Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 351 (2006) 14–26

Theoretical Computer Science

www.elsevier.com/locate/tcs

Stability versus speed in a computable algebraic model

Martin Ziegler[1]

Heinz Nixdorf Institute, University of Paderborn, 33095, Germany

Abstract

Algebraic models of real computation and their induced notions of time complexity neglect stability issues of numerical algorithms. Recursive Analysis on the other hand appropriately describes stable numerical computations while, based on Turing Machines, usually lacks significant lower complexity bounds.

We propose a synthesis of the two models, namely a restriction of algebraic algorithms to computable primitives. These are thus inherently stable *and* allow for nontrivial complexity considerations. In this model, one can prove on a sound mathematical foundation the empirically well-known observation that stability and speed may be contradictory goals in algorithm design.

More precisely we show that solving the geometric point location problem among hyperplanes by means of a *total* computable decision tree (i.e., one behaving numerically stable for all possible input points) has in general complexity exponentially larger than when permitting the tree to be partial, that is, to diverge (behave in an instable way) on a 'small' set of arguments.

The trade-off between the extremes is investigated quantitatively for the planar case. Proofs involve both topological and combinatorial arguments.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Speed; Stability; Point location; Models of computation; Recursive analysis; Linear decision tree

1. Motivation

Stability is a crucial issue in numerical programming. In fact, straightforward implementations of algorithms often lead to numerical difficulties as they are usually devised for idealized computers capable of exact real arithmetic. Adapting such an algorithm to stably execute on actual systems can turn out to require painstaking re-design and in the end result in considerable slow-down in running times. In practice, Filtering [13] can often help to use fast hardware-`floats` whenever possible and automatically switch to more precise but expensive calculations (`double`, exact rational arithmetic) only when necessary, hoping the latter case to occur rarely. However, concerning worst-case complexity, the present work will prove in a sound model of computation that a significant decrease in performance is generally unavoidable in order to achieve stability.

Formally, we restrict usual algebraic models to primitives that are *computable* in the sense of Recursive Analysis (cf. Section 2.2). Such a synthesis is natural since the latter framework ensures stability while the algebraic one allows for a rich variety of techniques yielding lower complexity bounds [6]. More precisely consider the Point Location problem (cf. Section 2.4). Well-known in Computational Geometry, it consists in determining the position of some input point $\vec{x}$ w.r.t. a fixed arrangement of hyperplanes $\{H_1, \ldots, H_n\}$ in Euclidean space. This problem is usually solved with

E-mail address: ziegler@upb.de.

[1] Supported by DFG project Zi1009/1-1.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.09.053

Linear Decision Tree algorithms (cf. Section 2.1), that is, by repeatedly evaluating some affine linear function f at $\vec{x}$ and proceeding adaptively according to the sign of the result: $+$, 0, $-$. For arbitrary arrangements of n hyperplanes in fixed dimension d, such trees of depth as shallow as $\mathcal{O}(\log n)$ are known.

However, they are in generally not totally computable. First, the linear function f might involve uncomputable coefficients; and second, the middle branch $f(\vec{x}) \overset{?}{=} 0$ of the primitive ternary test is numerically instable and, in the sense of Recursive Analysis, undecidable.

On the other hand, inputs $\vec{x}$ lying *on* one of the hyperplanes of the arrangement are always points of discontinuity and therefore give rise to numerical instabilities *inherent* to the problem in the sense of YAP [8, Section 2]. For Linear Decision Trees compliant with Recursive Analysis where computability requires stability, the maximum domain is thus $\mathbb{R}^d \setminus \bigcup_{j=1}^{n} H_j$.

Yet, some algorithms might have domains even smaller, that is, diverge at further arguments in addition to those above. YAP calls the latter instabilities *induced.*

The first result of the present work now reveals that relaxing totality in this sense in general does pay off in terms of complexity: for certain arrangements, the feasible depth of a computable Linear Decision Tree 'as total as possible' is exponentially larger than of one whose domain lacks a further set of measure zero. We then investigate quantitatively the trade-off between complexity and totality.

For the convenience of readers unfamiliar with either Recursive Analysis or Computational Geometry, Section 2 formally introduces the relevant models of computation as well as the problem under consideration.

2. Introduction

Given two algorithms for some problem, there are many criteria upon which to prefer one over the other, speed, size, and stability being the major ones. The first two are put onto a quantitative basis for comparing algorithms by considering their worst-case running times and memory consumption, respectively; see for example [10].

Stability on the other hand is a property and thus, in the original sense, either present or not. However a *quantitative* measure is desirable also for stability in order to compare algorithms, calling one *more* stable than some other. One means to this end are condition numbers introduced in [22]. Other works measure (in-)stability of an algebraic algorithm by counting the real tests (such as "`f(x)==0`") it performs [16]. The present paper proposes a further way of quantifying stability: (lack of) totality of a recursive real function.

Indeed in the model of computation used in Recursive Analysis, any correct algorithm is by definition inherently stable on its domain anyway. But given two stable algorithms (say, A_1 and A_2) for the same certain problem, there is still a way of comparing them: If $\mathrm{dom}(A_1) \subsetneq \mathrm{dom}(A_2)$, then one may justly call A_2 '*more*' stable than A_1.

A first qualitative result in Section 3 reveals that *requiring* totality may infer an exponential increase in time complexity compared to when divergence is allowed on a 'small' subset of inputs. More precisely, we consider the Point Location problem among n distinct computable hyperplanes $H_1, \ldots, H_n$ in $\mathbb{R}^d$ and show that, for each n,

- a computable Linear Decision Tree can always solve this problem within depth $\mathcal{O}(\log n)$ but may diverge for arguments $\vec{x} \in \bigcup_{j=1}^{m} H'_j$ from a finite union of m *additional* hyperplanes;
- there exists an arrangement of n hyperplanes for which *any* computable Linear Decision Tree, which does not diverge for inputs other than $x \in \bigcup H_i$, has necessarily depth $\Omega(n)$.

Section 4 then investigates quantitatively the trade-off between these two extremes in terms of the 'size' of (formally: the number of hyperplanes covering) this set of diverging inputs. It turns out that in 2D,

- $m = \mathcal{O}(n^2)$ additional hyperplanes of divergence suffice to guarantee logarithmic depth;
- $m = \mathcal{O}(n)$ additional hyperplanes can always achieve depth $\mathcal{O}(\sqrt[3]{n} \cdot \log n)$;
- m additional hyperplanes in general require depth $\Omega(\frac{n}{m \cdot \alpha(m)})$ where $\alpha(m)$ denotes the slowly increasing Inverse Ackermann Function.

The proofs use two tools which may be of interest on their own:

- To any planar arrangement of n distinct lines, one can add linearly many further lines such that in the thus enhanced resulting arrangement, any cell has at most $\mathcal{O}(\sqrt[3]{n})$ walls. (Proposition 4.3)
- In any finite arrangement of distinct hyperplanes, there is one having roughly as many cells to its left as to its right.

The imbalance can be bounded in terms of the maximum number of walls, taken over all cells of the arrangement. (Proposition 4.5)

We now recall models of computation and the problem under consideration.

2.1. Linear decision tree

Syntactically, a Linear Decision Tree (LDT) for dimension d is a finite rooted binary tree T such that

- each leaf v is labeled arbitrarily
- to each internal node u is assigned some affine linear functional

$$f_u : \mathbb{R}^d \to \mathbb{R}, \quad \vec{x} \mapsto a_0 + \sum_{i=1}^{d} x_i a_i. \tag{1}$$

The semantics goes as follows: upon input of $x \in \mathbb{R}^d$, recursively proceed to either the left or to the right subtree of T depending on whether, for the linear functional associated with T's root w, it holds $f_w(\vec{x}) < 0$ or $f_w(\vec{x}) \geqslant 0$. If w is a leaf, then report its label and stop.

This model, although rather simple, turns out to encompass a lot of known algorithms. In Computational Geometry for instance, one can easily test by means of an LDT the position of a point $\vec{x} \in \mathbb{R}^d$ relative to an oriented hyperplane $H^+ = \{y : f(y) > 0\}$; Section 2.4 will later consider that application in more detail.

Another example, `quicksort` is based on repeatedly comparing certain input elements x_k with x_l and proceeding according to whether the linear functional $f(\vec{x}) := x_k - x_l$ is <0 or $\geqslant 0$. The leafs of an according T_{qsort} are then labeled with order types, i.e., a permutation $\pi \in \mathcal{S}_d$ such that $x_{\pi(1)} \leqslant \cdots \leqslant x_{\pi(d)}$. This in particular shows that any LDT for sorting n reals has necessarily at least $n!$ many leafs and thus depth $\Omega(n \cdot \log n)$.

In fact, LDTs have proven useful not only in devising algorithms (i.e., upper complexity bounds) but also for proving lower bounds: any LDT for a problem in $\mathbb{R}^d$ with b_0 many connected components must have depth at least $\Omega(\log b_0)$. Pairwise `distinctness` of n given reals is an example where this method yields the exact asymptotic complexity $\Theta(n \cdot \log n)$ in the LDT-model; whereas a realRAM lacks such lower bounds. Even more astonishingly, the real `Knapsack` problem can be lower-bounded by $\Omega(n^2)$ while upper bounds, that is, LDT-algorithms for fixed n, run as fast as $\mathcal{O}(n^5 \cdot \log^2 n)$; cf. Sections 11.2 and 11.3 in [6].

Observe that in LDTs, evaluation of (1) is counted as *one* step while on actual digital computers it may take up to $\mathcal{O}(d)$ operations if all a_i are non-zero. However, this possible discrepancy between model and reality will not transcend polynomial-time complexity. More serious is its crossing of the *computability* boundary; in fact from the viewpoint of Recursive Analysis, tests on real numbers like '$f(\vec{x}) \overset{?}{=} 0$' are provably undecidable. Section 2.3 proposes a more realistic variant of the LDT-model.

2.2. Recursive analysis

Historically, computability questions for real numbers were treated before and even only initiated what is nowadays called 'classical' recursion theory, i.e., discrete computability. In fact, the title of TURING's seminal work [21] refers to *real* numbers. According to his definition, $x \in \mathbb{R}$ is computable iff some Turing Machine without input is capable of generating (the binary encodings of respective nominators and denominators of) a sequence of rationals

$$(q_1, \varepsilon_1, q_2, \varepsilon_2, \ldots, q_n, \varepsilon_n, \ldots) \quad \text{s.t.} \quad |q_n - x| \leqslant \varepsilon_n \to 0 \text{ as } n \to \infty. \tag{2}$$

Extending TURING's work, [15] and many others call a real function $f : \mathbb{R} \to \mathbb{R}$ computable iff some Turing Machine is able to, upon input of a sequence of rationals (q_n, ε_n) for $x \in \mathbb{R}$ as in (2), output a corresponding sequence (p_n, δ_n) for $y = f(x)$. Observe how this formalizes Interval Arithmetic as a way of operating on approximations of real numbers. Here, inaccuracies are taken into account not only for intermediate computations but also to the input itself; this contrasts a quite common (yet doubtful) presumption in Numerical Analysis. Consequently, stability is inherent to that model: any computable function is necessarily continuous and thus, small perturbations of the argument may not lead to unbounded changes on the result; cf., e.g., [5, p. 23] and [24, Theorem 4.3.1]. It is therefore generally agreed that this theoretical notion [2] captures quite well the principal capabilities of actual digital computers in connection with real numbers. For instance using this model it was proven formally that diagonalization of symmetric matrices is in general uncomputable (just as numerical experience suggests) while it becomes computable when restricted to matrices having no multiple eigenvalues [27].

Lacking continuity, characteristic functions

$$\mathbf{1}_{\{0\}}, \quad \mathbf{1}_{(-\infty,0]}, \quad \mathbf{1}_{(-\infty,0)}, \quad \mathbf{1}_{[0,\infty)}, \quad \mathbf{1}_{(0,\infty)}$$

are *not* computable and hence tests like

$$=0, \quad \leqslant 0, \quad <0, \quad \geqslant 0, \quad >0, \quad \neq 0$$

are undecidable. On the other hand, "<0" and ">0" both turn out to be decidable when restricted to non-zero arguments x while diverging for $x = 0$. We shall exploit that in Section 2.3 to design a *computable* LDT with only such partial predicates.

So continuity is necessary for computability but not sufficient: as there are uncountably many reals $a \in \mathbb{R}$ but only countably many Turing Machines, even most constant functions $f(x) \equiv a$ follow to be uncomputable. In fact to every (classically) non-recursive subset $A \subseteq \mathbb{N}$, there corresponds a non-computable real $a = \sum_{n\in A} 2^{-n}$. Usual 'practical' continuous functions however turn out to be computable, such as

$$+, \; -, \; \cdot, \; /, \quad |\cdot|, \quad \sqrt{\cdot}, \quad \sin, \cos, \exp, \ln, \ldots$$

Further literature on computable real functions is vast [19] and also investigates, although more scarcely, their complexity [17,24, Section 7]. However being based on Turing Machines, lower bounds for concrete problems are notoriously difficult whereas in algebraic models like the LDT, a rich variety of methods provide the latter [6].

The following subsection thus introduces a synthesis of LDT and Computable Analysis with the goal to gather the advantages of both into one single model. In fact, Brattka and Hertling [3] already did similarly for the realRAM and Computable Analysis. However for complexity considerations in Computational Geometry, LDTs are more appropriate; the synthesis we propose is thus the

2.3. Computable LDT

Consider the primitive of a LDT: testing, for input $\vec{x} \in \mathbb{R}^d$ and according to (1), whether $f_u(\vec{x}) < 0$ or $f_u(\vec{x}) \geqslant 0$. In order to make this predicate computable, we shall impose two natural conditions:

(i) All coefficients a_i appearing in functional f_u must be computable reals in the sense of Recursive Analysis.
(ii) For $f_u(\vec{x}) = 0$, the test diverges.

Property (ii) reflects that tests "$x < 0$" and "$x > 0$" become decidable only when restricting to non-zero arguments. Property (i) is obviously equivalent to $f_u : \mathbb{R}^d \to \mathbb{R}$ being a computable function. Here, a *vector* $\vec{x} = (x_1, \ldots, x_d)$ gets fed into the Turing Machine by means of an interleaved sequence of rationals

$$q_{1,1}, \varepsilon_{1,1}, \; q_{1,2}, \varepsilon_{1,2}, \; \ldots, \; q_{1,d}, \varepsilon_{1,d}, \quad q_{2,1}, \varepsilon_{2,1}, \; q_{2,2}, \varepsilon_{2,2}, \; \ldots, \; q_{2,d}, \varepsilon_{2,d}, \quad q_{3,1}, \; \cdots$$
$$\cdots\cdots, \; \varepsilon_{n-1,d}, \quad q_{n,1}, \varepsilon_{n,1}, \; q_{n,2}, \varepsilon_{n,2}, \; \ldots, \; q_{n,d}, \varepsilon_{n,d}, \quad q_{n+1,1}, \; \cdots\cdots\cdots\cdots,$$

where $|q_{n,i} - x_i| \leqslant \varepsilon_{n,i} \to 0$ for each $i = 1, \ldots, d$.

Definition 2.1. A computable LDT T is a LDT in which to each internal node u is assigned a linear functional f_u with only *computable* coefficients. Similarly, the leafs' labels have to be computable. [3] Furthermore, T's semantics gets slightly modified as follows:

For any input $\vec{x} \in \mathbb{R}^d$ which, during traversal of T, encounters an internal node u with $f_u(\vec{x}) = 0$, this test diverges and we thus set $T(\vec{x}) := \perp$.

In contrast to general LDTs, the domain $\{\vec{x} : T(\vec{x}) \neq \perp\}$ of a computable one is thus a *proper* subset of $\mathbb{R}^d$. Also observe that any *computable* LDT T induces a *computable* function F_T on $\mathrm{dom}(T)$, namely mapping $\vec{x}$ to the label of

[2] See [24, Section 4.1] for a survey on equivalent ones.
[3] Which is trivially the case for integers or finite strings.

the (unique) leaf of T which the above traversal semantics ends in. Borrowing from usual LDTs, we call T's depth its running time. For a fixed computable function F on $\mathrm{dom}(F) \subseteq \mathbb{R}^d$, the question about its complexity thus asks for the minimum depth of a computable LDT T with $F \subseteq F_T$. For the Point Location problem introduced now, this turns out to be of worst-case order $\Omega(n)$, n the number of hyperplanes involved.

2.4. Point location

The problem is well-known in Computational Geometry [2]: fix a family $\mathcal{H} = \{H_1, \ldots, H_n\}$ of n oriented hyperplanes in $\mathbb{R}^d$; given $\vec{x} \in \mathbb{R}^d \setminus \bigcup \mathcal{H}$, determine which component of $\mathbb{R}^d \setminus \bigcup \mathcal{H}$ $\vec{x}$ belongs to. Formally

Definition 2.2. For $j = 1, \ldots, n$, let

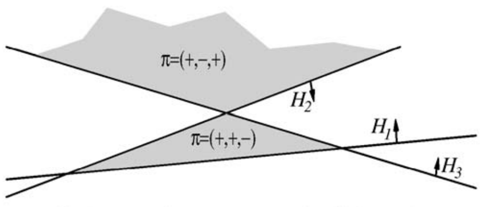
$$g_j : \mathbb{R}^d \to \mathbb{R}, \quad \vec{x} \mapsto a_{j0} + \sum_i x_i \cdot a_{ji}, \quad (a_{j1}, \ldots, a_{jd}) \neq 0$$

denote an affine linear function and $H_j = g_j^{-1}(0)$ its induced oriented hyperplane, $H_j^+ = g_j^{-1}(0, \infty)$ the positive halfspace and $H_j^- = g_j^{-1}(-\infty, 0)$ the negative one. For $\vec{x} \in \mathbb{R}^d$,

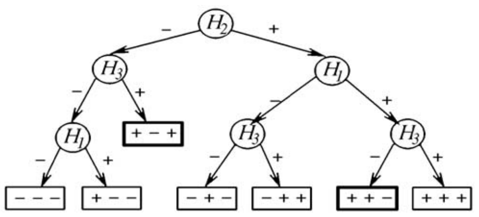
$$\pi(\vec{x}) = (\mathrm{sign}\, g_j(\vec{x}))_{j=1,\ldots,n} \in \{-1, 0, +1\}^n$$

is called its position vector. A cell of $\mathcal{H} = \{H_1, \ldots, H_n\}$ is a subset $\pi^{-1}(\sigma)$ of $\mathbb{R}^d$ for some $\sigma \in \{-1, +1\}^n$. Point Location is the problem of computing, to fixed $\mathcal{H}$ and upon input of some point $\vec{x}$ the cell that $\vec{x}$ lies in, i.e., the function

$$F : \mathbb{R}^d \setminus \bigcup \mathcal{H} \to \{-1, +1\}^n, \qquad \vec{x} \mapsto \pi(\vec{x}). \tag{3}$$



3 GENERIC LINES INDUCE 7 CELLS.



A LDT FOR THIS ARRANGEMENT.

Keep in mind that $\mathcal{H}$ is regarded as fixed and one asks for the mere existence of an algorithm solving Point Location for this specific arrangement; see also Remark 3.6. This corresponds to *non-uniform* lower and upper bounds prevalent in Algebraic Complexity Theory [6]. As common in Computational Geometry, issues related to the actual construction of an according data structure (pre-processing $\mathcal{H}$, so to speak) are treated separately.

The complexity of Point Location in the LDT-model is very well understood. For fixed d, a variety of techniques [11,7] yield according (algorithms giving rise to) trees of depth $\mathcal{O}(\log n)$ whose leafs are labeled with position vectors $\sigma \in \{-1, +1\}^n$. Conversely, since arrangements can induce up to

$$b_0 = \sum_{i=0}^{d} \binom{n}{i} = \Theta(n^d) \tag{4}$$

many cells (i.e., connected components) [14], depth $\Omega(\log n)$ is also necessary.

However when querying hyperplanes not belonging to the original arrangement $\mathcal{H}$, the equality tests in these algorithms when considered as computable LDTs will in general lead to a domain *strictly* contained in $\mathbb{R}^d \setminus \bigcup \mathcal{H}$. The next section reveals that and to what extend this is unavoidable.

3. Total versus partial point location

Before addressing the *complexity* of Point Location in the new model from Section 2.3, better make sure that it is at all *computable*:

Lemma 3.1. *For a hyperplane $H \subseteq \mathbb{R}^d$, the following are equivalent*:

(a) *H's Euclidean distance function $\mathbb{R}^d \ni \vec{x} \mapsto \min_{\vec{y}\in H} \|\vec{x}-\vec{y}\|_2$ is computable*;
(b) *$H = g^{-1}(0)$ for some $g(\vec{x}) = a_0 + \sum_i x_i a_i$ with all coefficients a_i, a_0 computable*;
(c) *the Point Location problem for $\mathcal{H} = \{H\}$ (i.e., with $n = 1$) is computable w.r.t. arbitrary orientation of H.*

Proof. See Corollaries 10, 13, and 15 in [4] plus Corollary 4.13 in [26]. □

So from now on consider only arrangements $\mathcal{H}$ of *computable* hyperplanes in the sense of either (a) or (b) from the above lemma. For these, one can easily upper bound the complexity:

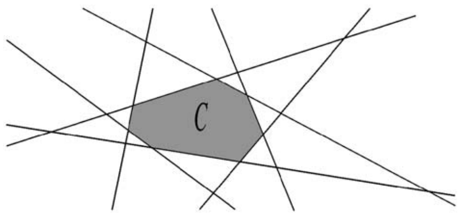
Observation 3.2. *For any arrangement $\mathcal{H}$ of n computable hyperplanes in $\mathbb{R}^d$, Point Location can be solved by a computable LDT of depth n.*

Proof. Upon input of $\vec{x}$, sequentially query for each $j = 1, \dots, n$ whether $\vec{x} \in H_j^+$ or $\vec{x} \in H_j^-$: this determines the entire position vector. It is easy to see that this ‘algorithm’ can be formulated as a computable LDT T. □

Observe that this T queries only hyperplanes $H\in\mathcal{H}$ and thus attains domain $\mathbb{R}^d\setminus\bigcup\mathcal{H}=\mathrm{dom}(F)$ according to Eq. (3). Since F ‘jumps’ at any $\vec{x} \in \bigcup\mathcal{H}$, there is also no way of further enlarging $\mathrm{dom}(T)$ without loosing computability. It turns out that for computable LDTs which are ‘maximally total’ in this sense, depth n is in general best possible:

Theorem 3.3. *There exist arrangements $\mathcal{H}$ of n computable hyperplanes such that any computable LDT with* $\mathrm{dom}(T) = \mathbb{R}^d\setminus\bigcup\mathcal{H}$ *solving the corresponding Point Location problem has depth at least n.*

Proof. Imagine a computable arrangement of n hyperplanes all touching one common cell C; these hyperplanes may even be chosen to have rational coefficients and in particular be computable. Then a LDT for this arrangement will have depth at least n. □



More generally, an argument due to E. Ukkonen from a different context [23] implies here that the number of walls of any cell in the arrangement lower bounds the depth of a ‘total’ computable LDT:

Proposition 3.4. *For a finite arrangement $\mathcal{H}$ of n hyperplanes in $\mathbb{R}^d$ and a cell C of $\mathcal{H}$, let $w(C)$ denote its number of ‘walls’, i.e.,*

$$w(C) = \#\{H \in \mathcal{H} \mid H \cap \overline{C} \neq \emptyset,\ \dim(H \cap \overline{C}) = d-1\}$$

and $w(\mathcal{H}) = \max_C w(C)$, the maximum taken over all cells C of $\mathcal{H}$. Then any LDT for Point Location among the hyperplane arrangement induced by $\mathcal{H}$ querying no hyperplanes other than those from $\mathcal{H}$ has depth at least $w(\mathcal{H})$.

Proof. See Section 5.1. □

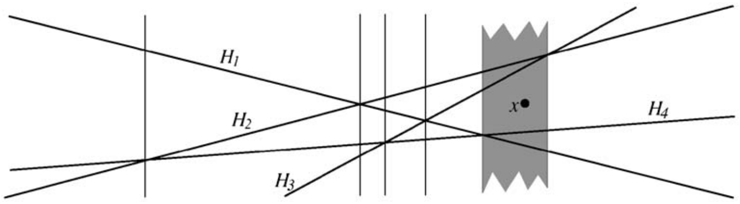
On the other hand a slight relaxation of the requirement “$\mathrm{dom}(T) = \mathbb{R}^d\setminus\bigcup\mathcal{H}$” does permit a computable LDT of depth $\mathcal{O}(\log n)$ for *any* computable arrangement of n hyperplanes:

Lemma 3.5. *To any arrangement $\mathcal{H}$ of n computable hyperplanes in $\mathbb{R}^d$, there exists another finite arrangement $\mathcal{H}'$ of computable hyperplanes and a computable LDT of depth $\mathcal{O}(\log n)$ with* $\operatorname{dom}(T) = (\mathbb{R}^d \setminus \bigcup \mathcal{H}) \setminus \bigcup \mathcal{H}'$ *for Point Location among $\mathcal{H}$.*

Observe that this domain is still dense in $\mathbb{R}^d$.

Proof. Many algorithms in Computational Geometry attain logarithmic time for Point Location by querying a finite number of additional hyperplanes. For instance, the following rather simple construction of Dobkin and Lipton [11] immediately leads to a computable LDT. [4]

In 2D, draw a vertical line through each vertex (=crossing point of two lines).



This will add at most $m \leqslant n \cdot (n-1)/2$ new lines H'. Within the enhanced arrangement $\mathcal{H} \cup \mathcal{H}'$, proceed as follows:

- use binary search to locate the vertical stripe that input $\vec{x}$ lies in;
- within a stripe, again use binary search to locate the cell that $\vec{x}$ lies in.

The latter is possible since, within each stripe, lines do not intersect and its cells thus can be ordered totally. Knowing the (sub-)cell of $\mathcal{H} \cup \mathcal{H}'$ that $\vec{x}$ lies in obviously includes knowledge about the cell of $\mathcal{H}$ containing $\vec{x}$.

It is easy to confirm that both steps can be realized by LDTs; in fact, by *computable* LDTs since each vertical line through the intersection point of two computable lines is again a computable one. First step takes depth $\mathcal{O}(\log m) = \mathcal{O}(\log n)$, second adds another $\mathcal{O}(\log n)$.

For higher dimensions, we proceed inductively similar to [11]. Let $\mathcal{H}$ for instance denote a 3D arrangement. Then consider, for each pair (H_i, H_j) of distinct planes from $\mathcal{H}$, their intersection $H_i \cap H_j$. This is either empty (trivial case) or a line in space. Let $\tilde{\mathcal{H}}$ denote the arrangement obtained by projecting those lines onto the plane $z = 0 : \mathcal{H}$ is a computable 2D arrangement of size at most $\tilde{n} = \mathcal{O}(n^2)$. By induction hypothesis, it allows Point Location by a computable LDT within depth $\mathcal{O}(\log \tilde{n}) = \mathcal{O}(\log n)$. Let $\tilde{C}$ denote the 2D cell of $\tilde{\mathcal{H}}$ that the (accordingly projected) input $\tilde{x}$ lies in; thus $\vec{x} \in \tilde{C} \times \mathbb{R}$, and it remains to determine which of the cells C towering over $\tilde{C}$ contains $\vec{x}$. As, again, the planes $\mathcal{H}$ do not intersect within $\tilde{C} \times \mathbb{R}$, this is possible by binary search within time $\mathcal{O}(\log n)$. □

Remark 3.6 (*Degenerate arrangements*). Constructions like the one used in the above proof are in Computational Geometry usually preceded by some sort of *non-degeneracy* presumption. In 2D this might require for instance that no three lines intersect in a common point or that all $n \cdot (n-1)/2$ intersection points have distinct x-coordinates.

On the other hand such cases can be dealt with by considering, out of several coinciding verticals, only one instance or by slightly rotating the entire setting, respectively. One can easily check that this does yield a LDT with the claimed properties.

It should be emphasized that Lemma 3.5 as well as the other upper complexity bounds in the present work claim just the *existence* of a computable LDT T for an arbitrary but *fixed* $\mathcal{H}$. Accordingly, *effectivity* of mappings $\mathcal{H} \mapsto T$ constructed in our proofs is neither claimed nor true with respect to *varying*, possibly degenerate arrangements regarded as input given by approximations to their coefficients in the sense of Recursive Analysis.

Anyway, the lower bounds presented in Theorems 3.3 and 4.6 do not rely on degenerate arrangements.

[4] Also note that the more sophisticated algorithm presented in [7] is no Linear Decision *Tree*. Indeed already a LDT that merely *decides* whether input $\vec{x}$ belongs to $\bigcup \mathcal{H}$ must have size at least $\Omega(n^d)$—compare Eq. (4) and Observation 5.1(b)—as opposed to Theorem 4.2 in [7].

4. Trading totality for speed

Recall that in order to achieve the exponential speedup of the LDT T in Lemma 3.5 compared to Theorem 3.3, it sufficed to reduce T's domain by a *finite* union of hyperplanes $\bigcup \mathcal{H}'$, i.e., a nowhere dense set of measure zero. In the present section, we quantify in dimension $d = 2$ the trade-off between both extremes by counting *how many* hyperplanes (= lines, here) are to be removed from $\operatorname{dom}(T)$ in dependence of the aimed running time.

Corollary 4.1. *To any arrangement $\mathcal{H}$ of n computable lines in $\mathbb{R}^2$, there exists another arrangement $\mathcal{H}'$ of $m = |\mathcal{H}'| \leqslant n \cdot (n-1)/2 = \mathcal{O}(n^2)$ computable lines and a computable LDT of depth $\mathcal{O}(\log n)$ with* $\operatorname{dom}(T) = (\mathbb{R}^2 \setminus \bigcup \mathcal{H}) \setminus \bigcup \mathcal{H}'$ *for Point Location among $\mathcal{H}$.*

Proof. See the proof of Lemma 3.5. □

When reducing $\operatorname{dom}(T)$ by querying only linearly rather than quadratically many further lines, significantly sublinear running time can still be attained:

Theorem 4.2. *To any arrangement $\mathcal{H}$ of n computable lines in $\mathbb{R}^2$, there exists*[5] *another arrangement $\mathcal{H}'$ of $m = |\mathcal{H}'| = \mathcal{O}(n)$ computable lines and a computable LDT T of depth $\mathcal{O}(\sqrt[3]{n} \cdot \log n)$ with* $\operatorname{dom}(T) = (\mathbb{R}^2 \setminus \bigcup \mathcal{H}) \setminus \bigcup \mathcal{H}'$ *for Point Location among $\mathcal{H}$.*

Proof. Apply Proposition 4.5 for $d = 2$ to $\widetilde{\mathcal{H}} := \mathcal{H} \cup \mathcal{H}'$ from Proposition 4.3. □

Proposition 4.3. *Let $\mathcal{H}$ denote an arrangement of n computable lines in the plane. There exist $m = \mathcal{O}(n)$ computable lines $\mathcal{H}'$ such that each cell of $\widetilde{\mathcal{H}} := \mathcal{H} \cup \mathcal{H}'$ has at most $\mathcal{O}(\sqrt[3]{n})$ walls.*

Proof. See Section 5.2. □

Remark 4.4. We wonder whether the $\mathcal{O}(\sqrt[3]{n})$ cell size obtained in Proposition 4.3 by a linear number of additional lines is optimal or not.

This tool reveals the lower bound in Proposition 3.4 to be close to optimal:

Proposition 4.5. *To any arrangement $\widetilde{\mathcal{H}}$ of n computable hyperplanes in $\mathbb{R}^d$, each cell of which has at most w walls, there exists a computable total LDT T of depth $\mathcal{O}(w \cdot d \cdot \log n)$ for Point Location within $\widetilde{\mathcal{H}}$.*

Proof. See Section 5.3. □

The following result complements the upper bounds in Corollary 4.1 and Theorem 4.2 by lower bounding the worst-case running time in terms of the number of lines removed from $\operatorname{dom}(T)$.

Theorem 4.6. *There are 2D arrangements $\mathcal{H}$ of n computable lines such that, for any arrangement $\mathcal{H}'$ of m further lines, a computable LDT T for Point Location among $\mathcal{H}$ with* $\operatorname{dom}(T) \supseteq (\mathbb{R}^2 \setminus \bigcup \mathcal{H}) \setminus \bigcup \mathcal{H}'$ *has depth at least $\Omega(n/m \cdot \alpha(m))$.*

Here, α denotes the very, very slowly increasing Inverse Ackermann function; $\alpha(m) \leqslant 4$ for all practical values of m.

The proof of Theorem 4.6 requires some additional notation:

Definition 4.7. For any LDT T—computable or not—and some node u of T, let T_u denote the subset of inputs $\vec{x} \in \mathbb{R}^d$ which, according to T's semantics from Section 2.1, passes through u on its way from the root.

[5] Formally: there exist functions $f, g : \mathbb{N} \to \mathbb{N}$ with $f(n) = \mathcal{O}(n)$ and $g(n) = \mathcal{O}(\sqrt[3]{n} \cdot \log n)$ such that to any computable arrangement $\mathcal{H}$ there exists a computable arrangement $\mathcal{H}'$ of size $|\mathcal{H}'| \leqslant f(|\mathcal{H}|)$ and a computable LDT T of depth $\leqslant g(|\mathcal{H}|)$.

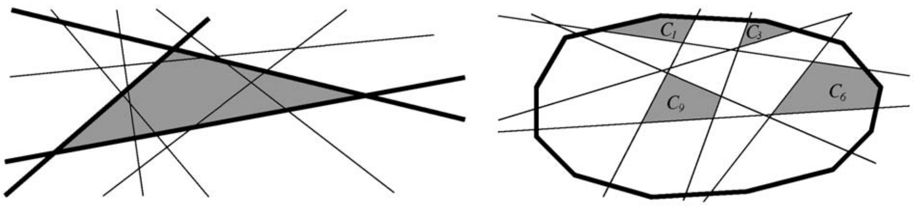


Fig. 1. Left: arrangement (thin + bold) refining a super-cell (gray); Right: Cell C (bold) has $n = 12$, each sub-cell at most $t = 5$ walls.

Thus $T_r = \mathbb{R}^d$ for the root r of T. If T is computable, then induction on the depth of u within T reveals that every T_u is an open convex polyhedron. Moreover for any two nodes $u \neq v$ in T, none being a descendant of the other, $T_u \cap T_v = \emptyset$.

Definition 4.8. Consider a finite collection $\mathcal{H}$ of hyperplanes in $\mathbb{R}^d$. Let $\mathcal{A}(\mathcal{H})$ denote the family of cells C it induces; $\mathcal{A}(\emptyset) := \{\mathbb{R}^d\}$.

An arrangement $\mathcal{A}'$ of pairwise disjoint convex open polyhedra $C' \subseteq \mathbb{R}^d$ partially refines $\mathcal{A}(\mathcal{H})$ if each $C' \in \mathcal{A}'$ is contained in some $C \in \mathcal{A}$. A super-cell of $\mathcal{A}(\mathcal{H})$ is a cell of $\mathcal{A}(\mathcal{H}')$ for some $\mathcal{H}' \subseteq \mathcal{H}$.

A LDT T solves $\mathcal{A}'$ if each non-empty open T_v is contained in some $C' \in \mathcal{A}'$. T is total for $\mathcal{A}'$ if $C' \in \mathcal{A}'$ is covered by appropriate T_v. T decides an open convex polyhedron C if it solves $\mathcal{A}' := \{C, \mathbb{R}^d \setminus C\}$.

Proof (*Theorem 4.6*). Like in the proof to Theorem 3.3, take a computable arrangement $\mathcal{H}$ of n lines, all forming walls of one common cell C. Consider a computable LDT deciding C and let $v_1, \ldots, v_n$ denote those of its leafs with non-empty open $T_{v_i} \subseteq C$. According to Proposition 3.4, each $C_i := T_{v_i}$ has at most t walls where t denotes the depth of T. As $\overline{\bigcup C_i} = C$, the C_i induce a partition of the n walls of C. In particular, at least n/t many C_i meet the boundary line $\gamma := \partial C$ of C; see Fig. 1.

Now suppose some arrangement $\mathcal{H}'$ of m lines covers $(\mathbb{R}^2 \setminus \bigcup \mathcal{H}) \setminus \mathrm{dom}(T)$. Then the arrangement $\mathcal{A}(\mathcal{H} \cup \mathcal{H}')$ refines the collection $\{T_v \neq \emptyset : v \text{ leaf of } T\}$. In particular, at least n/t many cells C' of $\mathcal{A}(\mathcal{H}')$ meet γ. The lemma below then implies $n/t = \mathcal{O}(m \cdot \alpha(m))$. □

Lemma 4.9. *Let γ denote a convex planar curve and $\mathcal{H}'$ an arrangement of m lines in the plane. Then, γ meets at most $\mathcal{O}(m \cdot \alpha(m))$ cells from $\mathcal{H}'$.*

Proof. See Remark 4 on p. 334 of [12]; alternatively, see Corollary 2.16 plus Theorem 5.11 in [20] and observe that any two distinct lines intersect in at most one point ($s = 1$) while convex γ meets any line at most twice ($t = 2$). □

5. Proofs

The present section collects the proofs to some technical propositions in Sections 3 and 4 which there would have interrupted the train of thoughts.

5.1. Proof of Proposition 3.4

Observation 5.1. *Fix some finite arrangement $\mathcal{H}$ of hyperplanes in $\mathbb{R}^d$.*

(a) *Any LDT T computing the position vector $\vec{x} \mapsto F(\vec{x})$* w.r.t. *$\mathcal{H}$ according to Equation* (3) *in Section* 2.4 *solves $\mathcal{A}(\mathcal{H})$ in the sense of Definition* 4.7. *T is total iff* $\mathrm{dom}(T) \supseteq \mathbb{R}^d \setminus \bigcup \mathcal{H}$.

(b) *Conversely, any LDT T solving $\mathcal{A}(\mathcal{H})$ can compute $\vec{x} \mapsto F(\vec{x})$ on* $\mathrm{dom}(T) \setminus \bigcap \mathcal{H}$ *by labeling each leaf v having non-empty open T_v with the* (*common*!) *position vector of all $\vec{x} \in T_w$.*

(c) *If $\mathcal{A}'$ partially refines $\mathcal{A}$ and LDT T solves $\mathcal{A}'$, then T also solves*[6] *$\mathcal{A}$. Furthermore a LDT that decides C for each $C \in \mathcal{A}$ also solves $\mathcal{A}$.*

(d) *Conversely for $C \in \mathcal{A}$, each LDT T solving $\mathcal{A}$ also decides C. If T is in addition computable and total, then $C = T_v$ for some unique leaf v of T.*

[6] We have already used that implicitly in the proof of Theorem 4.2.

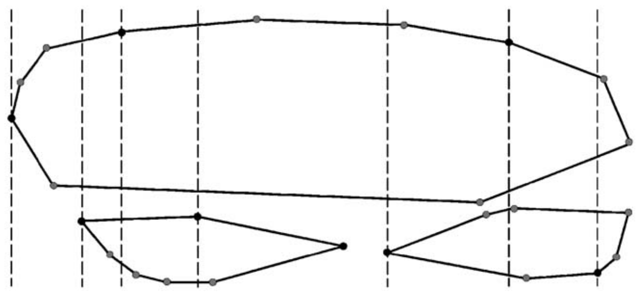


Fig. 2. Three 'large' cells, independently cut into smaller pieces by vertical lines through any 4th of their respective vertices: in the thus refined arrangement, each cell has at most 6 walls.

Indeed according to Definition 4.7, C' is a disjoint finite union of non-empty open sets T_v for certain leafs $v_1, \ldots, v_n$ of T. But if $n > 1$, then this union cannot be connected [25, Definition 26.1] contradicting C' being even convex.

Proposition 3.4 now follows from Claims (a + c) combined with the following

Lemma 5.2. *Let C be an open convex polyhedron with w walls. Then any computable total LDT T deciding C has depth at least w.*

Proof. Let $H_1, \ldots, H_w$ denote the hyperplanes that form the walls of $\overline{C}$. Consider some point $\vec{x} \in C$ and its way through T from the root to the leaf v with $T_v = C$. So any $\vec{x} \in C$ takes the same path though T. Now move $\vec{x}$ continuously towards one of the walls $\overline{C} \cap H_j$ of C. Immediately after $\vec{x}$ has crossed H_j, its position vector changes. Therefore, $\vec{x}$ must now end up in a different leaf $v' \neq v$ of T. The least common ancestor u of v and v' thus is a query of hyperplane H_j. Since this argument holds for any of C's walls $H_1, \ldots, H_w$, v has at least w different ancestors $u_1, \ldots, u_w$ and in particular depth $\geqslant w$ within T. □

5.2. Proof of Proposition 4.3

This section proves the already announced.

Proposition 4.3. *Let $\mathcal{H}$ denote an arrangement of n computable lines in the plane. There exist $m = \mathcal{O}(n)$ computable lines $\mathcal{H}'$ such that each cell of $\mathcal{H} \cup \mathcal{H}'$ has at most $\mathcal{O}(\sqrt[3]{n})$ walls.*

Proof. Let $w := \lceil \sqrt[3]{n} \rceil$. Consider some cell C of $\mathcal{H}$ having more than w walls, i.e., a 'large' one. Take its vertices ordered w.r.t. increasing x-coordinates; draw a vertical line H' through every wth vertex. These lines cut the large cells into smaller pieces, each having at most $w + 2$ vertices. Being in 2D this also means that each one has at most $w + 2$ walls. Cells that *used* to be 'small' (i.e., having at most w walls) might be intersected by these new lines; but here, too, the number of walls of any piece will not grow beyond $w + 2$.

We apply the above procedure not only to one but simultaneously to all cells that initially used to be 'large'; cf. Fig. 2. Let $\mathcal{H}'$ denote the collection of newly introduced vertical lines; then any cell in the arrangement $\mathcal{H} \cup \mathcal{H}'$ has at most $w + 2 \leqslant \mathcal{O}(\sqrt[3]{n})$ walls. As the vertices of an arrangement of computable lines are computable, $\mathcal{H}'$ is computable as well. Of course, the crucial question is for the size of $\mathcal{H}'$.

So let $\mathcal{A}'$ denote the collection of cells that used to be large and N their number. Each $C \in \mathcal{A}'$ having by definition more than $w = n^{1/3}$ walls, this induces a total of at least $\Omega(N \cdot n^{1/3})$ many walls. On the other hand, by virtue of Lemma 5.3 below, it can give rise to a total of at most $\mathcal{O}(N^{2/3} \cdot n^{2/3} + n)$ many walls. Therefore, $N = \mathcal{O}(n)$, and $\mathcal{A}'$ contains at most $\mathcal{O}(n^{4/3})$ many walls/vertices. Having introduced additional lines only through every wth of these vertices, we arrive at $m = |\mathcal{H}'| = \mathcal{O}(n)$. □

The following tool, applied in the above proof, is actually a deep result in combinatorial geometry taken from [9, Section 3].

```
Upon input of $\vec{x} \in \mathbb{R}^d$:
 a) Initialize $C$ to be the 'supercell' $\mathbb{R}^d$.
 b) Choose some new $H_i \in \mathcal{H}$;
    query the position of $\vec{x}$ with respect to $H_i$.
 c) If $\vec{x} \in H_i^+$, set $C := C \cap H_i^+$;
    similarly for $\vec{x} \in H_i^-$.   Diverge for $\vec{x} \in H_i$.
 d) Continue with b) until $C$ intersects only a
    constant number of hyperplanes from $\mathcal{H}$;
    let them be denoted by $\mathcal{H}'$.
 e) Locate $\vec{x}$ within $\mathcal{A}(\mathcal{H}')$ according to Observation 3.2
```

Fig. 3. Total computable Point Location within $\mathcal{A}(\mathcal{H})$.

Lemma 5.3. *For a family $\mathcal{A}(\mathcal{H})$ of cells induced in 2D by $n = |\mathcal{H}|$ lines, consider a sub-collection $\mathcal{A}' \subseteq \mathcal{A}(\mathcal{H})$ of size $N = |\mathcal{A}'|$. Then, the cells of $\mathcal{A}'$ have a total of at most $\mathcal{O}(N^{2/3} \cdot n^{2/3} + n)$ walls.*

It makes no difference whether, for two cells $c_1, c_2 \in \mathcal{A}'$ sharing a wall, this wall is counted once or twice: a factor of 2 is neglected by the $\mathcal{O}$-notation anyway. For similar reasons, the $\mathcal{O}(N^{2/3} \cdot n^{2/3} + n)$ bound also holds for the total number of *vertices* rather than walls in 2D.

5.3. Proof of Proposition 4.5

For reader's convenience, we repeat the claim:

Proposition 4.5. *To any arrangement $\mathcal{H}$ of n computable hyperplanes in $\mathbb{R}^d$, each cell of which has at most w walls, there exists a computable total LDT T of depth $\mathcal{O}(w \cdot d \cdot \log n)$ for Point Location within $\mathcal{H}$.*

Proof. The idea is to adopt Theorem 5 from [10] to our setting of computable LDTs. Consider the *meta*-algorithm in Fig. 3 for Point Location within $\mathcal{A}(\mathcal{H})$. This obviously terminates after at most $\mathcal{O}(n)$ steps and, for $\vec{x} \in \mathbb{R}^d \setminus \bigcup \mathcal{H}$, correctly locates $\vec{x}$ within $\mathcal{A}(\mathcal{H})$. Unrolling it with respect to all possible outcomes of these finitely many queries[7] yields a LDT T. Moreover, as $\mathcal{H}$ was presumed computable and only semi-decidable queries "$\vec{x} \in H_i^+$" or "$\vec{x} \in H_i^-$" are made, this T is a total computable LDT.

It remains to perform the choice in (b) adaptively in such a way that the loop is executed only $\mathcal{O}(w \cdot d \cdot \log n)$ times rather than $\mathcal{O}(n)$. Optimally, each query in (c) reduces the number of cells from $\mathcal{A}(\mathcal{H})$ that $\vec{x}$ might lie in by a factor of two. In other words: $H_i \in \mathcal{H}$ should be such as to have, for those cells from $\mathcal{A}(\mathcal{H})$ contained within C, about as many on its positive as on its negative side. Starting with a total of at most $\mathcal{O}(n^d)$ many cells according to Equation (4) this would lead to a running time $\mathcal{O}(d \cdot \log n)$.

However, neither this nor a strict fifty–fifty divide-and-conquer is in general feasible for large w. Instead, Lemma 5.4 below yields that H_i in (b) may always be chosen such as to at least reduce the number of cells from $\mathcal{A}(\mathcal{H})$ lying in C by a factor of roughly $1 - 1/w$. In other words, the running time can be bounded by $\mathcal{O}(\log_b(n^d)) = \mathcal{O}(d \cdot \log n / \log b)$, the logarithm taken to base $b = 1/(1 - 1/w)$ rather than base 2. As $1/\log(b) = \mathcal{O}(w)$, this proves the claim. □

Lemma 5.4. *Fix some collection $\mathcal{H}$ of n hyperplanes in $\mathbb{R}^d$ and a* super*cell C of $\mathcal{A}(\mathcal{H})$. Let $\mathcal{A}|_C$ denote the sub-arrangement of cells in $\mathcal{A}(\mathcal{H})$ lying within C, $N := \#\mathcal{A}|_C$ their number. Suppose that any such cell has at most w walls. Then there exists some hyperplane $H \in \mathcal{H}$ such that (for appropriate orientation of H) the numbers N^+ of those cells from $\mathcal{A}|_C$ lying on the positive side of H satisfies*

$$\frac{N-1}{w} \leqslant N^+ \leqslant \frac{N}{2}. \tag{5}$$

The following argument is due to Finschi, confer [10]:

[7] For degenerate arrangements, provisions similar to Remark 3.6 are to be made.

Proof. For $H \in \mathcal{H}$, let $N^+(H)$ denote the number of cells from $\mathcal{A}|_C$ on the positive side of H. W.l.o.g. (re-)orient each hyperplane such that $N^+(H) \leqslant N/2$. We will prove by contradiction that some $H \in \mathcal{H}$ satisfies (5). So suppose that

$$w \cdot N^+(H) < N - 1 \quad \forall H \in \mathcal{H}. \tag{6}$$

Now let $\mathcal{H}' \subseteq \mathcal{H}$ denote some arbitrary subset of the given hyperplanes.[8] For each such $\mathcal{H}' \subseteq \mathcal{H}$ consider the sub-collection

$$\{c \in \mathcal{A}(\mathcal{H}) \mid c \subseteq C, \ \forall H \in \mathcal{H}' : \ c \subseteq H^-\} \subseteq \mathcal{A}(\mathcal{H}) \tag{7}$$

of those cells from $\mathcal{A}|_C$ which are on the negative side H^- of each $H \in \mathcal{H}'$ (rather than of all $H \in \mathcal{H}$). Among those $\mathcal{H}' \subseteq \mathcal{H}$ for which this sub-collection (7) is non-empty (i.e., consists of at least one cell), choose some $\mathcal{H}'$ that is maximal w.r.t. set inclusion.

Claim. *Then, this sub-collection* (7) *consists of exactly one cell* (*say*, c) *which is furthermore a member* (*see footnote* 8) *of* $\mathcal{A}(\mathcal{H}')$.

In other words: c is the unique cell from $\mathcal{A}(\mathcal{H})$ contained in the super-cell C lying on the negative side of each $H \in \mathcal{H}' \subseteq \mathcal{H}$; and each $H \in \mathcal{H}'$ forms a wall of c. According to the prerequisite, these are at most w many.

So how many cells from $\mathcal{A}|_C$ have the property of lying on the positive side of *some* $H \in \mathcal{H}'$? As $|\mathcal{H}'| \leqslant w$ (see above), these are strictly less than $N-1$ many because, by presumption (6), each such H has strictly less than $(N-1)/w$ many cells on its positive side. On the other hand, $\mathcal{A}|_C$ consists of N cells and c from the above claim is the only one *not* lying on the positive side of any $H \in \mathcal{H}'$; so at least $N-1$ cells from $\mathcal{A}|_C$ lie on the positive side of some $H \in \mathcal{H}'$ while we just saw that strictly less than $N-1$ have this property: contradiction. □

Proof (*of the Claim*). $\mathcal{C} := C \cap \bigcap_{H \in \mathcal{H}'} H^-$ is a super-cell of $\mathcal{A}(\mathcal{H})$ and in particular convex. It consists of several (ordinary) cells, namely exactly the ones given by (7).

Now suppose that these are more than one. Choose an adjacent pair c_1, c_2 and some hyperplane $H \in \mathcal{H} \backslash \mathcal{H}'$ separating them; w.l.o.g. $c_1 \in H^+$ and $c_2 \in H^-$. Then c_2 lies on the negative sides not only of all $H' \in \mathcal{H}'$ but also of this new H which thus can be added to $\mathcal{H}'$: a contradiction to $\mathcal{H}'$ having been chosen maximal w.r.t. set inclusion.

Hence $\mathcal{C}$, the cell from $\mathcal{A}(\mathcal{H}')$, consists of only one cell from $\mathcal{A}(\mathcal{H})$; in particular, this ordinary cell also belongs to $\mathcal{A}(\mathcal{H}')$. □

6. Conclusion

Models of computation are thought to formally describe what actual PCs could do in principle. The present work proceeded along these lines by putting a very practical observation onto a sound mathematical basis: that numerical stability of algorithms in general can be achieved only at the expense of significant loss in performance. To this end, we proposed a joint algebraic-computable model and considered the Point Location problem.

This turned out to reveal an interesting quantitative trade-off between speed and stability: the faster an according algorithm has to run, the more *additional* tests on affine real functions (i.e., further possible sources of instability) *have* to be employed.

Our proofs combine techniques from areas as distant as Recursive Analysis and Combinatorial Geometry. Indeed from the latter, two deep results [9,12] enter in the respective proofs of Theorems 4.2 and 4.6.

It looks promising to extend the above synthesis of Computable Analysis with algebraic models of computation to, say, algebraic computation trees—cf. Section 4.4 of [6]—and thus obtain a new kind of Algebraic Complexity Theory for *stable* algorithms. In fact, the construction applied in Lemma 3.5 is the special (linear) case of a 'Cylindrical Algebraic Decomposition' well-known in Computational Algebraic Geometry [1].

[8] Keep in mind that a cell of $\mathcal{A}(\mathcal{H})$ is in general only *part* of a cell from $\mathcal{A}(\mathcal{H}')$.

Acknowledgements

The author wishes to express his gratitude to the anonymous referee for many helpful comments on this work including the following very interesting suggestion: in place of a (deterministic) computable LDT T as in Definition 2.1, consider one in which the branching is permitted to behave nondeterministically. Consequently, the regions T_u from Definition 4.7 may now overlap for distinct leafs. Then, membership to a computable convex n-gon P becomes totally semi-decidable within logarithmic depth—as opposed to Theorem 3.3—by triangulating it and testing a triangles' boundary inside of P only nondeterministically, that is, without necessarily diverging on an additionally queried line.

References

[1] S. Basu, R. Pollack, M.-F. Roy, Algorithms in Real Algebraic Geometry, Springer, Berlin, 2003.
[2] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry, Springer, Berlin, 1997.
[3] V. Brattka, P. Hertling, Feasible real random access machines, J. Complexity 14 (4) (1998) 490–526.
[4] V. Brattka, M. Ziegler, Computability of linear equations, in: Proc. 2nd IFIP Internat. Conf. on Theoretical Computer Science 'TCS@Montreal', Kluwer, Dordrecht, 2002, pp. 95–106; see also M. Ziegler, V. Brattka, Computability in linear algebra, Theoret. Comput. Sci. 326 (1–3) (2004) 187–211.
[5] R.L. Burden, J.D. Faires, Numerical Analysis, fourth ed., PWS-Kent, Boston, MA, 1989.
[6] P. Bürgisser, M. Clausen, M.A. Shokrollahi, Algebraic Complexity Theory, Springer, Berlin, 1997.
[7] B. Chazelle, Cutting hyperplanes for divide-and-conquer, Discrete Comput. Geom. 9 (2) (1993) 145–158.
[8] Chee-Keng Yap, Symbolic treatment of geometric degeneracies, J. Symbolic Comput. 10 (1990) 349–370.
[9] K.E. Clarkson, H. Edelsbrunner, L.J. Guibas, M. Sharir, E. Welzl, Combinatorial complexity bounds for arrangements of curves and spheres, Discrete Comput. Geom. 5 (1990) 99–160.
[10] V. Damerow, L. Finschi, M. Ziegler, Point location algorithms of minimum size, in: Proc. 14th Canadian Conf. on Computational Geometry (CCCG'02), pp. 5–9.
[11] D. Dobkin, R.J. Lipton, Multidimensional searching problems, SIAM J. Comput. 5 (2) (1976) 181–186.
[12] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, M. Sharir, Arrangements of curves in the plane—topology, combinatorics, and algorithms, Theoret. Comput. Sci. 92 (1992) 319–336.
[13] S. Fortune, C.J. Van Wyk, Static analysis yields efficient exact integer arithmetic for computational geometry, ACM Trans. Graph. 15 (3) (1996) 223–248.
[14] B. Grünbaum, Convex Polytopes, Wiley, New York, 1967.
[15] A. Grzegorczyk, On the definitions of computable real continuous functions, Fund. Math. 44 (1957) 61–77.
[16] P. Hertling, Topological complexity of zero finding with algebraic operations, J. Complexity 18 (2002) 912–942.
[17] Ker-I Ko, Complexity Theory of Real Functions, Birkhäuser, Basel, 1991.
[19] M.B. Pour-El, J.I. Richards, Computability in Analysis and Physics, Springer, Berlin, 1989.
[20] M. Sharir, P.K. Agarwal, Davenport–Schinzel Sequences and Their Geometric Applications, Cambridge, Oxford, 1995.
[21] A.M. Turing, On computable numbers with an application to the entscheidungsproblem, Proc. London Math. Soc. 42 (2) (1936) 230–265.
[22] A.M. Turing, Rounding-off errors in matrix processes, Quart. J. Mech. Appl. Math. I (1948) 287–308.
[23] E. Ukkonen, Exponential lower bounds for some $\mathcal{NP}$-complete problems in restricted linear decision tree model, BIT 23 (2) (1983) 181–192.
[24] K. Weihrauch, Computable Analysis, Springer, Berlin, 2000.
[25] S. Willard, General Topology, Addison-Wesley, Reading, MA, 1970.
[26] M. Ziegler, Computability on regular subsets of Euclidean space, Math. Logic Quart. 48 (Suppl. 1) (2002) 157–181.
[27] M. Ziegler, V. Brattka, A computable spectral theorem, Computability and Complexity Analysis, Lecture Notes in Computer Science, Vol. 2064, Springer, Berlin, 2001, pp. 378–388.