{s}$ (which, being more than the total fab-time of the slice, is the same as not having any cooling constraints at all). Note that the selected fullpath is still formed by two bandpaths: $(0,17)$ and $(17,18)$. This result was found to be equal or better than the fullpath consisting of a single $(0,18)$-bandpath. This solution should be compared to the typical result of path-planing software, Figure 4, which has slightly better fab-time ($13.51\,\mathrm{s}$ vs. $13.59\,\mathrm{s}$).

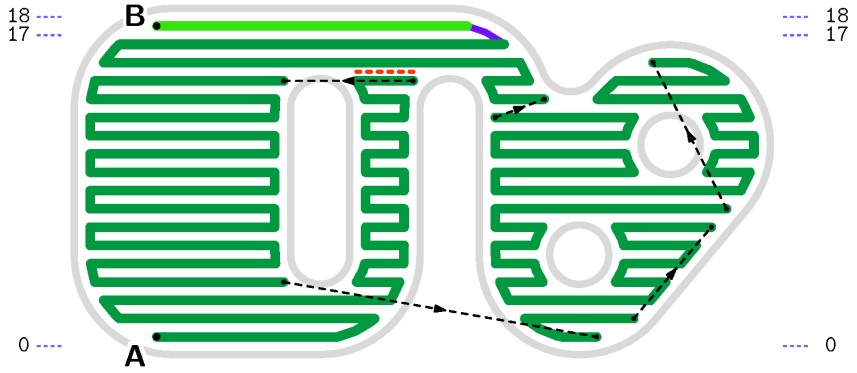
For comparison, Figure 20 shows the HotFill output with the same cooling limits but with μ set to 5. Note that the bandpaths have been limited to 5 scan-lines, and thus the result consists of five bandpaths: $(0,3)$, $(3,8)$, $(8,13)$, $(13,17)$ and $(17,18)$. The fab-time ($14.06\,\mathrm{s}$) is not much worse than the one obtained with $\mu = 50$ ($13.59\,\mathrm{s}$) and that of the path of Figure 4 ($13.51\,\mathrm{s}$).

Figure 18 – The HotFill solution with $\mu = 5$ and all cooling time limits set to $1.7\,\mathrm{s}$. The largest contact cooling time (red dashed line) is $1.69\,\mathrm{s}$. The total fab-time is $22.24\,\mathrm{s}$ including $10.66\,\mathrm{s}$ of extrusion and $11.58\,\mathrm{s}$ of air time.



Source: Own Authorship.

Figure 19 – The HotFill solution with $\mu = 50$ and all cooling time limits set to $50\,\mathrm{s}$. The largest contact cooling time (red dashed line) is $9.60\,\mathrm{s}$. The total fab-time is $13.59\,\mathrm{s}$ including $12.55\,\mathrm{s}$ of extrusion and $1.04\,\mathrm{s}$ of air time.



Source: Own Authorship.

3.5 Improvements and Computing Cost

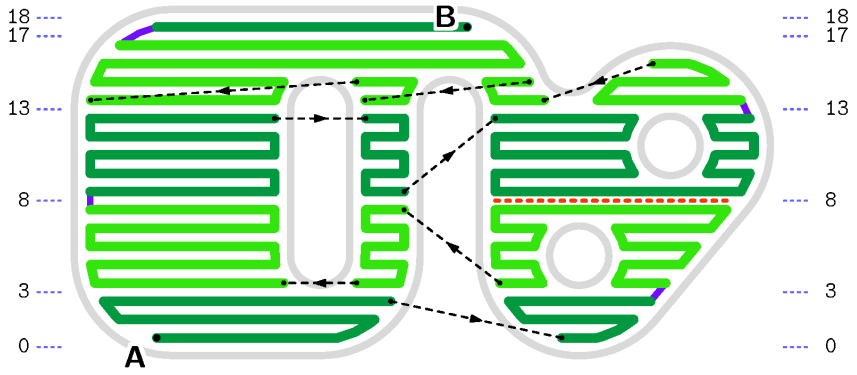
3.5.1 Algorithm Improvements

A straightforward implementation of Algorithm 1, as described in Chapter 3, would be too slow for production use. The following optimizations were needed to make it practical.

Virtual Fullpaths: Most of HotFill's running time, as described, would go into building the fullpaths $F_0[i,j]$ and $F_1[i,j]$. Concatenation of two paths, if implemented in the most straightforward way, takes time proportional to the number of moves in the resulting path. This item alone would contribute a factor of $n = \#\mathcal{R}$ to the total computing time.

That cost is substantially reduced by replacing the path tables F_0 and F_1 by four arrays T_0, T_1, K_0, and K_1, with the same size $(s+1) \times (s+1)$. Each entry $T_z[i,j]$ is set to $+\infty$ if the computed $F_z[i,j]$ would be Λ, otherwise it is set to what would be $T_{\mathsf{FAB}}(F_z[i,j])$. Each entry

Figure 20 – The HotFill solution with $\mu = 5$ and all cooling time limits set to $50\,\mathrm{s}$. The largest contact cooling time (red dashed line) is $6.50\,\mathrm{s}$. The total fab-time is $14.06\,\mathrm{s}$ including $12.44\,\mathrm{s}$ of extrusion and $1.62\,\mathrm{s}$ of air time.



Source: Own Authorship.

$K_z[i,j]$ is a pair (k,z') of an integer and a bit. It is set to $(-1,0)$ if $F_s[i,j]$ would be Λ or $i = 0$, otherwise it is such that $F_z[i,j]$ would be formed by concatenating $F_{z'}[k,i]$ with $B_0[i,j]$.

With these tables, it is not necessary to build and store the paths $F_0[i,j]$ and $F_1[i,j]$. The tables T_0, T_1, K_0, and K_1 are filled instead by slight modifications of Algorithms 1 and 2. The final tool-path H is then reconstructed, at the end of HotFill, by following the pointers $K_z[i,j]$ back to $(-1,0)$ and concatenating the bandpaths $B_z[i,j]$ identified by them.

Fast Fullpath Validity Check: The ValidPath procedure needs only check the cooling times of the contacts on cut-line i by the paths $F_z[i,j]$, which it can compute from the cover times of those contacts by the bandpaths $B_{z'}[k,i]$, and $B_z[i,j]$. To avoid repeated computations inside ValidPath, these cover times can be pre-computed, only once, when the bandpaths are created.

Specifically, after building a tentative bandpath $B_z[i,j]$, we consider each contact c on cut-line i, and compute the value $T^{\text{PRED}}_{\text{COOL}}(B_*,c)$ as $T_{\text{FAB}}(B_*)$, if B_* does not cover c, or $T_{\text{COV}}(B_*,c)$, if it does. Symmetrically, for each contact c on cut-line j, we compute $T^{\text{PRED}}_{\text{COOL}}(B_*,c)$ as 0, if B_* does not cover c, or $T_{\text{COV}}(\overleftarrow{B_*},c)$, if it does. These two lists of cover times are saved by HotFill, associated with the the path $B_z[i,j]$, and used by ValidPath.

Moreover, before returning from the BandPath procedure, we check whether any of those cover times, by itself, already exceeds the cooling time limit $\tau(c)$ of the corresponding contact. If that happens, that cooling constraint will be violated, no matter what other bandpath gets concatenated with $B_z[i,j]$. In that case, BandPath discards the candidate and returns Λ. Then the following call to MinFullPath immediately return Λ (Step 2), skipping the loop on k and z and all calls to ValidPath. As the width $j - i$ of the band increases, this optimization will be increasingly effective.

3.5.2 Programming Improvements

Even with the algorithmic improvements above, a straightforward implementation of HotFill would waste a considerable amount of time computing functions such as $T_{\mathsf{COV}}(P,c,r)$, $T_{\mathsf{COOL}}(P,c)$, $R^{\mathsf{MAX}}_{\mathsf{COOL}}(P,\mathcal{C})$ and so on. For example, computing $T_{\mathsf{COV}}(P,c,r)$ in the obvious way would require scanning the moves of the path P to locate the trace r, while computing and adding the fabtime of all moves of P up to that one; which would make the cost of T_{COV} proportional to the number $\#P$ of moves in the path, and the cost of $R^{\mathsf{MAX}}_{\mathsf{COOL}}(P,\mathcal{C})$ proportional to $\#P \times \#\mathcal{C}$. These costs in turn could make the cost of the innermost loop of HotFill proportional to size n of the input data, or even to n^2.

Our implementation uses several standard programming techniques to speed up those operations, often reducing the costs of those operations to a constant or to a multiple of $\log n$. Here are some of them.

Orientation Bits: A move r is represented as a pair $(\boldsymbol{r}, b)$ where $\boldsymbol{r}$ is a pointer to an object of a class Move, that describes the move in some arbitrary *native orientation*; and b is a bit that tells whether r is $\boldsymbol{r}$ ($b = 0$) or $\overleftarrow{\boldsymbol{r}}$ ($b = 1$). Thus reversing a move requires only complementing the bit b, without creating another object. The endpoints of the move are fields $\boldsymbol{r}.end[0]$ and $\boldsymbol{r}.end[1]$ in the order of the native orientation; so $p_{\mathsf{INI}}(r) = \boldsymbol{r}.end[b]$, $p_{\mathsf{FIN}}(r) = \boldsymbol{r}.end[1-b]$, and so on. This representation also makes it trivial to check whether a move is the reversal of another move.

This techniques is used for paths too. A path P is represented as a pair $(\boldsymbol{P},b)$ where $\boldsymbol{P}$ is an object of class Path that represents the path in some arbitrary native orientation, and the bit b tells whether P is $\boldsymbol{P}$ or its reversal. Then, for example, $P[k]$ is $\boldsymbol{P}.mv[k]$ if $b = 0$, and $\overleftarrow{\boldsymbol{P}.mv[k']}$ if $b = 1$; where n is the number of elements of $\boldsymbol{P}.mv$, and $k' = n - 1 - k$.

Pre-computed Timing Functions: The fabrication time $T_{\mathsf{FAB}}(r)$ of a move r is computed only once, when the Move object $\boldsymbol{r}$ is created, and stored as a field $\boldsymbol{r}.fabtime$.

Likewise, when a path object $\boldsymbol{P}$ is created, the values of $T_{\mathsf{FIN}}(\boldsymbol{P},k)$ for all k are stored in a table in the object record, namely $\boldsymbol{P}.tfin[k]$. Then the functions $T_{\mathsf{FAB}}(P)$, $T_{\mathsf{INI}}(P,k)$, and $T_{\mathsf{FIN}}(P,k)$, for $P = \boldsymbol{P}$ or $P = \overleftarrow{\boldsymbol{P}}$, can be computed from that list — in constant computing time, irrespective of the length of the path.

Contacts: Each contact c is represented by an object $\boldsymbol{c}$ of class Contact, that contains the two endpoints $c.end[0]$ and $c.end[1]$ and pointers $c.side[0]$ and $c.side[1]$ to the incident Move objects. The order of both pairs is irrelevant. The object $\boldsymbol{c}$ would have fields $\boldsymbol{c}.tcov[i]$ holding the pre-computed value of the cover time of the contact by the trace that is side i, in its native orientation; that is, $T_{\mathsf{FAB}}((\boldsymbol{c}.side[i],0),s)$ where s is the contact's midpoint.

Locating Contacts and Rasters: Another potential waste of computing time is scanning the list $\mathcal{C}$ to find the contacts that are closed or covered by a path, such as $B_0[i,j]$ or $B_1[i,j]$. This time is reduced by splitting the list $\mathcal{C}$ into s separate lists $C[0]$, $C[1]$, ..., $C[s]$, where $C[i]$ has all the contacts on cut-line i. Similarly, when locating the rasters in an (i,j)-band, we save time by splitting the raster set $\mathcal{R}$ into $s-1$ lists $R[0]$, $R[1]$, ..., $R[s-1]$, one per scan-line.

Locating Links: The lists $\mathcal{L}(P)$ and $\mathcal{L}(\overleftarrow{P})$ of link paths that connect to the endpoints of any partial tool-path P handled by HotFill — specifically each of the initial rasters, and each bandpath or fullpath P (including) — are stored as attributes of the relevant path object $\boldsymbol{P}$. Then the Concat function can quickly locate the bridging link by looking for a matching entry in the lists $\mathcal{L}(\overleftarrow{P'})$ and $\mathcal{L}(P'')$, stored in the objects $\boldsymbol{P'}$ and $\boldsymbol{P''}$. Note that these lists typically will have two entries or less, so this search will be very fast.

3.5.3 Computing Time Analysis

We now derive a formula for the asymptotic worst-case computing time (Knuth, 1998) of the HotFill algorithm, that tells how that time is expected to grow as the input size parameters n and s increase. We assume all the above optimizations have been implemented, and that the maximum band width μ is at most equal to the number of scan-lines s in the input.

The procedures BandPath and MinFullPath will be called $2(s-\mu)\mu+\mu(\mu+1)$ times each, which less than $2s\mu$. The asymptotic worst-case computing cost for each call to MinFullPath will be $\Theta(\mu)$, arising from the loop on k. Each call to BandPath is expected to cost $\Theta((n_{i,j})^2)$ for the construction of the bandpaths $B_0[i,j]$ and $B_1[i,j]$; where $n_{i,j} = \#\mathcal{R}_{i,j}$ is the number of rasters in the band. If we assume that there is an $O(1)$ upper bound on the number of rasters per scan-line, then n_{ij} will be $\Theta(\mu)$, and the computing cost of BandPath will be $\Theta(\mu^2)$.

Therefore, the cost of HotFill will be dominated by the calls of BandPath, namely $\Theta(s\mu^3)$ or $\Theta(n\mu^3)$ in the worst case, where $n = \#\mathcal{R}$ is the number of input rasters. This worst case will be achieved when the cooling time limits are large enough for all potential fullpaths $F_0[i,j]$ and $F_1[i,j]$ with $0 \le i < j \le s$ to be valid.

However, as μ approaches or exceeds the number of scan-lines s, the number of calls to BandPath and MinFullPath tends to the limit $s(s+1)$, which does not depend on μ. Thus the computation time will flatten out at $O(s^3)$ at those large values of μ.

4 EXPERIMENTS AND RESULTS

In this chapter we report some computational experiments that aim to demonstrate and quantify certain aspects of the HotFill algorithm. We used a Python 3 implementation of the algorithm (Nakonetchnei, 2021) that incorporates the optimizations described in Section 3.5.

The first set of tests aims to show that HotFill does achieve its goal, namely find a tool-path H that satisfies the specified cooling time constraints, if they can be achieved at all. See Section 4.2.

The second batch of tests aims to demonstrate that the imposition of reasonable cooling time constraints, which should improve the mechanical resistance of the part, generally has a small impact on the fabrication time T_{FAB} of the tool-path. See Section 4.3.

The third batch of tests aims to analyze the relationship between the input parameters – especially the maximum band width μ – and the computation time of HotFill. See Section 4.4.

To simplify the analysis and discussion of the results, in all tests the cooling time limit $\tau(c)$ of every contact c was set to the same value Δ. Thus the maximum relative cooling time $R^{\mathsf{MAX}}_{\mathsf{COOL}}(H,\mathcal{C})$ is simply $T^{\mathsf{MAX}}_{\mathsf{COOL}}(H,\mathcal{C})/\Delta$. For comparison, in each set of tests we show also the attributes of the tool-paths produced by two traditional tool-path planning programs, RP3 (Volpato, 2021) and Slic3r (Hodgson; Ranellucci; Moe, 2021) (which produce tool-paths according to their own criteria, without taking the cooling constraints into account), as well as the simple scan-line-order tool-paths, alternating (SCA) and non-alternating (SCN).

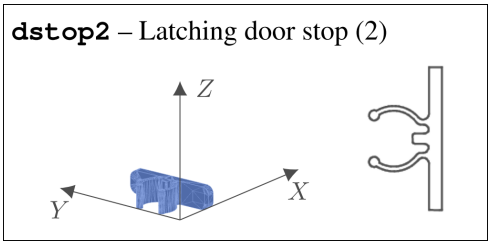
4.1 Dataset

To perform tests and compare the results obtained with the proposed algorithm, we obtained STL models for several objects where low mechanical strength could be an issue, such as human prostheses, mechanical parts, and coat hangers. Some models were created by ourselves, some were obtained from a public repository (MAKERBOT, 2021). From each object we selected a slice that was representative of the majority of the cross-sections.

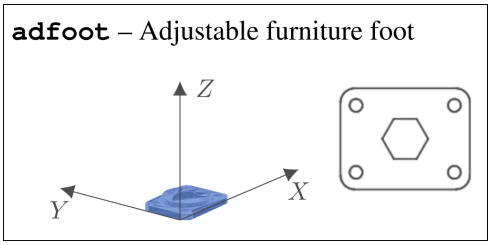
We tried to choose objects and slices such that the polygon D (the region to be filled) was substantially non-monotonic in either the X or Y direction (that is, where most scan-lines would have two or more raster lines); since, as observed in Section 2.5.2, there is no point in using the HotFill if D is monotonic. For this reason, we did not bother to apply the decomposition into sub-problems described in that section. These objects and the outlines of the selected slices are shown in Figure 21.

For all models the length unit of the STL file was assumed to be millimeters, except for **`runleg`** which was assumed to be in inches (and thus was scaled by 25.4).

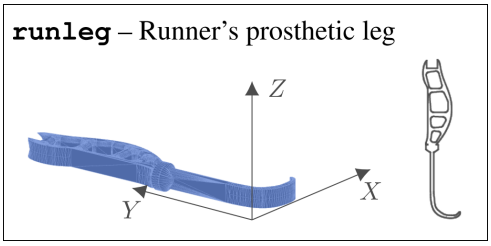
For each test slice, we performed two preliminary tests of the HotFill algorithm, with the slice rotated by $\theta = 0$ and $\theta = 90$ degrees. The rasters and link paths for input to HotFill were obtained by modifying the RP3 program so as to write out that information to a text file.



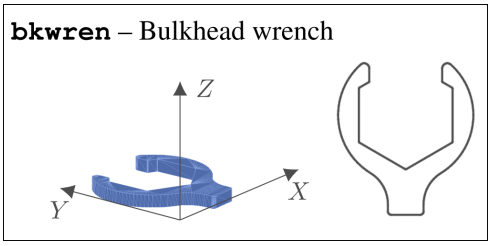
(a) (Matearoa, 2021)



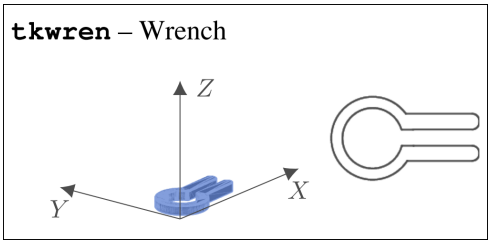
(b) (ffleurey, 2013)



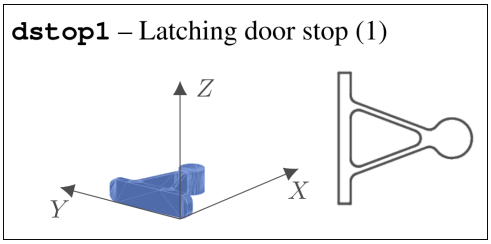
(c) (nillasarie, 2016)



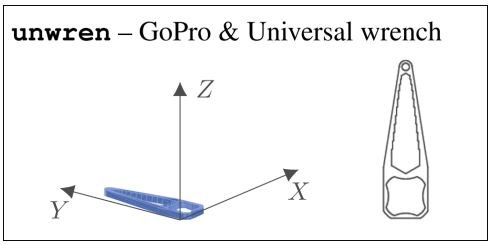
(d) (jephil08, 2020)



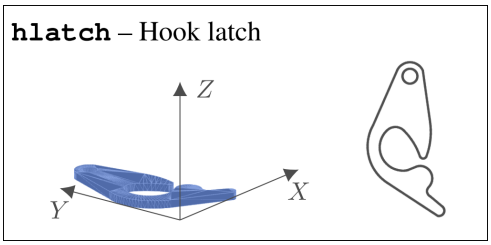
(e) (joraasen, 2016)



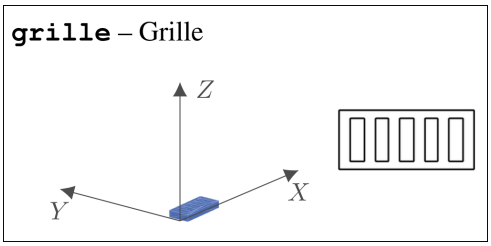
(f) (Matearoa, 2021)



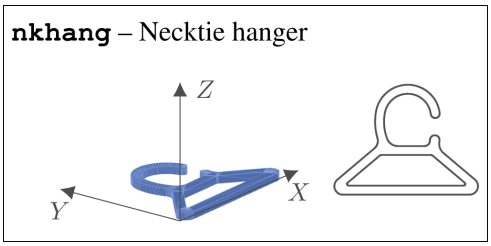
(g) (M600, 2014)



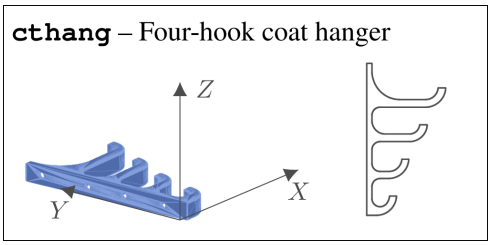
(h) (ecassiana, 2021c)



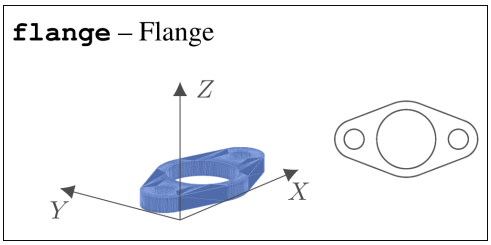
(i) (ecassiana, 2021b)



(j) (ecassiana, 2021d)



(k) (Alpha__Wolf, 2018)



(l) (ecassiana, 2021a)

Figure 21 – The STL models and slice outlines used in the tests (not to the same scale).

In each case, the RP3 software was asked to produce an infill-only tool-path (with zero contour layers) with assumed slice thickness $\sigma = 0.25\,\mathrm{mm}$, scan-line spacing (nominal raster width) $\lambda_{\mathsf{FILL}} = 0.400\,\mathrm{mm}$, and the specified raster orientation. The polygonal outline D of the area to be filled was therefore result of applying an inwards offset of $\lambda_{\mathsf{FILL}}/2$ to the slice's outline, to account for the actual extent of the material. Each link path produced by RP3 was the piece of the boundary of the polygon D between the relevant raster endpoints.

In order to make the fab-time comparisons with Slic3r meaningful, the parameters of the latter had to be adjusted so that the infill domain D was the same. We observed that Slic3r slightly adjusts the specified raster spacing depending on the slice geometry, so we had to specify the spacing as $\lambda_{\mathsf{FILL}} = 0.453\,\mathrm{mm}$ in order to get approximately the same number and total length of the raster lines as we got with RP3, when averaged over all datasets.

In the interest of brevity, we generally show the results for only one of the two datasets ($\theta = 0$ or $\theta = 90$ degrees) created from each STL model, namely the one which turned out to be the most challenging for HotFill (in the sense of Section 4.3). However we retained both datasets for the slice **`runleg`**, to show the impact of raster orientation on the attributes of the resulting tool-paths.

The parameters of the selected datasets are presented in Table 1. The total area to be filled, in mm^2, is approximately the total raster length L_{TOT} times the scanline spacing λ_{FILL}.

Table 1 – Datasets used for tests. The columns are the slice index (1 = bottom); the raster direction θ (0 or 90) in degrees, relative to the X-axis; the overall width X and height Y of the slice; the number $n = \#\mathcal{R}$ of raster lines; the number $s = \#\mathcal{S}$ of scan-lines; and the maximum L_{MAX}, average L_{AVG}, and total L_{TOT} length of the raster lines. All dimensions are in millimeters.

	Dataset	Slice	θ	X	Y	n	s	Raster length (mm)		
								L_{MAX}	L_{AVG}	L_{TOT}
1	**`dstop2:90`**	1	90	51.9	25.6	134	65	51.9	6.5	869.9
2	**`adfoot:0`**	5	0	44.7	34.4	165	87	44.7	19.9	3276.6
3	**`runleg:0`**	17	0	54.3	231.2	851	579	34.8	6.8	5787.4
4	**`bkwren:0`**	1	0	74.7	83.2	355	209	44.3	14.0	4977.7
5	**`tkwren:90`**	1	90	32.7	58.0	281	146	22.1	6.1	1717.9
6	**`dstop1:0`**	1	0	52.7	51.6	208	130	22.6	7.4	1538.1
7	**`unwren:90`**	1	90	70.4	19.2	138	49	27.8	6.8	936.0
8	**`hlatch:90`**	1	90	106.4	52.0	252	131	62.5	23.4	5895.4
9	**`grille:0`**	1	0	32.0	13.6	160	35	32.0	4.5	714.4
10	**`nkhang:90`**	1	90	75.7	98.8	554	248	41.1	8.4	4660.3
11	**`cthang:90`**	1	90	123.7	63.6	398	160	123.7	10.8	4301.2
12	**`flange:0`**	1	0	109.7	58.0	335	146	44.4	18.3	6143.9
13	**`runleg:90`**	17	90	231.7	53.6	443	135	106.0	13.0	5766.2

Source: Own Authorship.

4.2 Achieving the cooling constraints

The tests in this section aim to show that HotFill does achieve its goal; namely, find a viable tool-path H that satisfies the specified cooling time constraints (which in this case are $T_{\text{COOL}}(H,c) \leq \Delta$ for all $c \in \mathcal{C}$). The results are shown in Table 2. The entries marked "—" in the table are cases where HotFill failed to find a viable tool-path.

Table 2 – Maximum contact cooling times $T^{\text{MAX}}_{\text{COOL}}(H,\mathcal{C})$ of the tool-path H computed by Slic3r, RP3, SCN, and by HotFill for various cooling time limits Δ, with $\mu = 20$. All times are in seconds.

					HotFill with various values of Δ											
Dataset	Slic3r	RP3	SCN	SCA	1.4	2.0	2.8	4.0	5.7	8.0	11.3	16.0	22.6	32.0	45.3	64.0
1 **dstop2:90**	8.7	9.3	1.9	1.9	—	2.0	2.8	3.9	5.2	5.2	5.2	5.2	5.2	5.2	5.2	5.2
2 **adfoot:0**	78.2	27.5	1.9	2.6	—	1.9	2.8	3.8	5.7	7.8	11.3	15.1	16.1	16.1	16.1	16.1
3 **runleg:0**	115.9	64.8	1.3	1.4	1.4	2.0	2.8	4.0	5.7	7.7	9.0	9.0	9.0	9.0	9.0	9.0
4 **bkwren:0**	96.0	50.0	1.9	2.1	—	2.0	2.8	4.0	5.7	7.9	11.2	15.3	16.4	16.4	16.4	16.4
5 **tkwren:90**	49.2	26.3	1.0	1.1	1.4	2.0	2.8	4.0	5.7	7.2	8.2	8.2	8.2	8.2	8.2	8.2
6 **dstop1:0**	40.6	31.9	1.7	2.2	—	2.0	2.8	3.9	5.6	8.0	9.7	15.3	17.6	17.6	17.6	17.6
7 **unwren:90**	29.2	16.4	1.9	2.3	—	1.9	2.8	4.0	5.6	7.9	11.0	14.0	14.0	14.0	14.0	14.0
8 **hlatch:90**	115.8	96.2	3.3	4.7	—	—	—	4.0	5.7	7.9	10.8	15.9	22.4	31.9	41.5	41.5
9 **grille:0**	16.3	14.6	1.7	2.5	—	1.7	2.7	4.0	5.6	7.4	11.1	14.5	14.5	14.5	14.5	14.5
10 **nkhang:90**	107.4	119.6	2.2	2.9	—	—	2.8	4.0	5.7	8.0	11.3	15.9	20.9	20.9	20.9	20.9
11 **cthang:90**	83.2	58.3	4.2	5.2	—	—	—	4.0	5.7	8.0	11.2	15.8	22.1	31.5	35.2	35.2
12 **flange:0**	134.9	113.9	3.2	4.2	—	—	—	4.0	5.7	8.0	11.2	16.0	22.1	31.5	36.8	36.8
13 **runleg:90**	127.4	91.8	6.0	8.5	—	—	—	—	—	8.0	11.2	15.9	22.4	30.0	45.0	60.4

Source: Own Authorship.

As discussed in Section 2.4.2, the non-alternating scan-line solution (SCN) usually minimizes the maximum contact cooling time $T^{\text{MAX}}_{\text{COOL}}$. It follows that the $T^{\text{MAX}}_{\text{COOL}}$ of the SCN path is usually the minimum value of the cooling time limit Δ for which a solution exists. That observation is confirmed by Table 2. Indeed, in those tests, the HotFill procedure always found a solution whenever the specified Δ was higher than the SCN $T^{\text{MAX}}_{\text{COOL}}$. For example, the SCN path for the dataset **bkwren:0** has $T^{\text{MAX}}_{\text{COOL}} = 1.9\,\text{s}$, and the procedure succeeded once Δ was above that value ($2.0\,\text{s}$). However, for the **cthang:90** dataset, HotFill managed to find a solution even for $\Delta = 4.0$, which was lower than then $T^{\text{MAX}}_{\text{COOL}}$ of the SCN path.

Several factors determine the maximum cooling time of the SCN solution, which (as noted above) is usually the lowest cooling time limit Δ that allows a solution. The fabrication time of a scan-line depends on the number of rasters on it, and on the length of those rasters and of the gaps between them. This fab-time usually affects the cooling times of the contacts that involve those rasters. It can be seen that the datasets that have largest SCN cooling times, near the bottom of Table 2, also tend to have the longest raster lines (L_{MAX} on Table 1) and higher numbers of rasters per scan-line (n/s).

As expected, the reference programs Slic3r and RP3 often grossly exceed the limit Δ, since they mostly try to minimize the total fab-time with no regard for cooling times.

For every dataset there is a maximum value of Δ beyond which every bandpath and fullpath considered by HotFill will be valid, as if there was no cooling constraint (that is, as if Δ

was $+\infty$). Beyond that point, HotFill will return always the same tool-path, the one with minimum fab-time. Typically this path will still have better cooling time that the Slic3r and RP3 tool-paths, and its fab-time will not be much worse (see Section 4.3).

4.3 Impact on fabrication time

The tests in this section aim to show the impact of the cooling time constraints on the fabrication time $T_{\mathsf{FAB}}(H)$ of the computed tool-path. The results are shown in Table 3. For each dataset, besides the absolute fab-time T_{FAB} in seconds of each reference algorithm (Slic3r, RP3, SCN, SCA), we also give the absolute fab-time $T^{\mathsf{HF}}_{\mathsf{FAB}}$ of HotFill for each limit Δ, and the percent increase $D^{\mathsf{HF}}_{\mathsf{FAB}}$ of that fab-time relative to $T^{\mathsf{REF}}_{\mathsf{FAB}}$, the smallest between the Slic3r and RP3 fab-times. The total extrusion time T_{RAST} for the input rasters, which is the ideal lower bound for $T_{\mathsf{FAB}}(H)$, is also shown for reference. Each test was performed with $\mu = 20$.

Table 3 – Fabrication times $T_{\mathsf{FAB}}(H)$ of the tool-path H computed by RP3, SLic3r, SCN, SCA, and by HotFill with various cooling time limits Δ, as well as percentage increases in manufacturing time over the shortest time between RP3 and Slic3r. The total extrusion time T_{RAST} for the rasters alone (excluding jumps and links) is also given for reference. All tests used $\mu = 20$. All times are in seconds.

Dataset	T_{RAST}	Slic3r	RP3	SCN	SCA	HotFill for various values of Δ									
						1.4	2.0	2.8	4.0	5.7	8.0	11.3	16.0	32.0	64.0
1 **dstop2:90**	23.5	28.8	28.4	61.1	41.6	—	31.0	29.8	29.1	28.8	28.8	28.8	28.8	28.8	28.8
				+115%	+46%		+9%	+5%	+2%	+1%	+1%	+1%	+1%	+1%	+1%
2 **adfoot:0**	84.1	89.7	90.0	141.0	101.8	—	112.9	98.8	96.9	93.1	91.6	90.9	89.8	89.7	89.7
				+57%	+13%		+26%	+10%	+8%	+4%	+2%	+1%	+0%	+0%	+0%
3 **runleg:0**	156.0	178.3	181.9	402.4	249.7	225.0	199.8	190.6	186.1	183.5	183.0	183.0	183.0	183.0	183.0
				+126%	+40%	+26%	+12%	+7%	+4%	+3%	+3%	+3%	+3%	+3%	+3%
4 **bkwren:0**	129.2	137.8	139.5	314.4	203.9	—	221.0	171.9	155.2	148.7	145.3	143.2	142.5	142.4	142.4
				+128%	+48%		+60%	+25%	+13%	+8%	+5%	+4%	+3%	+3%	+3%
5 **tkwren:90**	46.7	53.5	55.4	124.8	82.6	70.2	63.4	59.9	57.9	57.0	56.5	56.4	56.4	56.4	56.4
				+133%	+54%	+31%	+19%	+12%	+8%	+7%	+6%	+5%	+5%	+5%	+5%
6 **dstop1:0**	41.2	47.4	48.2	101.4	65.6	—	72.0	60.3	54.4	51.5	50.5	49.4	48.9	48.8	48.8
				+114%	+38%		+52%	+27%	+15%	+9%	+7%	+4%	+3%	+3%	+3%
7 **unwren:90**	25.2	32.9	33.5	76.5	52.5	—	71.7	47.7	39.9	37.0	35.2	34.6	34.3	34.3	34.3
				+133%	+60%		+118%	+45%	+21%	+12%	+7%	+5%	+4%	+4%	+4%
8 **hlatch:90**	150.7	158.5	160.1	262.1	188.1	—	—	—	206.8	179.8	170.3	166.5	163.9	161.8	161.0
				+65%	+19%				+30%	+13%	+7%	+5%	+3%	+2%	+2%
9 **grille:0**	20.0	24.8	24.7	54.0	41.5	—	51.0	37.4	31.3	28.0	26.6	25.6	25.2	25.2	25.2
				+119%	+68%		+106%	+51%	+27%	+13%	+8%	+4%	+2%	+2%	+2%
10 **nkhang:90**	123.9	138.3	141.2	331.0	220.3	—	—	210.3	172.0	158.1	151.3	147.3	145.1	144.1	144.1
				+139%	+59%			+52%	+24%	+14%	+9%	+7%	+5%	+4%	+4%
11 **cthang:90**	112.8	126.1	125.5	275.6	188.5	—	—	—	176.7	148.6	138.7	133.1	130.7	128.4	128.3
				+120%	+50%				+41%	+18%	+11%	+6%	+4%	+2%	+2%
12 **flange:0**	158.1	168.4	171.8	329.2	228.8	—	—	—	230.6	200.5	187.7	181.2	177.1	173.6	172.9
				+95%	+36%				+37%	+19%	+11%	+8%	+5%	+3%	+3%
13 **runleg:90**	150.1	165.8	170.2	446.7	295.9	—	—	—	—	—	266.9	214.9	195.5	179.5	175.7
				+169%	+78%						+61%	+30%	+18%	+8%	+6%

Source: Own Authorship.

The fabrication and cooling times of all tool-paths produced by HotFill, RP3, and Slic3r were computed by the same Python program from the data extracted from the resulting g-code files, according to the formulas in Appendix A with the following parameters: acceleration and deceleration, $3000\,\mathrm{mm/s^2}$; maximum nozzle speed during extrusion, $40\,\mathrm{mm/s}$; maximum nozzle travel speed during jumps, $130\,\mathrm{mm/s}$; Jump start/end penalty, $0.05\,\mathrm{s}$. These parameter values were inferred from the actual fab-times of other tool-paths on the 3D Cloner model DH+ printer (3DCLONER, 2021).

The dependency of $T_{\mathsf{FAB}}(H)$ on the cooling limit Δ, for each of the test parts, is plotted on Figure 22. In order to magnify the variations and to make the plots of different datasets more comparable, the vertical axis is the percent increase in the *connection time* $T_{\mathsf{CONN}} = T_{\mathsf{FAB}} - T_{\mathsf{RAST}}$, the air time plus fab-time of links, compared to that of the reference tool-path: namely, $R^{\mathsf{HF}}_{\mathsf{CONN}} = T^{\mathsf{HF}}_{\mathsf{CONN}} / T^{\mathsf{REF}}_{\mathsf{CONN}}$, where $T^{\mathsf{HF}}_{\mathsf{CONN}}$ is the connection time of the HotFill path, and $T^{\mathsf{REF}}_{\mathsf{CONN}}$ is the same value for the reference tool-path (by RP3 or Slic3r, as in Table 3). Thus a value of 100% means that the Hotfill path is as efficient as the reference path, while a value of 200% means that it spends twice as much time with links and jumps.

It must be noted that the relative *total* fab-time $T^{\mathsf{HF}}_{\mathsf{FAB}} / T^{\mathsf{REF}}_{\mathsf{FAB}}$ are usually much smaller than $R^{\mathsf{HF}}_{\mathsf{CONN}}$. For example, according to Table 3, for $\Delta = 8\,\mathrm{s}$ and $\mu = 20$, the HotFill tool-path of that dataset has $T^{\mathsf{HF}}_{\mathsf{CONN}} = 266.9 - 150.1 = 116.1\,\mathrm{s}$, which is about 7 times the connection time of the reference tool-path $T^{\mathsf{REF}}_{\mathsf{CONN}} = T^{\mathsf{SLIC3R}}_{\mathsf{CONN}} = 165.8 - 150.1 = 15.7\,\mathrm{s}$; that is, $R^{\mathsf{HF}}_{\mathsf{CONN}} \approx 700\%$. However, the ratio of the total fab-times is only $266.9/165.8 = 161\%$ ($+61\%$ in Table 3).

Figure 22 also shows the dependency of fab-time on the maximum band width μ. The lower left plot of that figure is for $\mu = 20$, the value used in Table 3.

In Table 3, as in Table 2, we observe that HotFill produces a solution whenever the specified cooling time limit Δ is higher than the max cooling time $T^{\mathsf{MAX}}_{\mathsf{COOL}}$ of the SCN path (given in Table 2).

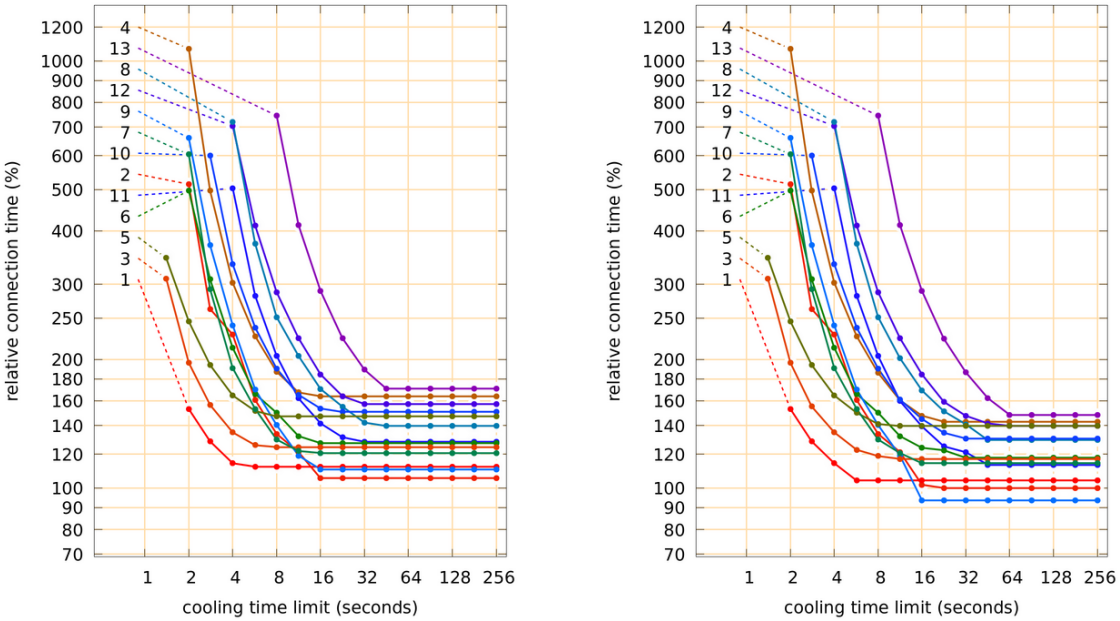
For small values of Δ, just above the minimum value, the fab-times are much higher than those of RP3 and Slic3R, but generally smaller than those of SCN. Likewise, as soon as Δ is greater than the $T^{\mathsf{MAX}}_{\mathsf{COOL}}$ of the alternating scan-line path (SCA), HotFill finds a path that has better fab-time than that of that path.

When the cooling constraints are not too restrictive (say, $\Delta \geq 8\,\mathrm{s}$ in these tests), the paths generated by HotFill, which satisfy these constraints, generally have fab-times that are less than 15% greater than those produced by RP3 or Slic3r (which often grossly violate the constraints). This trend is easier to see in Figure 22. In these tests, the only exception for $\Delta = 8\,\mathrm{s}$ was the **`runleg:90`** dataset (number 13, purple in the figure) for which the HotFill path had 61% higher fab-time ($266.9\,\mathrm{s}$) than Slic3r ($165.8\,\mathrm{s}$). For $\Delta = 16\,\mathrm{s}$, this difference dropped to 18%, while it was 5% or less for all the other datasets.

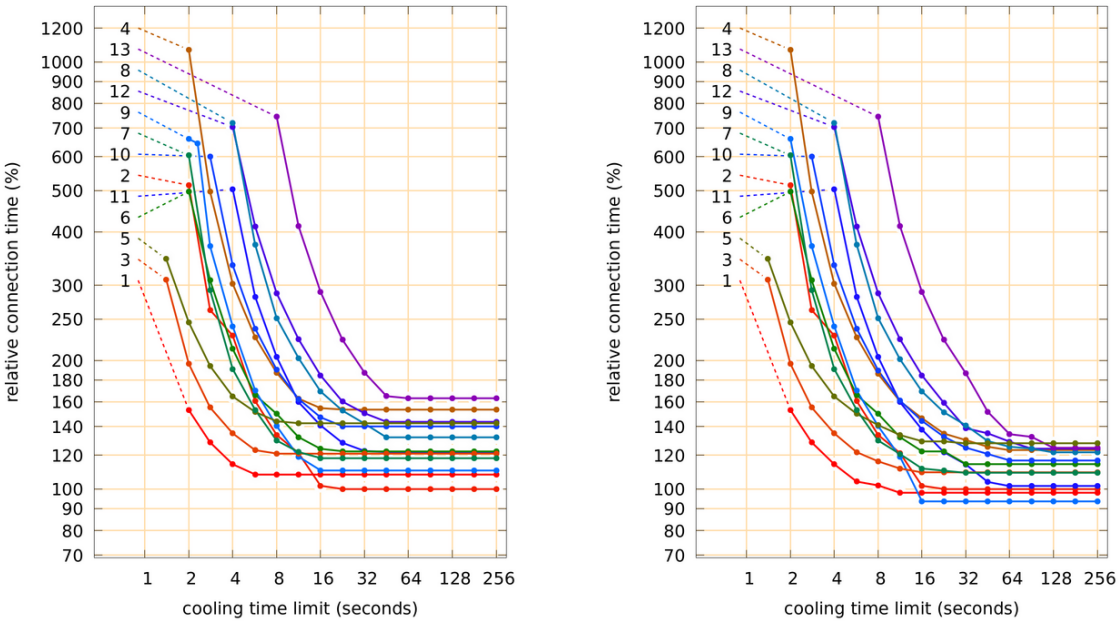
As seen in Figure 22, increasing the maximum band width μ yields only very small improvements in fab-time, and only for the larger values of Δ. For $\mu = 80$ (bottom right graph) and $\Delta \geq 128\,\mathrm{s}$, the HotFill paths had less than 30% higher connection time than the reference

Figure 22 – Relative connection time $R^{\text{HF}}_{\text{CONN}}$ as a function of the cooling time limit Δ, for different values of the maximum band width μ. Each curve is a different dataset. The numbers refer to Table 1.

(a) $\mu = 15$ **(b)** $\mu = 25$



(c) $\mu = 20$ **(d)** $\mu = 80$



Source: Own Authorship.

path. However, the running time of HotFill increased steeply as μ increased beyond 10. See Section 4.4. On the other hand, tests with smaller values of μ yielded paths with significantly larger fab-times, with not much gain in computing time. We concluded that the value $\mu = 20$ was a convenient compromise for these input files.

4.4 Analysis of the computation times

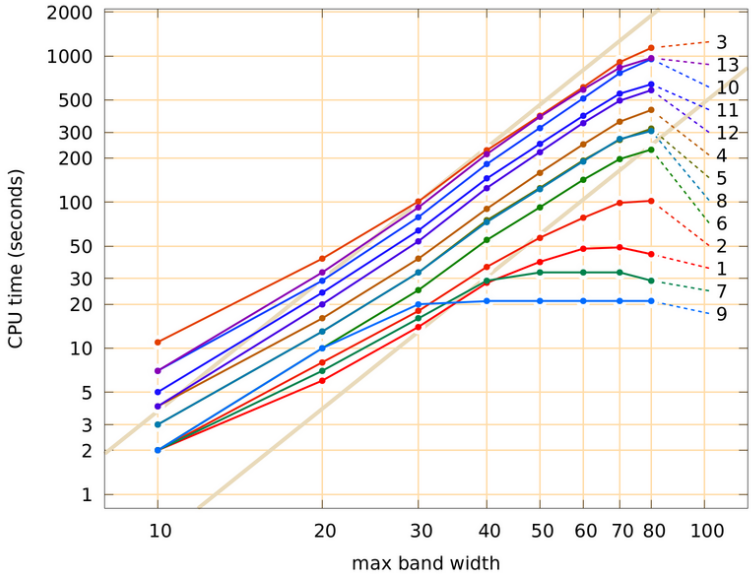
The tests in this section aim to determine the relation between the total computation time T_{CPU} of HotFill and the three most obvious parameters: the number of rasters $\#\mathcal{R}$, the maximum band width μ, and the cooling time limit Δ. The results are shown in Table 4 and Figure 23.

Table 4 – Computation times T_{CPU} of the HotFill algorithm for $\Delta = 8.0\,\text{s}$ and various values of the maximum band width μ. All times are in seconds. Columns n and m represent the number of raster lines and the number of scan-lines, respectively.

	Dataset	n	s	μ 10	20	30	40	50	60	70	80
1	**dstop2:90**	134	65	1	6	14	24	33	39	40	41
2	**adfoot:0**	165	87	2	7	15	27	42	59	74	85
3	**runleg:0**	851	579	9	40	99	190	327	515	744	1031
4	**bkwren:0**	355	209	4	16	38	75	131	200	293	399
5	**tkwren:90**	281	146	3	13	34	65	107	161	226	297
6	**dstop1:0**	208	130	2	9	23	44	73	119	159	203
7	**unwren:90**	138	49	2	7	14	22	25	25	25	25
8	**hlatch:90**	252	131	3	11	28	56	94	144	205	271
9	**grille:0**	160	35	2	10	19	21	20	20	20	20
10	**nkhang:90**	554	248	6	28	72	148	260	423	627	887
11	**cthang:90**	398	160	5	22	59	122	206	316	454	603
12	**flange:0**	335	146	4	18	47	97	168	258	373	503
13	**runleg:90**	443	135	6	29	81	168	305	458	673	896

Source: Own Authorship.

Figure 23 – Computing time T_{CPU} as a function of the maximum band width μ, for $\Delta = 64.0\,\text{s}$ and the datasets listed in Table 4. Each graph is a different dataset, with colors assigned arbitrarily. The broad tan lines are the plots of $A\mu^3$ for $A = 0.00370$ and $A = 0.000481$.



Source: Own Authorship.

The algorithms were implemented in Python3 (Nakonetchnei, 2021). The times were measured on an Intel® Core™ i7-10700 PC with a $2.9\,\text{GHz}$ clock, $256\,\text{kiB}$:$2\,\text{MiB}$:$16\,\text{MiB}$

L1:L2:L3 cache, $128\,\mathrm{GiB}$ of RAM, under the Linux (Ubuntu 20.04.2 LTS) operating system. The program was run in single-thread mode, without any GPU acceleration.

As observed in Section 3.5.3, the running time of HotFill should be dominated by a term that grows proportionally to $s\mu^3$ or $n\mu^3$ — as long as Δ is large enough, there is a small limit to the number of rasters per scan-line, and $\mu \ll s$; where s is the number of scan-lines, and $n = \#\mathcal{R}$ is the number of raster elements.

As shown in Figure 23, the measured computing times in Table 4 generally grow in proportion to μ^3 for μ between 30 and 40. For values of μ below 30, growth is slower because the terms of lower order are still significant. At the other end of the range, growth slows down again when μ approaches or exceeds s, as predicted by the analysis. This effect is evident for datasets that have fewer than 80 scan-lines, like **`unwren:90`** (graph 7, $s = 49$) and **`grille:0`** (graph 9, $s = 35$).

Figure 23 also indicates that, for any fixed μ, the computation time increases with increasing n and s, as expected. The largest times are seen for **`runleg:0`** (graph 3, $n = 851$, $s = 579$), **`runleg:90`** (graph 13, $n = 443$, $s = 135$), and **`cthang:90`** (graph 10, $n = 398$, $s = 160$).

5 CONCLUSIONS AND FUTURE WORK

We described HotFill, an algorithm to plan a tool-path for solid (airgap 0) raster infill for thermoplastic material extrusion 3D printers. Unlike the paths produced by commonly used path planning programs like Slic3r and RP3, the HotFill solution satisfies prescribed limits on the cooling time between deposition of adjacent raster lines. These cooling time constraints are expected to improve the mechanical strength of the object, as verified experimentally by previous research (SUN *et al.*, 2008; COSTA; DUARTE; COVAS, 2017; AKHOUNDI; BEHRAVESH, 2018; FERRARIS; ZHANG; Van Hooreweder, 2019; VOLPATO; ZANOTTO, 2019). Notwithstanding, experimental tests will be important to confirm and analyze any practical issues that might arise from the generated tool path.

The tool-paths considered by HotFill include the plain scan-line-order path (SCN), which generally has the minimum contact cooling times among all paths with a prescribed set of raster lines. Therefore, HotFill will generally find a valid solution, if one exists for the specified cooling constraints. However, even for the most stringent constraints, the path selected by HotFill usually has smaller fabrication time than the scan-line solution. Tests with a dozen non-trivial slices showed that, for reasonable cooling limits, the fab-time of the tool-path produced by HotFill is not much higher than that of the paths computed by Slic3r, RP3, and similar programs. As the cooling constraints are relaxed, the fab-time of the HotFill solution usually drops to only a few percent more than that of Slic3r.

The computation time of HotFill depends on the size (number of rasters and scan-lines) of the problem. A full search for the minimum fab-time can be expensive; however, the algorithm has a parameter μ that lets the user control the depth of the search, thus reducing the computation time for a modest increase in fabrication time. Anyway, the implementation can be improved in many ways (e. g., by replacing Python3 by a more efficient programming language) that should considerably reduce the computation time to a few seconds, even for large instances.

5.0.1 Future Work

There are still several parts of the algorithm that could be improved to reduce the computation time and/or the fabrication time of the resulting tool-paths, such as developing a more efficient and/or effective BandPath procedure. Also, one may consider using a more flexible partition of the rasters into bands, by using “topological” cut-lines instead of straight horizontal ones (EDELSBRUNNER; GUIBAS, 1989).

The bidirectional greedy BandPath heuristic that we used (see Section 3.3) has cost proportional to the square of the number $n_{i,j}$ of rasters in the band. One can surely find other heuristics that are much faster, but still likely to yield valid bandpaths with low enough fab-time.

BIBLIOGRAPHY

3DCLONER. *3DCloner DH Plus*. 2021. Code repository at http://3dcloner.ind.br/upload/20200724170856r7ginb.pdf. Accessed on 2021-12-13.

AGARWALA, M. K. *et al.* Structural quality of parts processed by fused deposition. **Rapid Prototyping Journal**, v. 2, n. 4, p. 4–19, 1996.

AKHOUNDI, B.; BEHRAVESH, A. H. Effect of filling pattern on the tensile and flexural mechanical properties of FDM 3D printed products. **Experimental Mechanics**, v. 59, p. 883–897, 1 2018.

BELLMAN, R. E. **Dynamic Programming**. Mineola, N.Y: Courier Dover Publications, 1957. ISBN 0486428095.

BERMAN, B. 3-D Printing: The New Industrial Revolution. **Business Horizons**, v. 55, n. 2, p. 155 – 162, 2012. ISSN 0007-6813. Disponível em: http://www.sciencedirect.com/science/article/pii/S0007681311001790.

BIKAS, H.; STAVROPOULOS, P.; CHRYSSOLOURIS, G. Additive Manufacturing Methods and Modelling Approaches: A Critical Review. **The International Journal of Advanced Manufacturing Technology**, v. 83, n. 1, p. 389–405, 2016. ISSN 1433-3015. Disponível em: https://doi.org/10.1007/s00170-015-7576-2.

CHEN, Y. *et al.* A full migration BBO algorithm with enhanced population quality bounds for multimodal biomedical image registration. **Applied Soft Computing**, v. 93, p. 106335, 2020. ISSN 1568-4946.

Cheng, C.-B.; Mao, C.-P. A modified ant colony system for solving the travelling salesman problem with time windows. **Mathematical and Computer Modelling**, v. 46, n. 9, p. 1225–1235, 2007.

COSTA, S. F.; DUARTE, F. M.; COVAS, J. A. Estimation of filament temperature and adhesion development in fused deposition techniques. **Journal of Materials Processing Technology**, Elsevier, v. 245, p. 167–179, 2017.

CROES, A. A method for solving traveling salesman problems. **Operations Research**, v. 5, p. 791–812, 1958.

DUMAS, Y. *et al.* An optimal algorithm for the traveling salesman problem with time windows. **Operations research**, v. 43, n. 2, p. 367–371, 1995.

EDELSBRUNNER, H.; GUIBAS, L. J. Topologically sweeping an arrangement. **Journal of Computer and System Sciences**, v. 38, n. 1, p. 165–194, 1989.

FAES, M.; FERRARIS, E.; MOENS, D. Influence of inter-layer cooling time on the quasi-static properties of ABS components produced via fused deposition modelling. **Journal Procedia CIRP**, Elsevier, v. 42, p. 748–753, 2016.

FERRARIS, E.; ZHANG, J.; Van Hooreweder, B. Thermography based in-process monitoring of fused filament fabrication of polymeric parts. **CIRP Annals**, CIRP, v. 68, n. 1, p. 213–216, 2019.

FLOOD, M. M. The traveling-salesman problem. **Operations Research**, v. 4, n. 1, p. 61–75, 1956.

Fok, K. *et al.* An ACO-based tool-path optimizer for 3-D printing applications. **IEEE Transactions on Industrial Informatics**, v. 15, n. 4, p. 2277–2287, 2019.

Fok, K. *et al.* A 3D printing path optimizer based on Christofides algorithm. In: **2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE)**. [*S.l.*: *s.n.*], 2016. p. 1–2.

GANGANATH, N. *et al.* Trajectory planning for 3D printing: A revisit to traveling salesman problem. In: **2016 (2nd) International Conference on Control, Automation and Robotics (ICCAR)**. [*S.l.*: *s.n.*], 2016. p. 287–290.

GIBSON, I.; ROSEN, D.; STUCKER, B. Additive manufacturing technologies - 3D printing, rapid prototyping, and direct digital manufacturing. In: **Rapid Manufacturing Association**. Second. [*S.l.*]: Springer, 2015. p. 510. ISBN 9781493921126.

GUO, N.; LEU, M. C. Additive Manufacturing: Technology, Applications and Research Needs. **Frontiers of Mechanical Engineering**, v. 8, n. 3, p. 215–243, 2013. ISSN 2095-0241. Disponível em: https://doi.org/10.1007/s11465-013-0248-8.

GURRALA, P. K.; REGALLA, S. P. Part strength evolution with bonding between filaments in fused deposition modelling. **Virtual and Physical Prototyping**, v. 9, n. 3, p. 141–149, 2014.

Hodgson, G.; Ranellucci, A.; Moe, J. **Slic3r Manual**. 2021. Online document at https://manual.slic3r.org/. Acessed on 2021-07-24.

Jin, Y.-A. *et al.* Optimization of tool-path generation for material extrusion-based additive manufacturing technology. **Additive Manufacturing**, v. 1-4, p. 32–47, 2014.

Knuth, D. E. **The Art of Computer Programming**. 3. ed. Boston: Addison-Wesley, 1998. v. 1. 780 p. ISBN 978-0201896831.

KOCH, C.; Van Hulle, L.; RUDOLPH, N. Investigation of mechanical anisotropy of the fused filament fabrication process via customized tool path generation. **Additive Manufacturing**, v. 16, p. 138–145, 2017. ISSN 2214-8604.

Komineas, G. *et al.* Build time estimation models in thermal extrusion additive manufacturing processes. **Procedia Manufacturing Journal**, v. 21, p. 647 – 654, 2018.

KULKARNI, P.; MARSAN, A.; DUTTA, D. Review of process planning techniques in layered manufacturing. **Rapid Prototyping Journal**, v. 6, n. 1, p. 18–35, 2000.

LEAL, R. *et al.* Additive Manufacturing Tooling for the Automotive Industry. **The International Journal of Advanced Manufacturing Technology**, v. 92, n. 5, p. 1671–1676, 2017. ISSN 1433-3015. Disponível em: https://doi.org/10.1007/s00170-017-0239-8.

Lepoivre, A. *et al.* Heat transfer and adhesion study for the FFF additive manufacturing process. **Procedia Manufacturing**, v. 47, p. 948–955, 2020. ISSN 2351-9789. 23rd International Conference on Material Forming.

LI, H.; ALIDAEE, B. Tabu search for solving the black-and-white travelling salesman problem. **Journal of the Operational Research Society**, Springer, v. 67, n. 8, p. 1061–1079, 2016.

LI, H. *et al.* MLFS-CCDE: multi-objective large-scale feature selection by cooperative coevolutionary differential evolution. **Memetic Computing**, v. 13, p. 1–18, 2021. ISSN 1865-9292.

LI, X.; LI, W.; HE, F. A multi-granularity nc program optimization approach for energy efficient machining. **Advances in Engineering Software**, v. 115, p. 75–86, 2018. ISSN 0965-9978.

LIANG, Y.; HE, F.; ZENG, X. 3D mesh simplification with feature preservation based on whale optimization algorithm and differential evolution. **Integrated Computer-Aided Engineering**, v. 27, p. 417–435, 2020. ISSN 1875-8835.

LIANG, Y. *et al.* An improved loop subdivision to coordinate the smoothness and the number of faces via multi-objective optimization. **Integrated Computer-Aided Engineering**, v. 29, p. 23–41, 2022. ISSN 1875-8835.

LIM, S. *et al.* Developments in Construction-Scale Additive Manufacturing Processes. **Automation in Construction**, v. 21, p. 262 – 268, 2012. ISSN 0926-5805. Disponível em: http://www.sciencedirect.com/science/article/pii/S0926580511001221.

LIN, S.; KERNIGHAN, B. W. An effective heuristic algorithm for the traveling-salesman problem. **Operations Research**, v. 21, n. 2, p. 498–516, abr. 1973.

LIU, W. *et al.* Toolpath planning for additive manufacturing using sliced model decomposition and metaheuristic algorithms. **Advances in Engineering Software**, v. 149, p. 102906, 2020. ISSN 0965-9978.

LóPEZ-IBáñEZ, M.; BLUM, C. Beam-ACO for the travelling salesman problem with time windows. **Computers & Operations Research**, v. 37, n. 9, p. 1570 – 1583, 2010. ISSN 0305-0548.

MAKERBOT. **Thingiverse - Digital Designs for Physical Objects**. 2021. 3D model repository at www.thingiverse.com/. Accessed on 2021-07-24.

MIAO, H.; TIAN, Y.-C. Dynamic robot path planning using an enhanced simulated annealing approach. **Applied Mathematics and Computation**, Elsevier, v. 222, p. 420–437, 2013.

MINETTO, R. *et al.* An optimal algorithm for 3D triangle mesh slicing. **Computer-Aided Design**, v. 92, p. 1–10, nov. 2017.

MLADENOVIĆ, N.; TODOSIJEVIĆ, R.; UROŠEVIĆ, D. An efficient general variable neighborhood search for large travelling salesman problem with time windows. **Yugoslav Journal of Operations Research**, University of Belgrade, v. 23, n. 1, p. 19–30, 2013.

MORRISON, D. R. *et al.* Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. **Discrete Optimization**, v. 19, p. 79–102, 2016. ISSN 1572-5286.

Nakonetchnei, E. C. **Hotfill**. 2021. Code repository at https://github.com/ecassiana/hotfill. Accessed on 2021-07-28.

NAKONETCHNEI, E. C. *et al.* Hotfill: A Cooling Time Constrained Raster-Fill Planning Algorithm for Extrusion 3D Printing. **Advances in Engineering Software, Elsevier. Under revision**, 2022.

NGO, T. D. *et al.* Additive Manufacturing (3D Printing): A Review of Materials, Methods, Applications and Challenges. **Composites Part B: Engineering**, v. 143, p. 172 – 196, 2018. ISSN 1359-8368. Disponível em: http://www.sciencedirect.com/science/article/pii/S1359836817342944.

PEARL, J. **Heuristics: Intelligent Search Strategies for Computer Problem Solving**. USA: Addison-Wesley Longman Publishing Co., Inc., 1984. ISBN 0201055945.

PREPARATA, F. P.; SHAMOS, M. I. **Computational Geometry – An Introduction**. New York, NY: Springer, 1985. ISBN 0-387-96131-3.

RESENDE, M. G.; RIBEIRO, C. C. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In: **Handbook of metaheuristics**. [*S.l.*]: Springer, 2010. p. 283–319.

ROSENKRANTZ, D. J.; STEARNS, R. E.; LEWIS II, P. M. An analysis of several heuristics for the traveling salesman problem. **SIAM Journal on Computing**, v. 6, n. 3, p. 563–581, 1977.

ROTHLAUF, F. **Design of modern heuristics: principles and application**. [*S.l.*]: Springer, 2011. v. 8.

SAVELSBERGH, M. Local search in routing problems with time windows. **Annals of Operations research**, Springer, v. 4, p. 285–305, 1985.

Schiller, G. J. Additive Manufacturing for Aerospace. In: **2015 IEEE Aerospace Conference**. [*S.l.*: *s.n.*], 2015. p. 1–8. ISSN 1095-323X.

SINGH, S.; RAMAKRISHNA, S. Biomedical Applications of Additive Manufacturing: Present and Future. **Current Opinion in Biomedical Engineering**, v. 2, p. 105 – 115, 2017. ISSN 2468-4511. Additive Manufacturing. Disponível em: http://www.sciencedirect.com/science/article/pii/S2468451117300296.

SUN, Q. *et al.* Effect of processing conditions on the bonding quality of FDM polymer filaments. **Rapid Prototyping Journal**, v. 14, n. 2, p. 72–80, 2008.

Alpha__Wolf, E. Farkas. **4-Hook Coat Hanger**. 2018. 3D model at www.thingiverse.com/thing:3315713. Accessed on 2021-07-24.

ecassiana, E. Cassiana. **Flange**. 2021. 3D model at https://www.thingiverse.com/thing:4916943. Accessed on 2021-07-28.

ecassiana, E. Cassiana. **Grille**. 2021. 3D model at https://www.thingiverse.com/thing:5161470. Accessed on 2021-12-11.

ecassiana, E. Cassiana. **Hook Latch**. 2021. 3D model at https://www.thingiverse.com/thing:4919047. Accessed on 2021-07-28.

ecassiana, E. Cassiana. **Necktie hanger**. 2021. 3D model at https://www.thingiverse.com/thing:4916952. Accessed on 2021-07-28.

ffleurey Frank. **Adjustable Foot for Tables and Furnitures**. 2013. 3D model at www.thingiverse.com/thing:148764. Accessed on 2021-07-24.

jephil08, J. Phillips. **1-Inch Bulkhead Hand Wrench**. 2020. 3D model at www.thingiverse.com/thing:4356560. Accessed on 2021-07-24.

joraasen, A. Jøraasen. **22mm Wrench**. 2016. 3D model at www.thingiverse.com/thing:1909039. Accessed on 2021-07-25.

M600 Anthony L. **GoPro Tool with Universal Wrench**. 2014. 3D model at www.thingiverse.com/thing:601212. Accessed on 2021-07-25.

Matearoa, M. Charpentier. **Door stop**. 2021. 3D model at www.thingiverse.com/thing:4889702. Accessed on 2021-07-24.

nillasarie, N. Illasarie. **Runners MK2 prosthetic leg**. 2016. 3D model at www.thingiverse.com/thing:1960573. Accessed on 2021-07.

VOLPATO, N. **Manufatura Aditiva: Tecnologias e Aplicações da Impressão 3D**. [*S.l.*]: Editora Blucher, 2017.

Volpato, N. **RP3 Basic User Guide (v2021)**. [*S.l.*], 2021.

VOLPATO, N. *et al.* Uma otimização da estratégia de preenchimento do processo fdm. In: ABCM RIO DE JANEIRO. **4º Congresso Brasileiro de Engenharia de Fabricação—COBEF. Águas de São Pedro**. [*S.l.*], 2007.

Volpato, N. *et al.* Combining heuristics for tool-path optimisation in material extrusion additive manufacturing. **Journal of the Operational Research Society**, v. 0, n. 0, p. 1–11, 2019.

VOLPATO, N.; ZANOTTO, T. T. Analysis of deposition sequence in tool-path optimization for low-cost material extrusion additive manufacturing. **International Journal of Advanced Manufacturing Technology**, v. 101, n. 5-8, p. 1855–1863, abr. 2019.

WAH, P. K. *et al.* Tool path optimization in layered manufacturing. **IIE Transactions**, Institute of Industrial Engineers, v. 34, n. 4, p. 335–347, 2002.

XIA, L.; LIN, S.; MA, G. Stress-based tool-path planning methodology for fused filament fabrication. **Additive Manufacturing**, v. 32, p. 101020, 2020. ISSN 2214-8604.

Yang, J. *et al.* Fractal scanning path generation and control system for selective laser sintering (SLS). **International Journal of Machine Tools and Manufacture**, v. 43, n. 3, p. 293–300, 2003.

YANG, W. Optimal path planning in rapid prototyping based on genetic algorithm. In: **Chinese Control and Decision Conference**. [*S.l.*]: Institute of Electrical and Electronics Engineers (IEEE), 2009. p. 5068–5072.

ZHAO, D.; GUO, W. Shape and performance controlled advanced design for additive manufacturing: A review of slicing and path planning. **Journal of Manufacturing Science and Engineering**, v. 142, p. 1–87, 01 2020.

APPENDIX A – Move timing functions

The move fabrication time functions $T_{\mathsf{FAB}}(m)$ and $T_{\mathsf{COV}}(m,u)$ defined in Section A can be estimated as described by (Komineas *et al.*, 2018). The key parameters are the *acceleration* a at the start of the move (which is also the deceleration at the end), and the maximum *cruise speed* v. These parameters depend on the printer and on whether m is a move or a trace.

Let d be the distance to be covered, namely $d = |q - p|$ where $p = p_{\mathsf{INI}}(m)$ and $q = p_{\mathsf{FIN}}(m)$. We assume that the nozzle is stationary at the beginning of every move, and comes to a full stops at the end of it. Therefore, if d is large enough, the nozzle will accelerate for a time $t_{\mathsf{a}} = v/a$, while covering a distance $d_{\mathsf{a}} = at_{\mathsf{a}}^2/2 = vt_{\mathsf{a}}/2 = v^2/(2a)$. At the end, it will decelerate for the same distance and time. In between, it will cruise at constant speed v for a distance $d_{\mathsf{c}} = d - 2d_{\mathsf{a}}$, which will take time $t_{\mathsf{c}} = d_{\mathsf{c}}/v$.

However, if d is less than v^2/a, the nozzle will have to start decelerating before reaching the cruise speed v. Acceleration and deceleration will each last for a distance $d_{\mathsf{a}} = d/2$ and take time $t_{\mathsf{a}} = \sqrt{d/a}$. The distance and time spent at cruise speed will be $d_{\mathsf{c}} = t_{\mathsf{c}} = 0$.

Either way, the total execution time $T_{\mathsf{FAB}}(m)$ will be $2t_{\mathsf{a}} + t_{\mathsf{c}}$.

Depending on the printer and on operator choices, it may be required to raise the nozzle and/or retract a couple mm of the filament at the begging of a jump, and to lower the nozzle and/or reposition the filament at the end. In that case, if m is a jump, the time t_{z} needed to perform these operations at each end of jump must be added to t_{a}.

For the passage time $T_{\mathsf{COV}}(m,q)$, let d_{u} be the distance from p to the point on the mid-line of m that is closest to the point u; namely $d_{\mathsf{u}} = \max\{0, \min\{d, (u-p)\cdot(q-p)/d\}\}$. We have three cases, depending on the part of the move where the passage occurs (accelerating, decelerating or cruising):

$$T_{COV} = \begin{cases} \sqrt{d_{\mathsf{u}}/a}, & \text{if } d_{\mathsf{u}} \le d_{\mathsf{a}} \\ 2t_{\mathsf{a}} + t_{\mathsf{c}} - \sqrt{(d - d_{\mathsf{u}})/a}, & \text{if } d_{\mathsf{u}} \ge d - d_{\mathsf{a}} \\ t_{\mathsf{a}} + (d_{\mathsf{u}} - d_{\mathsf{a}}),/v & \text{otherwise} \end{cases}$$