Discrete Applied Mathematics 161 (2013) 2293–2315

Contents lists available at SciVerse ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Two efficient algorithms for computing the characteristics of a subsegment of a digital straight line☆

Jacques-Olivier Lachaud [a], Mouhammad Said [a,b,*]

[a] Laboratoire de Mathématiques, UMR CNRS 5127, Université de Savoie, 73376 Le Bourget du Lac, France
[b] LIRIS, UMR CNRS 5611, Université Lumière Lyon 2, 69676 Bron, France

ARTICLE INFO

Article history:
Received 23 October 2011
Received in revised form 17 August 2012
Accepted 31 August 2012
Available online 6 October 2012

Keywords:
Discrete geometry
Standard lines
Digital straight segment recognition
Stern–Brocot tree

ABSTRACT

We address the problem of computing the exact characteristics of any subsegment of a Digital Straight Line (DSL) with known characteristics (a slope $\frac{a}{b}$, a shift to origin μ). We present here two new algorithms that solve this problem, whose correctnesses are fully proved. Their principle is to descend/climb the Stern–Brocot tree of fractions in a top-down/bottom-up way. The top-down algorithm **SmartDSS** has a time complexity which depends on the sum of the *quotients* of the continued fraction of the output slope and on the number of pattern repetitions. As a corollary, its time complexity is bounded by the sum of the *quotients* of the continued fraction of the input slope, Said and Lachaud (2009) [18]. The bottom-up algorithm **ReversedSmartDSS** has a time complexity proportional to the difference of *depth* of the input slope and the slope of the output segment, Said and Lachaud (2011) [17]. As a corollary, its complexity is thus logarithmic in the coefficients of the input slope. These algorithms are also efficient in practice, as shown by a series of experiments comparing these new algorithms with standard arithmetic digital straight segment recognition.

1. Introduction

Digital Straight Lines (DSL) have many definitions: digitization of Euclidean straight lines, word combinatorics description as Christoffel words or Sturmian words, arithmetic description with diophantine linear inequalities. Their first definition explains why they arise naturally in a digital shape analysis: the boundary of digital shape is indeed composed of finite parts of digital straight lines, called Digital Straight Segments (DSS). As such, they describe the first order geometry of digital shape and are used for coding, geometric estimation and feature detection. This explains why they have been so deeply studied (see the survey [13] or [12]).

Algorithms for recognizing if a sequence of digital points is a DSS have been known for a long time. Given a DSS, there are infinitely many DSL that cover it but only one of them has *minimal* characteristics (size of slope numerator and denominator, see [19] for more details about minimal characteristics). The characteristics of this DSL define the *(exact) characteristics of the DSS*. In one of the first recognition methods, Freeman [10] suggested to analyze the regularity in the pattern of the directions in the chain code [9] of a digital curve. Anderson and Kim [1] have presented a deep analysis of the properties of DSS and suggested a different algorithm based on calculating the convex hull of the points of digital curves to be analyzed. Modern DSS recognition algorithms achieve a computational complexity of $O(m)$, if m is the number of input points and with

☆ With the support of ANR project GeoDIB (ANR-BLAN-).
* Corresponding author at: Laboratoire de Mathématiques, UMR CNRS 5127, Université de Savoie, 73376 Le Bourget du Lac, France. Fax: +33 04 79 75 81 42.
E-mail addresses: jacques-olivier.lachaud@univ-savoie.fr (J.-O. Lachaud), mouhammadsaiid@hotmail.com (M. Said).

0166-218X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2012.08.038

Table 1
Comparison between the three DSS recognition algorithms in terms of necessary input data and in terms of time complexity. By hypothesis, the digital points $A = P_1, \ldots, P_m = B$ are a connected piece of DSL, whose slope is a/b and whose shift is μ. Furthermore, the *n-th convergent* of the continued fraction of a/b is $[u_0; u_1, \ldots, u_n]$. Note that a'/b' is a reduced fraction of a/b or a/b itself.

Algorithm	Input data	Output	Complexity
DR95	Points $P_1, \ldots, P_m$	Characteristics of DSS $[AB]$: slope $a'/b' = [u_0; u_1, \ldots, u_{n'}]$, shift μ', number of patterns δ	$\Theta(m)$
SmartDSS	Points A, B, integers a, b, μ		$\Theta(\delta + \sum_{i=0}^{n'} u_i)$
Reversed-SmartDSS	Points A, B, integers a, b, μ, $\frac{p_i}{q_i} = (u_i)_{i=0..n}$		$\Theta(n - n')$

a computing model where arithmetic operations take $O(1)$ time. Interestingly, this complexity is obtained by algorithms based on arithmetic properties [15,14,3], combinatorial properties [20], or dual-space construction [7].

These algorithms are optimal, when no further information is known. However, there are specific cases where we already know that the observed set of points is included in some DSL of known characteristics. The recognition algorithm has therefore only to determine the *exact* characteristics of the input set, and has not to decide whether or not this set is a piece of DSL. As an illustrating example, this situation occurs when computing the multiresolution geometry of a digital object, since analytic formulas give the multiresolution of DSL (see the thesis of Figueiredo [8] or the paper [18]).

We show in this paper that one can compute in sublinear time the exact characteristics of a DSS when it is a subsegment of a DSL with known characteristics. We give two algorithms whose principle is related to the Stern–Brocot tree of irreducible fractions. A variant of the first algorithm was presented in [18]. A variant of the second algorithm was presented in [17]. In this paper, we present these algorithms in a more unified way that is easier to understand and whose correctness and time complexity are easier to demonstrate. Full proofs are given and a new exhaustive experimental evaluation is conducted.

Many works deal with the relations between irreducible rational fractions and digital lines (see [6] for characterization with Farey series, and [21] for a link with decomposition into continuous fractions). In [2], Debled and Reveillès first introduced the link between the Stern–Brocot tree and the recognition of digital line. Recognizing a piece of digital line is like going down the Stern–Brocot tree up to the directional vector of the line. To sum up, the classical online DSS recognition algorithm **DR95** [2] (also reported in [12]) updates the DSS slope when adding a point that is just exterior to the current line (weak exterior points). In [4], de Vieilleville and Lachaud have related the arithmetically-based DSS recognition algorithm with new parameters related to a combinatorial representation of DSS. New analytic relations have been established and the relation with the Stern–Brocot tree has been made explicit.

We present two fast algorithms which compute the exact (minimal) characteristics of a DSS that is some subset of a DSL of known characteristics. More precisely, the first algorithm, called **SmartDSS** computes the exact characteristics of a DSS that is a subset of a known DSL, given the endpoints of the DSS, by moving in a top-down way along the Stern–Brocot tree. Its correctness is established by Proposition 2. Its worst-case computational complexity is $\Theta((\sum_{i=0}^{n'} u_i) + \delta)$, where $[u_0; u_1, \ldots, u_{n'}]$ is the continued fraction of the slope of the output DSS and δ is the number of patterns in this DSS (Proposition 4). The expectation of this sum for fractions $\frac{a}{b}$ with $a+b = N$ is experimentally lower than $\log^2 N$, and this sum is upper bounded by N. On the other hand, the best case is $\Theta(1)$. For the second algorithm, the input DSL, say D, is given as the continued fraction of its slope. The DSS is specified by the positions of its two endpoints A and B. Furthermore, the two lower leaning points of D surrounding A and B are given as input. This algorithm, called **ReversedSmartDSS**,[1] determines the characteristics of the DSS by moving in a bottom-up way along the Stern–Brocot tree. We prove the correctness of this algorithm in Proposition 12. We further show in Proposition 13 that its worst-case computational complexity is $\Theta(n - n')$, where $[u_0; u_1, \ldots, u_n]$ is the continued fraction of the slope of the input DSL and $[u_0; u_1, \ldots, u_{n'}]$ is the continued fraction of the slope of the output DSS. This result assumes a computing model where standard arithmetic operations are in $O(1)$, which is a reasonable assumption when analyzing digital shapes. The properties and specificities of these DSS recognition algorithms are summed up in Table 1.

This is to compare with the complexity of classical DSS recognition algorithms (e.g., DR95 [3], see also [13]), whose complexities are at best $\Theta(m)$, where m is the number of input points. All these algorithms have been implemented. Experimental results show that algorithm **ReversedSmartDSS** performs better than **SmartDSS** algorithm and are much faster than classical DSS recognition algorithms (see Table 2 in Section 5).

This paper is organized as follows. First, we recall in Section 2 some definitions and properties about the rational fractions, more particularly the relation between the rational fractions and the Stern–Brocot tree. In Sections 3 and 4, we describe these two new algorithms. We show the correctness of each algorithms, as well as their respective computational complexity. Section 5 compares the performances of the algorithms SmartDSS, ReversedSmartDSS and the standard arithmetic DSS recognition.

2. Digital straightness and continued fractions

In this section, we recall some links between DSS, patterns, continued fraction of the slope, and the Stern–Brocot tree representation of fractions.

[1] This name is in opposition to the **SmartDSS** algorithm, because it moves along the Stern–Brocot in a reversed direction.

A *Digital Straight Line (DSL)* of integer characteristics (a, b, μ), $\gcd(a, b) = 1$, is the infinite subset of the digital plane $\{(x, y) \in \mathbb{Z}^2, \mu \leq ax - by < \mu + |a| + |b|\}$ [16]. It is well-known that these DSL are 4-connected subset of the plane and may thus be defined as a sequence of Freeman moves in the plane. They are often called *standard*. The fraction a/b is the *slope* of the DSL, while μ is related to the shift at the origin. A *Digital Straight Segment (DSS)* is a finite 4-connected piece of DSL. Any DSS is included in an infinite number of DSL, but the *characteristics of the DSS* are the characteristics of the DSL containing it with minimal $|a|$. A DSS is uniquely determined from its characteristics and the starting and ending points. The *remainder* of a DSS – or a DSL – of characteristics (a, b, μ) is the function $(x, y) \mapsto ax - by$. *Upper leaning points* have remainder μ. *Lower leaning points* have remainder $\mu + |a| + |b| - 1$. It is easy to see that the convex hull of these points forms a strip in the plane of slope a/b which contains all points of the DSL. Weakly exterior points are the points of the digital plane closest to the strip but not within. *Upper weakly exterior points* have remainder $\mu - 1$ while *lower weakly exterior points* have remainder $\mu + |a| + |b|$. These points are fundamental when recognizing DSS [2].

Given a standard line (a, b, μ), we call *pattern* of characteristics (a, b) the succession of Freeman moves between any two consecutive upper leaning points. The Freeman moves defined between any two consecutive lower leaning points is the previous word read from back to front and is called the *reversed pattern* (see [4]). We say that a DSS is *primitive* whenever it contains one pattern of its slope or one reversed pattern of its slope (but not one of each).

As noted by several authors (e.g. see [12], or the work of Berstel reported in [4]), the pattern of any slope can be constructed from the continued fraction of the slope. We recall that a *simple continued fraction* is an expression:

$$z = \frac{a}{b} = [u_0, u_1, u_2, \ldots, u_i, \ldots, u_n] = u_0 + \cfrac{1}{u_1 + \cfrac{1}{\ldots + \cfrac{1}{u_{n-1} + \cfrac{1}{u_n}}}},$$

where n is the *depth* of the fraction, and u_0, u_1, etc., are all integers and called the *partial quotients*. We call *k-th convergent* the simple continued fraction formed of the k first partial quotients: $z_k = \frac{p_k}{q_k} = [u_0, u_1, u_2, \ldots, u_k]$. The function E takes a continued fraction z as input to build recursively the pattern of a DSS of slope z in the first quadrant.

$$E(z_{-2}) = 0, \qquad E(z_{-1}) = 1, \quad \text{and}, \quad \forall i \geq 0, \quad \begin{cases} E(z_{2i+1}) = E(z_{2i})^{u_{2i+1}} E(z_{2i-1}), \\ E(z_{2i}) = E(z_{2i-2}) E(z_{2i-1})^{u_{2i}}. \end{cases} \tag{1}$$

Let us take for example the fraction $\frac{7}{16} = [0; 2, 3, 2]$. The pattern of any DSS with this slope is thus:

$$\begin{aligned} E([0; 2, 3, 2]) &= E([0; 2, 3])^2 \cdot E([0; 2]) && 00010010010001001001 \cdot 001 \\ E([0; 2, 3]) &= E([0]) \cdot E([0; 2])^3 && 0 \cdot 001001001 \\ E([0; 2]) &= 001 && 001 \\ E([0]) &= 0 && 0. \end{aligned}$$

The role of partial quotients can be visualized with a structure called the *Stern–Brocot tree* (see [11] for a complete definition) which is a hierarchy containing all the positive irreducible rational fractions. The idea under its construction is to begin with the two fractions $\frac{0}{1}$ and $\frac{1}{0}$ and to repeat the insertion of the median of these two fractions as follows: insert the median $\frac{m+m'}{n+n'}$ between $\frac{m}{n}$ and $\frac{m'}{n'}$. The sequence of partial quotients defines the sequence of right and left moves down the tree. An illustration of this tree is proposed in Fig. 1. Many works deal with the relations between irreducible rational fractions and digital lines (see [6] for characterization with Farey series, and [21] for a link with decomposition into continuous fractions). In [2], Debled and Reveillès first introduced the link between this tree and the recognition of digital line. Recognizing a piece of digital line is like going down the Stern–Brocot tree up to the directional vector of the line. To sum up, the classical online DSS recognition algorithm **DR95** [2] (also reported in [12]) updates the DSS slope when adding a point that is just exterior to the current line (weak exterior points). The slope evolution is analytically given by the next property.

Proposition 1 *([4]). The slope evolution in* **DR95** *depends on the parity of the depth of its slope, the type of weakly exterior point added to the right. This is summed up in the table below, where the slope is* $[0, u_1, \ldots, u_k]$, $k = 2i$ *even or* $k = 2i + 1$ *odd, and the number of patterns contained in the DSS is denoted by* δ*, while the number of reversed patterns is denoted by* δ'*.*

	Evenk	*Odd k*
Upper weakly exterior	$[0, u_1, \ldots, u_{2i}, \delta]$	$[0, u_1, \ldots, u_{2i+1} - 1, 1, \delta]$
Lower weakly exterior	$[0, u_1, \ldots, u_{2i} - 1, 1, \delta']$	$[0, u_1, \ldots, u_{2i+1}, \delta']$

We may look again at our example of fraction $\frac{7}{16}$. The path in the Stern–Brocot tree from the root $\frac{0}{1}$ to this fraction is the list of nodes $\frac{0}{1}$, $\frac{1}{1}$, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{2}{5}$, $\frac{3}{7}$, $\frac{4}{9}$, $\frac{7}{16}$. Any DSS in a DSL of slope $\frac{7}{16}$ has a slope which is one of these fractions. We notice that the partial quotients of $\frac{7}{16}$ are a subset of the preceding list, i.e. $\frac{0}{1}$, $\frac{1}{2}$, $\frac{3}{7}$, $\frac{7}{16}$.

We will use the following technical lemmas for proving the validity of our algorithms. Let D be a DSL of slope a/b, $a > 0$, $b > 0$, and let w be a pattern of D.

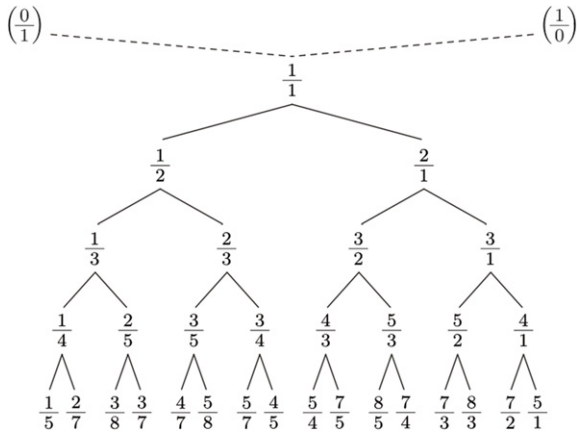


Fig. 1. Stern–Brocot tree: positive irreductible rational fractions.

Lemma 1. *The pattern w visits exactly one lower leaning point of D.*

Proof. A pattern is a path between two upper leaning points of a DSL. The associated remainder function will take all the values between $\mu+1$ and $\mu+a+b-1$ for all the $a+b-2$ points between the two upper leaning points. Therefore, only one point will have the value $\mu+a+b-1$. □

Lemma 2. *Let U, U' be the upper leaning points of w and L its lower leaning point. Then the vector $\vec{t} = \vec{UL} + (-1, 1)$ forms the Bézout coefficients of $\vec{UU'} = (b, a)$ (i.e. $\vec{UU'} \wedge \vec{t} = 1$). As a corollary, points $U + k\vec{UU'} + \vec{t}$, k integer, are the upper weakly exterior points to D. Furthermore, points $U + k\vec{UU'} + (1, -1)$ are the lower weakly exterior points to D.*

Proof. Since L has remainder $\mu+a+b-1$, the point $L+(-1, 1)$ has remainder $\mu-1$, hence the vector $\vec{t} = (t_x, t_y) = \vec{UL} + (-1, 1)$ has remainder -1. This means $-1 = at_x - bt_y = (t_x, t_y) \wedge (b, a)$ which concludes. □

Lemma 3. *Let $u01v$ be the decomposition of w into words such that the letters 01 stand around the lower leaning point. Any DSL containing w^k contains at least vw^ku, for a positive integer k. Furthermore, any path $w' = v'w^ku'$, with v' right factor of v and u' left factor of u, is a DSS with the same slope as w.*

Proof. Lemma 1 entails that the writing $w = u01v$ is valid and unique. Let U and U' be the two upper leaning points of w. It is well known that DSS slopes are governed by weakly exterior points (e.g. see [2]). Now, according to Lemma 2, the first weakly exterior point to the right of w^k is at the next lower leaning point plus $(-1, 1)$ (i.e. position $U' + \vec{u}1$). The first weakly exterior point to the left of w is at the previous lower leaning point plus $(-1, 1)$ (i.e. position $U - 0\vec{v}$). Therefore there cannot be any change in the slope of DSS between these points, and any DSL containing w^k contains vw^ku. □

3. Fast top-down DSS recognition algorithm when DSL container is known

Let S be some DSS included in a line D. We propose an output-sensitive algorithmic approach **SmartDSS** which computes all the characteristics of S in a time sublinear in the number of its points (Algorithm 1). The required input data are the characteristics (α, β, μ') of D, as well as the two extremities A, B, of S. Since we know that AB forms a DSS, a faster recognition algorithm than **DR95** can be designed. Indeed, most of the points between A and B do not lead to any slope evolution and we exploit this property in this algorithm.

In the whole section, D is a DSL in the first quadrant.

3.1. Overview of the algorithm

Starting from the initial correct quadrant, Algorithm 1 determines progressively the positions of the weakly exterior points. They are related to the Bézout coefficient of the current DSS slope $\frac{p_k}{q_k}$ (lines 6 and 10), see also Lemma 2. Since we update at each step the continued fraction of the slope, these coefficients are given in $O(1)$ time by the function `Bezout` (Algorithm 2), assuming a computation model where standard arithmetic operations are in $O(1)$.[2]

[2] A reasonable assumption since characteristics are bounded by $2\|\vec{AB}\|_\infty$, hence integers are bounded by the square of this value if the frame is centered on $[A, B]$.

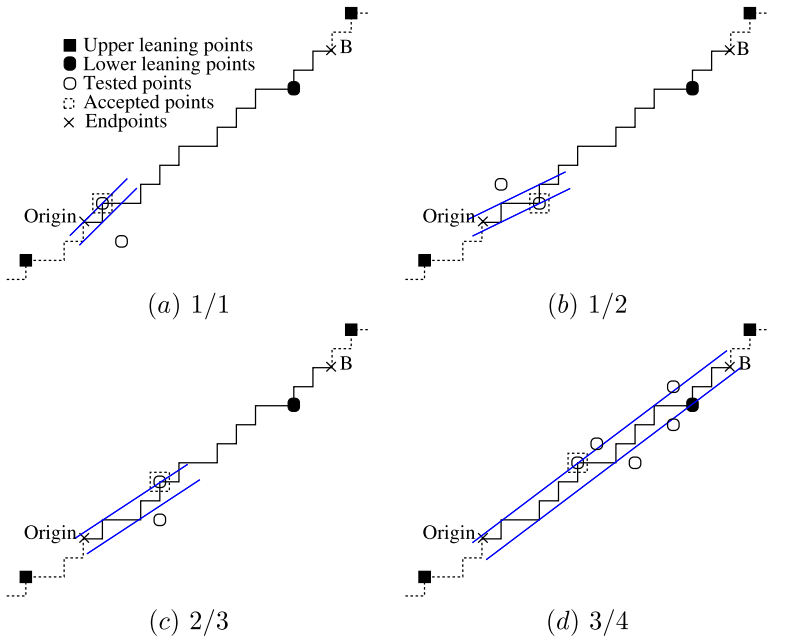


Fig. 2. Illustration of Algorithm 1. A digital straight line $D(13, 17, -5)$ with an odd slope. Compute the characteristics (a, b, μ) of a DSS S that is the subset of D between the origin and the point $B(12, 9)$. The slopes in the figures are drawn with solid blue lines and tested points are circled. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In the loop (while block at line 1), the algorithm checks in sequence upper and lower weakly exterior points so as to find the first that is in the DSL. If the algorithm has not overtaken the end point B, three cases arise. The first and second cases occur when a weakly exterior point happens to be part of the line. Once such a point is found, Proposition 1 indicates how to update the slope, depending if it is a slope increase (line 3) or decrease (line 7). This is effectively implemented in $O(1)$ time with Algorithm 3. It remains to update consistently the new first lower leaning point or upper leaning point. We have to be a little careful of whether the current DSS has a first leaning point that is an upper one (case ULU) or a first leaning point that is a lower one (case LUL): it is just to adapt δ to be the number of patterns or the number of reversed patterns. The third case indicates that there is one more pattern and one more reversed pattern in the DSS, and advances the next weakly exterior points accordingly.

The algorithm stops either when point B has been reached or when the slope of the DSL D has been reached.

The correctness of this algorithm is established in the next subsection (Proposition 2). An upper bound for its time complexity is given by Proposition 4. It is related to the partial quotients of the DSS slope.

An execution of this algorithm is illustrated in Fig. 2. 11 points have been tested compared with a DSS length of 23. For this execution, the table below displays the state of the main variables at the beginning of each iteration at line 1. The output is the DSS $(3, 4, 3 * 1 - 4 * 1)$, which has 3 patterns between $(0, 0)$ and $(12, 9)$. Note that $\frac{3}{4} = [0; 1, 3]$ and **SmartDSS** exits at iteration $1 + \delta + \sum u_i = 1 + 3 + (0 + 1 + 3)$.

Iter	U	L	(b, a)	U'	L'	δ	Case
1	(0, 0)	(0, 0)	(1, 0)	(0, 1)	(1, −1)	0	$\delta++$ (line 11)
2	(0, 0)	(0, 0)	(1, 0)	(1, 1)	(2, −1)	1	*UWE* (line 3)
3	(0, 0)	(1, 0)	(1, 1)	(1, 2)	(3, 1)	1	*LWE* (line 7)
4	(1, 1)	(1, 0)	(2, 1)	(4, 3)	(4, 1)	1	*UWE* (line 3)
5	(1, 1)	(3, 1)	(3, 2)	(5, 4)	(8, 4)	1	*UWE* (line 3)
6	(1, 1)	(3, 1)	(4, 3)	(6, 5)	(10, 6)	1	$\delta++$ (line 11)
7	(1, 1)	(3, 1)	(4, 3)	(10, 8)	(14, 9)	2	$\delta++$ (line 11)
8	(1, 1)	(3, 1)	(4, 3)	(14, 11)	(18, 12)	3	Exit (line 2)

3.2. Correctness of SmartDSS algorithm

We establish the proof of the correctness of algorithm **SmartDSS** in Proposition 2. We begin with some useful lemmas.

Lemma 4. *If* $(\frac{p_i}{q_i})_{i=0..k}$ *are the partial quotients of* $\frac{a}{b}$*, then Algorithm* 2 *returns the Bézout coefficients* (b', a') *of* (b, a)*, i.e.* $ba' - ab' = 1$.

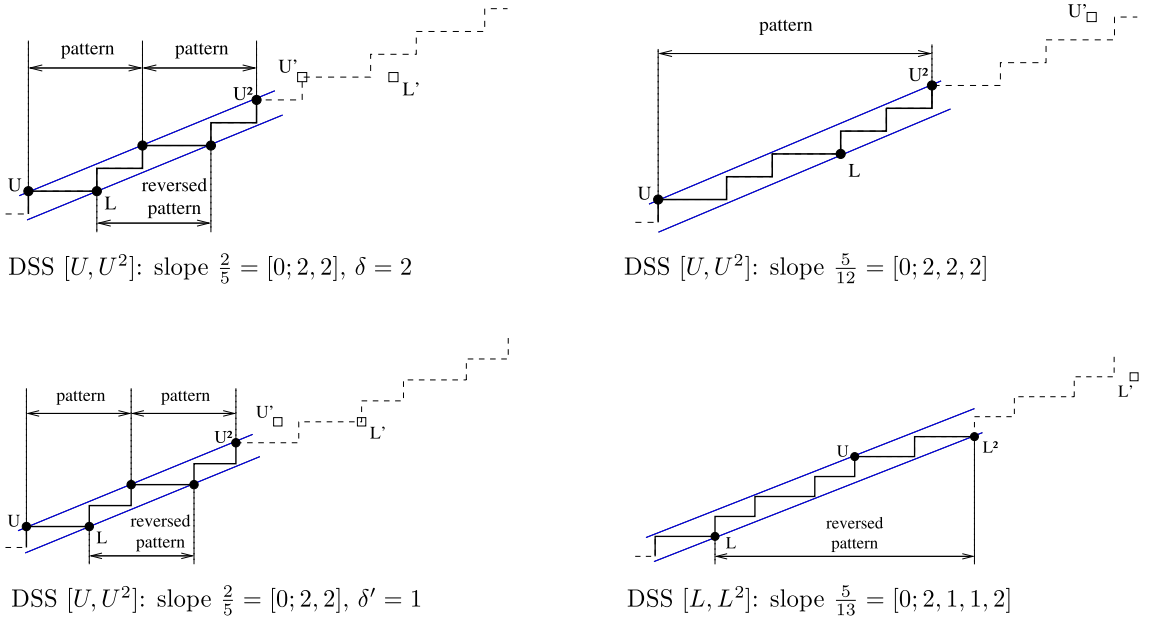


Fig. 3. Illustration of Lemma 5. Slope evolution of a ULU DSS of slope $\frac{2}{5}$ with two patterns (and one reverse pattern). Top row: the upper weakly exterior point U' belongs to D (left), slope becomes $\frac{5}{12}$ (right). Bottom row: the lower weakly exterior point L' belongs to D (left), slope becomes $\frac{5}{13}$ (right).

Proof. If k is odd, then $ba' - ab' = q_k(p_k - p_{k-1}) - p_k(q_k - q_{k-1}) = p_kq_{k-1} - q_kp_{k-1}$. For all continued fractions, this value is $(-1)^{k+1}$, hence 1 in this case. If k is even, then $ba' - ab' = q_kp_{k-1} - p_kq_{k-1} = -(-1)^{k+1}$, hence again 1. □

A DSS is said *ULU* (resp. *LUL*) if one of its leftmost leaning points is an upper leaning point (resp. a lower leaning point). Only DSS with trivial slope are both ULU and LUL. In the following, a subsegment from A to B of the DSL D is denoted by $[A, B]_D$ or more simply $[A, B]$ when no confusion may arise.

Lemma 5. *At each iteration of algorithm* `SmartDSS` *(line* 1*), the loop invariant is (letting* $U^2 = U + \delta(b, a), L^2 = L + \delta(b, a), \mu = aU_x - bU_y$*):*

- *the Boolean value (ulu or lul) is true.*
- *if ulu is true, then* $[A, U^2]$ *is a DSS of characteristics* (a, b, μ) *that is ULU and has* δ *patterns. Furthermore, U and L are respectively its leftmost upper and lower leaning points, while* U' *and* L' *are respectively its leftmost upper and lower weakly exterior points which lie to the right of* U^2*. There are* δ *patterns in* $[UU'[$*.*
- *if lul is true, then* $[A, L^2]$ *is a DSS of characteristics* (a, b, μ) *that is LUL and has* δ *reversed patterns. Furthermore, U and L are respectively its leftmost upper and lower leaning points, while* U' *and* L' *are respectively its leftmost upper and lower weakly exterior points which lie to the right of* L^2*. There are* δ *reversed patterns in* $[LL'[$*.*

Proof. The invariant is established by induction on the iteration number K. For $K = 1$, both *ulu* and *lul* are true. For $K = 1, U = L = U^2 = L^2$. It is a single point whose slope is by convention $(1, 0)$ and $\mu = -A_y$. $U' = U + (0, 1)$: it is the first point that is upper weakly exterior (used if the first step is a vertical step). $L' = L + (1, -1)$: it is the first point that is lower weakly exterior (never used since we are in the first quadrant). All invariant statements are true. Assuming the invariant was verified at iteration K, we show that it is true at iteration $K + 1$. According to the test at line 2, we know that the DSS $[A, B]$ touches at least either U' or L'. We examine all mutually exclusive cases:

$U'_y \le B_y$ *and* $U' \in D$. When *ulu* is true, an illustration is given in Fig. 3, top row. This situation is treated exactly as in the **DR95** algorithm when adding an upper weakly exterior point. The function `UpdateSlope` is the implementation of Proposition 1. The new slope a^*/b^* corresponds to the vector UU' (i.e. $\delta(b, a) + (b', a')$). The leaning points are updated as follows: U is not changed, $U^2 = U + (b^*, a^*)$, L becomes the last lower leaning point. We obtain a DSS $[A, U']$, that is ULU (*ulu* is true) and has exactly one pattern $[U, U']$. Hence U and L are the leftmost leaning points. Lemma 4 entails that the new positions U' and L' correspond to upper weakly exterior points. The test "not *lul*" is to take into account that if the segment is ULU then the upper weakly exterior point is before L', so L must be translated accordingly. Points U' and L' are the leftmost weakly exterior points to the right of U^2. Indeed subtracting (b^*, a^*) from their positions gives a vector shorter than (b^*, a^*) for U' and L' (we choose Bézout coefficients with $0 < a' \le a^*$).

```
Action SmartDSS( In D : DSL (α, β, μ'), In A, B : Points of ℤ²,
                                        Out S: DSS (a, b, μ) ) ;
Var u, p, q : array of integers /* Cont. frac. a/b = [u_0, ..., u_k] = p_k/q_k */ ;
Var U, L, U', L' : Point of ℤ² ;
Var ulu, lul, inside : boolean ;
Var k, loop : integer ;
begin
    k ← 0, u_0 ← 0, p_0 ← 0, q_0 ← 1, p_{-1} ← 1, q_{-1} ← 0 ;
    (b, a) ← (1, 0) ;
    (b', a') ← (0, 1) ; /* ba' − ab' = 1 */
    U ← A, L ← A /* U, L : first leaning points for the current DSS. */;
    U' ← U + (b', a'), L' ← L + (b − b', a − a') ;
    δ ← 0, ulu ← true, lul ← true ;
1   while p_k ≠ α do
        /* If ulu, [A, U + δ(b, a)] is a DSS (a, b, aU_x − bU_y). */;
        /* If lul, [A, L + δ(b, a)] is a DSS (a, b, aU_x − bU_y). */;
        /* U' is the next upper weakly exterior point to this DSS. */;
        /* L' is the next lower weakly exterior point to this DSS. */;
2       if (‖AU'‖_1 > ‖AB‖_1) and (‖AL'‖_1 > ‖AB‖_1) then
            break ;
        /* DSS [A, B] reaches at least U' or L'. */ ;
3       else if U'_y ≤ B_y and U' ∈ D then
            /* Increase slope with weak upper leaning point U' */ ;
4           UpdateSlope(true, δ, k, u, p, q ) ;
5           L ← L' − (b − b', a − a') ; /* Last lower leaning point. */
            if not lul then L ← L − (b, a);
            (a, b) ← (p_k, q_k) ;
6           (b', a') ← Bezout(p, q, k) ; /* ba' − ab' = 1 */
            U' ← U + (b + b', a + a'), L' ← L + (2b − b', 2a − a') ;
            δ ← 1, ulu ← true, lul ← false ;
7       else if L'_x ≤ B_x and L' ∈ D then
            /* Decrease slope with weak lower leaning point L' */ ;
8           UpdateSlope(false, δ, k, u, p, q) ;
9           U ← U' − (b', a') ; /* Last upper leaning point. */
            if not ulu then U ← U − (b, a);
            (a, b) ← (p_k, q_k) ;
10          (b', a') ← Bezout(p, q, k) ; /* ba' − ab' = 1 */
            U' ← U + (b + b', a + a'), L' ← L + (2b − b', 2a − a') ;
            δ ← 1, ulu ← false, lul ← false ;
11      else
            δ ← δ + 1 ; /* One more (reversed) pattern. */
            U' ← U' + (b, a);
            L' ← L' + (b, a);
    μ ← aU_x − bU_y ; /* (a,b,μ) are the characteristics of DSS [P, Q]. */
end
```

Algorithm 1: Computes the characteristics (a, b, μ) of a DSS that is some subset of a DSL D, given a starting point A and an ending point B ($A, B \in D$).

$L'_x \le B_x$ and $L' \in D$. When *ulu* is true, an illustration is given in Fig. 3, bottom row. The reasoning is similar to the previous case.

Otherwise. Both U' and L' are not in the DSL D, therefore the DSL goes straight until the next weakly exterior points. Since they are located (b, a) further, the DSS has one more pattern and one more reversed pattern. If *ulu*, $[A, U + (\delta + 1)(b, a)]$ is thus a DSS with the same characteristics. If *lul*, $[A, L + (\delta + 1)(b, a)]$ is thus a DSS with the same characteristics. The next weakly exterior points are just U' and L' translated by the pattern vector (b, a). □

Let $A^0, \ldots, A^m$ be an arbitrary finite 4-connected sequence of points in some DSL D. Such a sequence of points is by definition a DSS.

Function Bezout(**In** p, q, k) : (b', a') ;
p, q : array of integer /* *partial quotients* */ ;
k : integer /* *depth of continued fraction* */ ;
begin
  if *k is odd* **then**
    $b' \leftarrow q_k - q_{k-1}$;
    $a' \leftarrow p_k - p_{k-1}$;
  else
    $b' \leftarrow q_{k-1}$;
    $a' \leftarrow p_{k-1}$;
end

Algorithm 2: Computes in $O(1)$ the Bézout coefficients (b', a') of (p_k, q_k), i.e. $q_k a' - p_k b' = 1$, using the partial quotients $\frac{p_i}{q_i}$ of $\frac{p_k}{q_k}$.

Action UpdateSlope(**In** uw, δ, **InOut** k, u, p, q) ;
uw : Boolean /* *True iff upper weak leaning point* */ ;
δ : integer /* *number of (reversed) patterns* */ ;
k : integer /* *depth of slope continued fraction* */ ;
u, p, q : array of integers /* *slope cont. fraction* */ ;
begin
  if *(uw = true and k is odd) or (uw = false and k is even)* **then**
    $u_k \leftarrow u_k - 1, p_k \leftarrow p_k - p_{k-1}, q_k \leftarrow q_k - q_{k-1}$;
    $u_{k+1} \leftarrow 1, p_{k+1} \leftarrow p_k + p_{k-1}, q_{k+1} \leftarrow q_k + q_{k-1}$;
    $k \leftarrow k + 1$;
  if $\delta = 1$ **then**
    $u_k \leftarrow u_k + 1, p_k \leftarrow p_k + p_{k-1}, q_k \leftarrow q_k + q_{k-1}$;
  else
    $u_{k+1} \leftarrow \delta, p_{k+1} \leftarrow \delta p_k + p_{k-1}, q_{k+1} \leftarrow \delta q_k + q_{k-1}$;
    $k \leftarrow k + 1$;
end

Algorithm 3: Updates in $O(1)$ the slope of a DSS according to the addition of an upper leaning point (uw is *true*) or lower leaning point (uw is *false*), to the number of patterns or reversed patterns δ, and to the current continued fraction of the slope. This is the implementation of Proposition 1.

Lemma 6. *For a fixed D, the index m determines the number $K(m)$ of loops (line 1) in* SmartDSS(D, A^0, A^m). *Moreover, $K(m+1)$ is either equal to $K(m)$ or to $K(m) + 1$. The state of all variables depends only on $K(m)$.*

Proof. Point B intervenes only for exiting the loop (line 2). Since integers $\|AU'\|_1$ and $\|AL'\|_1$ are strictly increasing, the number of iterations $K(m)$ may not decrease. Lastly, U' and L' moves at least of a vector of 1-norm 1 since vectors (b, a) form increasing sequences. Since the 1-norm of $A^m A^{m+1}$ is 1, the exit condition (line 2) may be true only one more time, and $K(m+1) - K(m) \leq 1$. For a given D, the state of all variables depends only on $K(m)$ since B affects only $K(m)$ and no other computations. □

We prove below the correctness of function SmartDSS (Algorithm 1).

Proposition 2. *For any DSL D in the first quadrant such that $A, B \in D, A \neq B$, Algorithm 1 computes the characteristics of the segment $[A, B]$ included in D.*

Proof. We prove by induction on m that the output of SmartDSS(D, A^0, A^m) is the characteristics of the DSS $[A^0, A^m]$. For $m = 1$, SmartDSS(D, A^0, A^1) outputs $(0, 1, -A^0_y)$ if A^1 is $A^0 + (1, 0)$ directly (exit at beginning of iteration 1), or outputs $(1, 0, A^0_x)$ (exit at beginning of iteration 2) if A^1 is $A^0 + (0, 1)$. Both answers are correct.

Assuming the induction hypothesis is true for arbitrary $1 \leq k \leq m$, we prove the property holds for $m+1$. Let us assume that $[A^0, A^m]$ was a DSS (a^m, b^m, μ^m).

We know that SmartDSS(D, A^0, A^m) did output (a^m, b^m, μ^m) by induction hypothesis. According to Lemma 6, modifying the end point for this algorithm induces either $K(m+1) = K(m)$ or $K(m+1) = K(m) + 1$.

Case $K(m+1) = K(m)$. In this case, both SmartDSS(D, A^0, A^m) and SmartDSS(D, A^0, A^{m+1}) have exited at the same iteration at line 2. Since $[A^0, A^m]$ is a DSS (a^m, b^m, μ^m), we know by the loop invariant (Lemma 5) that U' and L' are the leftmost first weakly exterior points. But the exit condition entails both U' and L' are further away from A^{m+1}. Lemma 3 implies $[A^0, A^{m+1}]$ has the same characteristics as $[A^0, A^m]$. Since $K(m+1) = K(m)$, the returned value of SmartDSS(D, A^0, A^{m+1}) is also (a^m, b^m, μ^m), which concludes.

Case $K(m+1) = K(m) + 1$. In this case, at iteration $K(m)$, SmartDSS(D, A^0, A^m) exited at line 2 while SmartDSS (D, A^0, A^{m+1}) did not. We conclude easily that either $\|A^0U'\|_1 = \|A^0A^{m+1}\|_1$ or $\|A^0L'\|_1 = \|A^0A^{m+1}\|_1$. The properties of weakly exterior points entail that conditions $U' \in D$ and $L' \in D$ cannot be true at the same time. Since we are in the first quadrant, we have four cases to consider: $A^{m+1} = U'$, $A^{m+1} = L'$, $A^{m+1} = U' + (1, -1)$, $A^{m+1} = L' + (-1, 1)$.

$A^{m+1} = U'$. In this case, A^{m+1} is upper weakly exterior to DSS (a^m, b^m, μ^m) and therefore $[A^0A^{m+1}]$ is a DSS with characteristics as specified by Proposition 1. Now, $A^{m+1} = U' \Rightarrow U'_y \geq A^{m+1}_y$ and $A^{m+1} \in D \Rightarrow U' \in D$. The condition on line 3 is thus true. Lemma 5 applied at beginning of iteration $K(m+1)$ implies that $[A^0, U^2]$ has characteristics $(a^{m+1}, b^{m+1}, \mu^{m+1})$. Now U^2 is by construction A^{m+1} (according to UpdateSlope). We conclude that $[A^0, A^{m+1}]$ has characteristics $(a^{m+1}, b^{m+1}, \mu^{m+1})$. Since $K(m+1) = K(m) + 1$, the execution flow exits the loop immediately at line 2, thus the output is exactly $(a^{m+1}, b^{m+1}, \mu^{m+1})$, which is the correct answer.

$A^{m+1} = L'$. This case is completely symmetrical to the previous case, replacing U' by L' and U^2 by L^2, and the condition in line 7 being true (instead of line 3).

$A^{m+1} = U' + (1, -1)$. Let us compute the remainder of A^{m+1} for the DSS (a^m, b^m, μ^m). Since U' is upper weakly exterior (remainder $\mu^m - 1$) a short computation gives $\mu^m + a + b - 1$. Thus A^{m+1} is a lower leaning point of this DSS. Thus $[A^0, A^{m+1}]$ has characteristics (a^m, b^m, μ^m). We prove below that SmartDSS outputs also these characteristics.

Now, since $A^{m+1} \neq U'$ but $A^{m+1} \in D$, $U' \notin D$. Thus condition at line 3 is false at iteration $K(m)$. If $a^m = 0$, then condition $L' \in D$ at line 7 is false since the DSS is in the first quadrant. Otherwise, if *ulu* is true (*lul* is false), then U' is to the left of L': $U'_x + 1 < L'_x$. Thus condition $L'_x \leq A^{m+1}_x$ at line 7 is false because $A^{m+1}_x = U'_x + 1 < L'_x$. If *lul* is true, then L' is to the left of U'. But we know already that $[A^0, A^m]$ was a DSS. Being weakly exterior, L' touches one of these points, i.e. $L' + (-1, 1)$ is one of the A^i, $1 \leq i \leq m$. Since $L' + (-1, 1) \in D$, $L' \notin D$ by the rules of remainders. The condition in line 7 is thus also false in this case.

We conclude that the else statement (line 11) was executed at iteration $K(m)$. Both U' and L' are advanced and the DSS contains one more pattern and one more reversed pattern. The characteristics (a^m, b^m, μ^m) are not changed. At iteration $K(m) + 1$, the execution flow exits immediately at line 2 since both U' and L' are too far away (the vector (b, a) added to U' and L' has positive 1-norm). Therefore, SmartDSS(D, A^0, A^{m+1}) outputs (a^m, b^m, μ^m), which concludes.

$A^{m+1} = L' + (-1, 1)$. This case is completely symmetrical to the previous case, swapping points U' and L', vectors $(1, -1)$ and $(-1, 1)$, indices x and y, "to the left" and "bottom", and Booleans *ulu* and *lul*. □

3.3. Computational complexity of SmartDSS

Propositions 3 and 4 establish the time complexity of this algorithm with a tight output sensitive upper bound. Corollary 1 gives a non-tight input sensitive upper bound.

Proposition 3. *Let S be a* primitive *DSS of slope $\frac{a}{b} = [u_0, u_1, \ldots, u_k]$, contained in a DSL D. We denote by $T(S)$ the number of points tested by Algorithm* 1 *(test "$\in D$") to recognize its slope $\frac{a}{b}$. If $\sum_{i=0}^{k} u_i = n$, then $T(S) \leq 2n$ (it only depends on the sum of the partial quotients).*

Proof. We prove by induction on n that $T(S) \leq 2n$. For S of slope [0], the initial condition $T(S) = 0$ is obvious since S is just a one step horizontal segment and has slope 0. Algorithm 1 exits immediately from the loop at line 2 since tests $U'_y \leq B_y$ and $L'_x \leq B_x$ are false. Since A and the quadrant are known, the output is correct while no test "$\in D$" has been made. Assume that $T(S) \leq 2n$ for all primitive DSS S of slope $[u_0, u_1, \ldots, u_k]$ with $\sum_{i=0}^{k} u_i = n$. We shall prove that for any primitive DSS S' with a sum $n+1$, we have $T(S') \leq 2n + 2$.

Let S' be such a DSS, with slope $a'/b' = [u'_0, u'_1, \ldots, u'_{k'}]$ and sum $n+1$. The recognition process (Algorithm 1) is incremental, and corresponds exactly to a progressive descent in the Stern–Brocot tree. Therefore, the recognition process visits the father a/b of the node a'/b' at some point. According to Stern–Brocot tree and Proposition 1, there are only two possible evolutions for a slope of a DSS, either $[u_0, u_1, \ldots, u_k, \delta]$ or $[u_0, u_1, \ldots, u_k - 1, 1, \delta]$. Therefore, we have two cases for the partial quotients of $a/b = [u_0, u_1, \ldots, u_k]$:

- $k = k' - 1,\ \forall i = 0 \ldots k, u_i = u'_i, u'_{k'} = \delta$.
- $k = k' - 2, \forall i = 0, \ldots, k-1, u_i = u'_i, u_k = u'_k + 1, u'_{k'-1} = 1, u'_{k'} = \delta$.

In both cases, $\sum_{i=0}^{k} u_i = n + 1 - \delta$, $\delta \geq 1$. Therefore, at the moment the node a/b is reached, this first subpart of S' (called S hereafter) is itself a DSS with slope a/b and sum $\leq n$ (see Lemma 3 for more details). This subpart S is also primitive, even if S' itself contains δ repetitions of this pattern. We can apply the inductive hypothesis and we obtain $T(S) \leq 2(n + 1 - \delta)$.

Then, at each iteration of the loop at line 1, zero, one or two points are tested for their inclusion in D. Since the loop invariant guarantees that $[UU'[$ contains δ patterns or $[LL'[$ contains δ reversed patterns, the number of iterations between S and S' is exactly δ. It follows that at most 2δ points are tested between the recognition of S and the recognition of the slope of S'. Furthermore, once this slope is recognized, no other point is tested since S' was assumed primitive. Indeed, if any other point was tested (i.e. U' or L' is equal or before B), Lemma 3 implies that S' contains one pattern *and* one reversed pattern, which is contradictory with S' primitive.

Then, $T(S') \leq T(S) + 2\delta \leq 2(n+1-\delta) + 2\delta = 2n+2$. □

We may now give a bound on the time complexity of Algorithm 1.

Proposition 4. *For any DSL D of characteristics (α, β, ν), $\frac{\alpha}{\beta} = [u_0; u_1, \ldots, u_n]$ and for two points $A, B \in D$, Algorithm 1 computes the characteristics of the segment $S = [A, B]$ included in D with an output-sensitive time complexity $O(\delta + \sum_{i=0}^{k} u_i)$, where the slope of S is $[u_0, u_1, \ldots, u_k]$ and S contains δ patterns or reversed patterns. When S and D have same slope (i.e. $k = n$), the complexity is $O(\sum_{i=0}^{n} u_i)$.*

Proof. Proposition 2 tells that Algorithm 1 gives the correct output. Proposition 3 indicates that, when S is primitive, at most $2\sum_{i=0}^{k} u_i$ tests "$\in D$" are executed. This test is performed in $O(1)$ by the computation of the remainder $\alpha x - \beta y$ of the point in the DSL D. Looking closely at Algorithm 1 shows that all other operations are $O(1)$. Now, if S is not primitive and has δ patterns, two cases arise. If S does not have the same characteristics as D, then the loop iterates at most δ times for the δ patterns of S. If S has the same characteristics as D, the condition at line 1 exits immediately. This concludes. □

Corollary 1. *An input-dependent bound is $O(\sum_{i=0}^{n} u_i)$, where $[u_0, u_1, \ldots, u_n]$ is the slope of D. This bound is always greater than the one of Proposition 4.*

Proof. It is only reached as the worst case of the algorithm, when S has the same slope as D. □

Another equivalent way of presenting the time complexity is given by the following corollary.

Corollary 2. *The number of started iterations of the loop of* **SmartDSS** *is $1 + \delta + \sum_{i=0}^{k} u_i$.*

4. A coarsening algorithm for computing the characteristics of a subsegment included in a known DSL

In the next section, we describe our second algorithm **ReversedSmartDSS** for determining the characteristics of any subsegment S of a digital straight line D in the first quadrant, whose characteristics (α, β, μ) are known. We further assume that the input slope $\frac{\alpha}{\beta}$ is given as a continued fraction z_n with its convergents. We make use of the property that the slope of S is either $\frac{\alpha}{\beta}$ or any one of the ancestors of $\frac{\alpha}{\beta}$ in the Stern–Brocot tree ([19], see also Proposition 3 of [18]). The principle of our new algorithm is to follow a bottom-up way in the Stern–Brocot tree. Moreover, the algorithm jumps from principal convergent to previous principal convergent, except in the last iteration. It explains why this algorithm has a better worst-case complexity than **SmartDSS**, presented in the previous section.

Although **ReversedSmartDSS** seems to require more input data than **SmartDSS** for recognizing S, we note that the additional information is already known if the DSL D was recognized by a classical recognition algorithm (e.g., DR95 [3], or combinatorial algorithms [20]). Otherwise, the Euclid algorithm applied to $\frac{\alpha}{\beta}$ gives this additional information in $O(\log(\max(\alpha, \beta)))$ iterations.

In the next subsections, we begin by giving an overview of this new algorithm. Afterwards, we present subpatterns and the associated two main functions for computing them (**SmallestCovSubpattern** and **GreatestIncSubpattern**). We then discuss the case where $[A, B]$ is included in two patterns (function **DSSWithinTwoPatterns**). We finally take care of the case where $[A, B]$ is included in one pattern. Eventually, we show the complexity of the whole algorithm.

4.1. Overview of the algorithm

Algorithm 4 is the general algorithm for computing the exact characteristics of a segment S part of a DSL D, whose characteristics $(z_n = \frac{\alpha}{\beta}, \mu)$ are known. This algorithm thus computes the simplest DSL covering S. The segment S is defined by its two endpoints A and B, hereafter denoted by $[A, B]$. Lastly, we give also as input the two upper leaning points U_1 and U_2 of D which surround A and B.

Note that the points U_1 and U_2 can be computed in constant time from z_n and z_{n-1} with standard arithmetic. Without loss of generality, let us assume that $A = (0, 0)$ and $B = (B_x, B_y)$. Let (p_k/q_k) be the convergents of z_n. Then $(b, a) = (q_n, p_n)$, $(b', a') = Bezout(p, q, n)$. We compute $c_A = (\mu b') \div b$, then $U_1 = \mu(b', a') - c_A(b, a)$ and $U_2 = U_1 + (b, a)$.

The algorithm examines several cases listed below. The last case is the only one that makes a recursive call to itself. Cases are governed by the horizontal distance between U_1 and U_2, which in fact measures the number of patterns covering the segment. The whole algorithm is based on the notion of *subpattern*, and on finding either the smallest subpattern containing $[A, B]$ or the greatest included in $[A, B]$.

Function ReversedSmartDSS (**In** $(z = \frac{\alpha}{\beta}, \mu)$: DSL, **In** $U_1, U_2 \in \mathbb{Z}^2$, **In** $A, B \in \mathbb{Z}^2$) : DSL ;
Var Up_1, Up_2 : Points of $\mathbb{Z}^2$; /* *Updated upper leaning points.* */
Var dU : integer ; /* *Horizontal distance between* U_1 *and* U_2 */
begin
1 **if** $(A_x == B_x)$ **then return** $(\frac{1}{0}, A_x)$;
2 **if** $(A_y == B_y)$ **then return** $(\frac{0}{1}, -A_y)$;
$dU \leftarrow U_{2_x} - U_{1_x}$;
/* $[A, B]$ *contains at least one pattern of* z */;
3 **if** $(dU \geq 3\beta)$ *or* $(dU == 2\beta$ *and* $(A == U_1$ *or* $B == U_2))$ *or* $(A == U_1$ *and* $B == U_2)$ **then return** $(\frac{\alpha}{\beta}, \mu)$;
/* $[A, B]$ *is covered by two patterns of* z */;
4 **if** $(dU == 2\beta)$ **then**
return DSSWithinTwoPatterns $(\frac{\alpha}{\beta}, U_1, U_2, A, B)$;
/* $[A, B]$ *is covered by one pattern of* z */;
5 $(\frac{\alpha'}{\beta'}, m, Up_1, Up_2) \leftarrow$**SmallestCovSp**$(\frac{\alpha}{\beta}, U_1, U_2, A, B, true)$;
6 **if** $(Up_1 == U_1)$ *and* $(Up_2 == U_2)$ **then**
$(\frac{\alpha'}{\beta'}, m, Up_1, Up_2) \leftarrow$**GreatestIncSp**$(\frac{\alpha}{\beta}, U_1, U_2, A, B)$;
return $(\frac{\alpha'}{\beta'}, \alpha' Up_{1_x} - \beta' Up_{1_y})$;
7 **return ReversedSmartDSS**$((\frac{\alpha'}{\beta'}, \alpha' Up_{1_x} - \beta' Up_{1_y}), Up_1, Up_2, A, B)$;
end

Algorithm 4: Digital straight segment recognition algorithm with the bottom-up approach. The input is a digital straight line (DSL) specified as a pattern slope z and a shift to origin μ, two points A and B on this DSL such that $A < B$, hence forming a subsegment $[A, B]$. This algorithm computes the DSL with minimal characteristics that contains the subsegment $[A, B]$. The points U_1 and U_2 are the two upper leaning points of this DSL closest to A and B and such that $U_1 \leq A$ and $U_2 \geq B$.

1. *A and B have same abscissa or same ordinate*. The algorithm then stops and returns $(\frac{1}{0}, A_x)$ (or $(\frac{0}{1}, -A_y)$), which are the obvious results.
2. *The distance $U_{2_x} - U_{1_x}$ is*: (1) *three times greater than β*, (2) *equal to 2β*, A and U_1 *are superposed, or B and U_2 are superposed, or* (3) *A and U_1 are superposed, and B and U_2 are superposed.* The algorithm then stops and returns $(\frac{\alpha}{\beta}, \mu)$ (line 3). Indeed, in these cases, the DSS contains at least one pattern, so the characteristics immediately follow.
3. *This distance is 2β*. Segment $[A, B]$ is then strictly included in two patterns of D, with A in the first pattern and B in the second. The characteristics are computed by the function **DSSWithinTwoPatterns** (line 4), described by Algorithm 5.
4. *This distance is β*. Segment $[A, B]$ is included in one pattern of D. The function **SmallestCovSp** (Algorithm 6) attempts to extract a smaller subpattern that contains $[A, B]$. If this was impossible, then the exact characteristics of $[A, B]$ are the characteristics of the greatest subpattern included in $[A, B]$, that we obtain with a call to **GreatestIncSp** (Algorithm 7). Otherwise, segment $[A, B]$ is covered by a proper subpattern of D, whose slope is an ancestor of z_n in the Stern–Brocot. We recursively call **ReversedSmartDSS** on this simpler DSL.

4.2. Subpatterns of pattern

We begin by defining subpatterns as subsegments of a pattern anchored in the plane, i.e. words in $\{0, 1\}$ with a starting point in the plane (see Fig. 4(a) for an illustration).

Definition 1 (*Subpattern*)**.** Let $E(z_{2i+1}) = [U_1, U_2]$ be an odd pattern defined by its slope z_{2i+1} and its upper leaning points U_1 and U_2. Let us define the points $U^k = U_1 + k(q_{2i}, p_{2i})$ for $k \in \{1, \ldots, u_{2i+1}\}$. A *subpattern of* $[U_1, U_2]$ is any subsegment $[V_1, V_2]$ of the pattern $[U_1, U_2]$, for $V_1, V_2 \in \{U_1, U^1, \ldots, U^{u_{2i+1}}, U_2\}$ and $V_1 \leq V_2$.

Let $E(z_{2i}) = [U_1, U_2]$ be an even pattern defined by its slope z_{2i} and its upper leaning points U_1 and U_2. Let us define the points $U^k = U_2 - k(q_{2i}, p_{2i})$ for $k \in \{1, \ldots, u_{2i}\}$. A *subpattern of* $[U_1, U_2]$ is any subsegment $[V_1, V_2]$ of the pattern $[U_1, U_2]$, for $V_1, V_2 \in \{U_1, U^{u_{2i}}, \ldots, U^1, U_2\}$ and $V_1 \leq V_2$.

A *proper* subpattern is different from U_1U_2. A *null* subpattern is an empty segment.

The following lemma is obvious from the definition of points U^j, obtained by translation of a reduced partial of the slope z_n. Berstel formulas (1) induce the rules on the depth and slope of a subpattern.

Lemma 7. *A subpattern is a digital straight segment included in its pattern. A subpattern of an odd pattern $E(z_{2i+1})$ is a factor of the form $E(z_{2i})^m E(z_{2i-1})^{m'}$, where m, m' are integers and $0 \leq m \leq u_{2i+1}$, $0 \leq m' \leq 1$. A subpattern of an even pattern $E(z_{2i})$ is a factor of the form $E(z_{2i-2})^{m'} E(z_{2i-1})^m$, where m, m' are integers and $0 \leq m \leq u_{2i}$, $0 \leq m' \leq 1$.*

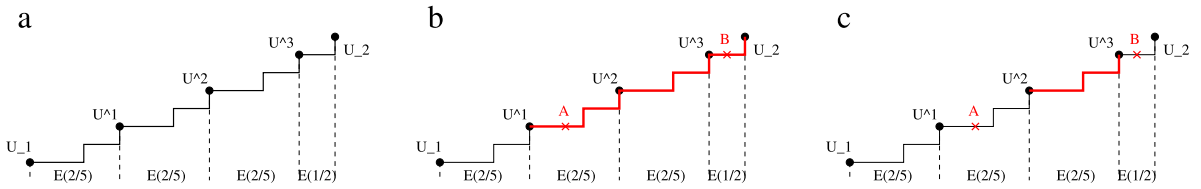


Fig. 4. Illustration of subpatterns: (a) odd pattern $E(\frac{7}{17}) = [0; 2, 2, 3])$, (b) smallest subpattern covering $[A, B]$ is $E(\frac{2}{5})^2E(\frac{1}{2})$, (c) greatest subpattern included in $[A, B]$ is $E(\frac{2}{5})$.

The depth and the slope of a subpattern of $w = E(z_n)$ *follows these rules:*

Depth/slope	$m' = 0$	$m' = 1$
$m = 0$	*(null)/(null)*	$n - 2/z_{n-2}$
$m = 1$	$n - 1/z_{n-1}$	$n - 1/[u_0, u_1, \ldots, u_{n-1} + 1]$
$2 \leq m \leq u_n$	$n - 1/z_{n-1}$	$n/[u_0, u_1, \ldots, u_{n-1}, m]$

It is clear that the slope of a proper non-null subpattern of z_n is always an ascendant of z_n in the Stern–Brocot tree. The slope is either a convergent with the same depth n but smaller quotient, or a principal convergent of depth $n - 1$ or $n - 2$. Two subpatterns are especially important in our context.

Definition 2. Given a pattern $E(z_n) = [U_1, U_2]$, $n \geq 1$, and a segment $[A, B]$ in this pattern, the *smallest subpattern of* $[U_1, U_2]$ *covering* $[A, B]$ is the subpattern $[U_1', U_2'] = E(z_{2i})^m E(z_{2i-1})^{m'}$ if $n = 2i + 1$ or the subpattern $[U_1', U_2'] = E(z_{2i-2})^{m'} E(z_{2i-1})^m$ if $n = 2i$, with the smallest integer m and smallest integer m' such that $[A, B]$ is a subsegment of $[U_1', U_2']$.

Symmetrically, the *greatest subpattern of* $[U_1, U_2]$ *included in* $[A, B]$ is the subpattern $[U_1', U_2'] = E(z_{2i})^m E(z_{2i-1})^{m'}$ if $n = 2i + 1$ or the subpattern $[U_1', U_2'] = E(z_{2i-2})^{m'} E(z_{2i-1})^m$ if $n = 2i$, with the greatest integer m and greatest integer m' such that $[U_1', U_2']$ is a subsegment of $[A, B]$.

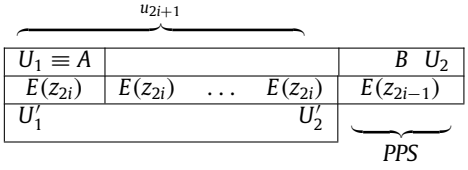
It is obvious that these subpatterns are unique and well-defined (see Fig. 4(b), (c) for an illustration).

Our algorithm is based on the following property of subpatterns.

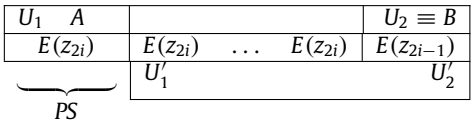
Proposition 5. *Let* $[A, B]$ *be some subsegment of a pattern* $[U_1, U_2]$. *If its smallest subpattern covering* $[A, B]$ *is the pattern itself, then its greatest subpattern included in* $[A, B]$ *defines the extremal upper leaning points of the digital straight segment* $[A, B]$. *Hence, the slope of this subpattern is the slope of* $[A, B]$.

Proof. We show only the case where $[U_1, U_2]$ is an odd pattern $E(z_{2i+1})$. The proof is made by the positions of A and B. Since the smallest covering subpattern is the pattern itself, the definition of smallest subpattern (Definition 2) implies that $A \in [U_1, U^1[$ and $B \in]U^{u_{2i+1}}, U_2]$. Otherwise said, Lemma 7 induces that point A belongs to the first $E(z_{2i})$ and point B belongs to the last $E(z_{2i-1})$. There are three possible cases (*PS* is previous slope z_{2i}, *PPS* is previous previous slope z_{2i-1}).

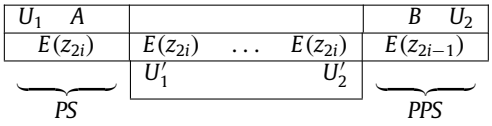
1. U_1 *and* A *are superposed*. The greatest subpattern included in $[A, B]$ is $[U_1', U_2'] = E(z_{2i})^{u_{2i+1}}$, where U_1' and U_2' are its upper leaning points.



2. U_2 *and* B *are superposed*. The greatest subpattern included in $[A, B]$ is $[U_1', U_2'] = E(z_{2i})^{u_{2i+1}-1} E(z_{2i-1})$.



3. *A and B are not respectively superposed to* U_1 *and* U_2. The greatest subpattern included in $[A, B]$ is $[U_1', U_2'] = E(z_{2i})^{u_{2i+1}-1}$.



For each case, the slope of $[A, B]$ is defined from the slope of the subpattern. This is due to the fact that the word $[A, U_1']$ is a strict right factor of $E(z_{2i})$ and hence does not modify the slope of a DSS $E(z_{2i})^m E(z_{2i-1})^{m'}$ when concatenated to the left. Furthermore, the word $[U_2', B]$ is a strict left factor of $E(z_{2i-1})$ and it does not modify the slope of a DSS $E(z_{2i})^m E(z_{2i-1})^{m'}$ when concatenated to the right. □

There are other cases where smallest subpatterns have a specific behavior, which will be used for proving the complexity of our algorithm.

Proposition 6. *Let $E(z_n)$ be some pattern, with $z_n = [u_0; u_1, \ldots, u_n]$, and $[A, B]$ a segment within. Let w be its smallest subpattern that covers $[A, B]$. If the slope of w has a depth n too, then w is a pattern and its smallest subpattern covering $[A, B]$ is w itself.*

Proof. We only prove the case where n is odd, i.e. $n = 2i + 1$. Lemma 7 implies w takes the form $E(z_{2i})^m E(z_{2i-1})^{m'}$, where m, m' are integers and $0 \leq m \leq u_{2i+1}$, $0 \leq m' \leq 1$. If the slope of w has a slope depth n too, then necessarily $m' = 1$ and $m > 1$. Indeed, if $m = 1$, then the slope is $[u_0; u_1, \ldots, u_{2i}, 1]$, which can be written as $[u_0; u_1, \ldots, u_{2i} + 1]$, a fraction of depth $2i = n - 1$. Since $m' = 1$ and $m > 1$, then w is a pattern of slope $z_n' = [u_0; u_1, \ldots, u_{n-1}, m]$. Thus, every subpattern of w is also a subpattern of $E(z_n)$, which entails immediately that the smallest pattern of w covering $[A, B]$ is the same as the smallest pattern of $E(z_n)$ covering $[A, B]$, which is w itself. This situation is illustrated below:

$[$	$\ldots$	A	$\ldots$	$[$	$]\ldots B\ldots]$
$E(z_{2i})$	$\ldots$	$E(z_{2i})$	$\ldots$	$E(z_{2i})$	$E(z_{2i-1})$
Smallest subpattern is		$E(z_{2i})$	$\ldots$	$E(z_{2i})$	$E(z_{2i-1})$

□

We mention the two following properties of subpatterns, which can be proven similarly:

Proposition 7. *The slope of the smallest subpattern of a pattern $E(z_n)$ covering a segment $[A, B]$ is either the slope of $[A, B]$ itself or one of its descendant in the Stern–Brocot tree.*

Proposition 8. *The slope of the greatest subpattern of a pattern $E(z_n)$ included in a segment $[A, B]$ is, when not null, either the slope of $[A, B]$ itself or one of its ascendant in the Stern–Brocot tree.*

The functions **SmallestCovSp** (Algorithm 6) and **GreatestIncSp** (Algorithm 7) compute subpatterns. The fact that function **SmallestCovSp**$(z_n, A, B, U_1, U_2, \mathit{false})$ computes exactly the smallest subpattern of $E(z_n) = [U_1, U_2]$ covering $[A, B]$ is discussed in Appendix. Similarly, the fact that function **GreatestIncSp**$(z_n, A, B, U_1, U_2, \mathit{false})$ computes exactly the greatest subpattern of $E(z_n) = [U_1, U_2]$ included in $[A, B]$ is discussed in Appendix.

4.3. Reversed subpatterns

We recall that a *reversed pattern* of slope z_n is the succession of Freeman moves between two consecutive *lower* leaning points of a DSL of slope z_n. The minimal characteristics of a digital straight segment may be defined by a reversed pattern and not by any pattern. We therefore also have to look for reversed patterns in a segment $[A, B]$ to determine in all cases its minimal characteristics.

Fortunately, reversed patterns share many characteristics with patterns. The most important one is that a reversed pattern of slope z_n is exactly the pattern of slope z_n read from right to left. Therefore Berstel formulas (1) must just be reversed for reversed pattern. Another way of doing that is to inverse the rules for odd and even patterns.

This is exactly what we do for extracting smallest covering and greatest included *reversed subpatterns*. The functions **SmallestCovSp** (Algorithm 6) and **GreatestIncSp** (Algorithm 7) computes also *reversed subpatterns* if the Boolean parameter *rev* is *true*. The inversion of odd and even rules is done in both cases at line 1. Nothing else is changed in the two functions.

One may question why everything is not symmetric in our algorithm, i.e. reversed subpatterns are checked wherever subpatterns are checked. In fact, for speed up purposes, as long there are at least two upper leaning points, we do not check for reversed patterns since they would induce the same characteristics. Therefore the only case where reversed patterns must be checked is when the sought segment has only one upper leaning point. In this case this segment is contained in two patterns and its upper leaning point is the middle one. This is why we take care of reversed patterns only in function **DSSWithinTwoPatterns** (Algorithm 5).

Proposition 9. *Let $[A, B]$ be some subsegment of a reversed pattern $[L_1, L_2]$ such that L_1 and L_2 are its lower leaning points and $L_1 \leq A < B \leq L_2$. If its smallest reversed subpattern covering $[A, B]$ is the reversed pattern $[L_1, L_2]$ itself, then its greatest reversed subpattern included in $[A, B]$ defines the extremal lower leaning points of the digital straight segment $[A, B]$. Hence, the slope of this reversed subpattern is the slope of $[A, B]$.*

Proof. The proof is exactly the same as the proof of Proposition 5, substituting odd and even and upper and lower leaning points. This proof holds since reversed patterns are patterns read from right to left. □

We add the following property which will be used in the next section.

Proposition 10. *If* $[U_1, U_m][U_m, U_2]$ *are two consecutive patterns of same slope,* L_1 *and* L_2 *their respective leaning points. Then any segment* $[A, B]$ *with* $U_1 \leq A \leq L_1$ *and* $L_2 \leq B \leq U_2$ *has the same characteristics as the segment* $[L_1, L_2]$.

Proof. Obviously since $[U_1, L_1]$ is a right factor of $[L_1, L_2]$ and $[L_2, U_2]$ is a left factor of $[L_1, L_2]$, adding them does not change the slope of the DSS (see Lemma 3 for more details). □

4.4. The segment $[A, B]$ *is included in two patterns of the DSL*

The function **DSSWithinTwoPatterns** (Algorithm 5) is really the core of this DSS recognition method. In principle, the whole algorithm could be written as three independent recursive functions: given some upper leaning point U_m within $[A, B]$, the first function looks for the best pattern with U_m as right upper leaning point, the second function looks for the best pattern with U_m as left upper leaning point, the third function looks for the best reversed pattern with only upper leaning point U_m.

However, this is not the method that achieves best complexity, since it would often occur that the first function ends up at the root of the Stern–Brocot tree, while the two other functions stop at a deepest slope (so in fewer calls). Therefore, we transform these three recursive functions into a loop where the three best patterns are sought progressively.

In the loop of **DSSWithinTwoPatterns**, three different patterns are tested progressively, so as to find the first that has exactly the sought slope. It is easy to see that, since the segment is included in two patterns, the middle upper leaning point U_m is the uppermost point of the segment. Therefore, if the slope of the segment is defined by a pattern (i.e. defined by two upper leaning points), then one of the extremities of the pattern is this point U_m. We thus test in sequence the possible patterns to the left and to the right of U_m. However, the slope of a DSS may also be defined by a reversed pattern (i.e. defined by two lower leaning points L_1 and L_2). We thus test also in sequence the possible reversed patterns.

The progressive computation of these three sequences of patterns (left pattern, right pattern, reversed pattern) is done in a parallel manner. More precisely, they are run consecutively one step at each time:

- line 3 computes the next covering subpattern that has U_m as its right upper leaning point. Either a smaller one is found that covers $[A, U_m]$ or it stops and z_{LU} is set to the greatest subpattern included in $[A, U_m]$.
- line 4 computes the next covering subpattern that has U_m as its left upper leaning point. Either a smaller one is found that covers $[U_m, B]$ or it stops and z_{RU} is set to the greatest subpattern included in $[U_m, B]$.
- line 5 computes the next covering reversed subpattern that has U_m as its upper leaning point. Either a smaller one is found that covers $[A, B]$ or it stops and z_L is set to the greatest reversed subpattern included in $[A, B]$ and containing U_m.

The conditions at lines 3, 4 and 5 correspond to three possible cases where respectively none, one, or two pattern computation(s) is/are stopped. At each iteration, the pattern, reversed or not, that lies in the deepest position in the Stern–Brocot tree is the candidate solution. It is indeed the slope of $[A, B]$ if it has stopped (Boolean b_* is *true*). The function then returns the characteristics of the elected pattern. Otherwise, the function loops and computes the new patterns.

Note that we use two functions **vUU**(z_n) and **vUL**(z_n). The first one simply returns the vector between two consecutive upper leaning points of the pattern z_n, i.e. (q_n, p_n). The second one returns the vector between an upper leaning point and the next lower leaning point. It is related to the Bézout vector of (q_n, p_n) as follows: **vUL**$(z_n = \frac{p_n}{q_n}) = (q_{n-1} + 1, p_{n-1} - 1)$ when n is even, and $(q_n - q_{n-1} + 1, p_n - p_{n-1} - 1)$ when n is odd.

We establish the proof of the correctness of algorithm **DSSWithinTwoPatterns** in Proposition 11.

Proposition 11. *Let* $[U_1, U_2]$ *be two patterns of slope* z_n *and let* $[A, B]$ *be a segment included in it, with* $U_1 \leq A < U_m < B \leq U_2$. *Then function* **DSSWithinTwoPatterns** *(Algorithm* 5*) calculates the greatest pattern or reversed pattern included in* $[A, B]$ *and containing* U_m. *Its slope is the slope of* $[A, B]$.

Proof. First of all, the algorithm stops. Indeed, at each iteration for each (reversed) pattern, either it is replaced by a strictly smaller (sub)pattern or its computation stops. At some point, all three will be stopped and condition at line 7 will be true: the algorithm then exits immediately.

The proof is given for the case of an odd pattern, $n = 2i + 1$. An illustration is given in the table below.

U_1		A			U_m	U_m			B	U_2
$E(z_{2i})$	...	$E(z_{2i})$	$E(z_{2i})$	$E(z_{2i-1})$		$E(z_{2i})$	⋯	$E(z_{2i})$	$E(z_{2i-1})$	
			U'_1		U_m	U_m	U'_2			

U_1	A	L_1		U_m		B	L_2	U_2
⋯		$E(z_{2i-1})$	$E(z_{2i})$	⋯	$E(z_{2i})$	$E(z_{2i})$	⋯	
		L'_1		L'_2				

We note first that all the points of $[A, B]$ except the point U_m are below the straight line (U_1U_2). Hence, since A and B are on different sides of U_m, we have by convexity that U_m is necessarily an upper leaning point of the DSS $[A, B]$.

We have two cases, depending on the fact that (i) $[A, B]$ has at least two upper leaning points or (ii) only one.

```
Function DSSWithinTwoPatterns
(In z_n = p_n/q_n : Pattern, /* Slope of the input two patterns. */
  In U_1, U_2 ∈ Z^2, /* Their first and last upper leaning points. */
  In A, B ∈ Z^2 /* The extremities of the segment. */
) : DSL ;
Var L_1, L_2, U_m, U'_1, U'_2, L'_1, L'_2, A', B' : ∈ Z^2; /* Leaning and other points. */
Var b_LU, b_RU, b_L : Boolean; /* Tells if the pattern is indeed the slope. */
Var z_LU, z_RU, z_L, z' : Pattern; /* Current patterns given as cont. fractions. */
begin
    b_LU ←false, b_RU ←false, b_L ←false ;
1   U_m ←U_1 + vUU(z_n); /* Middle upper leaning point between U_1 and U_2 */
2   L_1 ←U_1 + vUL(z_n) ; /* Lower leaning point in [U_1, U_m]. */
    L_2 ←L_1 + vUU(z_n) ; /* Lower leaning point in [U_m, U_2]. */
    z_LU ← z_n, z_RU ← z_n, z_L ← z_n ;
    while true do
        /* Subpattern left of U_m */
3       if ! b_LU then
            (z_LU, k, U'_1, U_m) ←SmallestCovSp(z_LU, U_1, U_m, A, U_m, false);
            if U'_1 = U_1 or k > 1 then
                (z_LU, k, U_1, U_m) ←GreatestIncSp(z_LU, U'_1, U_m, A, U_m, false);
                b_LU ← true ; /* Slope of [A, U_m] is z_LU. */
            else U_1 ← U'_1;
        /* Subpattern right of U_m */
4       if ! b_RU then
            (z_RU, k, U_m, U'_2) ←SmallestCovSp(z_RU, U_m, U_2, U_m, B, false);
            if U'_2 = U_2 or k > 1 then
                (z_RU, k, U_m, U_2) ←GreatestIncSp(z_RU, U_m, U_2, U_m, B, false);
                b_RU ← true ; /* Slope of [U_m, B] is z_RU. */
            else U_2 ← U'_2;
        /* Reversed subpattern containing U_m */
5       if ! b_L then
6           if A < L_1 then A' ← L_1 else A' ← A;
            if B > L_2 then B' ← L_2 else B' ← B;
            (z_L, k, L'_1, L'_2) ←SmallestCovSp(z_L, L_1, L_2, A', B', true);
            if L'_1 = L_1 and L'_2 = L_2 then
                (z_L, k, L_1, L_2) ←GreatestIncSp(z_L, L_1, L_2, A', B', true);
                b_L ← true ; /* Slope of [A, U_m, B] is z_L. */
            else L_2 ← U_m + vUL(z_L), L_1 ← L_2 − vUU(z_L) ;
        z' ← DeepestSlope(z_LU, z_RU, z_L) ; /* Deepest fraction z' = p'/q'. */
7       if (b_LU and z' = z_LU) or (b_RU and z' = z_RU) or (b_L and z' = z_L) then return (z', p'U_mx − q'U_my) ;
end
```

Algorithm 5: Given two patterns $E(z_n)^2 = [U_1, U_2]$ and a subsegment $[A, B]$, $U_1 < A < U_m < B < U_2$, computes the minimal characteristics of the segment $[A, B]$.

Case (i). Since U_m is already an upper leaning point, we have to look for another upper leaning point U' either before U_m or after U_m.

If the characteristics of $[A, B]$ are defined by $[U', U_m]$, then we claim that the pattern $[U_1, U_m]$ of slope z_{LU} will eventually be this pattern. We already have with Proposition 5 that z_{LU} is exactly this pattern when the smallest subpattern covering $[A, U_m]$ was the pattern itself. Otherwise, the smallest subpattern is not the pattern itself. The slope z_{LU} has then changed. If the subpattern is repeated ($k > 1$), then we can already conclude since $k - 1$ pattern(s) z_{LU} are included in $[A, U_m]$. Otherwise, the process will continue at the next iteration with one smaller pattern whose slope depth is either n, $n - 1$ or $n - 2$ (Lemma 7). At some point, the smallest subpattern covering $[A, U_m]$ will be the pattern itself, and Proposition 5 will conclude.

If the characteristics of $[A, B]$ are defined by $[U_m, U']$, then we claim that the pattern $[U_m, U_2]$ of slope z_{RU} will eventually be this pattern. The reasoning is identical to the previous one.

Case (ii). Then there are two lower leaning points L'_1 and L'_2, with $A \leq L'_1 < U_m < L'_2 \leq B$, such that the reversed pattern $[L'_1, L'_2]$ defines the slope of $[A, B]$. We claim that the pattern $[L_1, L_2]$ of slope z_L will eventually be this reversed pattern. We use first Proposition 10 to justify line 6: A and B are casted in $[L_1, L_2]$ before looking for reversed subpatterns. We then have

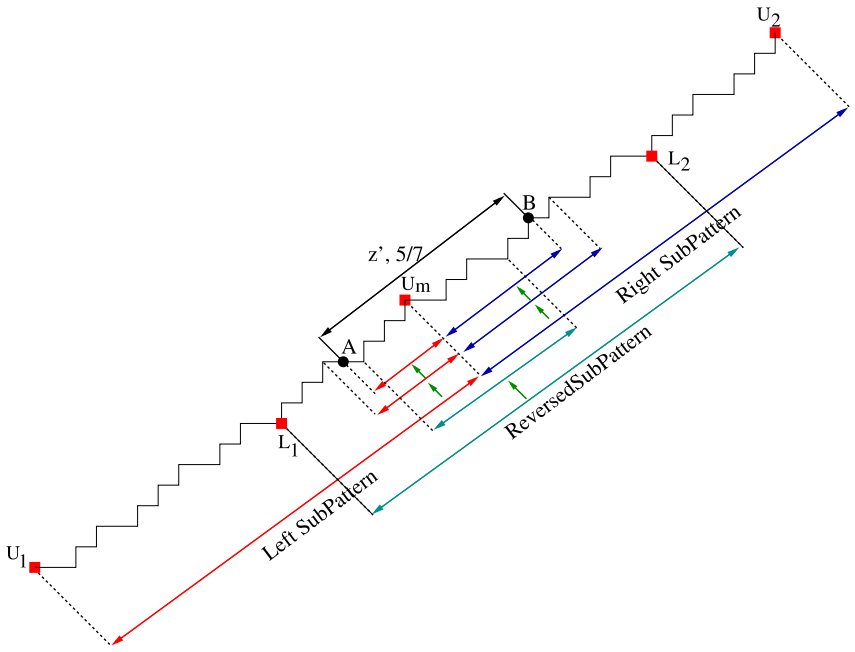


Fig. 5. Illustration of Algorithm 5. A DSL D of odd depth slope 13/18 with two patterns containing a segment $[A, B]$. U_1 and U_2 are respectively their leftmost and rightmost upper leaning points. Upper and lower leaning points are drawn as red boxes. The (red, blue or cyan) arrows represent bottom-up move along the Stern–Brocot Tree for respectively the patterns z_{LU}, z_{RU} and reversed pattern z_L. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

with Proposition 9 that z_L is exactly this reversed pattern when the smallest reversed subpattern covering $[A', B']$ was the reversed pattern itself.

Otherwise, the smallest reversed subpattern is not the pattern itself. The slope z_L has then been changed. Even if the reversed subpattern has k reversed patterns, we only keep the reversed subpattern that contains U_m since we look only for one reversed pattern (as noticed above, if there are two reversed patterns, it means there is at least one pattern and z_{LU} or z_{RU} will conclude). The process will continue at the next iteration with one smaller pattern whose slope depth is either n, $n-1$ or $n-2$ (Lemma 7). At some point, the smallest reversed subpattern covering $[A', B']$ will be the pattern itself, and Proposition 9 will conclude.

We have shown above that one of z_{LU}, z_{RU} or z_L will correspond to the sought slope. If the deepest slope corresponds to a "stopped" pattern z', the other patterns are then factor of this pattern (according to Berstel formulas), and hence will not modify its slope. $[A, B]$ has thus the slope of z'. □

Example. Let us look at a run of function **DSSWithinTwoPatterns** (Algorithm 5) for the DSL of slope $z_n = 13/18 = [0, 1, 2, 1, 1, 2]$ (Fig. 5), for the subsegment $[A, B]$. Here the segment is included in two patterns 13/18.

For the first step, pattern z_{LU} has slope 3/4 (call **SmallestCovSp**$(z, U_1, U_m, A, B, false)$, see Fig. 6(a)), right pattern has slope 5/7 (call **SmallestCovSp**$(z, U_m, U_2, A, B, false)$, see Fig. 6(c)), and reversed pattern has slope 5/7 (call **SmallestCovSp**$(z, L_1, L_2, A', B', true)$, see Fig. 6(b)). We compute the deepest slope of 3/4, 5/7 and 5/7, which is 5/7. As pattern z_L is stopped and has the deepest slope, Algorithm 5 exits and returns this slope (final result 5/7).

The whole process is illustrated in Fig. 5. All the computations for each of the three (reversed) patterns are displayed, although fewer computations are done. Indeed, since patterns are computed in a parallel manner, the first pattern that finds the correct answer makes the algorithm stop.

Let us now give some explanations of (Fig. 6(a)). In the first step, we fix X'_1 at U_1 and X'_2 at U_m, and we compute L the length of pattern $z_n = 13/18$ and L' the length of previous pattern $z_{n-1} = 5/7$ ($L = 13 + 18 = 31$ and $L' = 5 + 7 = 12$). Since the depth of z_n is odd (line 1, Algorithm **SmallestCovSp**), then we calculate k_1 ($k_1 = 2$) the number of previous pattern 5/7 just before or equal to A from X'_1 to the right. As $B = X'_2$, then B in the $E(z_{n-2})$ part and the function **SmallestCovSp** stops and returns $(z' = 3/4, k = 1, X'_1, X'_2)$, such that $X'_1 = X_1 + k_1(7, 5)$ and $X'_2 = X'_1 + k(4, 3)$. We have now a new slope $z' = 3/4$ between the new X'_1 and X'_2. In the second step, as $X'_1 \neq U_1$ and $k = 1$, then we recall the function **SmallestCovSp** with $z_n = 3/4$, $U_1 = X'_1$ and $U_2 = U_m$. Since the depth of z_n is even (line 2, Algorithm **SmallestCovSp**), then we calculate k_1 ($k_1 = 0$) the number of previous patterns 1/1 just after or equal to B from X'_2 to the left. As A is in the $E(z_{n-1})$ part, then $k_2 = 3$ and the function **SmallestCovSp** stops and returns $(z' = 1/1, k = k_2 - k_1, X'_1, X'_2)$, such that $X'_2 = X_2 - k_1(1, 1)$ and $X'_1 = X'_2 - k(1, 1)$. As X'_1 reaches A, this algorithm stops and returns the slope 1/1. The tables below display the states of patterns `left pattern`, `right pattern` and `reversed pattern` that are seen in Fig. 6.

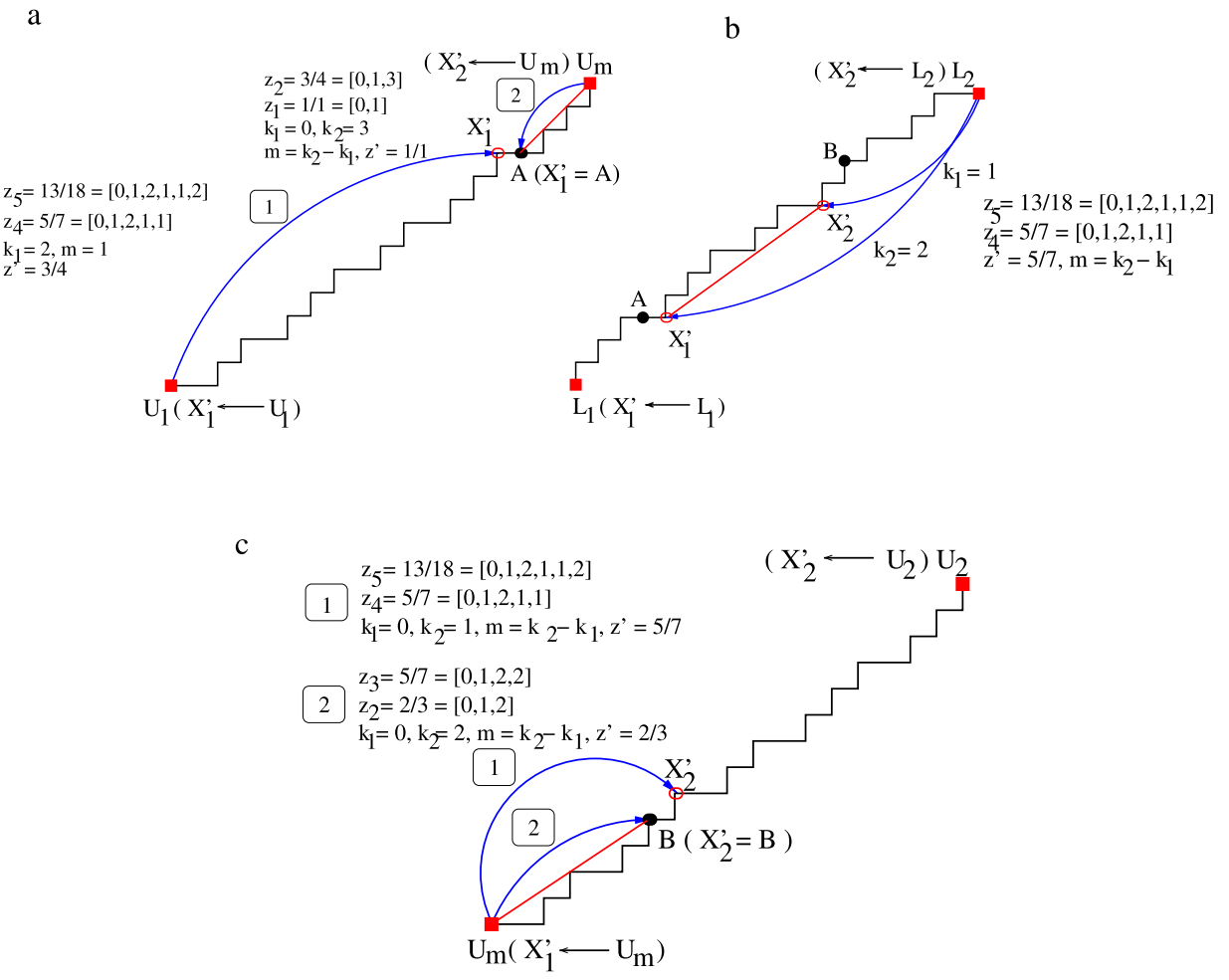


Fig. 6. Illustration of Algorithms **SmallestCovSp** and **GreatestIncSp**. The DSS L_1L_2 (resp. U_1U_m and U_mU_2) of characteristics (13/18, 0), which is a subset of U_1U_2 of Fig. 5. The blue arrows represent the move between the lower (upper) leaning points. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Iter	U_1	U_m	U_2	L_1	L_2	z_n	z_{LU}	z_{RU}	z_L	Case
	(0, 0)	(18, 13)				13/18	3/4			CovSp
1		(18, 13)	(36, 26)			13/18		5/7		CovSp
				(16, 10)	(23, 15)	13/18			5/7	CovSp, GIncSp
2	(14, 10)	(18, 13)				3/4	1/1			CovSp, GIncSp
		(18, 13)	(25, 18)			5/7		2/3		CovSp, GIncSp

For example, called separately, the `left pattern` would exit at iteration 2 ≤ 6-2 (6 is the depth of the input DSL (D) $\frac{13}{18}$ = [0, 1, 2, 1, 1, 2] and 2 is the depth of the output DSS (S) $\frac{1}{1}$ = [0, 1]). The output would be the DSS(1/1, $1*U_{m_x} - 1*U_{m_y}$). But `ReversedSubPattern` has already exited at iteration 1 with output DSS (5/7, $5*U_{m_x} - 7*U_{m_y}$), which was the deepest slope, and so is the final result.

4.5. Correctness of ReversedSmartDSS algorithm

We prove below the correctness of the function **ReversedSmartDSS** (Algorithm 4) for computing the minimal characteristics of some subsegment of a known DSL.

Proposition 12. *In the first quadrant, for any DSL D of slope z_n and shift μ, for any A, B ∈ D with A ≺ B, a call to* **ReversedSmartDSS**$((z_n, \mu), U_1, U_2, A, B)$ *computes the minimal characteristics of the segment [A, B] included in D, provided that U_1 and U_2 are the upper leaning points of D closest to A and B, $U_1 \preceq A$ and $B \preceq U_2$.*

Proof. We prove this result by induction on the depth n of the slope of the input DSL D. The initial steps $n = -1$ or $n = 0$ are obvious since D is just a vertical or horizontal segment and has slope $1/0$ or $0/1$. In this case, Algorithm 4 returns the correct characteristics $(\frac{1}{0}, A_x)$ or $(\frac{0}{1}, -A_y)$ (lines 1 and 2). The induction hypothesis is that this algorithm has a correct output for every segment $[A, B]$ that is a subset of a DSL of slope $[u_0; u_1, \ldots, u_n]$. We shall prove that the output is also correct for every segment $[A, B]$ that is a subset of some DSL D with a depth of its slope equal to $n + 1$.

If the horizontal distance between the upper leaning points U_1 and U_2 verifies one of the three conditions of line 3, then Algorithm 4 stops and returns the characteristics of the segment $[AB]$ which are trivially in this case the characteristics of D itself (i.e. depth is $n + 1$).

Otherwise, if this distance is equal to 2β, then Algorithm 4 returns the result of **DSSWithinTwoPatterns**$(z_{n+1}, U_1, U_2, A, B)$ at line 4. In this case, Proposition 11 concludes directly.

The last case occurs when S is included in only one pattern of D. At line 5, the smallest subpattern w of z_{n+1} covering $[A, B]$ is computed (let us denote its slope by z'). If it is z_{n+1} itself, then the function returns the characteristics of the greatest subpattern included in $[A, B]$ and Proposition 5 concludes.

Otherwise, the subpattern w covers $[A, B]$ hence $[A, B]$ is a subsegment of the DSL carried by the slope z'. Function **ReversedSmartDSS** recursively calls itself with this new slope and updated upper leaning points. Lemma 7 indicates that the subpattern w may have also a slope depth of $n + 1$, i.e. $z' = [u_0; u_1, \ldots, u_n, k]$, with $k < u_{n+1}$. In this case, the recursive call will end at line 6, since the smallest covering subpattern will be the pattern itself, and Proposition 5 will conclude. Otherwise the subpattern may have a slope depth n or $n-1$. In both cases, the induction hypothesis concludes the argument. □

4.6. Computational complexity of ReversedSmartDSS

We establish the time complexity of this algorithm in the proposition below, as a function of the depth of the slopes of the input DSL and output DSS.

We assume that we have stored all the convergents of the slope of D before running Algorithm 4. We further assume a computing model where standard arithmetic operations takes $O(1)$. Note that the largest integer used in the presented algorithms is lower than $\alpha^2 + \beta^2$, if the slope of the input DSL is $\frac{\alpha}{\beta}$, for a frame centered on the DSS.

Proposition 13. *Function* **ReversedSmartDSS** *(Algorithm 4) takes $O(n - n')$ time complexity, where n is the depth of the slope $\frac{\alpha}{\beta} = [u_0, u_1, \ldots, u_n]$ of the input DSL and n' is the depth of the slope $\frac{a}{b} = [u_0, u_1, \ldots, u_{n'-1}, u'_{n'}]$ of the output DSS $[A, B]$. The following table sums up the complexity of the different functions.*

Algorithm	*Complexity*
ReversedSmartDSS	$O(n - n')$
DSSWithinTwoPatterns	$O(n - n')$
SmallestCovSp	$O(1)$
GreatestIncSp	$O(1)$

Proof. First of all, it is clear that functions **SmallestCovSp** and **GreatestIncSp** (Algorithms 6 and 7) have constant time complexity assuming standard arithmetic operations are $O(1)$ and knowledge of convergents of z_n. Furthermore, computation of Algorithm 4 on line 1, 2 and 3 is clearly $O(1)$.

We prove the complexity of **ReversedSmartDSS** by induction on the depth of the input slope n. If $n = -2$, the input slope and the output slope are both ∞ and the function exits immediately at line1 in $O(1)$. If $n = -1$ both slopes are 0 and the function exits immediately at line 2 in $O(1)$. Assume now the complexity is correct for any n, $n \geq -1$, let us prove this complexity for an input slope of depth $n + 1$, whichever is the output slope n'. We know already that $n' \leq n$ since the slope of $[A, B]$ is an ascendant of the slope of $\frac{\alpha}{\beta}$ in the Stern–Brocot tree.

Easy cases are $O(1)$. If the segment $[A, B]$ is horizontal or vertical, the correct output slope is given in $O(1)$ at line1 or 2, which is some $O(n - n')$. If $[A, B]$ is included in three patterns or more or in two patterns but either A or B lies at some extremity, then the output slope is given in $O(1)$ at line 3, which is also some $O(n - n')$.

Segment $[A, B]$ lies over two patterns, complexity is $O(n + 1 - n')$. Otherwise, if $[A, B]$ is included in two patterns of the DSL (line 4), the function **DSSWithinTwoPatterns** is called (Algorithm 5). In this case, Proposition 11 indicates that the slope of $[A, B]$ is given either by z_{LU}, z_{RU} or by z_L. Since all operations within the main loop are $O(1)$ (mainly calls to **SmallestCovSp** and **GreatestIncSp**), the question is how many iterations are done. Let us consider z_{LU} for instance. Its depth is $n + 1$ at the beginning. Looking at line 3 shows that its smallest subpattern covering $[A, U_m]$ is computed. There are three cases:

(i) The smallest subpattern is z_{LU} itself or is some repetition of the same pattern. Then the computation is stopped and the number of iterations is 1.
(ii) The smallest subpattern has a slope of depth $n + 1$ too (see Lemma 7). In this case, Proposition 6 tells that the next call to **SmallestCovSp** will return the pattern itself. Thus the computation will stop at next iteration, and the number of iterations is 2.

(iii) The smallest subpattern has a slope strictly smaller than $n+1$, i.e. n or $n-1$. The computation of z_{LU} then continue with an input slope of depth strictly smaller than $n+1$.

Let n_{LU} be the depth of the exact slope of $[A, U_m]$. The three preceding cases induce immediately that the number of iterations to determine this slope is at most $n+1-n_{LU}+2$.

The same reasoning is applied to z_{RU} and z_L. Now, the computations of z_{LU}, z_{RU} and z_L are made in parallel, one step for each at each iteration. The exact output slope is either given by $[AU_m]$, $[U_mB]$ or a reversed pattern in $[AB]$. For each one, the number of iterations is upper bounded respectively by $n-n_{LU}+3$, $n-n_{RU}+3$ and $n-n_L+3$. If the output slope is $z'=z_{LU}$ (say), then $n'=n_{LU}$ and $n_{LU}\geq n_{RU}$ and $n_{LU}\geq n_L$. Thus z_{LU} stops after at most $n-n_{LU}+3$ iterations and is then necessarily one of the deepest slopes. The algorithm exits at line 7 after at most $n-n_{LU}+3=n+1-n'+2$ loops. Overall complexity is thus $O(n+1-n')$. Other cases are similar.

Segment $[A, B]$ *lies within one pattern, complexity is* $O(n+1-n')$. In this case, let w be the smallest subpattern of z_{n+1} covering $[A, B]$. Again, there are three cases:

(i) The smallest subpattern is z_{n+1} itself. Proposition 5 indicates that the depth n' of the output slope is either $n+1$, n or $n-1$. The output slope is immediately returned by a call to **GreatestIncSp** in $O(1)$.

(ii) The smallest subpattern has a slope of depth $n+1$ too (see Lemma 7). In this case, Proposition 6 tells first that the subpattern is a pattern and second that the next call to **SmallestCovSp** will return the pattern itself. Function **ReversedSmartDSS** is recursively called. It is easy to check that we are again in the case where the segment $[A, B]$ lies within one pattern, so **SmallestCovSp** returns the pattern w itself. Hence n' is again either the depth n' or the output slope. The output slope is immediately returned by a call to **GreatestIncSp** in $O(1)$.

(iii) The smallest subpattern has a slope of depth n or $n-1$ (see Lemma 7). Function **ReversedSmartDSS** is recursively called with this new digital straight line. We can apply the induction hypothesis and concludes that this function returns after $O(n-n')$ time complexity.

We have examined all cases. Time complexity of **ReversedSmartDSS** is some $O(n-n')$. □

Lamé's theorem implies that Algorithm 4 takes at most $O(\log(\max(\alpha,\beta)))$ time.

Corollary 3. *An input-sensitive bound of* **ReversedSmartDSS** *is* $O(\log(\max(\alpha,\beta)))$ *if* $z_n=\frac{\alpha}{\beta}$ *is the slope of* D*, or equivalently* $O(n)$*. This input-sensitive bound is greater or equal to the output-sensitive bound of Proposition 13.*

5. Experimentation

We have implemented the presented algorithms **SmartDSS** and **ReversedSmartDSS** in the open-source library `DGtal` [5], more specifically the Arithmetic package. We have then compared the running times of these two algorithms for increasing slope numerators and denominators. We have also run a standard arithmetic DSS recognition algorithm [3] both to validate the extracted characteristics and to compare the running times.

The experiments were conducted as follows for each one of the three algorithms (**ArithmeticDSS**, **SmartDSS**, **ReversedSmartDSS**):

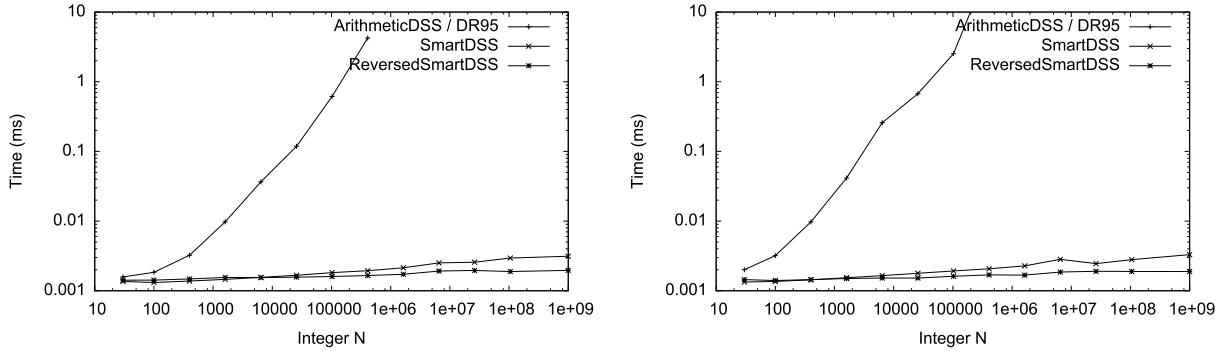
1. A maximal value N for α and β is chosen.
2. A maximal value M for A_x is chosen as either $N/2$ (one half of N) or $N/10$ (one tenth of N).
3. Each experiment is done for 40 000 randomly chosen α/β, where both α and β are randomly chosen in $\{1,\ldots,M\}$ with a uniform law.
4. For each slope α/β, five different shifts μ are randomly chosen in $\{0,\ldots,\alpha+\beta-1\}$ with a uniform law.
5. For each DSL $(\alpha/\beta,\mu)$, ten different abscissae A_x are randomly chosen in $\{0,\ldots,M-1\}$ with a uniform law. The ordinate A_y is chosen so that A belongs to the DSL. The abscissa B_x is randomly chosen in $\{A_x+1,\ldots,A_x+M\}$ with a uniform law. The ordinate B_y is chosen so that B belongs to the DSL.
6. The chosen algorithm is then executed to compute the exact characteristics of the subsegment $[A, B]$ of the digital straight line $(z=\frac{\alpha}{\beta},\mu)$.
7. We measure the total running time T in ms of these $40\,000\times5\times10$ executions. The time per call in ms is then $T/2\,000\,000$.

Note that each algorithm was fed with the same sequence of input parameters.

The running times for these experiments are given in Fig. 7. The linear complexity of **ArithmeticDSS** with respect to $|AB|_1$ is clear, while both **SmartDSS** and **ReversedSmartDSS** are logarithmic (see Fig. 8). **ReversedSmartDSS** is generally better than **SmartDSS**, especially for big segments with complex slopes. It is worthy to note that our new algorithms are already 20% faster than **ArithmeticDSS** for segments of size ≈5. Speed-ups are given in Table 2.

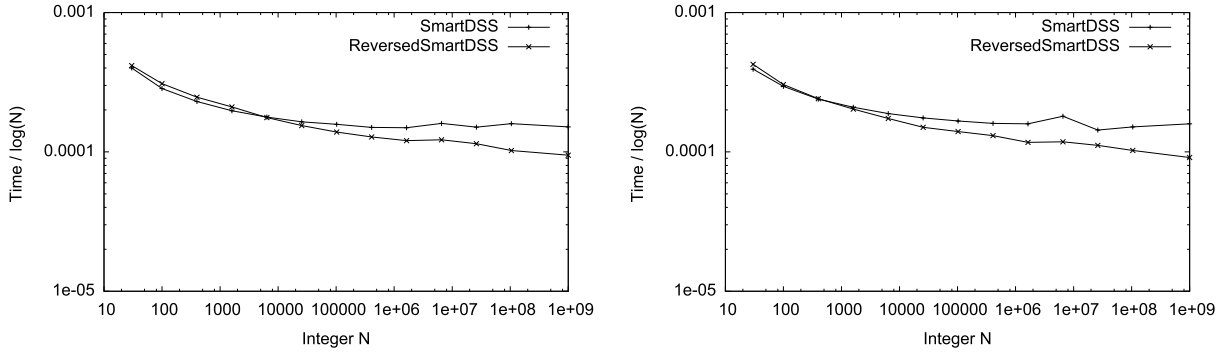
Other experiments were presented in [17].

Implementation details. Implementing fractions so as to have constant-time access to convergents of a fraction is not trivial. The naive approach that copies quotients has of course a complexity $\Theta(n)$, where n is the depth of the fraction. We adopt a completely different approach. We construct the Stern–Brocot on demand. Each fraction is thus computed once and placed forevermore in its correct position in the tree, with its correct fathers (left and right). Getting a convergent is thus just moving to the left or right ascendant node, which takes $O(1)$ time complexity. This is done in class `SternBrocot`, repertory `DGtal/math/arithmetic`.



(a) M is $N/10$. (b) M is $N/2$.

Fig. 7. Execution times (in log-scale) for determining the minimal characteristics of a digital straight segment included in a known digital straight line: in abscissa, the increasing sequence of parameter N; in ordinate, the running time in ms. Algorithms **SmartDSS** and **ReversedSmartDSS** are compared to the standard arithmetic DSS recognition (i.e. **DR95** [3] mentioned in [13]). See text for an explanation of the parameters N and M.



(a) M is $N/10$. (b) M is $N/2$.

Fig. 8. Comparing execution times of Algorithms **SmartDSS** and **ReversedSmartDSS** with the logarithmic bound $O(\log(N))$: in abscissa, the increasing sequence of parameter N; in ordinate, the time divided by $\log(N)$. See text for an explanation of the parameters N and M.

Table 2
Speed-up factors of our new DSS recognition algorithms with respect to the standard arithmetic recognition algorithm (i.e. **DR95** [3]). The slope parameters α and β of the digital straight line are randomly chosen in $\{1, \ldots, N\}$. The distance between the extremities of the segment is randomly chosen in $\{1, \ldots, M\}$.

N	Speed-up factor wrt **ArithmeticDSS**			
	SmartDSS		**ReversedSmartDSS**	
	$M = N/10$	$M = N/2$	$M = N/10$	$M = N/2$
30	1, 2	1, 5	1, 1	1, 4
100	1, 4	2, 3	1, 3	2, 3
400	2, 3	6, 8	2, 2	6, 8
1 600	6, 7	26, 9	6, 3	27, 7
6 400	23, 5	156, 7	23, 7	169, 8
25 600	70, 9	378, 3	75, 5	441, 9
102 400	338, 1	1312, 9	383, 7	1564, 2
409 600	2195, 0	22 274, 8	2574, 1	27 239, 4

6. Conclusion

We have presented two novel fast DSS recognition algorithm, in the special case where a DSL container is known. Their principle is to move either in a bottom-up or top-down way along the Stern–Brocot tree of irreducible fractions, starting from the root or from the initial known DSL slope. We have proved their correctness and we have given input and output-sensitive bounds, the latter ones being tight. Finally, we have implemented these algorithms in the open-source library `DGtal` [5]. Experiments have shown the considerable speed-up that they achieve with respect to standard DSS recognition algorithm, even for small segments. Our **ReversedSmartDSS** algorithm is only sensitive to the depth of the input DSL and output DSS, and is clearly sublinear. **SmartDSS** algorithm is more sensitive to the partial quotient values of the output DSS, but is generally sublinear.

```
Function SmallestCovSp
(In z_n = p_n/q_n : Pattern, /* Input slope as a continued fraction. */
  In X_1, X_2 ∈ Z^2, /* Upper or lower leaning points. */
  In A, B ∈ Z^2, /* Extremities of segment [A, B]. */
  In rev : Boolean /* when true, consider z_n as a reversed pattern. */
  ) : (z' : Pattern, m : integer, X'_1, X'_2 : Points of Z^2);
Var k_1, k_2 : integer;
begin
  if p_n = 0 or q_n = 0 then return (p_n/q_n, 1, X_1, X_2) ;
  L ← p_n + q_n ; /* Length of pattern z_n */
  L' ← p_{n-1} + q_{n-1} ; /* Length of previous pattern z_{n-1} */
1 if (! rev and odd(n) ) or (rev and even(n) ) then
    /* Case E(z_n) = E(z_{n-1})^{u_n} E(z_{n-2}). */
    k_1 ← ⌊|A − X_1|/L'⌋ ; /* Position of A wrt the left. */
    if |B − X_1| > u_n L' then
      /* B in the E(z_{n-2}) part. */
      z' ← [u_0; u_1, ..., u_{n-1}, u_n − k_1] ; /* k_1-th father of z_n */
      m ← 1 ; /* One covering pattern. */
    else
      k_2 ← ⌈|B − X_1|/L'⌉ ; /* B in some E(z_{n-1}). */
      z' ← z_{n-1}; /* Previous convergent of z_n. */
      m ← k_2 − k_1 ; /* k_2 − k_1 covering patterns z_{n-1}. */
    X'_1 ← X_1 + k_1(q_{n-1}, p_{n-1}) ; /* New position of leaning point. */
    X'_2 ← X'_1 + m(q', p') ; /* Where z' = p'/q'. */
2 else
    /* Case E(z_n) = E(z_{n-2})E(z_{n-1})^{u_n}. */
    k_1 ← ⌊(L − |B − X_1|)/L'⌋ ; /* Position of B wrt the right. */
    if |A − X_1| < L − u_n L' then
      /* A in the E(z_{n-2}) part. */
      z' ← [u_0; u_1, ..., u_{n-1}, u_n − k_1] ; /* k_1-th father of z_n */
      m ← 1 ; /* One covering pattern. */
    else
      k_2 ← ⌈(L − |A − X_1|)/L'⌉ ; /* A in some E(z_{n-1}). */
      z' ← z_{n-1}; /* Previous convergent of z_n. */
      m ← k_2 − k_1 ; /* k_2 − k_1 covering patterns z_{n-1}. */
    X'_2 ← X_2 − k_1(q_{n-1}, p_{n-1}) ; /* New position of leaning point. */
    X'_1 ← X'_2 − m(q', p') ; /* Where z' = p'/q'. */
  return (z', m, X'_1, X'_2);
end
```

Algorithm 6: **Input**: If *rev* is false, a pattern $E(z_n) = [X_1, X_2]$ with its two upper leaning points X_1 and X_2, or if *rev* is true, a reversed pattern $\hat{E}(z_n) = [X_1, X_2]$ with its two lower leaning points X_1 and X_2. A and B are two points of this segment such that $X_1 \leq A < B \leq X_2$. **Output**: the smallest (reversed if *rev* is true) subpattern $E(z')^m = [X'_1, X'_2]$ that covers $[A, B]$.

An interesting question is whether or not it is possible to compute the exact characteristics of a subsegment of a DSL in constant time. Even better, is it possible to obtain an analytical description as for DSL [8,18]? We intend to explore these open problems in future works.

Appendix. Functions computing subpatterns

The functions **SmallestCovSp** (Algorithm 6) and **GreatestIncSp** (Algorithm 7) compute subpatterns.

*A.1. Overview of **SmallestCovSp** algorithm*

Let X_1 and X_2 be the upper (lower if *rev* is true) leaning points of $z_n = \frac{p_n}{q_n}$ and A and B are the extremities of segment $[A, B]$. The function **SmallestCovSp** (Algorithm 6) computes the smallest covering (reversed if *rev* is true) subpatterns $E(z')^m = [X'_1, X'_2]$ that cover $[A, B]$, such that X'_1 and X'_2 are the new positions (upper or lower) leaning points. It is easy to see that this function is related to the depth of the current pattern z_n and the Boolean *rev* (line 1, 2).

```
Function GreatestIncSp
(In z_n = p_n/q_n : Pattern, /* Input slope as a continued fraction. */
  In X_1, X_2 ∈ ℤ^2, /* Upper or lower leaning points. */
  In A, B ∈ ℤ^2, /* Extremities of segment [A, B]. */
  In rev : Boolean /* when true, consider z_n as a reversed pattern. */
  ) : (z' : Pattern, m : integer, X'_1, X'_2 : Points of ℤ^2);
Var k_1, k_2 : integer;
begin
  if p_n = 0 or q_n = 0 then return (p_n/q_n, 1, X_1, X_2) ;
  L ← p_n + q_n ; /* Length of pattern z_n */
  L' ← p_{n-1} + q_{n-1} ; /* Length of previous pattern z_{n-1} */
1 if (! rev and odd(n) ) or (rev and even(n) ) then
    /* Case E(z_n) = E(z_{n-1})^{u_n} E(z_{n-2}). */
    k_1 ← ⌈|A − X_1|/L'⌉ ; /* Position of A wrt the left. */
    if B = X_2 then
      /* B at right extremity of E(z_{n-2}) part. */
      z' ← [u_0; u_1, ..., u_{n-1}, u_n − k_1] ; /* k_1-th father of z_n */
      m ← (k_1 < u_n)?1 : 0 ; /* One or zero included pattern. */
    else
      k_2 ← ⌊|B − X_1|/L'⌋ ; /* B strictly inside E(z_n). */
      z' ← z_{n-1}; /* Previous convergent of z_n. */
      m ← max(k_2 − k_1, 0) ; /* 0 or k_2 − k_1 incl. patterns z_{n-1}. */
    X'_1 ← X_1 + k_1(q_{n-1}, p_{n-1}) ; /* New position of leaning point. */
    X'_2 ← X'_1 + m(q', p') ; /* Where z' = p'/q'. */
  else
    /* Case E(z_n) = E(z_{n-2})E(z_{n-1})^{u_n}. */
    k_1 ← ⌈(L − |B − X_1|)/L'⌉ ; /* Position of B wrt the right. */
    if A = X_1 then
      /* A at left extremity of E(z_{n-2}) part. */
      z' ← [u_0; u_1, ..., u_{n-1}, u_n − k_1] ; /* k_1-th father of z_n */
      m ← (k_1 < u_n)?1 : 0 ; /* One or zero included pattern. */
    else
      k_2 ← ⌊(L − |A − X_1|)/L'⌋ ; /* A strictly inside E(z_n). */
      z' ← z_{n-1}; /* Previous convergent of z_n. */
      m ← max(k_2 − k_1, 0) ; /* 0 or k_2 − k_1 incl. patterns z_{n-1}. */
    X'_2 ← X_2 − k_1(q_{n-1}, p_{n-1}) ; /* New position of leaning point. */
    X'_1 ← X'_2 − m(q', p') ; /* Where z' = p'/q'. */
  return (z', m, X'_1, X'_2);
end
```

Algorithm 7: **Input**: If *rev* is false, a pattern $E(z_n) = [X_1, X_2]$ with its two upper leaning points X_1 and X_2, or if *rev* is true, a reversed pattern $\hat{E}(z_n) = [X_1, X_2]$ with its two lower leaning points X_1 and X_2. A and B are two points of this segment such that $X_1 \le A < B \le X_2$. **Output**: the greatest (reversed if *rev* is true) subpattern $E(z')^m = [X'_1, X'_2]$ included in $[A, B]$.

In the condition (if at line 1), the algorithm examines two cases, either the slope of z_n is *ULU* and odd depth or *LUL* (*rev* is true) and even depth. Otherwise (line 2), the slope of z_n is *ULU* and even depth or *LUL* (*rev* is true) and odd depth.

In the following, we have tested one case where $[A, B]$ is included in one pattern z_n. We assume that z_n is *ULU* and has an odd depth. In this case, we calculate the position of A from the left ($X'_1 = X_1 + k_1(q_{n-1}, p_{n-1})$), such that k_1 is the largest integer smallest or equal to $|A - X_1|/L'$. $z_{n-1} = \frac{p_{n-1}}{q_{n-1}}$ is the previous convergent of z_n. We thus test the position of B, if $|B - X_1| > u_n L'$, then B in the $E(z_{n-2})$ part and $m = 1$ (it means that $[A, B]$ is included in one pattern $E(z') = [u_0; u_1, \ldots, u_{n-1}, u_n - k_1]$). Otherwise, the point B in some $E(z_{n-1})$ and $m = k_2 - k_1$, then $[A, B]$ is included in $(k_2 - k_1)$—covering patterns $z' = z_{n-1}$ (previous covergent of z_n) and $X'_2 = X'_1 + m(q', p')$ ($z' = \frac{p'}{q'}$). Finally, the function **SmallestCovSp** returns the new pattern z' repeated m times and its new leaning points (X'_1, X'_2). We pursue the same reasoning for the other cases.

*A.2. Overview of **GreatestIncSp** algorithm*

The function **GreatestIncSp** (Algorithm 7) computes the greatest (reversed if *rev* is true) subpatterns $E(z')^m = [X'_1, X'_2]$ included in $[A, B]$. In this section, we focus to explain the red parts. For example, In the case where $[A, B]$ is included in one

odd pattern *ULU* z_n. We then calculate the position of A from the left ($X'_1 = X_1 + k_1(q_{n-1}, p_{n-1})$), such that k_1 is the smallest integer greater or equal to $|A - X_1|/L'$. If $B = X_2$, then B at the right extremity of $E(z_{n-2})$ part and $m = (k_1 < u_n ? 1 : 0)$ (it means that the greatest subpattern included in $[A, B]$ is $E(z') = [u_0; u_1, \ldots, u_{n-1}, u_n - k_1]$) repeated zero or one times. Otherwise, the point B strictly inside $E(z_n)$ and $m = \max(k_2 - k_1, 0)$ (k_2 is the largest integer smaller or equal to $|B - X_1|/L'$), then the greatest subpattern included in $[A, B]$ is $E(z') = E(z_{n-1})$ repeated zero or $(k_2 - k_1)-$ times. We pursue also the same reasoning for the other cases.

References

[1] T.A. Anderson, C.E. Kim, Representation of digital line segments and their preimages, Computer Vision, Graphics, and Image Processing 30 (3) (1985) 279–288.
[2] I. Debled-Rennesson, Etude et reconnaissance des droites et plans discrets, Ph.D. Thesis, Université Louis Pasteur, Strasbourg, 1995.
[3] I. Debled-Rennesson, J.-P. Reveillès, A linear algorithm for segmentation of discrete curves, International Journal of Pattern Recognition and Artificial Intelligence 9 (1995) 635–662.
[4] F. de Vieilleville, J.-O. Lachaud, Revisiting digital straight segment recognition, in: A. Kuba, K. Palágyi, L.G. Nyúl (Eds.), Proc. Int. Conf. Discrete Geometry for Computer Imagery, DGCI'2006, in: LNCS, vol. 4245, Springer, Szeged, Hungary, 2006, pp. 355–366.
[5] DGtal: digital geometry tools and algorithms library. http://liris.cnrs.fr/dgtal.
[6] L. Dorst, A.W.M. Smeulders, Discrete representation of straight lines, IEEE Transactions on Pattern Analysis and Machine Intelligence 6 (1984) 450–463.
[7] L. Dorst, A.W.M. Smeulders, Discrete Straight Line Segments: Parameters, Primitives and Properties, in: Vision Geometry, Series Contemporary Mathematics, American Mathematical Society, 1991, pp. 45–62.
[8] O. Figueiredo, Advances in discrete geometry applied to the extraction of planes and surfaces from 3D volumes, Ph.D. Thesis, EPFL, Lausanne, 1994.
[9] H. Freeman, On the encoding of arbitrary geometric configurations, IRE Transactions on Electronic Computers 10 (2) (1961) 260–268.
[10] H. Freeman, Computer processing of line-drawing images, ACM Computing Surveys 6 (1) (1974) 57–97.
[11] G.H. Hardy, E.M. Wright, An Introduction to the Theory of Numbers, Oxford Society, 1989.
[12] R. Klette, A. Rosenfeld, Digital Geometry—Geometric Methods for Digital Picture Analysis, Morgan Kaufmann, San Francisco, 2004.
[13] R. Klette, A. Rosenfeld, Digital straightness–a review, Discrete Applied Mathematics 139 (1–3) (2004) 197–230. The 2001 International Workshop on Combinatorial Image Analysis.
[14] V.A. Kovalevsky, New definition and fast recognition of digital straight segments and arcs. in: International Conference on Pattern Analysis and Machine Intelligence, 1990, pp. 31–34.
[15] V.A. Kovalevsky, S. Fuchs, Theoretical and experimental analysis of the accuracy of perimeter estimates, in: Forster, Ruwiedel (Eds.), Robust Computer Vision, 1992, pp. 218–242.
[16] J.-P. Reveillès, Géométrie discrète, calcul en nombres entiers et algorithmique, Thèse d'etat, Université Louis Pasteur, Strasbourg, France, 1991 (in French).
[17] M. Said, J.-O. Lachaud, Computing the characteristics of a subsegment of a digital straight line in logarithmic time, in: Proc. International Conference on Discrete Geometry for Computer Imagery, DGCI, in: LNCS, vol. 6607, Springer, Nancy, France, 2011, pp. 320–332.
[18] M. Said, J.-O. Lachaud, F. Feschet, Multiscale discrete geometry, in: Proc. International Conference on Discrete Geometry for Computer Imagery, DGCI, in: LNCS, vol. 5810, Springer, Montréal, Québec Canada, 2009, pp. 118–131.
[19] I. Sivignon, F. Dupont, J.-M. Chassery, Digital intersections: minimal carrier, connectivity, and periodicity properties, Graphical Models 66 (4) (2004) 226–244.
[20] A. Troesch, Interprétation géométrique de l'algorithme d'euclide et reconnaissance de segments, Theoretical Computer Science 115 (2) (1993) 291–319.
[21] J. Yaacoub, Enveloppes convexes de réseaux et applications au traitement d'images, Ph.D. Thesis, Université Louis Pasteur, Strasbourg, 1997.