Highlights:

- Strength of a 3D printed part depends on good bonding between infill rasters.
- Bond strength depends on time lapsed between deposition of adjacent rasters.
- Goal: generate a tool-path with prescribed limits for those cooling times.
- A practical dynamic programming algorithm for this goal, HotFill, is proposed.
- The algorithm is tested on some representative part slices.

⋆[Neri:]*Uma alternativa para o terceiro highlight: [*The problem of tool-path generation with prescribed cooling time limits is tackled.*]* ⋆[Stolfi:]*Alterei o item 3. Acho que "is tackled" não soa bem.*

HotFill: A Cooling Time Constrained Raster-Fill Planning Algorithm for Extrusion Additive Manufacturing

⋆[Stolfi:]*Acho que deveria ser "3D printers" em vez de "additive manufacturing". É mais específico e faz sentido para muito mais gente.*

Elis C. Nakonetchnei[1] Jorge Stolfi[2] Neri Volpato[1]
Ricardo D. da Silva[1] Rodrigo Minetto[1]

[1]*Federal University of Technology - Paraná (UTFPR), Curitiba, Brazil* [2]*State University of Campinas (UNICAMP), São Paulo, Brazil*

Abstract

One of the factors responsible for the mechanical strength of objects manufactured by thermoplastic extrusion 3D printers is the adhesion between adjacent material beads in the same layer. This, in turn, depends on the time lapsed between their deposition. In this paper 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 the algorithm is fast and produces tool-paths that have low fabrication time while respecting the cooling time constraints. The datasets and the HotFill source code are available at GitHub.

⋆[Neri:]*Depois de ler todo o artigo, ainda considero que o que foi apresentado trata só de "limit". Deixei um comentário nas conclusões também sobre isso.*⋆[Stolfi:]*Mas no formalismo, no algoritmo, e no programa tem um limite $\tau(c)$ para cada contato. Tanto assim que o algoritmo trabalha com a razão $R_{COOL}(P,c) = Tcool(P,c)/\tau(c)$ e não com o $T_{COOL}(P,c)$ diretamente. Só nos testes que usamos o mesmo limite $\tau(c) = \Delta$ para todos os contatos, só para não confundir.*

Keywords: Time constrained extrusion ; Tool-path planning ; 3D printing ; Raster filling ; Dynamic programming

[2]Corresponding author: Dr. Rodrigo Minetto, Tel. +55 41 998961525, Fax. +55 41 3310 4998, e-mail: rminetto@utfpr.edu.br.

1 Introduction

Thermoplastic polymer extrusion is the most common technology used in 3D printers [1, 2, 3]. It builds the target object one layer (*slice*) at a time, using a heated nozzle to deposit a thin filament of material following a suitable *tool-path*. ⋆[Stolfi:]*Neri, entendo que sua área é additive manufacturing em geral, mas este artigo não é sobre AM, é exclusivamente sobre thermoplastic extrusion 3D printers. Não é relevante nem para extrusão de outros materiais, como concreto, nem para outros processos térmicos, como fusão de pó a laser ou deposição de metal por arco elétrico. Então não vejo porque precisa mencionar AM, no título ou na introdução...*

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 [4], 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* [2, 5].

Figure 1 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 RP3 [6] or Slic3r [4]. 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 1; in which case the nozzle will have to jump between these sections without extruding any material [7].

Algorithms for tool-path generation typically try to minimize the total fabrication time for each slice [8, 5, 9, 10], which consists of the *extrusion time* plus the non-productive *air time* spent during the jumps [5]. 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 that produce "good enough" tool-paths, but not necessarily optimal ones.

There is a substantial literature of such heuristics. In 2009 Weidong tested a genetic algorithm (GA) meta-heuristic for this purpose [10]. Gnaganath et al. [11] and Fok et al. [12] considered different heuristic algorithms to solve the traveling salesman problem (TSP) and adapted them to

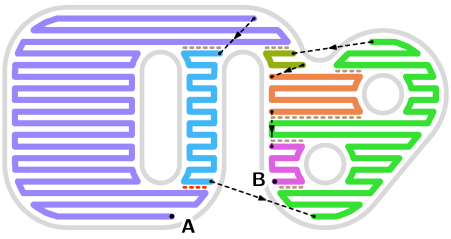

Figure 1. A possible tool-path A–B for the solid filling of a slice of a hypothetical part, whose extruded contour is shown in gray. Each color identifies a separate continuous extrusion section of the path. The black dashed lines denote nozzle movements without extrusion. The part measures $39\,\mathrm{mm}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

the tool-path planning problem. The main heuristics compared were: nearest neighbor selection (NN), Christofides's algorithm, and the k-opt algorithm [13, 14, 15, 16]. In 2019, Fok et al. experimented with the ant colony optimization (ACO) meta-heuristic [17]. Volpato et al. [5] compared the NN heuristic to the nearest insertion heuristic (NI).

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 [18, 19]; and that the orientation of filling rasters affects the quality of raster corners [8, 20, 21].

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 [7, 22, 23, 24, 25]. This effect was noted by several authors, starting with Sun et al. in 2008 [22, 26, 27, 28]. Studies of this effect using numerical models and experimental methods were published by Akhoundi and Behravesh in 2018 [27] and by Volpato and Zanotto in 2019 [25], respectively.

In the example of Figure 1, 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 Section 7.) Even if one followed the common practice of rotating the direction of the filling rasters by 90 degrees between successive layers [2], 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. ⋆[Stolfi:]*Acho que este parágrafo não deve ser misturado com a revisão bibliográfica, pois esta "solução" não é mencionada por nenhum paper. Confere? Acho que deve ficar antes do prior work, ou depois; não no meio*

In 2019, Volpato and Zanotto [25] 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 *hot tool-path problem* (HTPP).

The HTPP is somewhat similar to the traveling salesman problem with time windows (TSP-TW), in which each node must be visited in a specific time interval [29]. This problem was proved to be NP-complete by Savelsbergh [30] in 1985; a property that is believed to imply that there is no practical algorithm so solve it exactly. Approximate solutions for the TSP-TW were described by Cheng et al. [31], using an ant colony meta-heuristic; by Lopez et al. [32], using a combination of ant colony and beam search; and by Mladenovic et al. [33], using a two-stage variable neighborhood search.

However, none of these TSP-TW solutions were applied to the 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.

The strength of a 3D-printed part is affected by many other factors besides the filament-to-filament

bonding in the infill, such as the time elapsed between successive slices, the raster direction within each slice, and the extrusion temperature [24, 2]. These issues will not be considered in this work.

1.1 Contributions

In this paper we describe HotFill, an algorithm that uses dynamic programming to solve the hot tool-path problem for the special case of solid raster filling (RF-HTPP). The algorithm was implemented in Python3 and is available as open source [34]. Experiments show that it is fast and produces tool-paths that have low fabrication time while satisfying the specified cooling time constraints.

1.2 Structure of the paper

The remainder of the paper is organized as follows. In Section 2 we define some basic concepts and the notation used in the rest of the article. In Section 3 we formally define the RF-HTPP. Section 4 has some general considerations about the problem. Sections 5 describes the HotFill algorithm. Section 6 discusses some algorithm improvements and programming tricks that are necessary to achieve a practical computing time. In Section 7 we report tests of the algorithm, with these improvements, on some typical object slices. Our conclusions are in Section 8. In the appendix (Section 9), we detail the assumed printer dynamics model.

1.3 List of symbols

⋆[Stolfi:]*Completar e conferir as seções.*

⋆[Neri:]m *é usado como move e também é utilizado como number scan-lines...*⋆[Stolfi:]*Troquei* m = *número de scanlines por s...* ⋆[Stolfi:]*Eliminei a seção "Example variables" (u, P, c, etc.) pois são todas variáveis "locais". Se quiserem de volta está em* `JUNK-TEXT/0212-symbol-local.tex`.

⋆[Stolfi:]*Neri, É* B_z *mesmo, não* B_n.

Constants, functions and operators

Symbol	Meaning	Sec.
σ	layer (slice) thickness	2.1
$p_{\text{INI}}(m)$, $p_{\text{INI}}(P)$	initial position of a move m or path P	2.1
$p_{\text{FIN}}(m)$, $p_{\text{FIN}}(P)$	final position of a move m or path P	2.1
$\overleftarrow{m}$, $\overleftarrow{P}$	reversal of a move m or path P	2.1
$\lambda(m)$	nominal width of trace m	2.1
$\langle\rangle$	empty path	2.1
Λ	invalid path	2.1
$T_{\text{FAB}}(P)$	estimated fabrication time of path P	2.1
$T_{\text{COOL}}(P, c)$	cooling time of contact c with path P	2.2
$\tau(c)$	cooling time limit of a contact c	2.2
$R_{\text{COOL}}(P, c)$	cooling time ratio $T_{\text{COOL}}(P, c)/\,\tau(c)$ of contact c with path P	2.2
$R^{\text{MAX}}_{\text{COOL}}(P, \mathscr{C})$	maximum cooling time ratio among contacts $\mathscr{C}$ with path P	2.2
RF-HTPP inputs		
$\mathscr{R}$	set of infill raster elements	3.1
$\mathscr{L}(r)$	set of link paths that connect to raster r	3.1
$\mathscr{C}$	set of relevant contacts	3.1
s	number of scan-lines	3.1
$n = \#\mathscr{R}$	number of raster elements	3.1
$\mathbb{P}(\mathscr{R})$	set of all candidate paths using the rasters in $\mathscr{R}$	4.6
$\mathbb{V}(\mathscr{R}, \mathscr{C})$	paths in $\mathbb{P}(\mathscr{R})$ that satisfy the cooling limits of contacts $\mathscr{C}$	4.6
Parameters and variables of the HotFill algorithm		
μ	maximum number of scan-lines in a bandpath	5.3
i, j, k	cut-line or scan-line indices.	5.3
$B_z[i, j]$	valid bandpath for (i, j)-band with general direction z (0 or 1)	5.3
$F_z[i, j]$	best valid (i, j)-fullpath that ends with $B_z[i, j]$	5.3
H	output tool-path of HotFill	5.3
Test parameters		
Δ	cooling time limit assigned to all contacts	7.1
θ	infill raster angle rel. to X axis	7.1
L_{AVG}, L_{MAX}, L_{TOT}	average, maximum, and total raster length.	7.1
T_{RAST}	estimated total raster extrusion time (excl. links and jumps)	7.3
$T^{\text{HF}}_{\text{FAB}}$, $T^{\text{REF}}_{\text{FAB}}$	estimated fab-times of the HotFill and reference paths	7.3
$T^{\text{HF}}_{\text{CONN}}$, $T^{\text{REF}}_{\text{CONN}}$	estimated link+jump times of HotFill and reference paths	7.3
$R^{\text{HF}}_{\text{CONN}}$	ratio $T^{\text{HF}}_{\text{CONN}} \,/\, T^{\text{REF}}_{\text{CONN}}$	7.3
T_{CPU}	computer time in seconds	7.4

2 Basic concepts and notation

2.1 Tool-path model

Moves: In this paper, 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 $p_{\text{INI}}(m)$ and $p_{\text{FIN}}(m)$. The *reversal* of a move m is a move $\overleftarrow{m}$ such that $p_{\text{INI}}(\overleftarrow{m}) = p_{\text{FIN}}(m)$ and $p_{\text{FIN}}(\overleftarrow{m}) = p_{\text{INI}}(m)$. 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 same width $\lambda(m) = \lambda_{\text{FILL}}$, with the mid-lines spaced λ_{FILL} apart.

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 2.

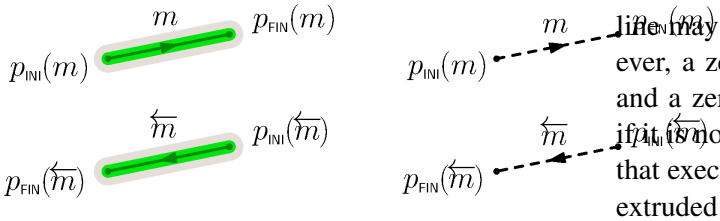

Figure 2. Graphical representation of a trace (left) and a jump (right), showing the effect of reversal on the attributes p_{INI} and p_{FIN}. The dark thin line in the trace is the nozzle's trajectory. The light gray area is the approximate extent of the material deposited by the nozzle. The green "thick segment" is the conventional representation of the trace used in this paper.

Tool-paths: For this paper, 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, $p_{\text{INI}}(P[k]) = p_{\text{FIN}}(P[k-1])$ whenever the two indices are valid. The initial and final points of the path are then $p_{\text{INI}}(P) = p_{\text{INI}}(P[0])$ and $p_{\text{FIN}}(P) = p_{\text{FIN}}(P[n-1])$. See Figure 3 (left).

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 3 (right).

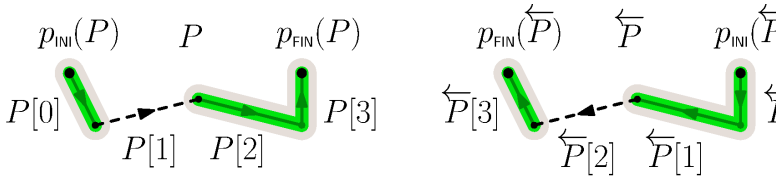

Figure 3. A path P with three traces and a jump (left) and its reversal $\overleftarrow{P}$ (right).

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 $p_{\text{INI}}(P) = p_{\text{FIN}}(P)$ is some arbitrary point. A move m too may have zero length (that is, its mid-line may be a single point $p_{\text{INI}}(m) = p_{\text{FIN}}(m)$. 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 $T_{\text{FAB}}(m)$, 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, $T_{\text{FAB}}(m)$ includes the time needed to extrude the material. For a jump, it includes the time needed to displace the nozzle from point $p_{\text{INI}}(m)$ to

point $p_{\text{FIN}}(m)$ 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, $T_{\text{FAB}}(m)$ 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 Section 9.

For simplicity, we assume that the *total fabrication time* $T_{\text{FAB}}(P)$ needed to perform all traces and jumps of a tool-path P is just the sum of the individual move times; that is, $\sum_{k=0}^{n-1} T_{\text{FAB}}(P[k])$ 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* $T_{\text{INI}}(P, k)$ and the *final time* $T_{\text{FIN}}(P, k)$ when the nozzle will start or finish the move $P[k]$, relative to the starting time of P, are given by

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

$$\begin{aligned} T_{\text{FIN}}(P, k) &= \sum_{i=0}^{k} T_{\text{FAB}}(P[i]) \\ &= T_{\text{INI}}(P, k) + T_{\text{FAB}}(P[k]) \end{aligned} \quad (2)$$

It is convenient also to define $T_{\text{INI}}(P, n) = T_{\text{FAB}}(P)$ and $T_{\text{FIN}}(P, -1) = 0$; then $T_{\text{FIN}}(P, k-1) = T_{\text{INI}}(P, k)$ 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 $p_{\text{FIN}}(T)$ to $p_{\text{INI}}(P)$ 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 $T_{\text{FAB}}(\langle\rangle) = 0$ and $T_{\text{FAB}}(\Lambda) = +\infty$.

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 4. 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.

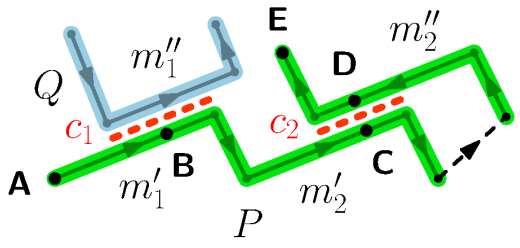

Figure 4. Graphical representation of two contacts c_1 and c_2 (red dashed lines) between traces m_1', m_1'' and m_2', m_2'', of two paths, P (green) and Q (blue).⋆[Neri:]*Por que nesta figura os traces são indicados como m? Como um m pode ser um trace ou um jump, não faz sentido ter um contato com um jump.*⋆[Stolfi:]*Não entendi a objeção. Sempre usamos m para um move, quer seja trace ou jump. Se a legenda diz que são traces, e a figura mostra traces, qual é a confusão?*

We will denote by $\text{sides}(c)$ the pair of traces that surround the contact c. In Figure 4, $\text{sides}(c_1)$ would be the pair of traces $\{m_1', m_1''\}$. For any path P and any set $\mathscr{C}$ of contacts, we will denote by $\text{contacts}(P, \mathscr{C})$ the elements of $\mathscr{C}$ that have at least one side that is a trace P or its reversal.

Cover times: We define $T_{\text{COV}}(m, u)$ 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 Section 9 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* $T_{\text{COV}}(P, c, m)$ *of* that side *by* P as $T_{\text{INI}}(P, k) + T_{\text{COV}}(P[k], u)$, where u is the midpoint of c. Note that the second term takes into account the orientation of m in P. In Figure 4, $T_{\text{COV}}(P, c_1, m'_1)$ is the fab-time of the path P from **A** to **B**, while $T_{\text{COV}}(\overleftarrow{P}, c_1, m'_1)$ 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* $T_{\text{COOL}}(P, c)$ *of* c *in* P as the absolute difference between the two cover times, namely $|T_{\text{COV}}(P, c, m') - T_{\text{COV}}(P, c, m'')|$. In Figure 4, $T_{\text{COOL}}(P, c_2)$ 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 pratical 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 $T_{\text{COOL}}(P, c)$ in the final tool-path P. See Section 3. The *cooling time ratio* $R_{\text{COOL}}(P, c)$ *of* c *in* any path P that closes it is the quotient $T_{\text{COOL}}(P, c)/\tau(c)$; 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 $R_{\text{COOL}}(P, c)$ will always be zero).

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

3 Statement of the problem

We can now specify formally the raster-fill version of the hot tool-path problem (RF-HTPP) that is the topic of this paper.

3.1 Input

Formally, the input for the RF-HTPP consists of a set of n *raster elements* $\mathscr{R}$, a set $\mathscr{C}$ 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. ⋆[Neri:]*Se só pode horizontal, então tem que inserir um comentário que , apesar do usuário poder utilizado qualquer raster angle, o sistema de planejamento de processo pode utilizar a estratégia de aplicar uma rotação no contorno para que as linhas fiquem sempre paralelas. Isso é feito para facilitar os cálculos.* ⋆[Stolfi:]*acrescentei "For simplicity of exposition, we assume that ". Suponho que o leitor vai saber o que fazer se não forem horizontais. Mas, se bem me lembro, o programa não exige isso: recebe vetores unitários que definem os eixos, e/ou toma o ângulo como parâmetro.* 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 5 there are 18 scan-lines. A relevant contact is a contact between distinct rasters that has a finite cooling time limit. ⋆[Neri:]*Sugiro padronizar a maneira que se indica ao leitor para ver determinada figura. Tem indicação no texto sem e com parênteses. Eu prefiro passar tudo para entre parênteses, como estava neste caso! (A indicação da See Figure, estava no final do parágrafo e eu acabei mudando para o final da frase no meio do parágrafo, para chamar a atenção para a definição de scan-line, que considero relevante para entender o método.* ⋆[Stolfi:]*Neri, não entendo porque é necessário padronizar o uso de parênteses quendo se faz referência a figuras. Acho que a decisão tem que ser caso por caso, como para qualquer outro texto. Se achamos que olhar para a figura é importante para a compreensão do texto a seguir, o "veja"* ***não*** *deve ficar entre parênteses. Se "veja a figura" é só para o caso improvável de alguém ter dúvida, pode ser entre parênteses.*

Each raster element r in $\mathscr{R}$ also has a set $\mathscr{L}(r)$ 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 $\mathscr{L}(r)$. There

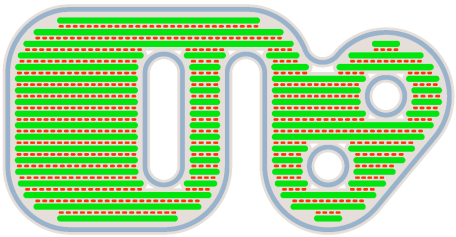

Figure 5. A possible input data set for the RF-HTPP problem, suitable for planning the solid raster filling of the slice shown in Figure 1. There are 56 raster elements (green thick segments) on 18 scan-lines. The total fab-time for those rasters, not counting the time to move between them, is 10.10 seconds. The red lines are the relevant contacts $\mathscr{C}$. The blue-gray lines are the contour of the slice (shown for reference only, not part of the input data).

is a separate set $\mathscr{L}(\overleftarrow{r})$ 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 6. Normally $\mathscr{L}(r)$ and $\mathscr{L}(\overleftarrow{r})$ may have 0, 1, or 2 paths each.

⋆[Neri:]*Sugiro inserir uma frase chamando a atenção para este ponto, pois é um diferencial que os outros métodos não consideram (pelo menos eu desconheço). Que tal algo do tipo: The possibility to analyze up to two alternative links for each* $p_{\text{INI}}(s)$ *or* $p_{\text{FIN}}(s)$ *is innovative in this tool-path generation method because, to our knowledge, it has not been considered before.* ⋆[Stolfi:]*Mas o RP3 e Slic3r consideram isso internamente, não? Tanto assim que nós pegamos estes links do arquivo* `.txt` *que o RPG gera.*

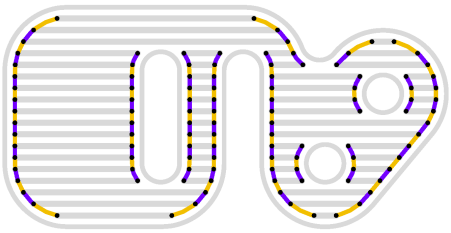

Figure 6. A possible collection of link paths for the input elements of Figure 5.

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

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 5 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 $\mathscr{C}$.

3.2 Output

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

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 $\overleftarrow{s} \in \mathscr{L}(\overleftarrow{r'})$ and $s \in \mathscr{L}(r'')$.

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 Section 9); 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 fill-

ing some gaps between the two that would not be filled by the raster elements alone.

4 General considerations about the problem

4.1 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.) ⋆[Stolfi:]*Como o "Note that" é uma senteça completa e independente do contexto, tem que ser em maiúscula com ponto final, mesmo entre parênteses.* Finding the best order and orientation for the H_i would be a separate problem, that could be addressed by the heuristics mentioned in Section 1.

4.2 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 1. ⋆[Stolfi:]*A path da figura 1 não é greedy; foi escolhida na mão. Talvez devessemos calcular a verdadeira solução greedy e colocar aqui também. Mas não aguento mais programar...*

4.3 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 7. 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. ⋆[Neri:]*Por que não NSC e o outro ASC? Não segue mais a lógica do nome em inglês?* ⋆[Stolfi:]*Pensamos nisso, mas teria que mudar as tabelas, e deu uma preguiiiça...* ⋆[Neri:]*É correto dizer que esta solução não utiliza a lista de raster links? Se sim, talvez fosse bom inserir uma frase sobre isso!* ⋆[Stolfi:]*Na verdade, do jeito que programamos, pode usar links em alguns casos, porque é automático quando a path é montada com os rasters na ordem escolhida. Por exemplo, se uma scan-line termina no mesmo X que a scanline seguinte começa, a solução SCN, pelas definições, pode usar um link nesse ponto, em vez de um jump. Portanto prefiro nem mencionar esta questão. Do jeito que está, está correto, mesmo que possa ter umas surpresas como essa...*

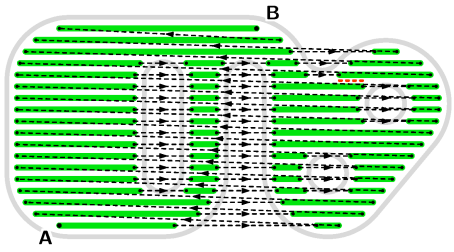

Figure 7. A non-alternating scan-line-order tool-path for the solid raster fill data of Figures 5 and 6. Its fab-time is 23.71 s, including 10.10 s of extrusion and 13.61 s of air time. The red dashed line is the contact with the largest cooling time, 1.67 s.

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 coor-

dinate, but reverses the order of rasters and the direction of extrusion from one scan-line to the next. See Figure 8.

Figure 8. An alternating scan-line-order tool-path for the solid raster fill data of Figures 5 and 6. Its fab-time is 17.95 s, including 11.02 s of extrusion, and 6.93 s of air time. The red dashed line at upper right is the contact with largest cooling time, 2.25 s.

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 7 to Figure 8, the maximum contact cooling time increased from 1.67 to 2.25 s.

4.4 Monotonic infill regions

The RF-HTPP is usually trivial if the input set $\mathscr{R}$ 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 [35].

Therefore, if the raster set $\mathscr{R}$ is monotonic, it is almost never necessary to use the HotFill algorithm described in Section 5. 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 9: if the cooling time limits $\tau(c)$ of all contacts were set to 0.41 s, the SCN solution (at left) would not be valid, but the SCA solution (at right) would be.

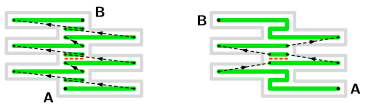

Figure 9. A monotonic instance of the RF-HTPP for which the maximum contact cooling time of the SCN solution at left (0.42 s) is higher than that of the SCA solution at right (0.40 s).

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 10.

Figure 10. A monotonic instance of the RF-HTPP for which the fabrication time of the SCA solution at left (2.38 s) is significantly higher than that of the tool-path at right (2.03 s), which does not follow the scan-line order. The maximum contact cooling times are 0.40 s and 0.73 s, respectively.

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 proaobly be adequate.

4.5 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 $\mathscr{R}'$, $\mathscr{R}^*$, and $\mathscr{R}''$; where $\mathscr{R}'$ consists of r' and all rasters below it, $\mathscr{R}''$ is r'' and all the rasters above it, and $\mathscr{R}^*$ is all the rasters between r' and r'', including both. Let P' and P'' be RF-HTPP solutions for $\mathscr{R}'$ and $\mathscr{R}''$, respectively, and P^* be the greedy solution for $\mathscr{R}^*$ starting at r'. If these partial solutions exist, a solution for the original input set $\mathscr{R}$ 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 Section 5 (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.

4.6 Exhaustive enumeration

In theory, the 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 $\mathscr{R}$; with any gap filled by the appropriate link if it exists, or by a jump otherwise. We denote by $\mathbb{P}(\mathscr{R})$ 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 $\mathbb{P}(\mathscr{R})$ that is valid, that is, whose maximum relative cooling time $R^{\text{MAX}}_{\text{COOL}}(H, \mathscr{C})$ does not exceed 1. We will denote the set of valid candidate paths by $\mathbb{V}(\mathscr{R}, \mathscr{C})$.

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

5 HotFill: Finding a valid tool-path

The HotFill algorithm is specialized to find a valid tool-path for solid raster filling. As observed in Section 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.

5.1 Cut-lines and bands

Cut-lines: We define a *cut-line* as a horizontal line that runs between successive scan-lines. If the rasters of $\mathscr{R}$ 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 11.) For any i, j with $0 \le i < j \le s$, we define the (i, j)-*band* $\mathscr{R}_{i,j}$ as the set of rasters between cut-lines ℓ_i and ℓ_j.

5.2 Band-paths and full-paths

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 $\mathscr{R}_{i,j}$ – 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 11. 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.

⋆[Neri:]*Não seria band-path, para manter o padrão?*⋆[Stolfi:]*Podemos discutir isto. Um problema é que se for "(i, j)-band-path" ficam dois hífens; acho estranho. Outro problema é que "full-path" pode confundir com a path completa final H. Os dois termos são bem especializados e são usados só no algoritmo HotFill; não seriam usados em*

outros algoritmos, mesmo usando a mesma notação para moves, paths, etc. Poderia ser outra palavra em vez de "full", mas se não começar com "F" vai dar um trabalhão mudar as figuras...

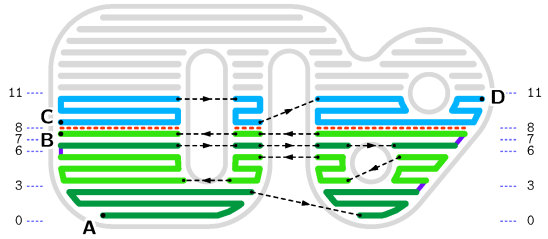

Figure 11. Illustration of the partial state at some point during the execution of the HotFill procedure (Algorithm 1) for the rasters and links of Figure 5 and 6. The colored paths comprise an $(8, 11)$-fullpath, specifically $F_0[8, 11]$, consisting of a $(7, 8)$-fullpath $F_1[7, 8]$ = **A**–**B** (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.

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.

5.3 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 $T_{\text{FAB}}(\langle\rangle)$ be zero and $T_{\text{FAB}}(\Lambda)$ be $+\infty$.

The HotFill heuristic (Algorithm 1) uses the *dynamic programming* approach [36] 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 $\mathscr{R}$ of horizontal rasters on s distinct scan-lines, with the associated links; a set $\mathscr{C}$ of relevant contacts between them; and a band height limit μ.

output: A valid fullpath H that uses all the rasters $\mathscr{R}$ 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
7      end
8  end
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
             H ← F_z[i, s] ;
13     end
14 end
15 return H
```

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

Algorithm 2: MinFullPath

input : A set $\mathscr{R}$ of horizontal rasters on s distinct scan-lines, with the associated link paths; a set $\mathscr{C}$ 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 $\mathscr{R}$ 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.

```
F_* ← Λ
if B ≠ Λ then
    if i = 0 then
        F ← B
    else
        for k in 0, 1, ..., i − 1 with i − k ≤ μ do
            for z in {0, 1} do
                if F_z[k, i] ≠ Λ then
                    T ← Concat(F_z[k, i], B)
                    if ValidPath(T, 𝒞) then
                        if T_FAB(T) < T_FAB(F_*)
                          then F_* ← T ;
                    end
                end
            end
        end
    end
end
return F_*
```

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 $B_1[i,j]$.

5.4 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 12 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 5 and 6. Light gray is used for entries $F_0[i',j']$ that will never be set nor used because $i' \ge j'$ 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.

The MinFullPath algorithm also uses the subroutine ValidPath$(P, \mathscr{C})$ that checks whether the cooling constraints of the contacts $\mathscr{C}$ 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,

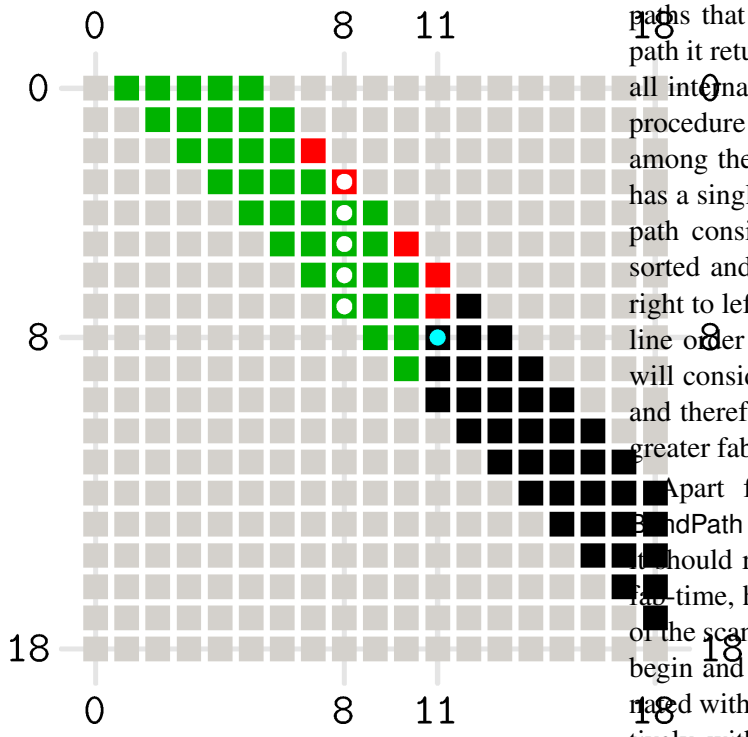

Figure 12. Typical status of the array F_0 upon entry to MinFullPath with parameters $\mu = 5$, row index $i = 8$, and column index $j = 11$, with the input data of Figures 5 and 6. For the color codes, see the text.

it should return Λ. If $p_{\text{FIN}}(P') \neq p_{\text{INI}}(P'')$, 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 $\overleftarrow{S} \in \mathscr{L}(\overleftarrow{P'})$ and $S \in \mathscr{L}(P'')$.

5.5 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 subroutine BandPath$(\mathscr{R}, \mathscr{C}, i, j, z)$. Its goal is to assemble the band rasters $\mathscr{R}_{i,j}$ 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 practically unsolvable problem. Therefore, BandPath must be some heuristic that considers only some small subset of all possible paths that can be built from the rasters $\mathscr{R}_{i,j}$. The path it returns must satisfy the cooling constraints of all internal contacts (between rasters of $\mathscr{R}_{i,j}$). 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 article, 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 rightmost 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 $\mathscr{R}_{i,j}$ or their reversals, so that $p_{\text{INI}}(r')$ is closest to $p_{\text{FIN}}(P')$. It then selects a raster r'', distinct from r', such that $p_{\text{FIN}}(r'')$ is closest to $p_{\text{INI}}(P'')$. 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 $\mathscr{R}_{i,j}$, 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.

5.6 Sample solutions

⋆[Neri:]*Não daria para considerar este caso como um primeiro resultado (preliminar)? Sendo uma seção inicial dos resultados? Poderia ser algo do tipo: alguns resultados com a camada da geometria hipotética.* ⋆[Stolfi:] *Acho importante colocar neste momento exemplos de saída para um slice bem pequeno, para que o leitor entenda o comportamento do algoritmo. Se movesse para a seção de testes, teria que ser uma subseção completamente separada e diferente das outras três. Acho que ficaria mais confuso...*

For illustration purposes, we show below the outputs of HotFill for the input data of Figure 5 and 6, with the cooling time limits set to different values. More realistic examples are given in Section 7.

Figures 13, 14, and 15 show the HotFill tool-paths with the cooling time limits $\tau(c)$ of every contact c set to 3.5 s, 2.3 s, and 1.7 s, respectively. The red dashed lines show the contacts with maximum cooling time.

The last two example solutions should be compared to those of the scan-line solutions, alternating (Figure 8) and non-alternating (Figure 7). The HotFill solutions have the same maximum cooling times, but slightly lower fab-times (17.68 s vs. 17.95 s and 22.24 s vs. 23.71 s, respectively).

Figure 16 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 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 four bandpaths: $(0, 15)$, $(15, 16)$, $(16, 17)$ 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 1, which has slightly better fab-time (13.51 s vs. 13.59 s).

For comparison, Figure 17 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 $(17, 18)$. Still the fab-time (14.06 s) is not much worse than the one obtained with $\mu = 50$ (13.59 s) and that of the path of Figure 1 (13.51 s).

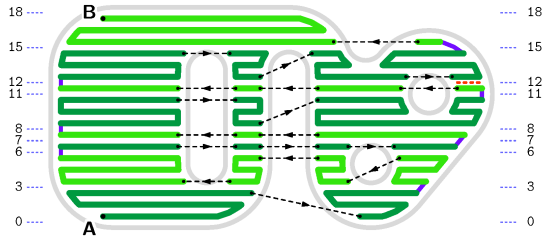

Figure 13. The HotFill solution with $\mu = 5$ and all cooling time limits set to 3.5 s. The blue horizontal dotted lines are the cut-lines that separate the band-paths that comprise the solution. The largest contact cooling time (red dashed line) is 3.36 s. The total fab-time is 15.26 s including 11.97 s of extrusion and 3.29 s of air time.

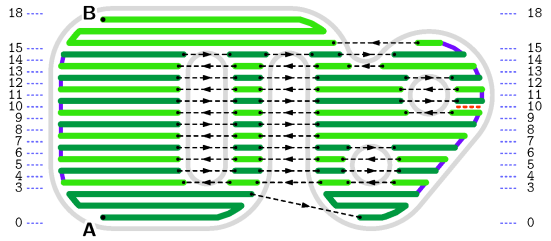

Figure 14. The HotFill solution with $\mu = 5$ and all cooling time limits set to 2.3 s. The largest contact cooling time (red dashed line) is 2.25 s. The total fab-time is 17.68 s including 11.17 s of extrusion and 6.51 s of air time.

6 Improvements and computing cost

⋆[Neri:]*Complexity foi abordada na seção 7? A seção 6 e a 7 poderiam ser uma só, não?* ⋆[Stolfi:]*São sentidos diferentes de "complexidade". Mudei os nomes das seções.*

6.1 Algorithm improvements

⋆[Neri:]*Se na seção 5 o método foi totalmente explicado, por que não passar esta parte em mais um apêndice? Acho que somente os leitores mais da computação gostarão de ler esta seção. Os da mecânica se contentarão se entenderem a lógica do método!!!* ⋆[Stolfi:]*Talvez, mas o método é inviável sem estas alterações. Não é "future work". O código disponibilizado tem estas melhorias. E pode ser artigo de mecânica para você, mas para mim é artigo de computação ;-) Tanto assim que não tem nenhum experimento de laboratório – só algoritmos e testes no computador...*

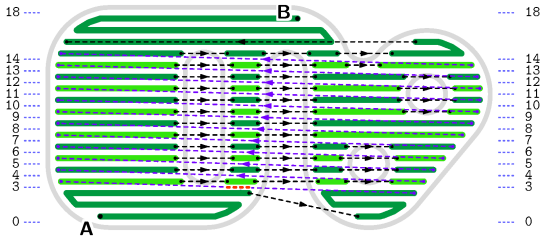

Figure 15. The HotFill solution with $\mu = 5$ and all cooling time limits set to 1.7 s. The largest contact cooling time (red dashed line) is 1.69 s. The total fab-time is 22.24 s including 10.66 s of extrusion and 11.58 s of air time.

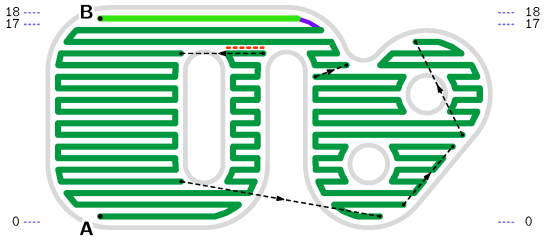

Figure 16. The HotFill solution with $\mu = 50$ and all cooling time limits set to 50 s. The largest contact cooling time (red dashed line) is 9.60 s. The total fab-time is 13.59 s including 12.55 s of extrusion and 1.04 s of air time.

The running time of Algorithm 1 can be reduced in many ways.

Virtual fullpaths: Most of HotFill's running time, as described, goes 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 $n = \#\mathscr{R}$ to the total computing time.

We can reduce that cost substantially 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 $(s+1) \times (s+1)$.

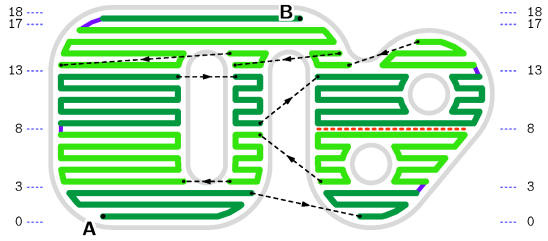

Figure 17. The HotFill solution with $\mu = 5$ and all cooling time limits set to 50 s. The largest contact cooling time (red dashed line) is 6.50 s. The total fab-time is 14.06 s including 12.44 s of extrusion and 1.62 s of air time.

Each entry $T_z[i,j]$ should be set to $+\infty$ if the computed $F_z[i,j]$ is Λ, otherwise it should be set to $T_{\text{FAB}}(F_z[i,j])$. Each entry $K_z[i,j]$ would be a pair (k,z') of an integer and a bit. It should be set to $(-1,0)$ if $F_s[i,j]$ is Λ or $i = 0$, otherwise it should be such that $F_z[i,j]$ was formed by concatenating $F_{z'}[k,i]$ with $B_0[i,j]$.

With these tables, it is not necessary to actually 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 can be filled instead by slight modifications of Algorithms 1 and 2. The final toolpath H can be 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 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 $B_z[i,j]$.

Early BandPath termination: The procedure BandPath can be speeded up also by monitoring each contact c in the cut-lines i and j as the bandpath B is being built, and computing a lower bound $T^{\text{PRED}}_{\text{COOL}}(B_*,c)$ based on the current incomplete bandpath B_*, for the cooling time of c on any fullpath that may includes B.

Specifically, for each contact c on cut-line i, we can define $T^{\text{PRED}}_{\text{COOL}}(B_*,c)$ as $T_{\text{FAB}}(B_*)$, if B_* does not cover c, or $T_{\text{COV}}(B_*,c)$, if it does. For each contact c on cut-line j, we can define $T^{\text{PRED}}_{\text{COOL}}(B_*,c)$ to be 0, if B_* does not cover c, or $T_{\text{COV}}(\overleftarrow{B_*},c)$, if it does. The BandPath procedure can stop and return Λ as soon as $T^{\text{PRED}}_{\text{COOL}}(B_*,c)$ exceeds the cooling time limit $\tau(c)$ for any of those contacts. As the width $j-i$ of the band increases, this early termination of BandPath will be increasingly likely to happen.

6.2 Programming improvements

Even with the algorithmic improvements above, a straightforward implementation of HotFill would waste a considerable amount of time computing functions such as $T_{\text{COV}}(P,c,r)$, $T_{\text{COOL}}(P,c)$, $R^{\text{MAX}}_{\text{COOL}}(P,\mathscr{C})$ and so on. For example, computing $T_{\text{COV}}(P,c,r)$ 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 $R^{\text{MAX}}_{\text{COOL}}(P,\mathscr{C})$ proportional to $\#P \times \#\mathscr{C}$. 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.

Fortunately there are several simple programming tricks that can be used to speed up those operations, possibly reducing those "unit costs" to a constant or to a multiple of $\log n$. Here are some of them.

Orientation bits: A move r can be 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 would be fields $\boldsymbol{r}.end[0]$ and $\boldsymbol{r}.end[1]$ in the order of the native orientation; so $p_{\text{INI}}(r) = \boldsymbol{r}.end[b]$, $p_{\text{FIN}}(r) = \boldsymbol{r}.end[1-b]$, and so on. This representation also makes it trivial to check whether a move is the reversal of another move.

This trick is used for paths too. A path P can be 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, $P[k]$ would be $\boldsymbol{P}.mv[k]$ 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 can be 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 can be 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 precomputed 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.

Similarly, the cover times of each contact c on cut-line i by the bandpath $B_z[i, j]$ can be computed when that bandpath is constructed, and can be stored in a list $\boldsymbol{B}.tcovs[0]$ of the Path object. Another list $\boldsymbol{B}.tcovs[1]$ would hold the cover times by $B_z[j, i]$ of the contacts on cut-line j

Locating contacts and rasters: Another potential waste of computing time is scanning the list $\mathscr{C}$ 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 can be reduced by splitting the list $\mathscr{C}$ 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 can save time by splitting the raster set $\mathscr{R}$ into $s - 1$ lists $R[0]$, $R[1]$, ..., $R[s - 1]$, one per scan-line.

Locating links: For efficiency, the lists $\mathscr{L}(P)$ and $\mathscr{L}(\overleftarrow{P})$ 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) — should be attributes of the relevant path object $\boldsymbol{P}$. Then the Concat function then can quickly locate the bridging link by looking for a matching entry in the lists $\mathscr{L}(\overleftarrow{P'})$ and $\mathscr{L}(P'')$, 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.

6.3 Computing time analysis

We can assume 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 a $2(s - \mu)\mu + \mu(\mu + 1)$ times each, which less than $2s\mu$. With the above optimizations, the asymptotic worst-case computing cost [37] 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 $n_{i,j} = \#\mathscr{R}_{i,j}$ 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 $n = \#\mathscr{R}$ 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.

7 Experiments and results

In this section 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 [34] that incorporates the optimizations described in Section 6.

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 7.2.

The second batch of tests aims to demonstrate that the imposition of reasonable cooling time constraints, which should improve the mechanical resis-

tance of the part, generally has a small impact on the fabrication time T_{FAB} of the tool-path. See Section 7.3.

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 7.4.

In all tests, the parameter $\tau(c)$ of every contact c was set to the same value Δ. Thus the maximum relative cooling time $R^{\text{MAX}}_{\text{COOL}}(H, \mathscr{C})$ is $T^{\text{MAX}}_{\text{COOL}}(H, \mathscr{C})/\Delta$. 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 [38] and Slic3r [4] (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).

In order to make the fab-time comparisons with Slic3r meaningful, the nominal contour offset and raster spacing parameters of that program had to be adjusted so that, on the average, the number and total length of the raster lines it used were as close as possible to those used by RP3 and HotFill.

⋆[Stolfi:]*Listar os parãmetros de offset etc fornecidos ao RP3.*

⋆[Stolfi:]*Listar aqui os parâmetros fornecidos ao Slic3r, inclusive o* λ_{FILL}.

7.1 Test data

STL models: For testing, 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 [39]. 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 4.4, 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 18. 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).

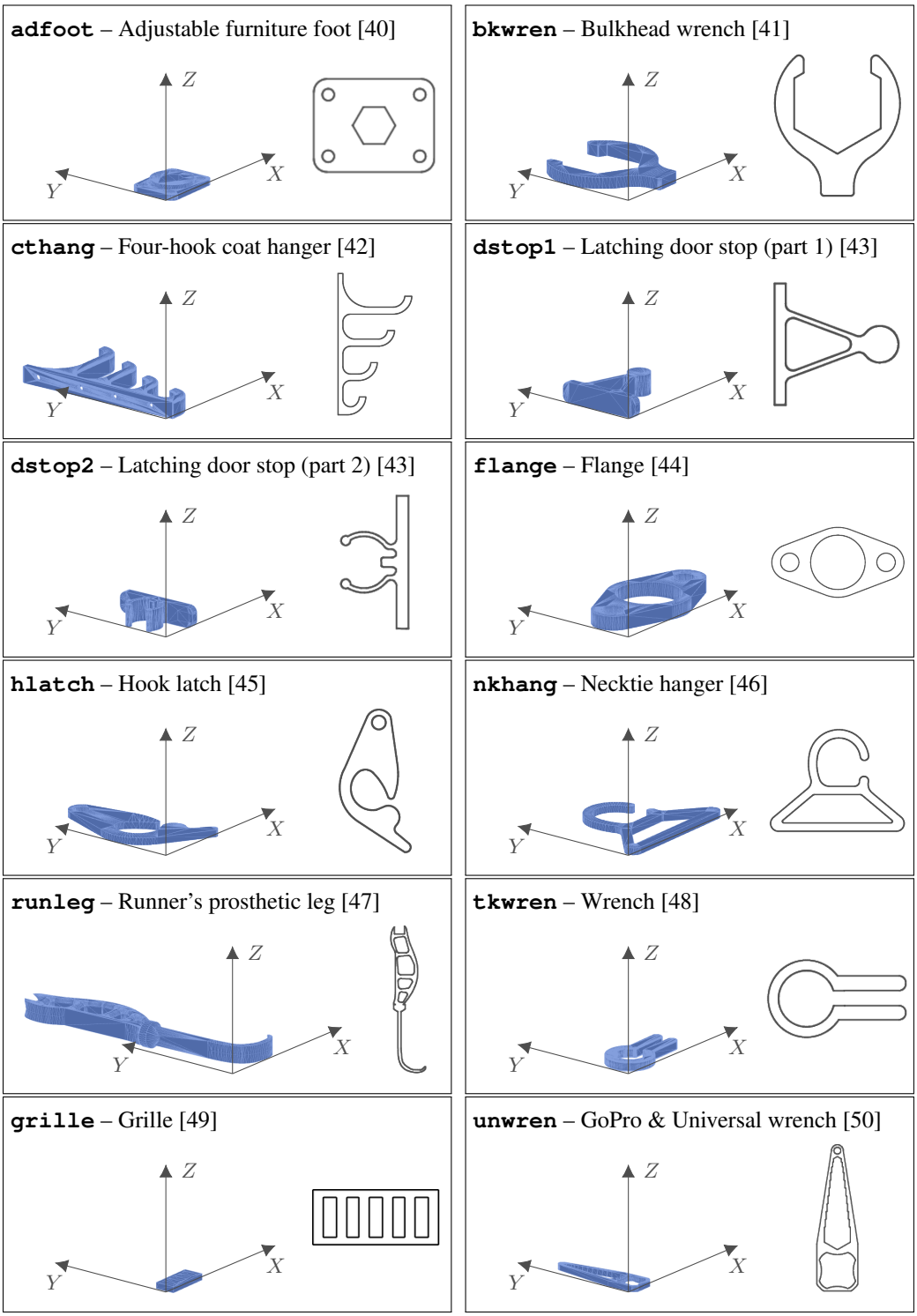

Figure 18. The STL models and slice outlines used in the tests (not to the same scale).

Input datasets: For each test slice, we performed two preliminary tests of HotFill algorithm, with the slice rotated by $\theta = 0$ or $\theta = 90$ degrees. The RP3 software was used to extract the polygonal outline D of the area to be filled, rotated by the specified angle θ, and to compute the endpoints of the raster lines, with assumed slice thickness $\sigma = 0.25\,\mathrm{mm}$ and scan-line spacing (nominal raster width) $\lambda_{\text{FILL}} = 0.4\,\mathrm{mm}$. The RP3 software also computed the link paths that could be used to connect the endpoints of every pair of adjacent rasters, if they were to be extruded in sequence with opposite orientations.

Specifically, the polygon D was the result of applying an inwards offset of $\delta = 0.15\,\mathrm{mm}$ to the slice's outline, to account for the thickness of the contour trace. The trajectory of each link path was the piece of the boundary of the polygon D between the relevant raster endpoints. ⋆[Stolfi:]*Esta é uma propriedade exclusiva dos dados que usamos nos testes, só porque pegamos os links do RP3 e ele faz assim. O algoritmo e o programa não fazem nenhuma suposição sobre os links. Poderiam por exemplo ser simples traços retos, o que economizaria tempo mensurável. Ou poderiam ser pedaços de D, mas simplificados.* The RP3 program was extended to output the raster and link information to a text file, that was read by the HotFill implementation. ⋆[Stolfi:]*Cada link path está pendurada em dois rasters. Mas, do jeito que paths são representadas, o custo dessa duplicação é pequeno; a maior parte da informação (pontos, tempos, etc) é compartilhada entre S e* $\overleftarrow{S}$*. Aliás esta é uma das razões para manter a seção "Program improvements" no artigo: o gasto de memória seria muitas vezes maior sem esses truques.*

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 7.3). 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	m	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

7.2 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 $c \in \mathscr{C}$). The results are shown in Table 2. ⋆[Neri:]*O objetivo já foi dito logo acima. Sugiro retirar ou reescrever! Sugestão de alteração de frase: [*The HotFill results finding a viable tool-path H that satisfies the specified cooling time constraints, which are $T_{\text{COOL}}(H, c) \leq \Delta$ for all $c \in \mathscr{C}$,are shown in Table 2.*]* ⋆[Stolfi:]*Aquela frase na introdução da seçaõ está muito longe e é muito sucinta. Acho que precisa explicar aqui claramente o objetivo desta série de testes* The entries marked "—" in the table are cases where HotFill failed to find a viable tool-path.

Table 2. Maximum contact cooling times $T^{\text{MAX}}_{\text{COOL}}(H, \mathscr{C})$ 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

Discussion: As discussed in Section 4.3, 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 7.3).

7.3 Impact on fabrication time

The tests in this section aim to show the impact of the cooling time constraints on the fabrication time $T_{\text{FAB}}(H)$ of the computed tool-path. The results are shown in Table 3. ⋆[Neri:]*O objetivo já foi dito logo acima. Sugestão de frase: [*The results of the impact of the cooling time constraints on the fabrication time $T_{\text{FAB}}(H)$ of the computed tool-path are shown in Table 3.*]* ⋆[Stolfi:]*Mesma resposta da seção anterior.* 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 $T^{\text{HF}}_{\text{FAB}}$ of HotFill for each limit Δ, and the percent increase $D^{\text{HF}}_{\text{FAB}}$ of that fab-time relative to $T^{\text{REF}}_{\text{FAB}}$, the smallest between the Slicr3 and RP3 fab-times. The total extrusion time T_{RAST} for the input rasters, which is the ideal lower bound for $T_{\text{FAB}}(H)$, is also shown for reference. Each test was performed with $\mu = 20$.

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 Section 9 with the following parameters: acceleration and deceleration, $3000\,\text{mm/s}^2$; maximum nozzle speed during extrusion, $40\,\text{mm/s}$; maximum nozzle travel speed during jumps, $130\,\text{mm/s}$; Jump start/end penalty, $0.05\,\text{s}$. These parameter values were inferred from the actual fab-times of other tool-paths on the 3D Cloner model DH+ printer [51].

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 the percent fab-time increases of the latter relative to 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.

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

The dependency of $T_{\text{FAB}}(H)$ on the cooling limit Δ, for each of the test parts, is plotted on Figure 19. 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* $T_{\text{CONN}} = T_{\text{FAB}} - T_{\text{RAST}}$, the air time plus fab-time of links, compared to that of the reference tool-path: namely, $R^{\text{HF}}_{\text{CONN}} = T^{\text{HF}}_{\text{CONN}} / T^{\text{REF}}_{\text{CONN}}$, where $T^{\text{HF}}_{\text{CONN}}$ is the connection time of the HotFill path, and $T^{\text{REF}}_{\text{CONN}}$ 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 $T^{\text{HF}}_{\text{FAB}} / T^{\text{REF}}_{\text{FAB}}$ are usually much smaller than $R^{\text{HF}}_{\text{CONN}}$. For example, according to Table 3, for $\Delta = 8\,\text{s}$ and $\mu = 20$, the HotFill tool-path of that dataset has $T^{\text{HF}}_{\text{CONN}} = 266.9 - 150.1 = 116.1\,\text{s}$, which is about 7 times the connection time of the reference tool-path $T^{\text{REF}}_{\text{CONN}} = T^{\text{SLIC3R}}_{\text{CONN}} = 165.8 - 150.1 = 15.7\,\text{s}$; that is, $R^{\text{HF}}_{\text{CONN}} \approx 700\%$. However, the ratio of the total fab-times is only $266.9/165.8 = 161\%$ ($+61\%$ in Table 3).

Figure 19 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.

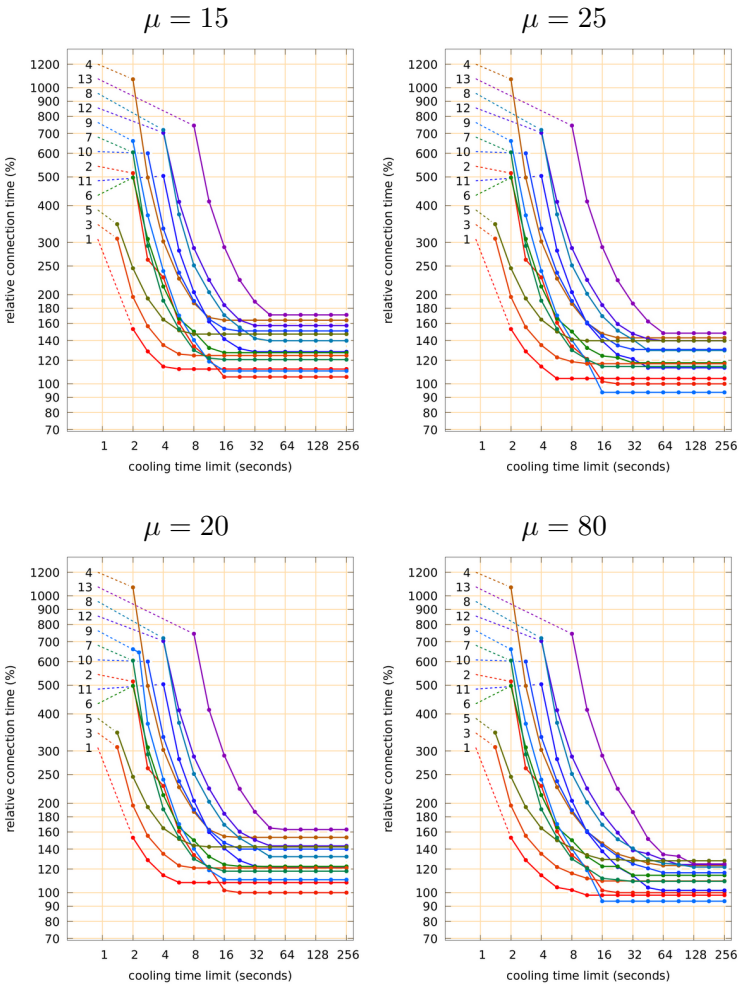

Figure 19. Relative connection time $R^{\text{HF}}_{\text{CONN}}$ as a function of the cooling time limit Δ, for different values of the maximum band width μ. Each curve is a different dataset. The numbers refer to Table 1.

Discussion: 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 $T^{\text{MAX}}_{\text{COOL}}$ 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 $T^{\text{MAX}}_{\text{COOL}}$ 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, $\Delta \geq 8\,\text{s}$ 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 19. In these tests, the only exception for $\Delta = 8\,\text{s}$ was the **`runleg:90`** dataset (number 13, purple in the figure) for which the HotFill path had 61% higher fab-time ($266.9\,\text{s}$) than Slic3r ($165.8\,\text{s}$). For $\Delta = 16\,\text{s}$, this difference dropped to 18%, while it was 5% or less for all the other datasets.

As seen in Figure 19, 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 $\Delta \geq 128\,\text{s}$, the HotFill paths had less than 30% higher connection time than the reference path. However, the running time of HotFill increased steeply as μ increased beyond 10. See Section 7.4. 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.

7.4 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 $\#\mathscr{R}$, the maximum band width μ, and the cooling time limit Δ. The results are shown in Table 4 and Figure 20.

The algorithms were implemented in Python3 [34]. The times were measured on an Intel® Core™ i7-10700 PC with a 2.9 GHz clock, 256 kiB:2 MiB:16 MiB L1:L2:L3 cache, 128 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.

Table 4. Computation times T_{CPU} of the HotFill algorithm for $\Delta = 8.0\,\text{s}$ and various values of the maximum band height μ. All times are in seconds.

			μ							
Dataset	n	m	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

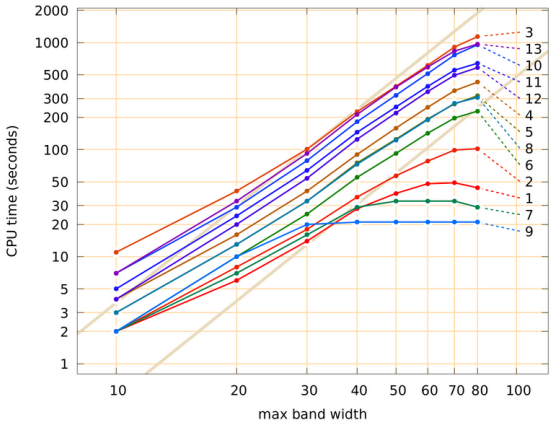

Figure 20. Computing time T_{CPU} as a function of the maximum band width μ, for $\Delta = 64.0\,\text{s}$ and the datasets listed in Table 4. Each graph is a different dataset, with colors assigned arbitrarily. The broad tan lines are the plots of $A\mu^3$ for $A = 0.00370$ and $A = 0.000481$.

Discussion: As observed in Section 6.3, 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 $n = \#\mathscr{R}$ is the number of raster elements.

As shown in Figure 20, 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 20 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, $n = 398$, $s = 160$

7.5 Sample HotFill outputs

Figures 21, 22 and 23 show some of the infill tool-paths generated by HotFill for some of the test datasets of Table 1. For each solution, the complete tool-path is shown on the left, and its constituent bandpaths are shown on the right.

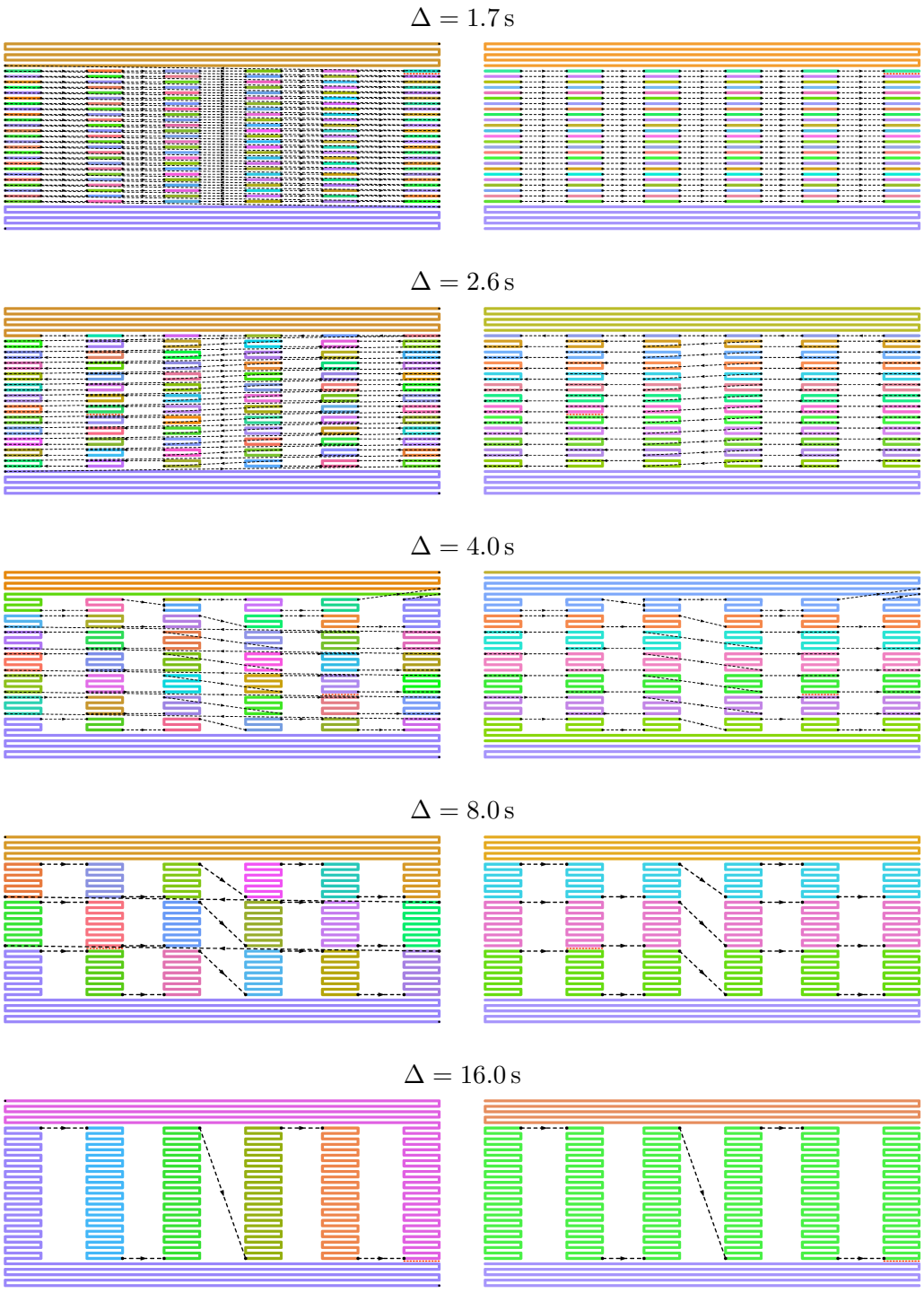

Figure 21. Tool-paths produced by HotFill for the **`grille:0`** dataset, with $\mu = 40$ and various cooling time limits Δ. The left side shows the complete final fullpath H, with each continous extrusion section drawn in a different color. The right side shows the bandpaths $B_z[i, j]$ that were joined to produce that tool-path, each in a different color, without the jumps or links that connect them.

In Figure 21, it can be seen that the smaller the limit Δ, the more jumps between rungs are required to satisfy the cooling constraints; which increases the fabrication time, as observed in Table 3. The top entry in Figure 21 was produced for the dataset **`grille:0`**, with $\Delta = 1.7$s, slightly above the maximum cooling time of the non-alternating scan-line-order path SCN. Note that the middle section of the tool-path, that fills the vertical bars of the grille, is essentially the same as the SCN path. The contact cooling times are larger in that region than in the horizontal bars, because of the penalties and the acceleration/deceleration time at each trace/jump boundary. On the two horizontal bars, however, HotFill was able to follow the alternating scan-line-order (SCA) strategy, without exceeding the limit Δ. Thus the total fab-time was smaller than that of the SCN: $51.04\,\mathrm{s}$ rather than $53.98\,\mathrm{s}$.

The second tool-path from the top was obtained with $\Delta = 2.6\,\mathrm{s}$, slightly greater than the $T^{\mathrm{MAX}}_{\mathrm{COOL}}$ of the alternating scan-line solution SCA. HotFill did consider the SCA tool-path, but found a slightly better one, shown in the figure. It executes two rasters in each vertical bar before moving to the next bar. Compared to the SCA solution, this strategy requires only half as many jumps between the vertical bars. On the other hand, the cooling time constraints between these band-paths force them to be executed all in the same general direction (from right to left), connected by long jumps. Still, the overall fab-time of the HotFill tool-path ($39.32\,\mathrm{s}$) is less that of the SCA solution ($41.51\,\mathrm{s}$).

The diagonal jumps in the middle of some band-paths, in this and the following examples, are a consequence of the fact that the greedy algorithm builds two half-paths, starting with rasters at two opposite corners of the band and grows them inwards, in both directions at the same time. The diagonal jump is where the two half-paths meet.

The last three entries, for $\Delta = 4.0 = 8.0$, and $16.0\,\mathrm{s}$, show how HotFill managed to reduce the fab-time even further (to $31.30\,\mathrm{s}$, $26.57\,\mathrm{s}$ and $24.43\,\mathrm{s}$, respectively) by using wider bandpaths. The last one could have used a single bandpath, instead of the three shown at right; however the resulting fullpath would have been the same.

Figures 22 and 23 shows HotFill results for two other datasets, **`cthang:90`** and **`dstop1:0`**, with $\mu = 40$ and various values of Δ. In both cases, notice how bandpath widths are smaller where the infill has longer and/or more numerous per scan-line, but generally increase as Δ increases. On the left column of Figure 22, the path for $\Delta = 32\,\mathrm{s}$ is visibly affected by the limit $\mu = 40$.

⋆[Neri:]*Acho que cabe um comentário geral do tipo: [*In general, it can be seen that for shorter Δ, the number of jumps increases ... fab-time increases ... *]* ⋆[Stolfi:]*Coloquei isso acima.* ⋆[Neri:]*[*and this will affect the part surface quality (due to strings generated). Therefore, there is a trade-off between part strength and surface quality.*]* ⋆[Stolfi:]*Está certo isto? Note que esses paths são só o infill; o contorno supostamente vai ser fabricado de uma só vez, antes ou depois do infill.*

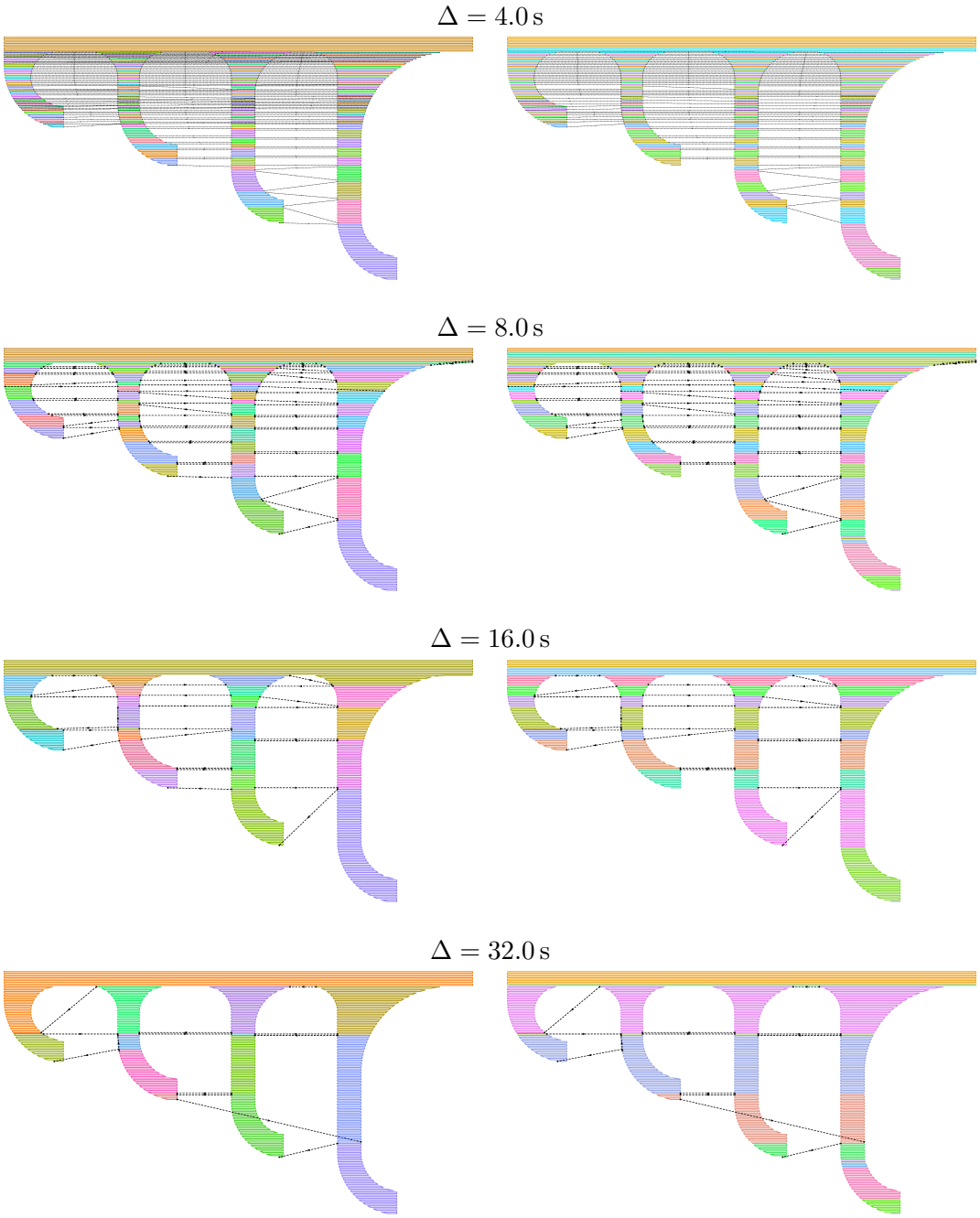

Figure 22. Tool-paths produced by HotFill for the input dataset **`cthang:90`**, with $\mu = 40$ and various cooling time limits Δ. In each case, the final tool-path is at left, the component bandpaths are at right, with the same color conventions of Figure 21.

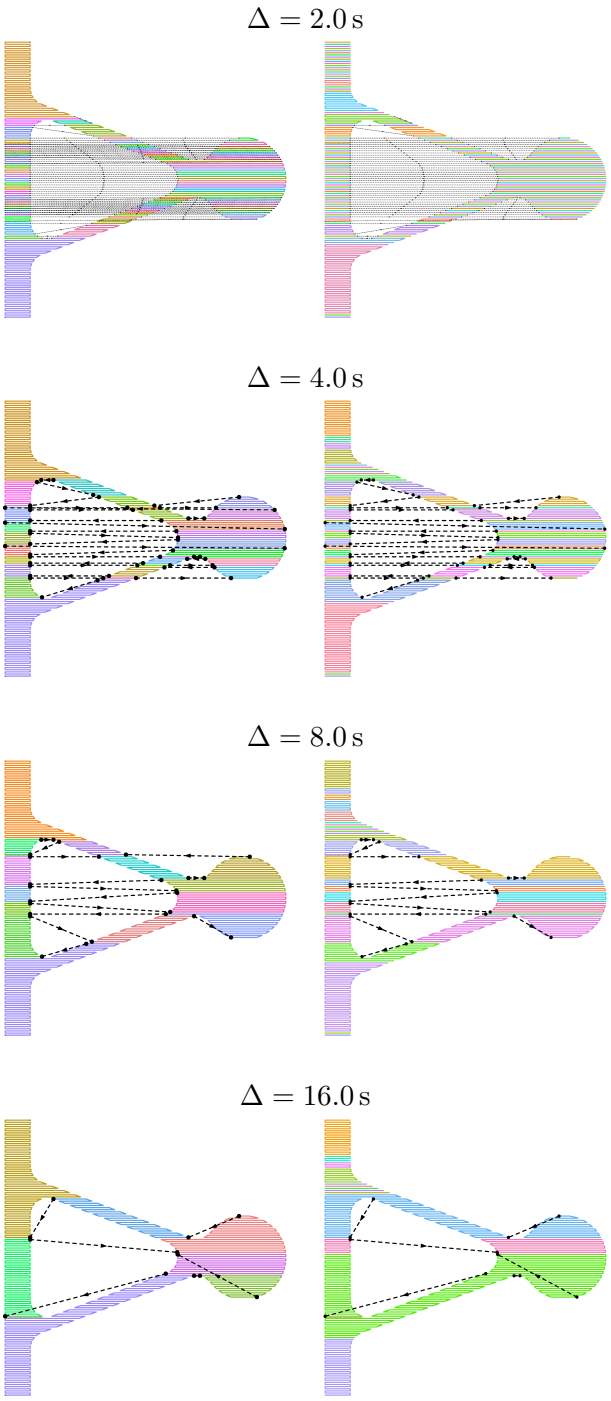

Figure 23. Tool-paths (fullpaths) produced by HotFill for the input dataset **`dstop1:90`**, with $\mu = 40$ and various cooling time limits Δ. In each case, the final tool-path is at left, the component bandpaths are at right, with the same color conventions of Figure 21.

8 Conclusions and future work

We described HotFill, an algorithm to plan a tool-path for solid (airgap 0) raster infill for thermoplastic extrusion 3D printers. Unlike the paths produced by commonly used path planning programs like Slic3r and RP3, the HotFill solution satisfies prescribed limits ⋆[Neri:]*???* 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 [22, 26, 27, 28, 25].

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 nontrivial 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. ⋆[Neri:]*???* 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.

Possibilities for development: 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 [52].

The bidirectional greedy BandPath heuristic that we used (see Section 5.5) 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.

9 Appendix: Move timing functions

The move fabrication time functions $T_{\mathsf{FAB}}(m)$ and $T_{\mathsf{COV}}(m,u)$ defined in Section 9 can be estimated as described by Komineas et al. [53]. 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):

$$\begin{cases} \sqrt{d_{\mathsf{u}}/a} & \text{if } d_{\mathsf{u}} \leq d_{\mathsf{a}} \\ 2t_{\mathsf{a}} + t_{\mathsf{c}} - \sqrt{(d-d_{\mathsf{u}})/a} & \text{if } d_{\mathsf{u}} \geq d - d_{\mathsf{a}} \\ t_{\mathsf{a}} + (d_{\mathsf{u}} - d_{\mathsf{a}})/v & \text{otherwise} \end{cases}$$

References

[1] P. Kulkarni, A. Marsan, D. Dutta, Review of process planning techniques in layered manufacturing, Rapid Prototyping Journal 6 (1) (2000) 18–35. doi:10.1108/13552540010309859.

[2] I. Gibson, D. Rosen, B. Stucker, Additive manufacturing technologies - 3D printing, rapid prototyping, and direct digital manufacturing, in: Rapid Manufacturing Association, 2nd Edition, Springer, 2015, p. 510. doi:10.1520/F2792-12A.2.

[3] R. Minetto, N. Volpato, J. Stolfi, R. M. Gregori, M. V. da Silva, An Optimal Algorithm for 3D Triangle Mesh Slicing, Computer-Aided Design 92 (2017) 1–10. doi:10.1016/j.cad.2017.07.001.

[4] G. Hodgson, A. Ranellucci, J. Moe, Slic3r manual, Online document at `https://manual.slic3r.org/`. Acessed on 2021-07-24. (2021).

[5] N. Volpato, L. C. Galvão, L. F. Nunes, R. I. Souza, K. Oguido, Combining heuristics for tool-path optimisation in material extrusion additive manufacturing, Journal of the Operational Research Society 0 (0) (2019) 1–11. doi:10.1080/01605682.2019.1590135.

[6] N. Volpato, RP3 basic user guide (v2021), Tech. rep., Federal University of Technology - Parana (UTFPR), Department of Mechanical Engineering, Additive Manufacturing and Tooling Group (NUFER) (2021).

[7] M. K. Agarwala, V. R. Jamalabad, N. A. Langrana, A. Safari, P. J. Whalen, S. C. Danforth, Structural quality of parts processed by fused deposition, Rapid Prototyping Journal 2 (4) (1996) 4–19. doi:10.1108/13552549610732034.

[8] Y.-A. Jin, Y. He, J.-Z. Fu, W.-F. Gan, Z.-W. Lin, Optimization of tool-path generation for material extrusion-based additive manufacturing technology, Additive Manufacturing 1-4 (2014) 32–47. doi:10.1016/j.addma.2014.08.004.

[9] P. K. Wah, K. G. Murty, A. Joneja, L. C. Chiu, Tool path optimization in layered manufacturing, IIE Transactions 34 (4) (2002) 335–347. doi:10.1080/07408170208928874.

[10] W. Yang, Optimal path planning in rapid prototyping based on genetic algorithm, in: Chinese Control and Decision Conference, Institute of Electrical and Electronics Engineers (IEEE), 2009, pp. 5068–5072. doi:10.1109/CCDC.2009.5194966.

[11] N. Ganganath, C.-T. Cheng, K.-Y. Fok, C. K. Tse, Trajectory planning for 3D printing: A revisit to traveling salesman problem, in: 2016 (2nd) International Conference on Control, Automation and Robotics (ICCAR), 2016, pp. 287–290. doi:10.1109/ICCAR.2016.7486742.

[12] K. Fok, N. Ganganath, C. Cheng, C. K. Tse, A 3D printing path optimizer based on Christofides algorithm, in: 2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE), 2016, pp. 1–2. doi:10.1109/ICCE-TW.2016.7520990.

[13] A. Croes, A method for solving traveling salesman problems, Operations Research 5 (1958) 791–812.

[14] M. M. Flood, The traveling-salesman problem, Operations Research 4 (1) (1956) 61–75.

[15] S. Lin, B. W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Operations Research 21 (2) (1973) 498–516. doi:10.1287/opre.21.2.498.

[16] D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis, II, An analysis of several heuristics for the traveling salesman problem, SIAM Journal on Computing 6 (3) (1977) 563–581. doi:10.1137/0206041.

[17] K. Fok, C. Cheng, N. Ganganath, H. H. Iu, C. K. Tse, An ACO-based tool-path optimizer for 3-D printing applications, IEEE Transactions on Industrial Informatics 15 (4) (2019) 2277–2287. doi:10.1109/TII.2018.2889740.

[18] J. Yang, H. Bin, X. Zhang, Z. Liu, Fractal scanning path generation and control system for selective laser sintering (SLS), International Journal of Machine Tools and Manufacture 43 (3) (2003) 293–300.

[19] D. Zhao, W. Guo, Shape and performance controlled advanced design for additive manufacturing: A review of slicing and path planning, Journal of Manufacturing Science and Engineering 142 (2020) 1–87. doi:10.1115/1.4045055.

[20] C. Koch, L. Van Hulle, N. Rudolph, Investigation of mechanical anisotropy of the fused filament fabrication process via customized tool path generation, Additive Manufacturing 16 (2017) 138–145. doi:https://doi.org/10.1016/j.addma.2017.06.003.
URL `https://www.sciencedirect.com/science/article/pii/S2214860417300465`

[21] L. Xia, S. Lin, G. Ma, Stress-based tool-path planning methodology for fused filament fabrication, Additive Manufacturing 32 (2020) 101020. doi:https://doi.org/10.1016/j.addma.2019.101020.
URL `https://www.sciencedirect.com/science/article/pii/S2214860419315659`

[22] Q. Sun, G. M. Rizvi, C. T. Bellehumeur, P. Gu, Effect of processing conditions on the bonding quality of FDM polymer filaments, Rapid Prototyping Journal 14 (2) (2008) 72–80. doi:10.1108/13552540810862028.

[23] P. K. Gurrala, S. P. Regalla, Part strength evolution with bonding between filaments in fused deposition modelling, Virtual and Physical Prototyping 9 (3) (2014) 141–149. doi:10.1080/17452759.2014.913400.

[24] M. Faes, E. Ferraris, D. Moens, Influence of inter-layer cooling time on the quasi-static properties of

ABS components produced via fused deposition modelling, Journal Procedia CIRP 42 (2016) 748–753. doi:10.1016/j.procir.2016.02.313.

[25] N. Volpato, T. T. Zanotto, Analysis of deposition sequence in tool-path optimization for low-cost material extrusion additive manufacturing, International Journal of Advanced Manufacturing Technology 101 (5-8) (2019) 1855–1863. doi:10.1007/s00170-018-3108-1.

[26] S. F. Costa, F. M. Duarte, J. A. Covas, Estimation of filament temperature and adhesion development in fused deposition techniques, Journal of Materials Processing Technology 245 (2017) 167–179. doi:10.1016/j.jmatprotec.2017.02.026.

[27] B. Akhoundi, A. H. Behravesh, Effect of filling pattern on the tensile and flexural mechanical properties of FDM 3D printed products, Experimental Mechanics 59 (2018) 883–897.

[28] E. Ferraris, J. Zhang, B. Van Hooreweder, Thermography based in-process monitoring of fused filament fabrication of polymeric parts, CIRP Annals 68 (1) (2019) 213–216. doi:10.1016/j.cirp.2019.04.123.

[29] Y. Dumas, J. Desrosiers, E. Gelinas, M. M. Solomon, An optimal algorithm for the traveling salesman problem with time windows, Operations research 43 (2) (1995) 367–371.

[30] M. Savelsbergh, Local search in routing problems with time windows, Annals of Operations research 4 (1985) 285–305. doi:10.1007/BF02022044.

[31] C.-B. Cheng, C.-P. Mao, A modified ant colony system for solving the travelling salesman problem with time windows, Mathematical and Computer Modelling 46 (9) (2007) 1225–1235. doi:10.1016/j.mcm.2006.11.035.

[32] M. López-Ibáñez, C. Blum, Beam-ACO for the travelling salesman problem with time windows, Computers & Operations Research 37 (9) (2010) 1570 – 1583. doi:10.1016/j.cor.2009.11.015.

[33] N. Mladenović, R. Todosijević, D. Urošević, An efficient general variable neighborhood search for large travelling salesman problem with time windows, Yugoslav Journal of Operations Research 23 (1) (2013) 19–30.

[34] E. Cassiana, Hotfill, Code repository at `https://github.com/ecassiana/hotfill`. Accessed on 2021-07-28 (2021).

[35] F. P. Preparata, M. I. Shamos, Computational Geometry – An Introduction, Springer, 1985.

[36] R. E. Bellman, Dynamic Programming, Courier Dover Publications, 1957.

[37] D. E. Knuth, The Art of Computer Programming, Vol. 1, Addison-Wesley, 1998.

[38] N. Volpato, Manual básico do programa RP^3 (v110211), Tech. rep., U. Tecnológica Federal do Paraná, Depto. de Mecânica, NUFER (2011).

[39] MakerBot, Thingiverse - Digital designs for physical objects, 3D model repository at `www.thingiverse.com/`. Accessed on 2021-07-24 (2021).

[40] Frank (@ffleurey), Adjustable foot for tables and furnitures, 3D model at `www.thingiverse.com/thing:148764`. Accessed on 2021-07-24 (2013).

[41] J. Phillips (@jephil08), 1-Inch bulkhead hand wrench, 3D model at `www.thingiverse.com/thing:4356560`. Accessed on 2021-07-24 (2020).

[42] E. Farkas (@Alpha__Wolf), 4-Hook coat hanger, 3D model at `www.thingiverse.com/thing:3315713`. Accessed on 2021-07-24 (2018).

[43] M. Charpentier (@Matearoa), Door stop, 3D model at `www.thingiverse.com/thing:4889702`. Accessed on 2021-07-24 (2021).

[44] E. Cassiana (@ecassiana), Flange, 3D model at `https://www.thingiverse.com/thing:4916943`. Accessed on 2021-07-28 (2021).

[45] E. Cassiana (@ecassiana), Hook latch, 3D model at `https://www.thingiverse.com/thing:4919047`. Accessed on 2021-07-28 (2021).

[46] E. Cassiana (@ecassiana), Necktie hanger, 3D model at `https://www.thingiverse.com/thing:4916952`. Accessed on 2021-07-28 (2021).

[47] N. Illasarie (@nillasarie), Runners MK2 prosthetic leg, 3D model at `www.thingiverse.com/thing:1960573`. Accessed on 2021-07 (2016).

[48] A. Jøraasen (@joraasen), 22mm Wrench, 3D model at `www.thingiverse.com/thing:1909039`. Accessed on 2021-07-25 (2016).

[49] E. Cassiana (@ecassiana), Grille, 3D model at `https://www.thingiverse.com/thing:5161470`. Accessed on 2021-12-11 (2021).

[50] Anthony L. (@M600), GoPro tool with universal wrench, 3D model at `www.thingiverse.com/thing:601212`. Accessed on 2021-07-25 (2014).

[51] 3DCloner, 3dcloner dh plus, Code repository at `http://3dcloner.ind.br/upload/20200724170856r7ginb.pdf`. Accessed on 2021-12-13 (2021).

[52] H. Edelsbrunner, L. J. Guibas, Topologically sweeping an arrangement, Journal of Computer and System Sciences 38 (1) (1989) 165–194. doi:10.1016/0022-0000(89)90038-X.

[53] G. Komineas, P. Foteinopoulos, A. Papacharalampopoulos, P. Stavropoulos, Build time estimation models in thermal extrusion additive manufacturing processes, Procedia Manufacturing Journal 21 (2018) 647 – 654. doi:10.1016/j.promfg.2018.02.167.