

1 Highlights:

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

8 ★[Neri:]*Uma alternativa para o terceiro highlight: [The problem of tool-path*
9 *generation with prescribed cooling time limits is tackled.]* ★[Stolfi:]*Altereí o item*
10 *3. Acho que “is tackled” não soa bem.*

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

Elis C. Nakonetchnei¹, Jorge Stolfi², Neri Volpato¹, Ricardo D. da Silva¹,
Rodrigo Minetto^{1,*}

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 $R_{cool}(P, c) = T_{cool}(P, c) / \tau(c)$ 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

*Corresponding Author - Av. Sete de Setembro, 3165, Curitiba, Brazil, Tel. +55 41 998961525, Fax. +55 41 3310 4998.

Email addresses: ??@dinf.utfpr.br (Elis C. Nakonetchnei), stolfi@ic.unicamp.br (Jorge Stolfi), ??@???.utfpr.br (Neri Volpato), ??@dinf.utfpr.br (Ricardo D. da Silva), rminetto@dinf.utfpr.br (Rodrigo Minetto) submitted to Journal of Additive Manufacturing December 28, 2021

¹Department of Informatics, Federal University of Technology - Paraná (UTFPR), Curitiba, Brazil

²State University of Campinas (UNICAMP), São Paulo, Brazil

28 1. Introduction

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

38 Typically, the tool-path consists of a *contour*, a set of poly-lines following the
39 perimeter of the slice; a *filling* or *infill* of its interior; and, if necessary, some tem-
40 porary *support* structures to hold up overhanging parts during fabrication. The
41 filling may use many different strategies [4], such as concentric poly-lines, hon-
42 eycomb patterns, or Hilbert curves. A common strategy for solid infill (100%
43 density, air gap 0) uses a set of parallel closely spaced *raster lines* [2, 5].

44 Figure 1 shows a possible tool-path for the solid filling of a slice of a hypo-
45 thetical part, as might be produced by process planning programs like RP3 [6] or
46 Slic3r [4]. Depending on the shape of the region to be filled, the tool-path may
47 have to be split into two or more *continuous extrusion sections* (CES), shown by
48 different colors in Figure 1; in which case the nozzle will have to jump between
49 these sections without extruding any material [7].

50 Algorithms for tool-path generation typically try to minimize the total fabri-
51 cation time for each slice [8, 5, 9, 10], which consists of the *extrusion time* plus
52 the non-productive *air time* spent during the jumps [5]. There is still no algorithm

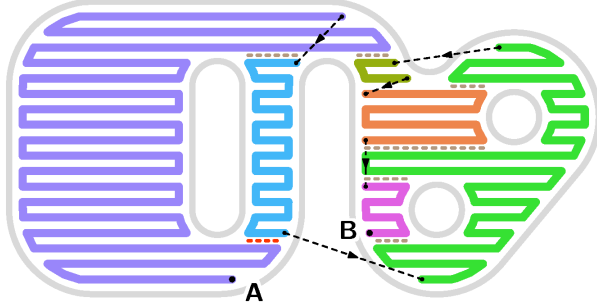


Figure 1. A possible tool-path A–B for the solid filling of a slice of a hypothetical part, whose extruded contour is shown in gray. Each color identifies a separate continuous extrusion section of the path. The black dashed lines denote nozzle movements without extrusion. The part measures $39\text{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

53 that will find the absolutely best tool-path in a practical amount of computer time;
 54 therefore, path-planning programs generally use heuristics that produce “good
 55 enough” tool-paths, but not necessarily optimal ones.

56 There is a substantial literature of such heuristics. In 2009 Weidong tested a
 57 genetic algorithm (GA) meta-heuristic for this purpose [10]. Gnaganath et al. [11]
 58 and Fok et al. [12] considered different heuristic algorithms to solve the traveling
 59 salesman problem (TSP) and adapted them to the tool-path planning problem. The
 60 main heuristics compared were: nearest neighbor selection (NN), Christofides’s
 61 algorithm, and the k -opt algorithm [13, 14, 15, 16]. In 2019, Fok et al. experi-
 62 mented with the ant colony optimization (ACO) meta-heuristic [17]. Volpato et
 63 al. [5] compared the NN heuristic to the nearest insertion heuristic (NI).

64 However, fabrication time is not the only criterion that matters when plan-
 65 ning the tool-path. Several authors have studied how the orientations of extruded

66 lines influence the tensile and flexural strength of an infill layer. For instance, it
67 was observed that fractal-like fillings, with extruded lines in multiple orientations,
68 generally improve these mechanical properties [18, 19]; and that the orientation
69 of filling rasters affects the quality of raster corners [8, 20, 21].

70 The order of extrusion also affects the strength of the infill because the bond-
71 ing between two adjacent raster lines is stronger if the second one is deposited
72 while the first is still hot [7, 22, 23, 24, 25]. This effect was noted by several au-
73 thors, starting with Sun et al. in 2008 [22, 26, 27, 28]. Studies of this effect using
74 numerical models and experimental methods were published by Akhoundi and
75 Behravesht in 2018 [27] and by Volpato and Zanotto in 2019 [25], respectively.

76 In the example of Figure 1, the red and brown horizontal dashed lines indicate
77 contacts between adjacent raster lines with significant cooling times. The contact
78 highlighted in red has the largest time, 7.26 seconds. For context, the predicted
79 fabrication time for the solid fill alone is 13.51 s including 12.59 s of extrusion and
80 0.91 s of air time. (The times in this and following examples assume the printer
81 parameters specified in Section 7.) Even if one followed the common practice
82 of rotating the direction of the filling rasters by 90 degrees between successive
83 layers [2], the part may be much weaker at those bonds than elsewhere.

84 Adhesion between adjacent filaments can be generally improved by fabricat-
85 ing the part inside a heated chamber. However this facility is not available in many
86 desktop printers. Anyway, the chamber temperature that would be needed to en-
87 sure really good adhesion, even between beads deposited a minute or more apart,
88 is likely to be so high that the deposited filament will not properly solidify and
89 the part may deform under its own weight. *★[Stolfi:]Acho que este parágrafo não
90 deve ser misturado com a revisão bibliográfica, pois esta “solução” não é men-*

91 *cionada por nenhum paper. Confere? Acho que deve ficar antes do prior work,*
92 *ou depois; não no meio*

93 In 2019, Volpato and Zanotto [25] suggested that a tool-path optimization al-
94 gorithm should take into account specified limits on the elapsed time between the
95 deposition of adjacent filaments. We call this the *hot tool-path problem* (HTPP).

96 The HTPP is somewhat similar to the traveling salesman problem with time
97 windows (TSP-TW), in which each node must be visited in a specific time in-
98 terval [29]. This problem was proved to be NP-complete by Savelsbergh [30] in
99 1985; a property that is believed to imply that there is no practical algorithm so
100 solve it exactly. Approximate solutions for the TSP-TW were described by Cheng
101 et al. [31], using an ant colony meta-heuristic; by Lopez et al. [32], using a com-
102 bination of ant colony and beam search; and by Mladenovic et al. [33], using a
103 two-stage variable neighborhood search.

104 However, none of these TSP-TW solutions were applied to the HTPP. More
105 importantly, the constraints of the latter are different from those of standard TSP-
106 TW, since we wish to impose a maximum *difference* between the times of arrival
107 at certain node pairs, rather than a specified time interval for reaching each node.

108 The strength of a 3D-printed part is affected by many other factors besides
109 the filament-to-filament bonding in the infill, such as the time elapsed between
110 successive slices, the raster direction within each slice, and the extrusion temper-
111 ature [24, 2]. These issues will not be considered in this work.

112 1.1. Contributions

113 In this paper we describe HotFill, an algorithm that uses dynamic program-
114 ming to solve the hot tool-path problem for the special case of solid raster filling
115 (RF-HTPP). The algorithm was implemented in Python3 and is available as open

116 source [34]. Experiments show that it is fast and produces tool-paths that have
117 low fabrication time while satisfying the specified cooling time constraints.

118 *1.2. Structure of the paper*

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

127 1.3. *List of symbols*

128 ★[Stolfi:] *Completar e conferir as seções.*

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

133 ★[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
$R_{\text{COOL}}^{\text{MAX}}(P, \mathcal{C})$	maximum cooling time ratio among contacts \mathcal{C} with path P	2.2

RF-HTPP inputs

\mathcal{R}	set of infill raster elements	3.1
$\mathcal{L}(r)$	set of link paths that connect to raster r	3.1
\mathcal{C}	set of relevant contacts	3.1
s	number of scan-lines	3.1
$n = \#\mathcal{R}$	number of raster elements	3.1
$\mathbb{P}(\mathcal{R})$	set of all candidate paths using the rasters in \mathcal{R}	4.6
$\mathbb{V}(\mathcal{R}, \mathcal{C})$	paths in $\mathbb{P}(\mathcal{R})$ that satisfy the cooling limits of contacts \mathcal{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

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.

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 $k = 0, 1, \dots, n - 1$; where n is the number of moves, denoted by $\#P$. Therefore, each move must begin where the previous

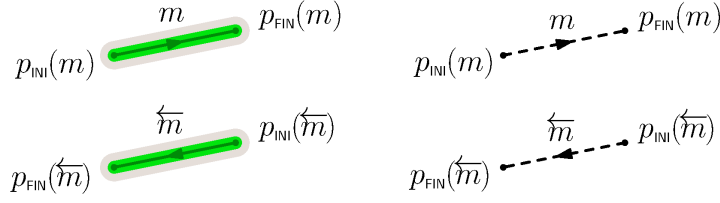


Figure 2. Graphical representation of a trace (left) and a jump (right), showing the effect of reversal on the attributes p_{INI} and p_{FIN} . The dark thin line in the trace is the nozzle's trajectory. The light gray area is the approximate extent of the material deposited by the nozzle. The green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

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 $k = 0, 1, \dots, n - 1$, where $k' = n - 1 - k$ and $n = \#P = \#\overleftarrow{P}$. See Figure 3 (right).

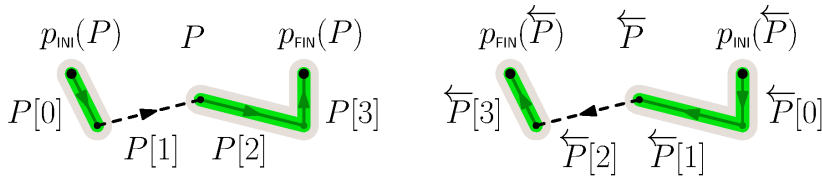


Figure 3. A path P with three traces and a jump (left) and its reversal \overleftarrow{P} (right).

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

167 $p_{\text{INI}}(P) = p_{\text{FIN}}(P)$ is some arbitrary point. A move m too may have zero length
 168 (that is, its midline may be a single point $p_{\text{INI}}(m) = p_{\text{FIN}}(m)$). However, a zero-
 169 length jump in a path makes no sense, and a zero-length trace in a path makes
 170 sense only if it is not preceded or followed by another trace, so that execution
 171 produces an isolated roundish drop of extruded material.

172 *Manufacturing time:*. We assume that there is a function $T_{\text{FAB}}(m)$, the *manufac-*
 173 *turing* or *fabrication time*, or *fab-time* for short, that gives the estimated time it
 174 will take for the printer to execute a move m . If m is a trace, $T_{\text{FAB}}(m)$ includes
 175 the time needed to extrude the material. For a jump, it includes the time needed
 176 to displace the nozzle from point $p_{\text{INI}}(m)$ to point $p_{\text{FIN}}(m)$ without extruding; and
 177 also the time needed to raise and lower the nozzle, and/or to suck back and re-feed
 178 the filament, if required by the printer. In both cases, $T_{\text{FAB}}(m)$ in our model should
 179 include the time needed to accelerate and decelerate the carriage, assuming that
 180 the nozzle is stationary or has to change direction at each end. The formulas we
 181 use for T_{FAB} are detailed in Section 9.

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

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

190 With this assumption, the *initial time* $T_{\text{INI}}(P, k)$ and the *final time* $T_{\text{FIN}}(P, k)$
 191 when the nozzle will start or finish the move $P[k]$, relative to the starting time of

192 P , are given by

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

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

193 It is convenient also to define $T_{\text{INI}}(P, n) = T_{\text{FAB}}(P)$ and $T_{\text{FIN}}(P, -1) = 0$; then
 194 $T_{\text{FIN}}(P, k - 1) = T_{\text{INI}}(P, k)$ for every move $P[k]$.

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

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

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

204 2.2. Cooling time constraints

205 *Contacts.*: We define a *contact* as a straight line segment between two adjacent
 206 traces that should be welded together. See Figure 4. For example, when com-
 207 pletely filling an area with horizontal traces, there will usually be a contact be-
 208 tween any two traces whenever the Y coordinates of their mid-lines differ by the
 209 common trace width λ_{FILL} , and their X ranges overlap.

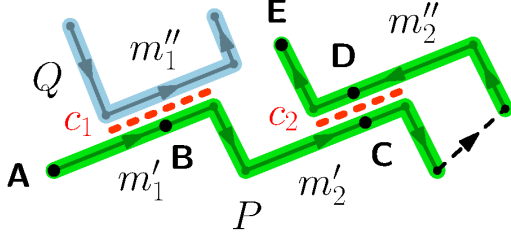


Figure 4. Graphical representation of two contacts c_1 and c_2 (red dashed lines) between traces m'_1 , m''_1 and m'_2 , m''_2 , of two paths, P (green) and Q (blue).
 ★[Neri:] *Por que nesta figura os traces são indicados como m? Como um m pode ser um trace ou um jump, não faz sentido ter um contato 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?*

210 We will denote by $\text{sides}(c)$ the pair of traces that surround the contact c . In Fig-
 211 ure 4, $\text{sides}(c_1)$ would be the pair of traces $\{m'_1, m''_1\}$. For any path P and any set
 212 \mathcal{C} of contacts, we will denote by $\text{contacts}(P, \mathcal{C})$ the elements of \mathcal{C} that have at
 213 least one side that is a trace P or its reversal.

214 *Cover times:.* We define $T_{\text{cov}}(m, u)$ as a function that gives the time interval be-
 215 tween the start of the extrusion of a trace m and the moment when the nozzle is at
 216 the point u' on the trace's mid-line that is closest to a given point u of the plane.
 217 See Section 9 for how this time can be estimated.

218 Moreover, if m is one of the side traces of a contact c , and P is a path such
 219 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
 220 $T_{\text{ini}}(P, k) + T_{\text{cov}}(P[k], u)$, where u is the midpoint of c . Note that the second term
 221 takes into account the orientation of m in P . In Figure 4, $T_{\text{cov}}(P, c_1, m'_1)$ is the

222 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}
 223 from **E** to **B**.

224 *Contact cooling time:*. If a path P contains both sides m' and m'' of a contact
 225 c , we define the *cooling time* $T_{\text{cool}}(P, c)$ of c in P as the absolute difference be-
 226 tween the two cover times, namely $|T_{\text{cov}}(P, c, m') - T_{\text{cov}}(P, c, m'')|$. In Figure 4,
 227 $T_{\text{cool}}(P, c_2)$ is the fab-time of P from **C** to **D**. It is an estimate of the interval of
 228 time elapsed between the deposition of material on the two sides of the contact;
 229 which is assumed to determine the strength of the corresponding physical weld.

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

235 *Cooling time limit:*. We assume that each contact c also has a *cooling time limit*
 236 $\tau(c)$, a user-specified maximum allowed value for $T_{\text{cool}}(P, c)$ in the final tool-path
 237 P . See Section 3. The *cooling time ratio* $R_{\text{cool}}(P, c)$ of c in any path P that closes
 238 it is the quotient $T_{\text{cool}}(P, c) / \tau(c)$; it is greater than 1 if and only if that limit would
 239 be violated if P were to be chosen as the tool-path. If there is no constraint on
 240 the cooling time limit, we may let $\tau(c)$ be $+\infty$ (so that $R_{\text{cool}}(P, c)$ will always be
 241 zero).

242 The *maximum cooling ratio* $R_{\text{cool}}^{\text{MAX}}(P, \mathcal{C})$ of a tool-path P and a set \mathcal{C} of con-
 243 tacts is the maximum value of $R_{\text{cool}}(P, c)$ among all contacts c in \mathcal{C} which have
 244 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 \mathcal{R} , a set \mathcal{C} of relevant contacts between those elements.

Each raster element is a single trace. For simplicity of exposition, we assume that every trace is horizontal and oriented from left to right. *★[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.*

269 ★[Stolfi:]Neri, não entendo porque é necessário padronizar o uso de parênteses
 270 quando se faz referência a figuras. Acho que a decisão tem que ser caso por caso,
 271 como para qualquer outro texto. Se achamos que olhar para a figura é importante
 272 para a compreensão do texto a seguir, o “veja” **não** deve ficar entre parênteses.
 273 Se “veja a figura” é só para o caso improvável de alguém ter dúvida, pode ser
 274 entre parênteses.

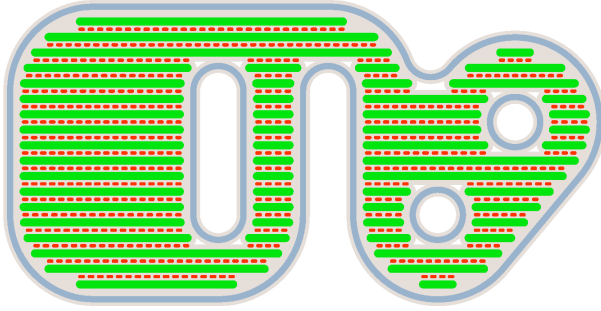


Figure 5. A possible input data set for the RF-HTPP problem, suitable for planning the solid raster filling of the slice shown in Figure 1. There are 56 raster elements (green thick segments) on 18 scan-lines. The total fab-time for those rasters, not counting the time to move between them, is 10.10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

275 Each raster element r in \mathcal{R} also has a set $\mathcal{L}(r)$ of associated *links*. A link is
 276 a short path that is to be included in the tool-path, in place of a jump, to connect
 277 other rasters (or their reversals) to r . Thus $p_{\text{FIN}}(S) = p_{\text{INI}}(S)$ for every link S in
 278 $\mathcal{L}(r)$. There is a separate set $\mathcal{L}(\overleftarrow{r})$ of links that can be used to connect other
 279 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.
 280 Normally $\mathcal{L}(r)$ and $\mathcal{L}(\overleftarrow{r})$ may have 0, 1, or 2 paths each.

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

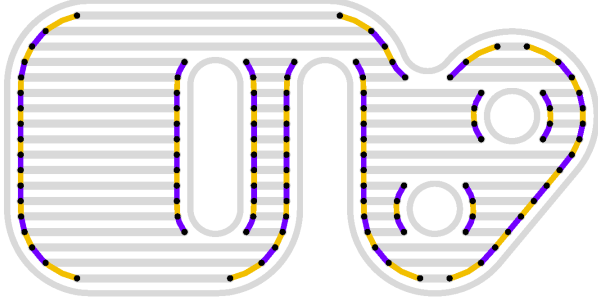


Figure 6. A possible collection of link paths for the input elements of Figure 5.

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

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

294 The contour paths in Figure 5 are shown only to indicate the shape of the slice,
 295 but are not part of the input data set; we assume that they will be fabricated before
 296 or after the whole raster filling. We also assume that the contacts between the

297 filling and the contour have no cooling time constraints, and therefore are omitted
 298 from \mathcal{C} .

299 3.2. Output

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

303 Any two raster elements that are consecutive in H must be connected by a
 304 link if available, or by a jump if there is no such link. If the rasters are r' and r''
 305 (in their original or reversed orientations), the link, if it exists, will be the only
 306 one that connects $p_{\text{FIN}}(r')$ to $p_{\text{INI}}(r'')$; that is, the only s such that $\overleftarrow{s} \in \mathcal{L}(\overleftarrow{r'})$ and
 307 $s \in \mathcal{L}(r'')$.

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

312 Ideally, among all the valid tool-paths, the solution should be the *best* one,
 313 meaning the one with smallest fabrication time. However, no efficient algorithm
 314 is known that will find this path in any case. Therefore, one can only expect
 315 the resulting path to be “good enough” for fabrication. We assume that the time
 316 T_{RAST} spent depositing the rasters is the same in any tool-path built from them (see
 317 Section 9); thus minimizing the total fab-time means minimizing the *connection*
 318 *time* $T_{\text{CONN}}(H)$, the time spent extruding links and executing jumps.

319 Using links instead of jumps will normally reduce the path’s fabrication time,
 320 and may also improve the adhesion between the filling and the contour by filling
 321 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 $\mathcal{R}_1, \mathcal{R}_2, \dots$, 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, \dots , 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 sentença 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 devêssemos

346 *calcular a verdadeira solução greedy e colocar aqui também. Mas não aguento*
347 *mais programar...*

348 4.3. Scan-line solutions

349 Another simple possible solution to the RF-HTPP is a *non-alternating scan-*
350 *line-order* (SCN) tool-path, which fabricates the scan-lines in order of increasing
351 *Y* coordinate, with the rasters each scan-line extruded in the same order and direc-
352 tion (all left-to-right, or all right-to-left). See Figure 7. Such a tool-path is likely to
353 be optimal or near-optimal with regards to contact cooling, in the sense of having
354 the minimum possible $R_{\text{COOL}}^{\text{MAX}}$. However, its fabrication time may be much larger
355 than that of the optimum valid path. ★[Neri:]*Por que não NSC e o outro ASC?*
356 *Não segue mais a lógica do nome em inglês?* ★[Stolfi:]*Pensamos nisso, mas teria*
357 *que mudar as tabelas, e deu uma preguiiça...* ★[Neri:]*É correto dizer que esta*
358 *solução não utiliza a lista de raster links? Se sim, talvez fosse bom inserir uma*
359 *frase sobre isso!* ★[Stolfi:]*Na verdade, do jeito que programamos, pode usar links*
360 *em alguns casos, porque é automático quando a path é montada com os rasters*
361 *na ordem escolhida. Por exemplo, se uma scanline termina no mesmo X que a*
362 *scanline seguinte começa, a solução SCN, pelas definições, pode usar um link*
363 *nesse ponto, em vez de um jump. Portanto prefiro nem mencionar esta questão.*
364 *Do jeito que está, está correto, mesmo que possa ter umas surpresas como essa...*

365

366 *Alternating scan-line solution:*. A slightly less trivial possible solution for the
367 RF-HTPP is an *alternating scan-line-order* (SCA) path, which also processes the
368 scan-lines in order of increasing *Y* coordinate, but reverses the order of rasters
369 and the direction of extrusion from one scan-line to the next. See Figure 8.

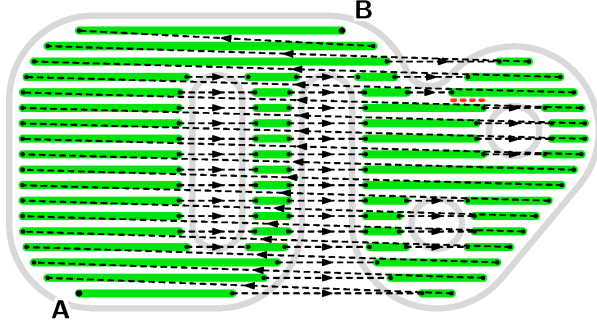


Figure 7. A non-alternating scan-line-order tool-path for the solid raster fill data of Figures 5 and 6. Its fab-time is 23.71 s, including 10.10 s of extrusion and 13.61 s of air time. The red dashed line is the contact with the largest cooling time, 1.67 s.

370 An alternating scan-line solution usually has much smaller fab-time than the
 371 non-alternating version, because the jumps between scan-lines are replaced by
 372 shorter jumps or by links. However, it is more likely to violate cooling constraints,
 373 if some cooling time limits are comparable to time required to extrude one scan-
 374 line. Note that, from Figure 7 to Figure 8, the maximum contact cooling time
 375 increased from 1.67 to 2.25 s.

376 4.4. Monotonic infill regions

377 The RF-HTPP is usually trivial if the input set \mathcal{R} has a single raster line on
 378 each scan-line. We call such a raster set *monotonic*, because it arises, in particular,
 379 if the infill region D is Y -monotonic in the geometric sense — meaning that every
 380 horizontal line intersects it in at most one line segment [35].

381 Therefore, if the raster set \mathcal{R} is monotonic, it is almost never necessary to
 382 use the HotFill algorithm described in Section 5. In that case, one can try an SCA
 383 (alternating scan-line) tool-path first, which will probably have fab-time close to



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.

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.

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

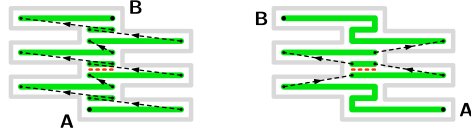


Figure 9. A monotonic instance of the RF-HTPP for which the maximum contact cooling time of the SCN solution at left (0.42 s) is higher than that of the SCA solution at right (0.40 s).

398 Figure 10.

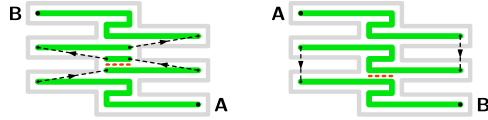


Figure 10. A monotonic instance of the RF-HTPP for which the fabrication time of the SCA solution at left (2.38 s) is significantly higher than that of the tool-path at right (2.03 s), which does not follow the scan-line order. The maximum contact cooling times are 0.40 s and 0.73 s, respectively.

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

403 4.5. Problem decomposition at monotonic sections

404 More generally, it is possible to split the RF-HTPP into smaller independent
 405 sub-problems if the input raster set contains monotonic sections. Specifically,
 406 suppose that there are one or more consecutive scan-lines that have a single raster
 407 each. Let r' and r'' be the lowest and highest of those rasters, respectively. Then

the problem can be split into three independent problems, with raster sets \mathcal{R}' , \mathcal{R}^* , and \mathcal{R}'' ; where \mathcal{R}' consists of r' and all rasters below it, \mathcal{R}'' is r'' and all the rasters above it, and \mathcal{R}^* is all the rasters between r' and r'' , including both. Let P' and P'' be RF-HTPP solutions for \mathcal{R}' and \mathcal{R}'' , respectively, and P^* be the greedy solution for \mathcal{R}^* starting at r' . If these partial solutions exist, a solution for the original input set \mathcal{R} may be obtained by concatenating P' , P^* , and P'' , or their reversals, removing the duplicate rasters r' and r'' .

This decomposition assumes that the path P' ends with raster r' , and that P'' will start with r'' . This will always be the case if the paths are computed with the algorithm of 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 \mathcal{R} ; with any gap filled by the appropriate link if it exists, or by a jump otherwise. We denote by $\mathbb{P}(\mathcal{R})$ the set of all those potential tool-paths.

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

A potential path is completely determined by an ordering and orientation of the raster elements in it. Therefore, $\#\mathbb{P}(\mathcal{R}) = 2^n n!$, where $n = \#\mathcal{R}$ is the number of raster elements. Clearly, the brute-force enumeration of all the paths

433 in $\mathbb{P}(\mathcal{R})$ is practically impossible, except for very small instances (with a couple
 434 dozen rasters at most).

435 5. HotFill: Finding a valid tool-path

436 The HotFill algorithm is specialized to find a valid tool-path for solid raster fill-
 437 ing. As observed in Section 4, the set \mathbb{P} of potential tool-paths is usually too large
 438 to be enumerated exhaustively. The HotFill algorithm gets around that problem by
 439 limiting the search to a proper subset of \mathbb{P} , that is expected to include tool-paths
 440 that are valid and have low enough fabrication times.

441 5.1. Cut-lines and bands

442 *Cut-lines:*. We define a *cut-line* as a horizontal line that runs between successive
 443 scan-lines. If the rasters of \mathcal{R} lie on s successive scan-lines, there are $s+1$ relevant
 444 cut-lines, from ℓ_0 (just below the lowest raster) to ℓ_m (just above the highest one).
 445 (See the blue dotted lines in Figure 11.) For any i, j with $0 \leq i < j \leq s$, we
 446 define the (i, j) -band $\mathcal{R}_{i,j}$ as the set of rasters between cut-lines ℓ_i and ℓ_j .

447 5.2. Band-paths and full-paths

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

453 The result of HotFill is built incrementally by concatenating one or more of
 454 those bandpaths, starting with some $(0, j)$ -bandpath; where the top of each band-
 455 path is the same cut-line as the bottom of the next bandpath. We define an (i, j) -

456 *fullpath* as being such a concatenation that ends with a (i, j) -bandpath. See Fig-
457 ure 11. A fullpath is valid if and only if all its bandpaths are valid, and it satisfies
458 the cooling time constraints of all contacts between its successive bands.

459 ★[Neri:]*Não seria band-path, para manter o padrão?*★[Stolfi:]*Podemos dis-*
460 *cutir isto. Um problema é que se for “(i, j)-band-path” ficam dois hffens; acho*
461 *estranho. Outro problema é que “full-path” pode confundir com a path completa*
462 *final H. Os dois termos são bem especializados e são usados só no algoritmo*
463 *HotFill; não seriam usados em outros algoritmos, mesmo usando a mesma notação*
464 *para moves, paths, etc. Poderia ser outra palavra em vez de “full”, mas se não*
465 *começar com “F” vai dar um trabalhão mudar as figuras...*

466 The solution returned by HotFill can be viewed as an elaboration of the scan-
467 line solutions, using bandpaths instead of individual scan-lines, in order to re-
468 duce the air time where possible. Indeed, a scan-line solution (alternating or non-
469 alternating) is a fullpath composed from bandpaths that span a single scan-line
470 each.

471 5.3. Dynamic programming

472 The HotFill algorithm considers at most two valid bandpaths for each (i, j) -
473 band, which are stored in the array elements $B_0[i, j]$ and $B_1[i, j]$. Either or both
474 of these elements may be set to Λ to signify that the bandpath was not created for
475 some reason. It also constructs at most two fullpaths $F_0[i, j]$ and $F_1[i, j]$ for each
476 (i, j) -band; where $F_0[i, j]$ is the best valid (i, j) -fullpath that ends with $B_0[i, j]$,
477 $F_1[i, j]$ is the best valid (i, j) -fullpath that ends with $B_1[i, j]$, and “best” means
478 “with minimum fab-time”. Either element is Λ if there is no such valid fullpath.
479 For this algorithm, it is convenient to let $T_{\text{FAB}}(\langle \rangle)$ be zero and $T_{\text{FAB}}(\Lambda)$ be $+\infty$.

480 The HotFill heuristic (Algorithm 1) uses the *dynamic programming* approach [36]

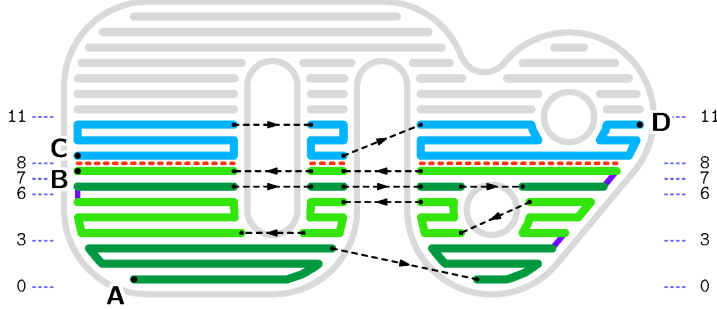


Figure 11. Illustration of the partial state at some point during the execution of the HotFill procedure (Algorithm 1) for the rasters and links of Figure 5 and 6. The colored paths comprise an $(8, 11)$ -fullpath, specifically $F_0[8, 11]$, consisting of a $(7, 8)$ -fullpath $F_1[7, 8] = \mathbf{A-B}$ (green and purple) and an $(8, 11)$ -bandpath $B_0[8, 11] = \mathbf{C-D}$ (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.

481 to find these optimal fullpaths. It enumerates all the (i, j) -bands from bottom to
 482 top, determining the optimal (i, j) -fullpaths $F_0[i, j]$ and $F_1[i, j]$ for each one. At
 483 the end, the desired solution will be one of the fullpaths $F_0[i, s]$ or $F_1[i, s]$, for all
 484 i in $\{0, 1, \dots, s - 1\}$ — whichever has the least fab-time.

485 For efficiency reasons, the algorithm only considers bandpaths that span at
 486 most a specified number μ of scan-lines. That is, it ensures and assumes that
 487 $B_z[i, j]$ is Λ if $j - i > \mu$. Therefore, each execution of the loops on i (in Al-
 488 gorithm 1) and k (in Algorithm 2) will perform at most μ iterations, instead of
 489 s .

Algorithm 1: HotFill

input : A set \mathcal{R} of horizontal rasters on s distinct scan-lines, with the associated links; a set \mathcal{C} of relevant contacts between them; and a band height limit μ .

output: A valid fullpath H that uses all the rasters \mathcal{R} and any applicable links; or Λ if it cannot find such a path.

```
1 for  $j$  in  $1, 2, \dots, s$  do
2   for  $i$  in  $0, 1, \dots, j - 1$  with  $j - i \leq \mu$  do
3     for  $z$  in  $\{0, 1\}$  do
4        $B_z[i, j] \leftarrow \text{BandPath}(\mathcal{R}, \mathcal{C}, i, j, z)$ 
5        $F_z[i, j] \leftarrow \text{MinFullPath}(\mathcal{R}, \mathcal{C}, \mu, i, j, F_0, F_1, B_z[i, j])$ 
6     end
7   end
8 end
9  $H \leftarrow \Lambda$ 
10 for  $i$  in  $0, 1, \dots, s - 1$  with  $s - i \leq \mu$  do
11   for  $z$  in  $\{0, 1\}$  do
12     if  $T_{FAB}(F_z[i, s]) < T_{FAB}(H)$  then  $H \leftarrow F_z[i, s]$  ;
13   end
14 end
15 return  $H$ 
```

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

- 492 1. if $i = 0$, the only (i, j) -fullpaths are the $(0, j)$ -bandpaths $B_0[0, j]$ and/or
493 $B_1[0, j]$, if they exist.

- 494 2. if $i > 0$, an (i, j) -fullpath P is a (k, i) -fullpath P' concatenated with an
495 (i, j) -bandpath B , either $B_0[i, j]$ or $B_1[i, j]$.
- 496 3. the path P above, if it exists, is valid if and only if P' and B are valid, and
497 the contacts between them (on the cut-line i) have their cooling constraints
498 satisfied.
- 499 4. the cooling times of these contacts depend only on the final (k, i) -bandpath
500 B of F'_0 (either $B_0[k, i]$ or $B_1[k, i]$) and the (i, j) -bandpath B .

501 It follows that the (i, j) -fullpath with minimum fab-time that ends with $B_0[i, j]$,
502 if it exists, must be a (k, i) -fullpath S with minimum fab-time, either $F_0[k, i]$ or
503 $F_1[k, i]$, concatenated with the (i, j) -bandpath $B_0[i, j]$, for some k in $0, 1, \dots, i -$
504 1 ; provided that the cooling constraints of the contacts between S and $B_0[i, j]$ are
505 satisfied by the concatenation of those two paths. The analogous conclusion holds
506 for the minimum (i, j) -fullpath that ends with $B_1[i, j]$.

507 5.4. The *MinFullPath* procedure

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

516 The number of non- Λ entries in each of the tables F_0 and F_1 is at most $\mu[(s -$
517 $\mu) + (\mu + 1)/2]$, which is less than $s\mu$. Figure 12 shows the typical situation

518 of the array F_0 before a call to `MinFullPath` with $\mu = 5$, $i = 8$, and $j = 11$,
 519 for the input data of Figures 5 and 6. Light gray is used for entries $F_0[i', j']$ that
 520 will never be set nor used because $i' \geq j'$ or $j' - i' > \mu$. Entries in red were
 521 set to Λ because all potential (i', j') -fullpaths ending with the bandpath $B_1[i', j']$
 522 were found to violate internal cooling constraints. Entries in black have yet to
 523 be computed. The remaining green entries $F_0[i', j']$ hold already computed valid
 524 (i', j') -fullpaths. The array F_1 has a similar appearance.

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

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

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

538 5.5. Choosing the bandpaths

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

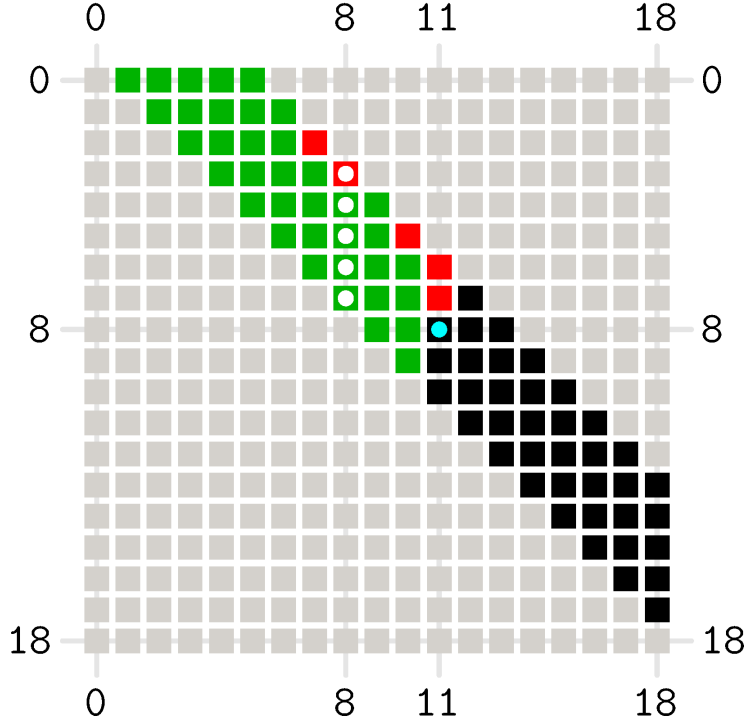


Figure 12. Typical status of the array F_0 upon entry to `MinFullPath` with parameters $\mu = 5$, row index $i = 8$, and column index $j = 11$, with the input data of Figures 5 and 6. For the color codes, see the text.

543 Finding the absolutely best (i, j) -bandpath is too hard; in fact, computing the
544 best possible valid $(0, s)$ -bandpath is the same as computing the best valid tool-
545 path for the input rasters, which, as we have noted, is still a practically unsolvable
546 problem. Therefore, `BandPath` must be some heuristic that considers only some
547 small subset of all possible paths that can be built from the rasters $\mathcal{R}_{i,j}$. The path it
548 returns must satisfy the cooling constraints of all internal contacts (between rasters
549 of $\mathcal{R}_{i,j}$). The procedure may return Λ if it cannot find such a path among the paths
550 it considers. However, if the band has a single scan-line, the procedure must return

551 the path consisting of all the rasters in that scan-line, sorted and oriented from left
 552 to right (if $z = 0$) or right to left (if $z = 1$). This ensures that, if the scan-line
 553 order paths SCN and/or SCA are valid, HotFill will consider them among all the
 554 possible fullpaths; and therefore it will never return a solution that has greater
 555 fab-time than those two.

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

561 *The bidirectional greedy bandpath.:* For this article, we have chosen the follow-
 562 ing heuristic for the BandPath procedure. When $z = 0$, the result of BandPath
 563 (that will become $B_0[i, j]$) starts with the leftmost raster on scan-line i (at the
 564 bottom of the band), and ends with the rightmost raster on scan-line $j - 1$, both
 565 oriented from left to right. When $z = 1$, the result (that will be $B_1[i, j]$) starts with
 566 the *rightmost* raster on scan-line i , and ends with the *leftmost* raster on scan-line
 567 $j - 1$, both oriented *from right to left*.

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

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

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

590 In particular, when the band has a single scan-line ($j - i = 1$), the bandpath
591 $B_0[i, j]$ will be the rasters on that scan-line oriented and concatenated from left to
592 right; and $B_1[i, j]$ will be the reverse of that path.

593 5.6. Sample solutions

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

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

604 Figures 13, 14, and 15 show the HotFill tool-paths with the cooling time limits
605 $\tau(c)$ of every contact c set to 3.5 s, 2.3 s, and 1.7 s, respectively. The red dashed
606 lines show the contacts with maximum cooling time.

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

611 Figure 16 shows the HotFill output when μ is set to 50 (which is the same as
612 not having any limit on band height) and all cooling time limits are set to 50 s
613 (which, being more than the total fab-time of the slice, is the same as not having
614 any cooling constraints at all). Note that the selected fullpath is still formed by
615 four bandpaths: (0, 15), (15, 16), (16, 17) and (17, 18). This result was found to
616 be equal or better than the fullpath consisting of a single (0, 18)-bandpath. This
617 solution should be compared to the typical result of path-planing software, Fig-
618 ure 1, which has slightly better fab-time (13.51 s vs. 13.59 s).

619 For comparison, Figure 17 shows the HotFill output with the same cooling
620 limits but with μ set to 5. Note that the bandpaths have been limited to 5 scan-
621 lines, and thus the result consists of five bandpaths: (0, 3), (3, 8), (8, 13), (13, 17)
622 and (17, 18). Still the fab-time (14.06 s) is not much worse than the one obtained
623 with $\mu = 50$ (13.59 s) and that of the path of Figure 1 (13.51 s).

Algorithm 2: MinFullPath

input : A set \mathcal{R} of horizontal rasters on s distinct scan-lines, with the associated link paths; a set \mathcal{C} of relevant contacts between them; a maximum band height μ ; a pair of cut-line indices i, j with $0 \leq i < j \leq s$; two $(s+1) \times (s+1)$ arrays F_0, F_1 whose elements are fullpaths or Λ ; and an (i, j) -bandpath B or Λ .

output: A (valid) (i, j) -fullpath F that uses all the rasters \mathcal{R} up to cut-line ℓ_j , ends with the bandpath B , and has minimum fab-time among such fullpaths; or Λ if there is no such valid fullpath.

```
1  $F_* \leftarrow \Lambda$ 
2 if  $B \neq \Lambda$  then
3   if  $i = 0$  then
4      $F \leftarrow B$ 
5   else
6     for  $k$  in  $0, 1, \dots, i-1$  with  $i-k \leq \mu$  do
7       for  $z$  in  $\{0, 1\}$  do
8         if  $F_z[k, i] \neq \Lambda$  then
9            $T \leftarrow \text{Concat}(F_z[k, i], B)$ 
10          if  $\text{ValidPath}(T, \mathcal{C})$  then
11            if  $T_{FAB}(T) < T_{FAB}(F_*)$  then  $F_* \leftarrow T$ ;
12          end
13        end
14      end
15    end
16  end
17 end
18 return  $F_*$ 
```

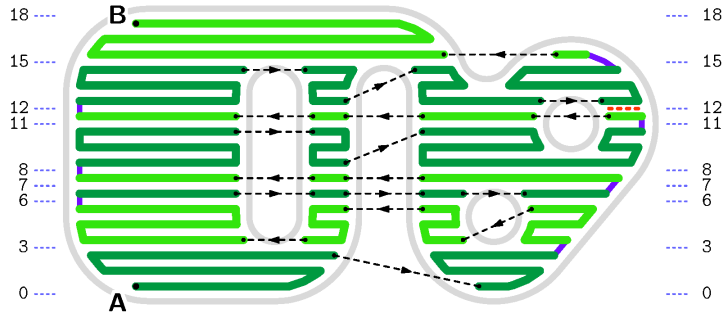


Figure 13. The HotFill solution with $\mu = 5$ and all cooling time limits set to 3.5 s.

The blue horizontal dotted lines are the cut-lines that separate the bandpaths 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.

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

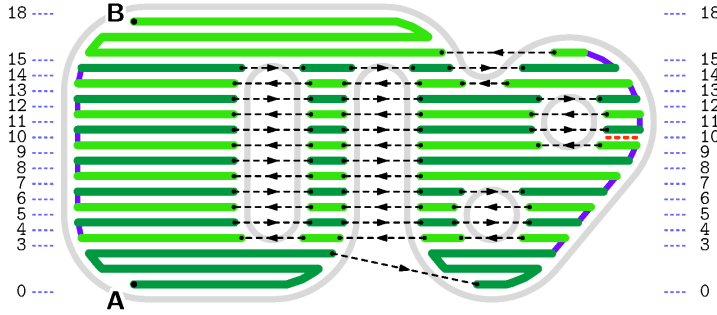


Figure 14. The HotFill solution with $\mu = 5$ and all cooling time limits set to 2.3 s.

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.

635 *computação ;-)* Tanto assim que não tem nenhum experimento de laboratório – só
 636 *algoritmos e testes no computador...*

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

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

643 We can reduce that cost substantially by replacing the path tables F_0 and F_1
 644 by four arrays T_0 , T_1 , K_0 , and K_1 , with the same size $(s+1) \times (s+1)$. Each entry
 645 $T_z[i, j]$ should be set to $+\infty$ if the computed $F_z[i, j]$ is Λ , otherwise it should be
 646 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
 647 bit. It should be set to $(-1, 0)$ if $F_s[i, j]$ is Λ or $i = 0$, otherwise it should be such
 648 that $F_z[i, j]$ was formed by concatenating $F_{z'}[k, i]$ with $B_0[i, j]$.

649 With these tables, it is not necessary to actually build and store the paths

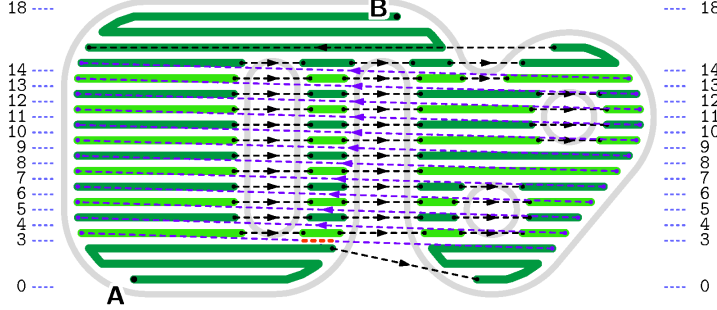


Figure 15. The HotFill solution with $\mu = 5$ and all cooling time limits set to 1.7 s. The largest contact cooling time (red dashed line) is 1.69 s. The total fab-time is 22.24 s including 10.66 s of extrusion and 11.58 s of air time.

650 $F_0[i, j]$ and $F_1[i, j]$. The tables T_0 , T_1 , K_0 , and K_1 can be filled instead by slight
 651 modifications of Algorithms 1 and 2. The final tool-path H can be reconstructed,
 652 at the end of HotFill, by following the pointers $K_z[i, j]$ back to $(-1, 0)$ and con-
 653 catenating the bandpaths $B_z[i, j]$ identified by them.

654 *Fast validity check.*: The ValidPath procedure needs only check the cooling times
 655 of the contacts on cut-line i by the paths $F_z[i, j]$, which it can compute from the
 656 cover times of those contacts by the bandpaths $B_{z'}[k, i]$, and $B_z[i, j]$.

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

661 Specifically, for each contact c on cut-line i , we can define $T_{\text{COOL}}^{\text{PRED}}(B_*, c)$ as
 662 $T_{\text{FAB}}(B_*)$, if B_* does not cover c , or $T_{\text{COV}}(B_*, c)$, if it does. For each contact c on
 663 cut-line j , we can define $T_{\text{COOL}}^{\text{PRED}}(B_*, c)$ to be 0, if B_* does not cover c , or $T_{\text{COV}}(\overleftarrow{B_*}, c)$,

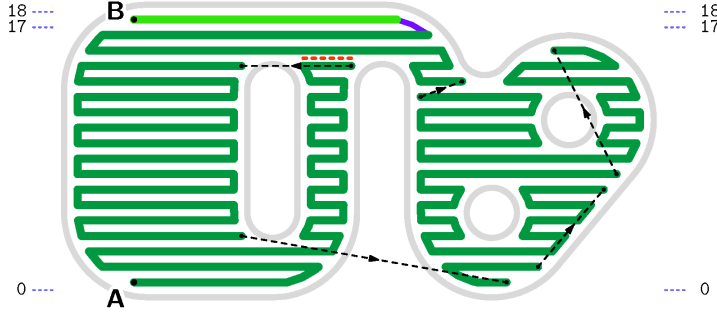


Figure 16. The HotFill solution with $\mu = 50$ and all cooling time limits set to 50 s. The largest contact cooling time (red dashed line) is 9.60 s. The total fab-time is 13.59 s including 12.55 s of extrusion and 1.04 s of air time.

664 if it does. The BandPath procedure can stop and return Λ as soon as $T_{\text{COOL}}^{\text{PRED}}(B_*, c)$
 665 exceeds the cooling time limit $\tau(c)$ for any of those contacts. As the width $j - i$ of
 666 the band increases, this early termination of BandPath will be increasingly likely
 667 to happen.

668 6.2. Programming improvements

669 Even with the algorithmic improvements above, a straightforward implemen-
 670 tation of HotFill would waste a considerable amount of time computing functions
 671 such as $T_{\text{COV}}(P, c, r)$, $T_{\text{COOL}}(P, c)$, $R_{\text{COOL}}^{\text{MAX}}(P, \mathcal{C})$ and so on. For example, computing
 672 $T_{\text{COV}}(P, c, r)$ in the obvious way would require scanning the moves of the path P
 673 to locate the trace r , while computing and adding the fabtime of all moves of P up
 674 to that one; which would make the cost of T_{COV} proportional to the number $\#P$ of
 675 moves in the path, and the cost of $R_{\text{COOL}}^{\text{MAX}}(P, \mathcal{C})$ proportional to $\#P \times \#\mathcal{C}$. These
 676 costs in turn could make the cost of the innermost loop of HotFill proportional to
 677 size n of the input data, or even to n^2 .

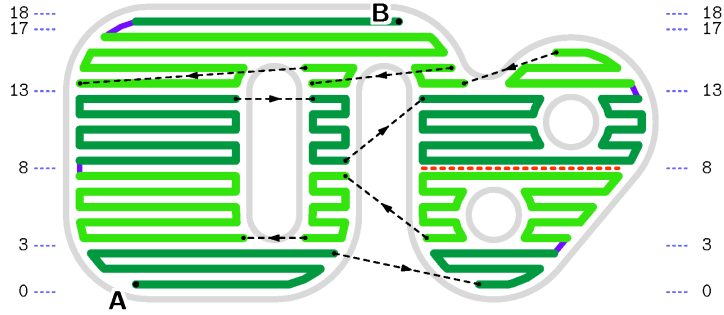


Figure 17. The HotFill solution with $\mu = 5$ and all cooling time limits set to 50 s. The largest contact cooling time (red dashed line) is 6.50 s. The total fab-time is 14.06 s including 12.44 s of extrusion and 1.62 s of air time.

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

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

689 This trick is used for paths too. A path P can be represented as a pair (P, b)
 690 where P is an object of class **Path** that represents the path in some arbitrary native
 691 orientation, and the bit b tells whether P is P or its reversal. Then, for example,
 692 $P[k]$ would be $P.mv[k]$ if $b = 0$, and $\overleftarrow{P.mv[k']}$ if $b = 1$; where n is the number of

693 elements of $P.mv$, and $k' = n - 1 - k$.

694 *Pre-computed timing functions:*. The fabrication time $T_{\text{FAB}}(r)$ of a move r can be
 695 computed only once, when the **Move** object r is created, and stored as a field
 696 $r.fabtime$.

697 Likewise, when a path object P is created, the values of $T_{\text{FIN}}(P, k)$ for all k
 698 are stored in a table in the object record, namely $P.tfin[k]$. Then the functions
 699 $T_{\text{FAB}}(P)$, $T_{\text{INI}}(P, k)$, and $T_{\text{FIN}}(P, k)$, for $P = \mathbf{P}$ or $P = \overleftarrow{\mathbf{P}}$, can be computed from
 700 that list — in constant computing time, irrespective of the length of the path.

701 *Contacts:*. Each contact c can be represented by an object c of class **Contact**,
 702 that contains the two endpoints $c.end[0]$ and $c.end[1]$ and pointers $c.side[0]$ and
 703 $c.side[1]$ to the incident **Move** objects. The order of both pairs is irrelevant. The
 704 object c would have fields $c.tcov[i]$ holding the precomputed value of the cover
 705 time of the contact by the trace that is side i , in its native orientation; that is,
 706 $T_{\text{FAB}}((c.side[i], 0), s)$ where s is the contact's midpoint.

707 Similarly, the cover times of each contact c on cut-line i by the bandpath
 708 $B_z[i, j]$ can be computed when that bandpath is constructed, and can be stored
 709 in a list $B.tcovs[0]$ of the **Path** object. Another list $B.tcovs[1]$ would hold the
 710 cover times by $B_z[j, i]$ of the contacts on cut-line j

711 *Locating contacts and rasters:*. Another potential waste of computing time is
 712 scanning the list \mathcal{C} to find the contacts that are closed or covered by a path, such as
 713 $B_0[i, j]$ or $B_1[i, j]$. This time can be reduced by splitting the list \mathcal{C} into s separate
 714 lists $C[0], C[1], \dots, C[s]$, where $C[i]$ has all the contacts on cut-line i . Similarly,
 715 when locating the rasters in an (i, j) -band, we can save time by splitting the raster
 716 set \mathcal{R} into $s - 1$ lists $R[0], R[1], \dots, R[s - 1]$, one per scan-line.

717 *Locating links:*. For efficiency, the lists $\mathcal{L}(P)$ and $\mathcal{L}(\overleftarrow{P})$ of link paths that con-
 718 nect to the endpoints of any partial tool-path P handled by HotFill — specifically
 719 each of the initial rasters, and each bandpath or fullpath P (including) — should
 720 be attributes of the relevant path object P . Then the Concat function then can
 721 quickly locate the bridging link by looking for a matching entry in the lists $\mathcal{L}(\overleftarrow{P})$
 722 and $\mathcal{L}(P'')$, stored in the objects P' and P'' . Note that these lists typically will
 723 have two entries or less, so this search will be very fast.

724 6.3. Computing time analysis

725 We can assume that the maximum band width μ is at most equal to the num-
 726 ber of scan-lines s in the input. The procedures BandPath and MinFullPath will
 727 be called a $2(s - \mu)\mu + \mu(\mu + 1)$ times each, which less than $2s\mu$. With the
 728 above optimizations, the asymptotic worst-case computing cost [37] for each call
 729 to MinFullPath will be $\Theta(\mu)$, arising from the loop on k . Each call to BandPath
 730 is expected to cost $\Theta((n_{i,j})^2)$ for the construction of the bandpaths $B_0[i, j]$ and
 731 $B_1[i, j]$; where $n_{i,j} = \#\mathcal{R}_{i,j}$ is the number of rasters in the band. If we assume
 732 that there is an $O(1)$ upper bound on the number of rasters per scan-line, then n_{ij}
 733 will be $\Theta(\mu)$, and the computing cost of BandPath will be $\Theta(\mu^2)$.

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

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

742 values of μ .

743 7. Experiments and results

744 In this section we report some computational experiments that aim to demon-
745 strate and quantify certain aspects of the HotFill algorithm. We used a Python
746 3 implementation of the algorithm [34] that incorporates the optimizations de-
747 scribed in Section 6.

748 The first set of tests aims to show that HotFill does achieve its goal, namely
749 find a tool-path H that satisfies the specified cooling time constraints, if they can
750 be achieved at all. See Section 7.2.

751 The second batch of tests aims to demonstrate that the imposition of reason-
752 able cooling time constraints, which should improve the mechanical resistance of
753 the part, generally has a small impact on the fabrication time T_{FAB} of the tool-path.
754 See Section 7.3.

755 The third batch of tests aims to analyze the relationship between the input
756 parameters – especially the maximum band width μ – and the computation time
757 of HotFill. See Section 7.4.

758 In all tests, the parameter $\tau(c)$ of every contact c was set to the same value
759 Δ . Thus the maximum relative cooling time $R_{\text{COOL}}^{\text{MAX}}(H, \mathcal{C})$ is $T_{\text{COOL}}^{\text{MAX}}(H, \mathcal{C})/\Delta$. For
760 comparison, in each set of tests we show also the attributes of the tool-paths pro-
761 duced by two traditional tool-path planning programs, RP3 [38] and Slic3r [4]
762 (which produce tool-paths according to their own criteria, without taking the cool-
763 ing constraints into account), as well as the simple scan-line-order tool-paths, al-
764 ternating (SCA) and non-alternating (SCN).

765 In order to make the fab-time comparisons with Slic3r meaningful, the nomi-

766 nal contour offset and raster spacing parameters of that program had to be adjusted
767 so that, on the average, the number and total length of the raster lines it used were
768 as close as possible to those used by RP3 and HotFill.

769 ★[Stolfi:]*Listar os parâmetros de offset etc fornecidos ao RP3.*

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

771 7.1. Test data

772 *STL models:*. For testing, we obtained STL models for several objects where low
773 mechanical strength could be an issue, such as human prostheses, mechanical
774 parts, and coat hangers. Some models were created by ourselves, some were
775 obtained from a public repository [39]. From each object we selected a slice that
776 was representative of the majority of the cross-sections.

777 We tried to choose objects and slices such that the polygon D (the region to
778 be filled) was substantially non-monotonic in either the X or Y direction (that is,
779 where most scan-lines would have two or more raster lines); since, as observed
780 in Section 4.4, there is no point in using the HotFill if D is monotonic. For this
781 reason, we did not bother to apply the decomposition into sub-problems described
782 in that section.

783 These objects and the outlines of the selected slices are shown in Figure 18.
784 For all models the length unit of the STL file was assumed to be millimeters,
785 except for **runleg** which was assumed to be in inches (and thus was scaled by
786 25.4).

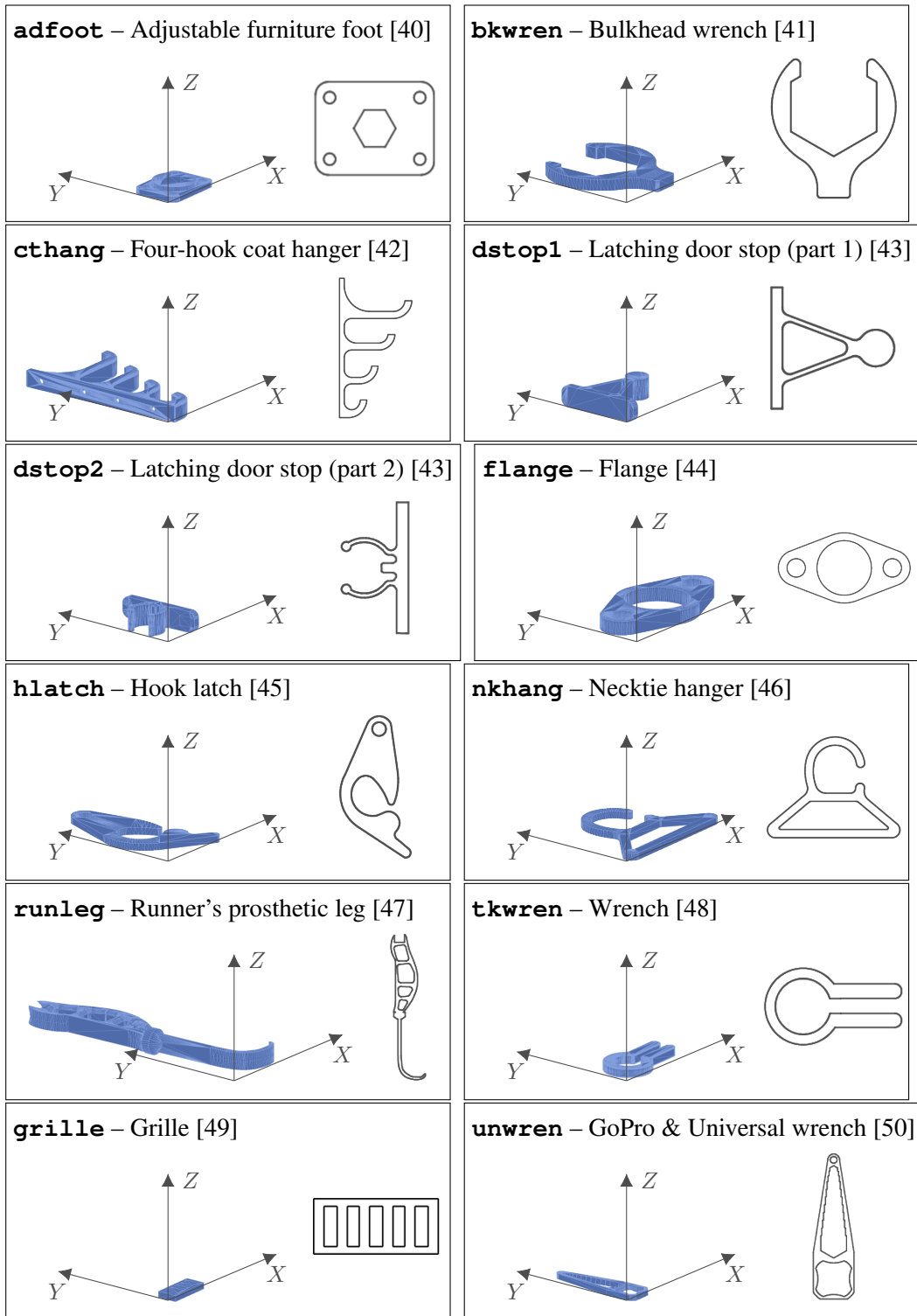


Figure 18. The STL models and slice outlines used in the tests (not to the same scale).

787 *Input datasets*:. For each test slice, we performed two preliminary tests of HotFill
788 algorithm, with the slice rotated by $\theta = 0$ or $\theta = 90$ degrees. The RP3 software
789 was used to extract the polygonal outline D of the area to be filled, rotated by the
790 specified angle θ , and to compute the endpoints of the raster lines, with assumed
791 slice thickness $\sigma = 0.25$ mm and scan-line spacing (nominal raster width) $\lambda_{\text{FILL}} =$
792 0.4 mm. The RP3 software also computed the link paths that could be used to
793 connect the endpoints of every pair of adjacent rasters, if they were to be extruded
794 in sequence with opposite orientations.

795 Specifically, the polygon D was the result of applying an inwards offset of
796 $\delta = 0.15$ mm to the slice's outline, to account for the thickness of the contour
797 trace. The trajectory of each link path was the piece of the boundary of the poly-
798 gon D between the relevant raster endpoints. *★[Stolfi:]Esta é uma propriedade*
799 *exclusiva dos dados que usamos nos testes, só porque pegamos os links do RP3 e*
800 *ele faz assim. O algoritmo e o programa não fazem nenhuma suposição sobre os*
801 *links. Poderiam por exemplo ser simples traços retos, o que economizaria tempo*
802 *mensurável. Ou poderiam ser pedaços de D , mas simplificados.* The RP3 pro-
803 gram was extended to output the raster and link information to a text file, that was
804 read by the HotFill implementation. *★[Stolfi:]Cada link path está pendurada em*
805 *dois rasters. Mas, do jeito que paths são representadas, o custo dessa duplicação*
806 *é pequeno; a maior parte da informação (pontos, tempos, etc) é compartilhada*
807 *entre S e \overleftarrow{S} . Aliás esta é uma das razões para manter a seção “Program im-*
808 *provements” no artigo: o gasto de memória seria muitas vezes maior sem esses*
809 *truques.*

810 In the interest of brevity, we generally show the results for only one of the
811 two datasets ($\theta = 0$ or $\theta = 90$ degrees) created from each STL model, namely

812 the one which turned out to be the most challenging for HotFill (in the sense of
813 Section 7.3). However we retained both datasets for the slice **runleg**, to show
814 the impact of raster orientation on the attributes of the resulting tool-paths.

815 The parameters of the selected datasets are presented in Table 1. The total area
816 to be filled, in mm^2 , is approximately the total raster length L_{TOT} times the scanline
817 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	cthange: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

818 7.2. Achieving the cooling constraints

819 The tests in this section aim to show that HotFill does achieve its goal; namely,
820 find a viable tool-path H that satisfies the specified cooling time constraints (which
821 in this case are $T_{\text{COOL}}(H, c) \leq \Delta$ for all $c \in \mathcal{C}$). The results are shown in Table 2.
822 ★[Neri:]*O objetivo já foi dito logo acima. Sugiro retirar ou reescrever! Sugestão*
823 *de alteração de frase:* [The HotFill results finding a viable tool-path H that sat-
824 isfies the specified cooling time constraints, which are $T_{\text{COOL}}(H, c) \leq \Delta$ for all
825 $c \in \mathcal{C}$, are shown in Table 2.] ★[Stolfi:]*Aquela frase na introdução da seção está*
826 *muito longe e é muito sucinta. Acho que precisa explicar aqui claramente o ob-*
827 *jetivo desta série de testes* The entries marked “—” in the table are cases where
828 HotFill failed to find a viable tool-path.

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

Dataset	Slic3r	RP3	SCN	SCA	HotFill with various values of Δ											
					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

829 *Discussion:*. As discussed in Section 4.3, the non-alternating scan-line solution
830 (SCN) usually minimizes the maximum contact cooling time $T_{\text{COOL}}^{\text{MAX}}$. It follows that
831 the $T_{\text{COOL}}^{\text{MAX}}$ of the SCN path is usually the minimum value of the cooling time limit Δ
832 for which a solution exists. That observation is confirmed by Table 2. Indeed, in
833 those tests, the HotFill procedure always found a solution whenever the specified
834 Δ was higher than the SCN $T_{\text{COOL}}^{\text{MAX}}$. For example, the SCN path for the dataset
835 **bkwren:0** has $T_{\text{COOL}}^{\text{MAX}} = 1.9\text{ s}$, and the procedure succeeded once Δ was above
836 that value (2.0 s). However, for the **cthang:90** dataset, HotFill managed to find
837 a solution even for $\Delta = 4.0$, which was lower than then $T_{\text{COOL}}^{\text{MAX}}$ of the SCN path.

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

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

849 For every dataset there is a maximum value of Δ beyond which every band-
850 path and fullpath considered by HotFill will be valid, as if there was no cooling
851 constraint (that is, as if Δ was $+\infty$). Beyond that point, HotFill will return always
852 the same tool-path, the one with minimum fab-time. Typically this path will still
853 have better cooling time than the Slic3r and RP3 tool-paths, and its fab-time will

854 not be much worse (see Section 7.3).

855 7.3. Impact on fabrication time

856 The tests in this section aim to show the impact of the cooling time constraints
857 on the fabrication time $T_{\text{FAB}}(H)$ of the computed tool-path. The results are shown
858 in Table 3. *★[Neri:]O objetivo já foi dito logo acima. Sugestão de frase: [The*
859 *results of the impact of the cooling time constraints on the fabrication time $T_{\text{FAB}}(H)$*
860 *of the computed tool-path are shown in Table 3.] ★[Stolfi:]Mesma resposta da*
861 *seção anterior.* For each dataset, besides the absolute fab-time T_{FAB} in seconds
862 of each reference algorithm (Slic3r, RP3, SCN, SCA), we also give the absolute
863 fab-time $T_{\text{FAB}}^{\text{HF}}$ of HotFill for each limit Δ , and the percent increase $D_{\text{FAB}}^{\text{HF}}$ of that
864 fab-time relative to $T_{\text{FAB}}^{\text{REF}}$, the smallest between the Slic3r and RP3 fab-times. The
865 total extrusion time T_{RAST} for the input rasters, which is the ideal lower bound for
866 $T_{\text{FAB}}(H)$, is also shown for reference. Each test was performed with $\mu = 20$.

867 The fabrication and cooling times of all tool-paths produced by HotFill, RP3,
868 and Slic3r were computed by the same Python program from the data extracted
869 from the resulting g-code files, according to the formulas in Section 9 with the fol-
870 lowing parameters: acceleration and deceleration, 3000 mm/s²; maximum nozzle
871 speed during extrusion, 40 mm/s; maximum nozzle travel speed during jumps,
872 130 mm/s; Jump start/end penalty, 0.05 s. These parameter values were inferred
873 from the actual fab-times of other tool-paths on the 3D Cloner model DH+ printer [51].

Table 3. Fabrication times $T_{\text{FAB}}(H)$ 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.

Dataset	T_{RAST}	Slic3r	RP3	SCN	SCA	HotFill for various values of Δ									
						1.4	2.0	2.8	4.0	5.7	8.0	11.3	16.0	32.0	64.0
1 dstop2:90	23.5	28.8	28.4	61.1 +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%

874 The dependency of $T_{\text{FAB}}(H)$ on the cooling limit Δ , for each of the test parts,
 875 is plotted on Figure 19. In order to magnify the variations and to make the plots of
 876 different datasets more comparable, the vertical axis is the percent increase in the
 877 connection time $T_{\text{CONN}} = T_{\text{FAB}} - T_{\text{RAST}}$, the air time plus fab-time of links, compared
 878 to that of the reference tool-path: namely, $R_{\text{CONN}}^{\text{HF}} = T_{\text{CONN}}^{\text{HF}} / T_{\text{CONN}}^{\text{REF}}$, where $T_{\text{CONN}}^{\text{HF}}$ is the
 879 connection time of the HotFill path, and $T_{\text{CONN}}^{\text{REF}}$ is the same value for the reference
 880 tool-path (by RP3 or Slic3r, as in Table 3). Thus a value of 100% means that the
 881 Hotfill path is as efficient as the reference path, while a value of 200% means that
 882 it spends twice as much time with links and jumps.

883 It must be noted that the relative *total* fab-time $T_{\text{FAB}}^{\text{HF}} / T_{\text{FAB}}^{\text{REF}}$ are usually much
 884 smaller than $R_{\text{CONN}}^{\text{HF}}$. For example, according to Table 3, for $\Delta = 8$ s and $\mu = 20$,
 885 the HotFill tool-path of that dataset has $T_{\text{CONN}}^{\text{HF}} = 266.9 - 150.1 = 116.1$ s, which
 886 is about 7 times the connection time of the reference tool-path $T_{\text{CONN}}^{\text{REF}} = T_{\text{SLIC3R}}^{\text{REF}} =$
 887 $165.8 - 150.1 = 15.7$ s; that is, $R_{\text{CONN}}^{\text{HF}} \approx 700\%$. However, the ratio of the total
 888 fab-times is only $266.9/165.8 = 161\%$ (+61% in Table 3).

889 Figure 19 also shows the dependency of fab-time on the maximum band width
 890 μ . The lower left plot of that figure is for $\mu = 20$, the value used in Table 3.

891 *Discussion:*. In Table 3, as in Table 2, we observe that HotFill produces a solution
 892 whenever the specified cooling time limit Δ is higher than the max cooling time
 893 $T_{\text{COOL}}^{\text{MAX}}$ of the SCN path (given in Table 2).

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

898 When the cooling constraints are not too restrictive (say, $\Delta \geq 8$ 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 $\Delta = 8$ s was the **runleg:90** dataset (number 13, purple in the figure) for which the HotFill path had 61% higher fab-time (266.9 s) than Slic3r (165.8 s). For $\Delta = 16$ 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 $\Delta \geq 128$ 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 $\#\mathcal{R}$, 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.

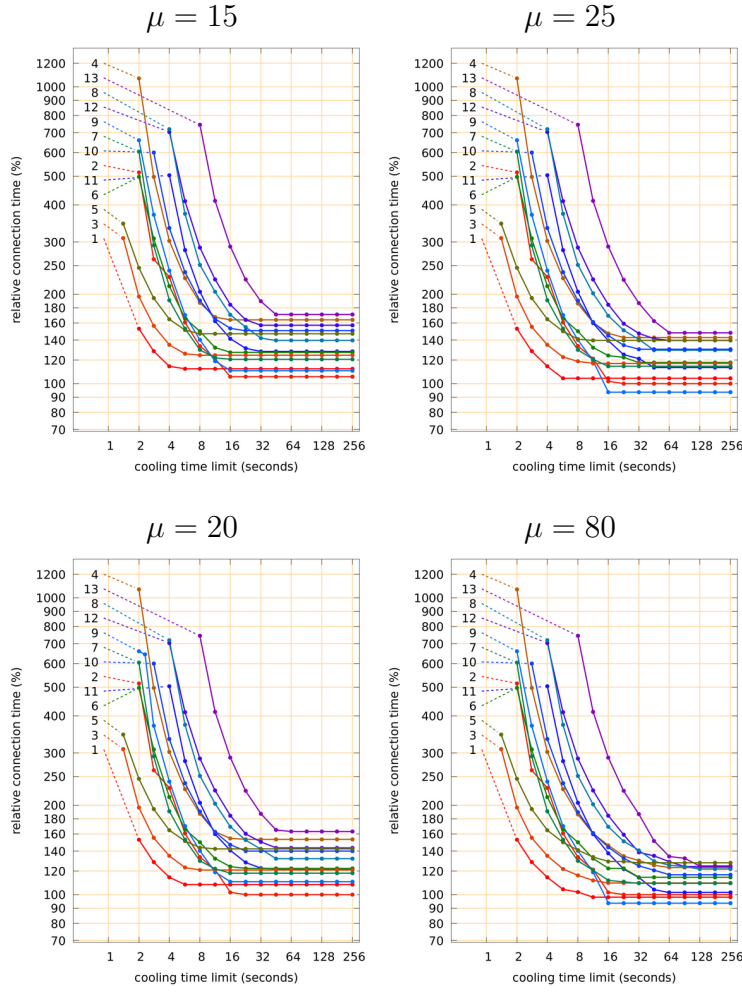


Figure 19. Relative connection time $R_{\text{CONN}}^{\text{HF}}$ as a function of the cooling time limit Δ , for different values of the maximum band width μ . Each curve is a different dataset. The numbers refer to Table 1.

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

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

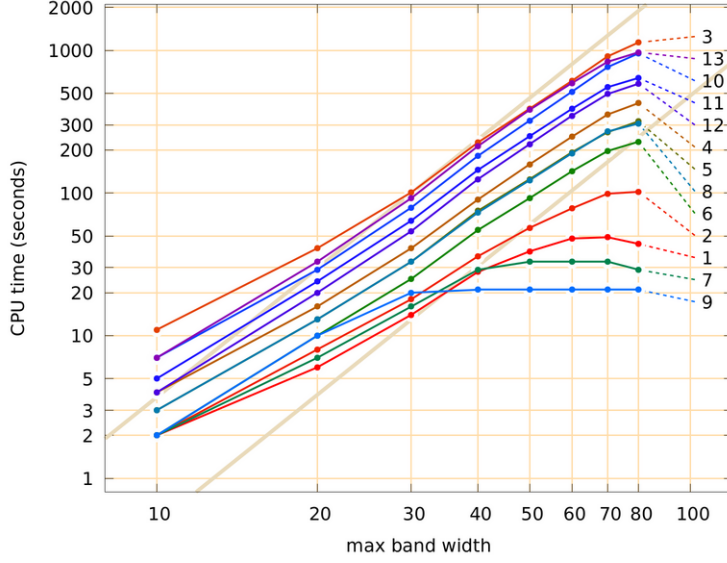


Figure 20. Computing time T_{CPU} as a function of the maximum band width μ , for $\Delta = 64.0\text{s}$ and the datasets listed in Table 4. Each graph is a different dataset, with colors assigned arbitrarily. The broad tan lines are the plots of $A\mu^3$ for $A = 0.00370$ and $A = 0.000481$.

924 *Discussion:*. As observed in Section 6.3, the running time of HotFill should be
 925 dominated by a term that grows proportionally to $s\mu^3$ or $n\mu^3$ — as long as Δ is
 926 large enough, there is a small limit to the number of rasters per scan-line, and
 927 $\mu \ll s$; where s is the number of scan-lines, and $n = \#\mathcal{R}$ is the number of raster
 928 elements.

929 As shown in Figure 20, the measured computing times in Table 4 generally
 930 grow in proportion to μ^3 for μ between 30 and 40. For values of μ below 30,
 931 growth is slower because the terms of lower order are still significant. At the
 932 other end of the range, growth slows down again when μ approaches or exceeds
 933 s , as predicted by the analysis. This effect is evident for datasets that have fewer

934 than 80 scan-lines, like **unwren:90** (graph 7, $s = 49$) and **grille:0** (graph 9,
935 $s = 35$).

936 Figure 20 also indicates that, for any fixed μ , the computation time increases
937 with increasing n and s , as expected. The largest times are seen for **runleg:0**
938 (graph 3, $n = 851$, $s = 579$), **runleg:90** (graph 13, $n = 443$, $s = 135$), and
939 **cthang:90** (graph 10, $n = 398$, $s = 160$)

940 *7.5. Sample HotFill outputs*

941 Figures 21, 22 and 23 show some of the infill tool-paths generated by HotFill
942 for some of the test datasets of Table 1. The constituent bandpaths are indicated
943 by different colors.

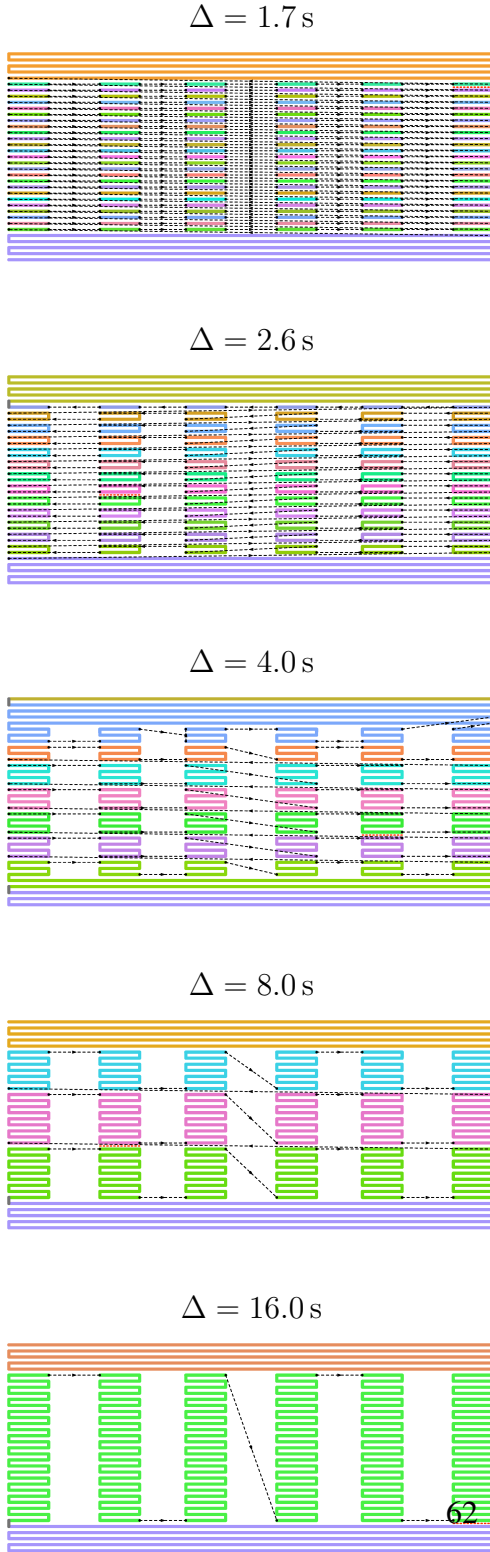


Figure 21. Tool-paths produced by HotFill for the **grille:0** dataset, with $\mu = 40$ and various cooling time limits Δ . Each of the bandpaths $B_z[i, j]$ that were joined to produce that tool-path is rendered in a different color. The links used

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

956 The second tool-path from the top was obtained with $\Delta = 2.6\text{s}$, slightly
 957 greater than the $T_{\text{COOL}}^{\text{MAX}}$ of the alternating scan-line solution SCA. HotFill did con-
 958 sider the SCA tool-path, but found a slightly better one, shown in the figure. It
 959 executes two rasters in each vertical bar before moving to the next bar. Compared
 960 to the SCA solution, this strategy requires only half as many jumps between the
 961 vertical bars. On the other hand, the cooling time constraints between these band-
 962 paths force them to be executed all in the same general direction (from right to
 963 left), connected by long jumps. Still, the overall fab-time of the HotFill tool-path
 964 (39.32 s) is less than that of the SCA solution (41.51 s).

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

969 half-paths meet.

970 The last three entries, for $\Delta = 4.0 = 8.0$, and 16.0 s, show how HotFill man-
971 aged to reduce the fab-time even further (to 31.30 s, 26.57 s and 24.43 s, respec-
972 tively) by using wider bandpaths. The last one could have used a single bandpath,
973 instead of the three shown at right; however the resulting fullpath would have been
974 the same.

975 Figures 22 and 23 shows HotFill results for two other datasets, **cthang:90**
976 and **dstop1:0**, with $\mu = 40$ and various values of Δ . In both cases, notice how
977 bandpath widths are smaller where the infill has longer and/or more numerous
978 per scan-line, but generally increase as Δ increases. On Figure 22, the path for
979 $\Delta = 32$ s is visibly affected by the limit $\mu = 40$.

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

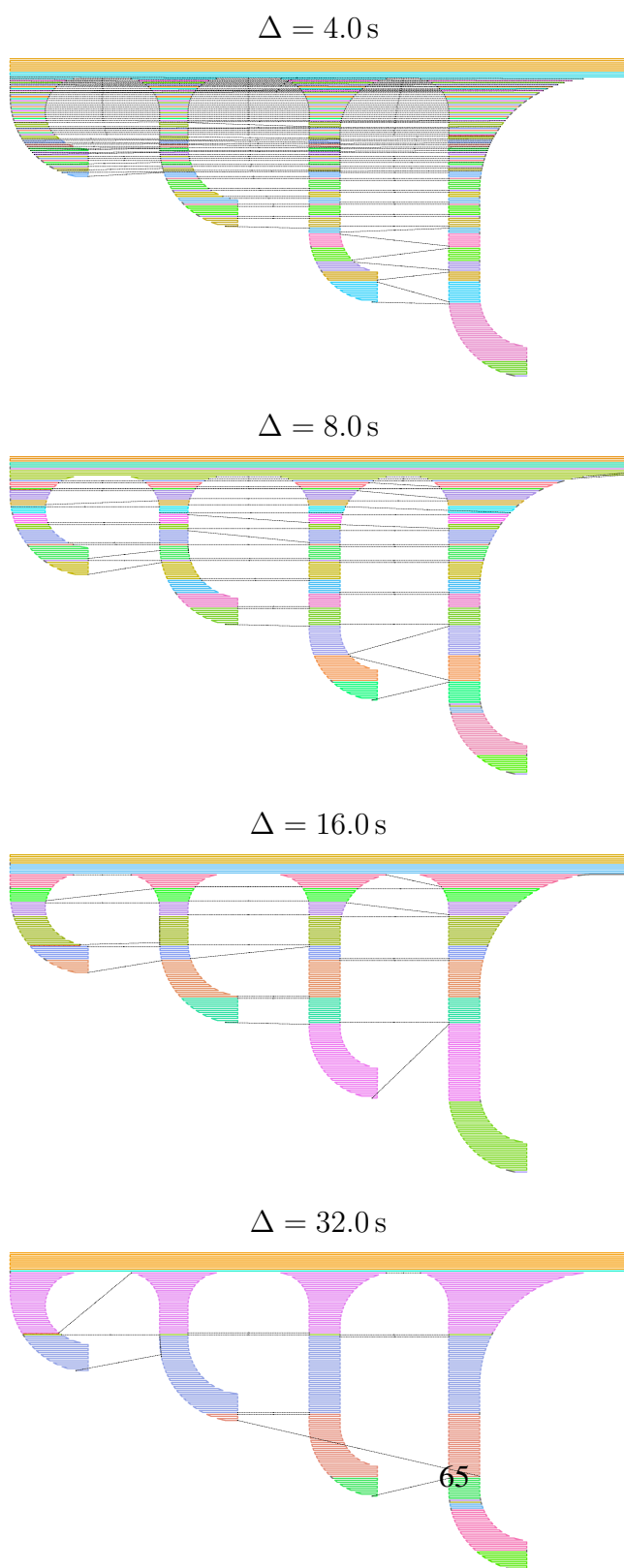


Figure 22. Tool-paths produced by HotFill for the input dataset **cthang:90**, with $\mu = 40$ and various cooling time limits Δ . The color conventions are the

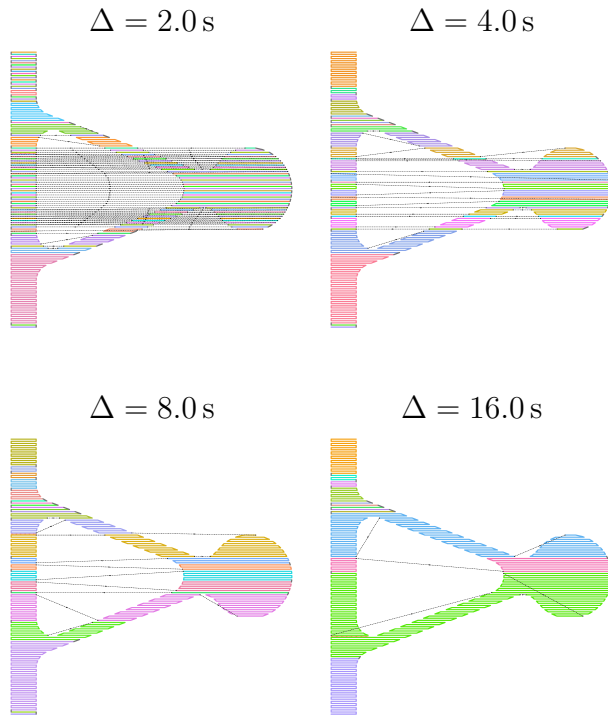


Figure 23. Tool-paths (fullpaths) produced by HotFill for the input dataset **dstop1:90**, with $\mu = 40$ and various cooling time limits Δ . The color conventions are the same as in Figure 21.

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 $\star[Neri:]???$ 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 non-trivial slices showed that, for reasonable cooling limits, the fab-time of the tool-path produced by HotFill is not much higher than that of the paths computed by Slic3r, RP3, and similar programs. $\star[Neri:]???$ 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.

1011 *Possibilities for development*:. There are still several parts of the algorithm that
 1012 could be improved to reduce the computation time and/or the fabrication time
 1013 of the resulting tool-paths, such as developing a more efficient and/or effective
 1014 BandPath procedure. Also, one may consider using a more flexible partition of the
 1015 rasters into bands, by using “topological” cut-lines instead of straight horizontal
 1016 ones [52].

1017 The bidirectional greedy BandPath heuristic that we used (see Section 5.5)
 1018 has cost proportional to the square of the number $n_{i,j}$ of rasters in the band. One
 1019 can surely find other heuristics that are much faster, but still likely to yield valid
 1020 bandpaths with low enough fab-time.

1021 9. Appendix: Move timing functions

1022 The move fabrication time functions $T_{\text{FAB}}(m)$ and $T_{\text{COV}}(m, u)$ defined in Sec-
 1023 tion 9 can be estimated as described by Komineas et al. [53]. The key parameters
 1024 are the *acceleration* a at the start of the move (which is also the deceleration at the
 1025 end), and the maximum *cruise speed* v . These parameters depend on the printer
 1026 and on whether m is a move or a trace.

1027 Let d be the distance to be covered, namely $d = |q - p|$ where $p = p_{\text{INI}}(m)$
 1028 and $q = p_{\text{FIN}}(m)$. We assume that the nozzle is stationary at the beginning of
 1029 every move, and comes to a full stops at the end of it. Therefore, if d is large
 1030 enough, the nozzle will accelerate for a time $t_a = v/a$, while covering a distance
 1031 $d_a = at_a^2/2 = vt_a/2 = v^2/(2a)$. At the end, it will decelerate for the same
 1032 distance and time. In between, it will cruise at constant speed v for a distance
 1033 $d_c = d - 2d_a$, which will take time $t_c = d_c/v$.

1034 However, if d is less than v^2/a , the nozzle will have to start decelerating before

1035 reaching the cruise speed v . Acceleration and deceleration will each last for a
 1036 distance $d_a = d/2$ and take time $t_a = \sqrt{d/a}$. The distance and time spent at
 1037 cruise speed will be $d_c = t_c = 0$.

1038 Either way, the total execution time $T_{\text{FAB}}(m)$ will be $2t_a + t_c$.

1039 Depending on the printer and on operator choices, it may be required to raise
 1040 the nozzle and/or retract a couple mm of the filament at the begging of a jump,
 1041 and to lower the nozzle and/or reposition the filament at the end. In that case, if
 1042 m is a jump, the time t_z needed to perform these operations at each end of jump
 1043 must be added to t_a .

For the passage time $T_{\text{cov}}(m, q)$, 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 $d_u = \max \{0, \min \{d, (u - p) \cdot (q - p)/d\}\}$.

We have three cases, depending on the part of the move where the passage occurs
 (accelerating, decelerating, or cruising):

$$\begin{array}{ll} \sqrt{d_u/a} & \text{if } d_u \leq d_a \\ 2t_a + t_c - \sqrt{(d - d_u)/a} & \text{if } d_u \geq d - d_a \\ t_a + (d_u - d_a)/v & \text{otherwise} \end{array}$$

1044

1045 References

- 1046 [1] P. Kulkarni, A. Marsan, D. Dutta, Review of process planning techniques in layered manu-
 1047 facturing, *Rapid Prototyping Journal* 6 (1) (2000) 18–35. doi:10.1108/13552540010309859.
- 1048 [2] I. Gibson, D. Rosen, B. Stucker, Additive manufacturing technologies - 3D printing, rapid
 1049 prototyping, and direct digital manufacturing, in: *Rapid Manufacturing Association*, 2nd
 1050 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/>. Accessed 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>

- 1101 [21] L. Xia, S. Lin, G. Ma, Stress-based tool-path planning methodology
1102 for fused filament fabrication, *Additive Manufacturing* 32 (2020) 101020.
1103 doi:<https://doi.org/10.1016/j.addma.2019.101020>.
1104 URL [https://www.sciencedirect.com/science/article/pii/](https://www.sciencedirect.com/science/article/pii/S2214860419315659)
1105 [S2214860419315659](https://www.sciencedirect.com/science/article/pii/S2214860419315659)
- 1106 [22] Q. Sun, G. M. Rizvi, C. T. Bellehumeur, P. Gu, Effect of processing conditions on the bond-
1107 ing quality of FDM polymer filaments, *Rapid Prototyping Journal* 14 (2) (2008) 72–80.
1108 doi:[10.1108/13552540810862028](https://doi.org/10.1108/13552540810862028).
- 1109 [23] P. K. Gurralla, S. P. Regalla, Part strength evolution with bonding between filaments
1110 in fused deposition modelling, *Virtual and Physical Prototyping* 9 (3) (2014) 141–149.
1111 doi:[10.1080/17452759.2014.913400](https://doi.org/10.1080/17452759.2014.913400).
- 1112 [24] M. Faes, E. Ferraris, D. Moens, Influence of inter-layer cooling time on the quasi-static
1113 properties of ABS components produced via fused deposition modelling, *Journal Procedia*
1114 *CIRP* 42 (2016) 748–753. doi:[10.1016/j.procir.2016.02.313](https://doi.org/10.1016/j.procir.2016.02.313).
- 1115 [25] N. Volpato, T. T. Zanutto, Analysis of deposition sequence in tool-path optimization for
1116 low-cost material extrusion additive manufacturing, *International Journal of Advanced Man-
1117 ufacturing Technology* 101 (5-8) (2019) 1855–1863. doi:[10.1007/s00170-018-3108-1](https://doi.org/10.1007/s00170-018-3108-1).
- 1118 [26] S. F. Costa, F. M. Duarte, J. A. Covas, Estimation of filament temperature and adhesion
1119 development in fused deposition techniques, *Journal of Materials Processing Technology*
1120 245 (2017) 167–179. doi:[10.1016/j.jmatprotec.2017.02.026](https://doi.org/10.1016/j.jmatprotec.2017.02.026).
- 1121 [27] B. Akhouni, A. H. Behraves, Effect of filling pattern on the tensile and flexural mechanical
1122 properties of FDM 3D printed products, *Experimental Mechanics* 59 (2018) 883–897.
- 1123 [28] E. Ferraris, J. Zhang, B. Van Hooreweder, Thermography based in-process monitoring
1124 of fused filament fabrication of polymeric parts, *CIRP Annals* 68 (1) (2019) 213–216.
1125 doi:[10.1016/j.cirp.2019.04.123](https://doi.org/10.1016/j.cirp.2019.04.123).
- 1126 [29] Y. Dumas, J. Desrosiers, E. Gelinas, M. M. Solomon, An optimal algorithm for the traveling
1127 salesman problem with time windows, *Operations research* 43 (2) (1995) 367–371.

- 1128 [30] M. Savelsbergh, Local search in routing problems with time windows, *Annals of Operations*
1129 *research* 4 (1985) 285–305. doi:10.1007/BF02022044.
- 1130 [31] C.-B. Cheng, C.-P. Mao, A modified ant colony system for solving the travelling salesman
1131 problem with time windows, *Mathematical and Computer Modelling* 46 (9) (2007) 1225–
1132 1235. doi:10.1016/j.mcm.2006.11.035.
- 1133 [32] M. López-Ibáñez, C. Blum, Beam-ACO for the travelling salesman problem with
1134 time windows, *Computers & Operations Research* 37 (9) (2010) 1570 – 1583.
1135 doi:10.1016/j.cor.2009.11.015.
- 1136 [33] N. Mladenović, R. Todosijević, D. Urošević, An efficient general variable neighborhood
1137 search for large travelling salesman problem with time windows, *Yugoslav Journal of Oper-*
1138 *ations Research* 23 (1) (2013) 19–30.
- 1139 [34] E. Cassiana, Hotfill, Code repository at [https://github.com/ecassiana/](https://github.com/ecassiana/hotfill)
1140 [hotfill](https://github.com/ecassiana/hotfill). Accessed on 2021-07-28 (2021).
- 1141 [35] F. P. Preparata, M. I. Shamos, *Computational Geometry – An Introduction*, Springer, 1985.
- 1142 [36] R. E. Bellman, *Dynamic Programming*, Courier Dover Publications, 1957.
- 1143 [37] D. E. Knuth, *The Art of Computer Programming*, Vol. 1, Addison-Wesley, 1998.
- 1144 [38] N. Volpato, Manual básico do programa RP³ (v110211), Tech. rep., U. Tecnológica Federal
1145 do Paraná, Depto. de Mecânica, NUFER (2011).
- 1146 [39] MakerBot, Thingiverse - Digital designs for physical objects, 3D model repository at [www.](http://www.thingiverse.com/)
1147 [thingiverse.com/](http://www.thingiverse.com/). Accessed on 2021-07-24 (2021).
- 1148 [40] Frank (@ffleurey), Adjustable foot for tables and furnitures, 3D model at [www.](http://www.thingiverse.com/thing:148764)
1149 [thingiverse.com/thing:148764](http://www.thingiverse.com/thing:148764). Accessed on 2021-07-24 (2013).
- 1150 [41] J. Phillips (@jephil08), 1-Inch bulkhead hand wrench, 3D model at [www.thingiverse.](http://www.thingiverse.com/thing:4356560)
1151 [com/thing:4356560](http://www.thingiverse.com/thing:4356560). Accessed on 2021-07-24 (2020).

- 1152 [42] E. Farkas (@Alpha_Wolf), 4-Hook coat hanger, 3D model at [www.thingiverse.com/](http://www.thingiverse.com/thing:3315713)
1153 [thing:3315713](http://www.thingiverse.com/thing:3315713). Accessed on 2021-07-24 (2018).
- 1154 [43] M. Charpentier (@Matearoa), Door stop, 3D model at [www.thingiverse.com/](http://www.thingiverse.com/thing:4889702)
1155 [thing:4889702](http://www.thingiverse.com/thing:4889702). Accessed on 2021-07-24 (2021).
- 1156 [44] E. Cassiana (@ecassiana), Flange, 3D model at [https://www.thingiverse.com/](https://www.thingiverse.com/thing:4916943)
1157 [thing:4916943](https://www.thingiverse.com/thing:4916943). Accessed on 2021-07-28 (2021).
- 1158 [45] E. Cassiana (@ecassiana), Hook latch, 3D model at [https://www.thingiverse.](https://www.thingiverse.com/thing:4919047)
1159 [com/thing:4919047](https://www.thingiverse.com/thing:4919047). Accessed on 2021-07-28 (2021).
- 1160 [46] E. Cassiana (@ecassiana), Necktie hanger, 3D model at [https://www.thingiverse.](https://www.thingiverse.com/thing:4916952)
1161 [com/thing:4916952](https://www.thingiverse.com/thing:4916952). Accessed on 2021-07-28 (2021).
- 1162 [47] N. Illasarie (@nillasarie), Runners MK2 prosthetic leg, 3D model at [www.thingiverse.](http://www.thingiverse.com/thing:1960573)
1163 [com/thing:1960573](http://www.thingiverse.com/thing:1960573). Accessed on 2021-07 (2016).
- 1164 [48] A. Jøraasen (@joraasen), 22mm Wrench, 3D model at [www.thingiverse.com/](http://www.thingiverse.com/thing:1909039)
1165 [thing:1909039](http://www.thingiverse.com/thing:1909039). Accessed on 2021-07-25 (2016).
- 1166 [49] E. Cassiana (@ecassiana), Grille, 3D model at [https://www.thingiverse.com/](https://www.thingiverse.com/thing:5161470)
1167 [thing:5161470](https://www.thingiverse.com/thing:5161470). Accessed on 2021-12-11 (2021).
- 1168 [50] Anthony L. (@M600), GoPro tool with universal wrench, 3D model at [www.](http://www.thingiverse.com/thing:601212)
1169 [thingiverse.com/thing:601212](http://www.thingiverse.com/thing:601212). Accessed on 2021-07-25 (2014).
- 1170 [51] 3DCloner, 3dcloner dh plus, Code repository at [http://3dcloner.ind.br/](http://3dcloner.ind.br/upload/20200724170856r7ginb.pdf)
1171 [upload/20200724170856r7ginb.pdf](http://3dcloner.ind.br/upload/20200724170856r7ginb.pdf). Accessed on 2021-12-13 (2021).
- 1172 [52] H. Edelsbrunner, L. J. Guibas, Topologically sweeping an arrangement, *Journal of Computer*
1173 *and System Sciences* 38 (1) (1989) 165–194. doi:10.1016/0022-0000(89)90038-X.
- 1174 [53] G. Komineas, P. Foteinopoulos, A. Papacharalampopoulos, P. Stavropoulos, Build time es-
1175 *timation models in thermal extrusion additive manufacturing processes*, *Procedia Manufac-*
1176 *turing Journal* 21 (2018) 647 – 654. doi:10.1016/j.promfg.2018.02.167.