JOURNAL OF COMPLEXITY **12,** 315–338 (1996)
ARTICLE NO. 0021

Topological Complexity with Continuous Operations

PETER HERTLING

Theoretische Informatik I, FernUniversität, D-58084 Hagen, Germany

Received May 1996

The topological complexity of algorithms is studied in a general context in the first part and for zero-finding in the second part. In the first part the *level of discontinuity* of a function f is introduced and it is proved that it is a lower bound for the total number of comparisons plus 1 in any algorithm computing f that uses only continuous operations and comparisons. This lower bound is proved to be sharp if arbitrary continuous operations are allowed. Then there exists even a balanced optimal computation tree for f. In the second part we use these results in order to determine the topological complexity of zero-finding for continuous functions f on the unit interval with $f(0) \cdot f(1) < 0$. It is proved that roughly $\log_2 \log_2 \varepsilon^{-1}$ comparisons are optimal during a computation in order to approximate a zero up to ε. This is true regardless of whether one allows arbitrary continuous operations or just function evaluations, the arithmetic operations $\{+, -, *, /\}$, and the absolute value. It is true also for the subclass of nondecreasing functions. But for the subclass of increasing functions the topological complexity drops to zero even for the smaller class of operations. © 1996 Academic Press, Inc.

1. INTRODUCTION

This paper consists of two parts. In the first part we analyze the topological complexity of algorithms which may use arbitrary continuous operations. In the second part we study the topological complexity of zero-finding.

In the first part we prove a general result on the topological complexity of algorithms in the sense of Smale (1987) and Vassiliev (1992). They gave estimations for the total number of branching nodes needed in a computation tree that computes a vector of zeros of a univariate complex polynomial up to a small error. We consider computation trees that may use arbitrary continuous operations. In fact, we consider *continuous computation trees* on arbitrary topological spaces. On the other hand, we introduce a simple and presumably new ordinal rank measuring the discontinuity of a function, which we call the *level* of the function. Our main result can be stated as follows: if f is a function of finite level then the minimum of the total number of branching nodes in any continuous computation tree computing f is equal to the level of f minus 1. Concerning the upper bound we show more: there is an optimal continuous computation tree of minimal depth, i.e., if the level of f is $n \geq 1$ then there is a continuous computation tree for f with $n - 1$ branching nodes and with at most $\lceil \log_2 n \rceil$ branching nodes on any path. Furthermore we give local definitions of the level of discontinuity of a function at a point and of the topological complexity of an algorithm at a point and show that also in the local case the analogous estimates hold true and are sharp. Note that the lower bound is valid for real number machines as introduced by Blum *et al.* (1989) and for real number oracle machines (with continuous operations) as introduced by Novak (1995) and Novak and Woźniakowski (1996).

In the last mentioned paper Novak and Woźniakowski (1996) study the topological complexity of zero-finding for continuous functions on the unit interval. They consider real number oracle machines which for a fixed $\varepsilon > 0$ solve the problem to compute an ε-approximation of a zero of f for any $f \in C[0, 1]$ with $f(0) < 0$ and $f(1) > 0$, i.e., the problem to compute a number x such that there is a zero x^* of f with $|x - x^*| \leq \varepsilon$. For example, the bisection algorithm uses $\lceil \log_2 \varepsilon^{-1} \rceil - 1$ comparisons during a computation (for $\varepsilon < 1$). One of the results of Novak and Woźniakowski (1996) states that this is optimal if the real number oracle machine may use arbitrary Hölder operations, i.e., operations that satisfy a Hölder condition on any bounded subset of their domain. We show that one can perform exponentially better with a machine which besides comparisons uses only function evaluations, the arithmetic operations $\{+, -, *, /\}$, and the absolute value $|\cdot|$. Observe that division is not Hölder. More precisely, we prove that there is a computation tree using these operations which contains exactly $\lceil \log_2(\varepsilon^{-1} + 2) \rceil - 2$ branching nodes, has minimal depth, and computes an ε-approximation of a zero even for any function $f \in C[0, 1]$ with $f(0) \cdot f(1) < 0$. Hence it contains at most

$$\lceil \log_2(\lceil \log_2(\varepsilon^{-1} + 2) \rceil - 1) \rceil$$

branching nodes on any path. On the other hand, using the characterization obtained in the first part we show that this is optimal even if arbitrary continuous arithmetic operations and arbitrary continuous information operations are allowed.

In fact, the lower bound is proved already for the subclass of nondecreasing functions. But for the slightly smaller class of increasing functions the zero can be approximated by a real number oracle machine that uses only the arithmetic operations $\mathrm{ARI}_{\mathrm{abs}} := \{+, -, *, /, |\cdot|\}$, function values, and no comparisons at all.

We conclude that the approximation of zeros—considered in the second part—of (nondecreasing) functions $f \in C[0, 1]$ with $f(0) \cdot f(1) < 0$ is an example of a problem where a purely topological lower bound—as considered in the first part—can be achieved by an algorithm that uses only simple arithmetic operations.

2. Continuous Computation Trees and the Level of Discontinuity

First we introduce *continuous computation trees*, which subsume the purely topological properties of computation trees arising from e.g. real number machines as introduced by Blum *et al.* (1989) or oracle machines, see Novak and Woźniakowski (1996). Then the *level of discontinuity* of a function is defined. In the last subsection results connecting continuous computation trees and the level of functions are given.

In the whole section X and Y are arbitrary topological spaces if nothing else is said.

2.1. *Continuous Computation Trees*

A *topological test* on X is a function $t :\subseteq X \to \{\text{TRUE}, \text{FALSE}\}$ such that $t^{-1}\{\text{TRUE}\}$ is an open subset of $\text{dom}(t)$. (By $f :\subseteq X \to Y$ we denote a (possibly partial) function f with $\text{dom}(f) \subseteq X$ and $\text{range}(f) \subseteq Y$).

Definition 2.1. A *continuous computation tree* (*CCT*) over (X, Y) is a finite binary tree T whose internal nodes contain topological tests on X and whose leaves contain continuous functions $g :\subseteq X \to Y$. The function $f_T :\subseteq X \to Y$ computed by T is defined recursively. If T consists just of one leaf, $T = (g)$ where $g :\subseteq X \to Y$ is a continuous function, then we set $f_T := g$. If $T =$ (if t then T_1 else T_2) where t is a topological test on X and T_1 and T_2 are continuous computation trees, then

$$f_T(x) := \begin{cases} f_{T_1}(x) & \text{if } t(x) \\ f_{T_2}(x) & \text{if } \neg t(x) \\ \text{undefined} & \text{if } x \notin \text{dom}(t) \end{cases}$$

for all $x \in X$.

The *computation path* that is followed during a computation on an input $x \in \text{dom}(f_T)$ is defined in the usual way as the sequence of nodes from the root to a leaf that are visited during the computation.

Before we define the global and the local topological complexity of a CCT we discuss why the CCT model expresses the purely topological properties of real number algorithms. For details on real number machines the reader is referred to Blum *et al.* (1989). We consider only algorithms that stop after a bounded number of steps for any input for which they give a result (if for an algorithm there is no upper bound for the length of the computations on valid inputs then there is also no upper bound for the number of comparisons executed during such computations). Their computation can be described by a finite rooted tree. The computation starts at the root where the input is read into certain registers and, if successful, it ends in one of the leaves where the output is contained in output registers. There are two types of internal nodes. The *branching nodes* contain comparisons "$r_i : r_j$" of real numbers where ":" $\in \{=, \neq, <, >, \leq, \geq\}$. The *computation nodes* contain arithmetic operations "$r_i := \text{op}(r_{i_1}, \ldots, r_{i_k})$" where op belongs to a prescribed class ARI of basic arithmetic operations. This might include only the four basic operations $\{+, -, *, /\}$ (and rational constants) or evaluation of polynomials or additionally some elementary functions like e.g. exp, log, or the absolute value $|\cdot|$. If a division by zero is tried or, more generally, if an argument does not lie in the domain of definition of the operation which is to be executed then the computation stops without result.

In this paper we are interested only in the topological complexity of an algorithm, i.e., in its branching nodes. Hence we rewrite the algorithm without arithmetic nodes. They can be eliminated by replacing the simple comparisons in the branching nodes by tests of the form "$T(x) : 0$" where x is the input and T is the function composed of the basic arithmetic operations that are executed on the sequence of nodes leading from the root to the corresponding branching node. In the same way one has to write the results in the leaves in the form "$g(x)$" for analogously resulting functions g. If all the basic arithmetic operations are continuous then the composed functions T and g are continuous too. Hence we have a continuous computation tree.

This can also be applied to real number algorithms with additional information operations. If the considered input space is not a finite-dimensional Euclidean space but, e.g., a sufficiently large subspace of $C[0, 1]$ then in the real number model one has to use other operations in order to obtain information about the input objects. Novak and Woźniakowski (1996) have introduced the real number oracle machines that have additional *information nodes.* In these nodes information operations $L: I \times \mathbb{R}^k \to \mathbb{R}$, defined on the input space I and on a real Euclidean space $\mathbb{R}^k$, are executed. If for example the input space is a subspace of $C[0,1]$ then the evaluation operator $\text{eval} : C[0, 1] \times [0, 1] \to \mathbb{R}$, $\text{eval}(f, x) = f(x)$ is a common information operator. If the used information operators and the arithmetic operations are continuous then one can eliminate them as above and again one obtains a continuous computation tree.

We conclude that any computation tree describing a real number machine or a real number oracle machine can be transformed canonically into a CCT if all used operations—besides the comparisons, of course—are continuous. Especially any lower bounds proved for CCTs are valid for real number (oracle) machines, too.

Following Smale (1987), in this section we consider the total number of branching nodes in a computation tree as the topological complexity of the algorithm. Since our trees are binary this is the number of leaves minus one. In Section 2.3 we will see that in this context the depth of a tree is a less sharp complexity measure since for any CCT there is an equivalent (i.e., computing the same function) balanced CCT of minimal size.

We define also a local version of the topological complexity of T at a point.

DEFINITION 2.2. The *size* $\mathrm{Size}(T)$ of a CCT T is the number of leaves of T. The *branching number* $\mathrm{size}(T, x)$ of T at a point $x \in X$ is the number of all computation paths such that in any neighborhood U of x there are points $x' \in \mathrm{dom}(f_T) \cap U$ such that on input x' this path is followed.

The branching number can also be expressed recursively. If T is a CCT over (X, Y) and $M \subseteq X$ is a subset of X, then the *restriction of T to M* is defined recursively by

$$T|_M := \begin{cases} (g|_M) & \text{if } T = (g) \\ (\text{if } t \text{ then } T_1|_M \text{ else } T_2|_M) & \text{if } T = (\text{if } t \text{ then } T_1 \text{ else } T_2). \end{cases}$$

LEMMA 2.3. *Let T be a CCT over (X, Y) and $x \in X$. If $T = (g)$ then*

$$\mathrm{size}(T, x) = \begin{cases} 1 & \text{if } x \in \overline{\mathrm{dom}(g)} \\ 0 & \text{else,} \end{cases}$$

and if $T = (\text{if } t \text{ then } T_1 \text{ else } T_2)$ then

$$\mathrm{size}(T, x) = \mathrm{size}(T_1|_{t^{-1}\{\mathrm{TRUE}\}}, x) + \mathrm{size}(T_2|_{t^{-1}\{\mathrm{FALSE}\}}, x).$$

The proof is easily done by induction. Obviously $\mathrm{size}(T, x) \leq \mathrm{Size}(T)$ for all $x \in X$.

Remarks. 1. The branching number can be described as the number of all computation paths that could be followed if x is disturbed slightly at the beginning of the computation. Another local version of the topological complexity seems to be interesting as well: the number of all computation paths that could be followed if at *any* internal node the input x might be disturbed independently. This *level of degeneracy* $size_\beta(T, x)$ of T at a point x can also be defined recursively: if $T = (g)$ then

$$\mathrm{size}_\beta(T, x) = \begin{cases} 1 & \text{if } x \in \overline{\mathrm{dom}(g)} \\ 0 & \text{else,} \end{cases}$$

and if $T =$ (if t then T_1 else T_2), then

$$\mathrm{size}_\beta(T, x) = \begin{cases} \mathrm{size}_\beta(T_1, x) + \mathrm{size}_\beta(T_2, x) & \text{if } x \in \overline{t^{-1}\{\mathrm{TRUE}\}} \cap \overline{t^{-1}\{\mathrm{FALSE}\}} \\ \mathrm{size}_\beta(T_1, x) & \text{if } x \in \overline{t^{-1}\{\mathrm{TRUE}\}} \setminus \overline{t^{-1}\{\mathrm{FALSE}\}} \\ \mathrm{size}_\beta(T_2, x) & \text{if } x \in \overline{t^{-1}\{\mathrm{FALSE}\}} \setminus \overline{t^{-1}\{\mathrm{TRUE}\}} \\ 0 & \text{if } x \in X \setminus \overline{\mathrm{dom}(t)}. \end{cases}$$

One easily checks $\mathrm{size}(T, x) \le \mathrm{size}_\beta(T, x) \le \mathrm{Size}(T)$ for all $x \in X$. Compare also the second remark in Section 2.3.

2. We have considered only binary tests. Sometimes also comparisons of real numbers with a subsequent ternary branching according to the result "less", "equal", or "larger" of the comparison are considered. If all of the used operations are continuous then these tests have the form $t :\subseteq X \to \{-1, 0, 1\}$ where $t^{-1}\{-1\}$ and $t^{-1}\{1\}$ are disjoint open subsets of $\mathrm{dom}(t)$. It is not difficult to prove that a continuous computation tree which may contain also such ternary tests can be transformed into a CCT according to Definition 2.1 (i.e., containing only binary topological tests) that contains at most as many branching nodes. Thus, if arbitrary continuous operations are allowed, then one cannot reduce the total number of branching nodes needed by using ternary tests.

2.2. *The Level of Discontinuity of a Function*

In the last section we considered the topological complexity of algorithms. Here we introduce a very simple, but presumably new, ordinal rank measuring the discontinuity of a function. In the next section we shall see that it is a lower bound for the topological complexity of an algorithm computing the function, and, in the case of CCTs, a sharp lower bound.

We introduce a global and a local version.

DEFINITION 2.4. Let $f :\subseteq X \to Y$ be a (possibly partial) function. We set $C_0(f) := \mathrm{dom}(f)$ and for any $n \in \mathbb{N} = \{0, 1, 2, \ldots\}$

$$C_{n+1}(f) := \mathrm{dom}(f) \cap \mathrm{closure}\{x \in C_n(f) \mid f|_{C_n(f)} \text{ is discontinuous in } x\}.$$

The *level of f* is $\mathrm{Lev}(f) := \min\{n \mid C_n(f) = \emptyset\}$ where we set $\min \emptyset := \infty$, and for any $x \in X$ the *level of f in x* is $\mathrm{lev}(f, x) := \min\{n \mid x \notin C_n(f)\}$.

The following properties are obvious.

LEMMA 2.5. *Let functions $f, g :\subseteq X \to Y$ be fixed.*

1. *The sequence $(C_n(f))_n$ is falling, i.e., $C_n(f) \subseteq C_m(f)$ for any $m < n$.*
2. $\mathrm{lev}(f, x) \le \mathrm{Lev}(f)$ *for all $x \in X$ and* $\mathrm{Lev}(f) = \max\{\mathrm{lev}(f, x) \mid x \in X\}$ *if* $\mathrm{Lev}(f)$ *is finite.*
3. *If $U \subseteq X$ is open and the functions f and g coincide on U, i.e., $f|_U = g|_U$,* then $\mathrm{lev}(f, x) = \mathrm{lev}(g, x)$ *for all $x \in U$.*
4. *If $f = g|_M$ is the restriction of g to a subset $M \subseteq X$ then* $\mathrm{lev}(f, x) \le \mathrm{lev}(g, x)$ *for all $x \in X$.*

The level is additive in the following sense.

LEMMA 2.6. *Let $f :\subseteq X \to Y$ be a function. Then $C_m(f|_{C_n(f)}) = C_{n+m}(f)$ for all $m, n \in \mathbb{N}$ and hence* $\mathrm{lev}(f, x) = n + \mathrm{lev}(f|_{C_n(f)}, x)$ *for all $x \in C_n(f)$, $n \in \mathbb{N}$.*

Proof. The second assertion is a consequence of the first. We prove the first by induction over m. It is trivial for $m = 0$. We obtain

$$\begin{aligned} C_{n+m+1}(f) &= \mathrm{dom}(f) \cap \mathrm{closure}\{x \in C_{n+m}(f) \mid f|_{C_{n+m}(f)} \text{ is discontinuous in } x\} \\ &= C_n(f) \cap \mathrm{closure}\{x \in C_m(f|_{C_n(f)}) \mid f|_{C_m(f|_{C_n(f)})} \\ &\quad \text{is discontinuous in } x\} \\ &= C_{m+1}(f|_{C_n(f)}) \end{aligned}$$

by Lemma 2.5(1) and by the induction hypothesis. ■

Remarks. 1. We have defined the sequence $C_n(f)$ only for finite index n. One can in a natural way extend the definition to arbitrary ordinal numbers and, thus, obtain transfinite levels of discontinuity. When one considers only the characteristic functions of sets the resulting subhierarchy coincides (essentially) with Hausdorff's (1935) difference hierarchy of sets; see Hertling (1996).

2. Instead of the sets $C_n(f)$ one could consider the sets $A_n(f)$ which are defined in the same way without taking the closure, i.e., $A_0(f) := \mathrm{dom}(f)$,

$$A_{n+1}(f) := \{x \in A_n(f) \mid f|_{A_n(f)} \text{ is discontinuous in } x\}.$$

The definition can also be extended to arbitrary ordinals. The level $\mathrm{Lev}'(f)$ of f, defined via this sequence of sets, has very similar properties to the level according to Definition 2.4. In general the level $\mathrm{Lev}(f)$ can be much larger than $\mathrm{Lev}'(f)$. But if the range of f is a regular space then $\mathrm{Lev}'(f) \le \mathrm{Lev}(f) \le \mathrm{Lev}'(f) + 1$, and, if additionally $\mathrm{Lev}(f)$ is finite, then even $\mathrm{Lev}(f) = \mathrm{Lev}'(f)$ can be proved. For more results on the levels see Hertling (1996). Altogether the level $\mathrm{Lev}(f)$, resp. $\mathrm{lev}(f, x)$ from Definition 2.4, seems to have more natural properties.

2.3. *The Topological Complexity and the Level of Discontinuity*

The first main result combining the topological complexity and the level states that the level of a function gives a lower bound for the topological complexity of any algorithm that computes the function and uses only comparisons and continuous arithmetic (and information) operations. In fact, it is a lower bound for the size of a CCT computing the function. This is true globally and locally. Second, if for the algorithm one allows arbitrary continuous operations, i.e., a CCT, then this estimation is sharp. One can even construct an optimal CCT which is balanced and contains only total tests.

THEOREM 2.7. *Let T be a CCT over (X, Y). Then* $\mathrm{Lev}(f_T) \le \mathrm{Size}(T)$ *and* $\mathrm{lev}(f_T, x) \le \mathrm{size}(T, x)$ *for all $x \in X$.*

A binary tree with n leaves is called *balanced* if it has minimal depth, i.e., if its longest path contains $\lceil \log_2 n \rceil$ branching nodes.

THEOREM 2.8. *Let $f :\subseteq X \to Y$ be a function with finite level* $0 < \mathrm{Lev}(f) < \infty$. *There is a balanced CCT T containing only total tests which computes f (i.e., $f = f_T$) and satisfies* $\mathrm{Size}(T) = \mathrm{Lev}(f)$ *and* $\mathrm{size}(T, x) = \mathrm{lev}(f, x)$ *for all $x \in \mathrm{dom}(f)$.*

Remarks. 1. If X is a metric space then any topological test "$t(x)$" can be written in the usual form "$T(x) > 0$" for a continuous function T. Namely, one uses $T :\subseteq X \to \mathbb{R}$ defined by $\mathrm{dom}(T) := \mathrm{dom}(t)$, by $T(x) := d(x, t^{-1}\{\mathrm{FALSE}\})$ for $x \in \mathrm{dom}(t)$ if $t^{-1}\{\mathrm{FALSE}\} \ne \emptyset$, and by $T(x) := 1$ for $x \in \mathrm{dom}(t)$ if $t^{-1}\{\mathrm{FALSE}\} = \emptyset$. Hence a function $f :\subseteq X \to Y$ of finite level, defined on a metric space X, can be computed by a balanced, optimal real number algorithm using continuous operations and the usual comparisons of real numbers.

2. Previous versions of these results have been presented by Hertling and Weihrauch (1994). There also the following issue is discussed. The level of discontinuity seems to be a scalar measure for the "inherent degen-

eracies" introduced by Yap (1990) for problems in computational geometry. Correspondingly, the branching number and the level of degeneracy (see the first remark at the end of Section 2.1) seem to be scalar measures for the "algorithm-induced degeneracies" introduced in the same paper. Under this assumption our Theorem 2.7 gives a quantitative proof of Yap's observation: "it seems that induced degeneracies subsume inherent degeneracies".

For the proof of Theorem 2.7 we need a simple statement about the branching number. A function $f : X \to \mathbb{N}$ on a topological space X is upper semicontinuous if for any point $x \in X$ there is a neighborhood U of x such that $f(x') \leq f(x)$ for all $x' \in U$.

LEMMA 2.9. *Let T be a CCT. Then the branching number function* $\mathrm{size}(T, .) : X \to \mathbb{N}$ *is upper semicontinuous.*

The proof is done by induction using Lemma 2.3.

Proof of Theorem 2.7. The first assertion follows immediately from the second assertion because

$$\mathrm{lev}(f_T, x) \leq \mathrm{size}(T, x) \tag{1}$$

for all $x \in X$ induces $\mathrm{Size}(T) \geq \max\{\mathrm{size}(T, x) | x \in X\} \geq \max\{\mathrm{lev}(f_T, x) | x \in X\} = \mathrm{Lev}(f_T)$.

Fix an $x \in X$. We prove (1) by induction over the structure of T. If $T = (g)$ consists of one leaf then $f_T = g$ is a continuous function, hence

$$\mathrm{lev}(f_T, x) = \mathrm{lev}(g, x) = \left\{\begin{matrix} 1 & \text{if } x \in \mathrm{dom}(g) \\ 0 & \text{else} \end{matrix}\right\} \leq \mathrm{size}(T, x).$$

Otherwise T has the form $T = (\text{if } t \text{ then } T_1 \text{ else } T_2)$ where t is a topological test and T_1 and T_2 are CCTs. We define $T'_1 := T_1|_{t^{-1}\{\mathrm{TRUE}\}}$ and

$$k := \mathrm{size}(T'_1, x).$$

If $\mathrm{lev}(f_T, x) \leq k$ then the proof is finished by Lemma 2.3. Let us assume $\mathrm{lev}(f_T, x) > k$, i.e., $x \in C_k(f_T)$. By Lemma 2.6 we obtain $\mathrm{lev}(f_T, x) = k + \mathrm{lev}(f_T|_{C_k(f_T)}, x)$. Hence by Lemma 2.3 we have to show only

$$\mathrm{lev}(f_T|_{C_k(f_T)}, x) \leq \mathrm{size}(T_2|_{t^{-1}\{\mathrm{FALSE}\}}, x). \tag{2}$$

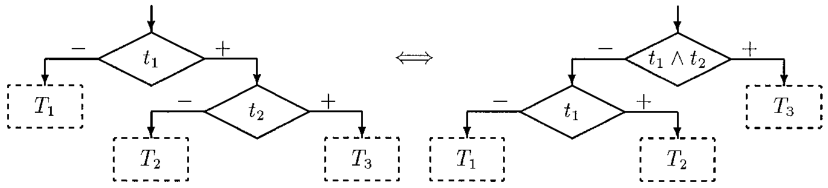

FIG. 1. Equivalent CCTs, part 1.

Let $U \subseteq X$ be open with $U \cap \mathrm{dom}(t) = t^{-1}\{\mathrm{TRUE}\}$. Then $f_T|_U = f_{T|_U} = f_{T_1'}$. For all $x' \in U$ we have by Lemma 2.5(3) and by the induction hypothesis

$$\mathrm{lev}(f_T, x') = \mathrm{lev}(f_T|_U, x') = \mathrm{lev}(f_{T_1'}, x') \le \mathrm{size}(T_1', x').$$

Since the branching number is upper semicontinuous by Lemma 2.9 there is an open neighborhood V of x such that $\mathrm{size}(T_1', x') \le \mathrm{size}(T_1', x)$ for all $x' \in V$. For all $x' \in U \cap V$ we obtain

$$\mathrm{lev}(f_T, x') \le \mathrm{size}(T_1', x) = k,$$

hence $C_k(f_T) \cap U \cap V = \varnothing$ and thus $C_k(f_T) \cap V \subseteq X \setminus U$. Using Lemma 2.5, $f_T|_{X\setminus U} = f_{T_2|_{t^{-1}\{\mathrm{FALSE}\}}}$, and the induction hypothesis we finally deduce the assertion (2):

$$\begin{aligned}\mathrm{lev}(f_T|_{C_k(f_T)}, x) &= \mathrm{lev}(f_T|_{C_k(f_T)\cap V}, x)\\ &\le \mathrm{lev}(f_T|_{X\setminus U}, x)\\ &= \mathrm{lev}(f_{T_2|_{t^{-1}\{\mathrm{FALSE}\}}}, x)\\ &\le \mathrm{size}(T_2|_{t^{-1}\{\mathrm{FALSE}\}}, x). \quad \blacksquare\end{aligned}$$

The main part of the construction in the proof of Theorem 2.8 will give an unbalanced tree. For the balancing we use the following lemma. Let t_1, $t_2 : X \to \{\mathrm{TRUE, FALSE}\}$ be total topological tests on X. Then their conjunction $t_1 \bigwedge t_2 : X \to \{\mathrm{TRUE, FALSE}\}$ and their disjunction $t_1 \bigvee t_2 : X \to \{\mathrm{TRUE, FALSE}\}$ are also total topological tests.

LEMMA 2.10. *Let t_1, $t_2 : X \to$ {TRUE, FALSE} be total topological tests and T_1, T_2, T_3 CCTs. Then the two CCTs in* Fig. 1 *are equivalent to each other and the two CCTs in* Fig. 2 *are equivalent to each other. Furthermore, in both cases the branching number is the same at each point.*

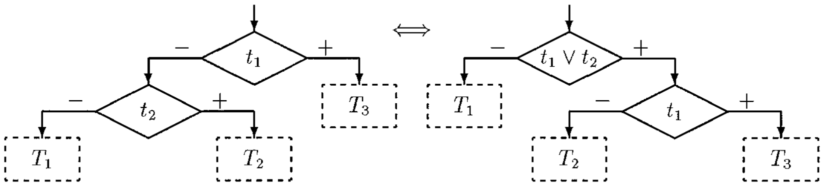

FIG. 2. Equivalent CCTs, part 2.

The proof follows immediately from the definitions and from Lemma 2.3.

Proof of Theorem 2.8. We construct a tree T which has all the requested properties except being balanced. With the last lemma one can transform this tree into a CCT which is additionally balanced.

If f is continuous with nonempty domain, i.e., $\mathrm{Lev}(f) = 1$, then f is computed by the CCT $T := (f)$ without any tests. Then $\mathrm{Size}(T) = 1 = \mathrm{Lev}(f)$ and $\mathrm{lev}(f, x) = 1 = \mathrm{size}(T, x)$ for all $x \in \mathrm{dom}(f)$.

Let f be a function with $\mathrm{Lev}(f) = n + 1$ for some $n \geq 1$. The function $f^{(-)} := f|_{C_1(f)}$ has level $\mathrm{Lev}(f^{(-)}) = n$ by Lemma 2.5(2) und Lemma 2.6. Hence, by the induction hypothesis there is a CCT $T^{(-)}$ with $n - 1$ total tests which computes $f^{(-)}$ and has the properties $\mathrm{lev}(f^{(-)}, x) = \mathrm{size}(T^{(-)}, x)$ for all $x \in \mathrm{dom}(f^{(-)})$. Using the continuous function $f^{(+)} := f|_{X \setminus C_1(f)}$ we define a CCT T by

$$T := (\text{if } x \in X \setminus \overline{C_1(f)} \text{ then } (f^{(+)}) \text{ else } T^{(-)}).$$

Note that the set $C_1(f)$ is closed in $\mathrm{dom}(f)$. It is obvious that the CCT T computes the function f. The tests in T are total. The tree T contains n tests. Hence it has size $\mathrm{Size}(T) = n + 1 = \mathrm{Lev}(f)$. We still have to show $\mathrm{size}(T, x) = \mathrm{lev}(f, x)$ for all $x \in \mathrm{dom}(f)$. By Theorem 2.7 we already know size $(T, x) \geq \mathrm{lev}(f, x)$ for all $x \in X$. Hence it is sufficient to prove

$$\mathrm{size}(T, x) \leq \mathrm{lev}(f, x) \tag{3}$$

for all $x \in \mathrm{dom}(f)$. Fix an $x \in \mathrm{dom}(f)$. By definition we have

$$\mathrm{size}(T, x) = \mathrm{size}((f^{(+)})|_{X \setminus \overline{C_1(f)}}, x) + \mathrm{size}(T^{(-)}|_{\overline{C_1(f)}}, x).$$

Obviously $\mathrm{size}((f^{(+)})|_{X \setminus \overline{C_1(f)}}, x) \leq 1$ and $\mathrm{size}(T^{(-)}|_{\overline{C_1(f)}}, x) \leq \mathrm{size}(T^{(-)}, x)$, hence

$$\text{size}(T, x) \leq 1 + \text{size}(T^{(-)}, x).$$

We distinguish the cases $x \in \text{dom}(f) \setminus C_1(f)$ and $x \in C_1(f)$.

First assume $x \in \text{dom}(f) \setminus C_1(f)$. Then $x \notin \overline{C_1(f)} = \overline{\text{dom}(f^{(-)})} = \overline{\text{dom}(f_{T^{(-)}})}$. This implies $\text{size}(T^{(-)}, x) = 0$. Thus, in the case $x \in \text{dom}(f) \setminus C_1(f)$ we have $\text{size}(T, x) \leq 1 = \text{lev}(f, x)$.

Now assume $x \in C_1(f)$. Then $x \in \text{dom}(f^{(-)})$. By the induction hypothesis we know $\text{size}(T^{(-)}, x) = \text{lev}(f^{(-)}, x)$. We obtain

$$\text{size}(T, x) \leq 1 + \text{lev}(f^{(-)}, x) = \text{lev}(f, x)$$

where for the last equation we used the additivity of the level (Lemma 2.6). We have proved (3) in both cases. This finishes the proof. ■

3. Approximating Zeros

3.1. *The Problem and the Results*

In their recent paper Novak and Woźniakowski (1996) studied the topological complexity of zero-finding for continuous functions on the unit interval under various assumptions. We give new results of the same type. Consider the space

$$F := \{f \in C[0, 1] \mid f(0) \cdot f(1) < 0\}$$

with the maximum norm $\|f\| = \max\{|f(x)| \mid x \in X\}$ for all $f \in F$ and its subspace of nondecreasing functions

$$F_{\text{nd}} := \{f \in C[0, 1] \mid f \text{ is nondecreasing and } f(0) < 0, f(1) > 0\}.$$

An *ε-approximation* of a number x^* is a number x with $|x - x^*| \leq \varepsilon$. For a fixed subspace G of F and a fixed $\varepsilon > 0$ we consider the problem to determine an ε-approximation of a zero of f, for any $f \in G$. We are interested in the minimal topological complexity of an algorithm that solves this problem. Here under an algorithm we understand a real number oracle machine as sketched in Section 2.1. For details see Novak and Woźniakowski (1996). In this section we study the topological complexity for the spaces F and F_{nd} and in the final section we shall consider the subspace of increasing functions.

A purely topological lower bound for the problem over the space F_{nd} is given by the following proposition, which will be proved in Section 3.2.

PROPOSITION 3.1. *Let $\varepsilon > 0$ and let $P_\varepsilon : F_{\mathrm{nd}} \to \mathbb{R}$ be a function such that $P_\varepsilon(f)$ is an ε-approximation of a zero of f for any $f \in F_{\mathrm{nd}}$. Then* $\mathrm{Lev}(P_\varepsilon) \geq \log_2(\varepsilon^{-1} + 2) - 1$.

By Theorem 2.7 this gives a lower bound for the total number of branching nodes in any CCT computing an ε-approximation for any $f \in F_{\mathrm{nd}}$. We will see that it is a sharp bound. In fact it is a sharp bound not only for F_{nd} but for F. Using this, Theorem 2.8 tells us that there is an optimal balanced CCT solving the problem. We shall prove that there is even an optimal balanced tree describing a real number oracle machine that—besides comparisons—uses only the arithmetic operations

$$\mathrm{ARI}_{\mathrm{abs}} := \{+, -, *, /, |\cdot|\},$$

i.e., the four basic arithmetic operations of addition, subtraction, multiplication, and division, and the absolute value, and for information operations it uses only the evaluation operator

$$\mathrm{eval}\colon F \times [0,1] \to \mathbb{R}, \qquad \mathrm{eval}(f, x) := f(x).$$

Note that the arithmetic operations in $\mathrm{ARI}_{\mathrm{abs}}$ and the information operator eval are continuous. The main properties of the algorithm are described in the next proposition. The proof is given in Section 3.3.

PROPOSITION 3.2. *For any $n \in \mathbb{N}$ there is a computation tree with the following properties:* (1) *for any function $f \in F$ it computes a* $(1/(2^{n+2} - 2))$*-approximation of a zero of f;* (2) *it uses only* $\mathrm{ARI}_{\mathrm{abs}}$, *the information operator* eval, *and comparisons; and* (3) *the total number of branching nodes is n and the maximal number of branching nodes on any computation path is* $\lceil \log_2(n + 1) \rceil$.

Especially, we conclude that the lower bound of Proposition 3.1 is sharp.

COROLLARY 3.3. *Let $\varepsilon > 0$ and G be a subspace with $F_{\mathrm{nd}} \subseteq G \subseteq F$. Then the minimum of the levels of all functions $P_\varepsilon : G \to \mathbb{R}$, such that $P_\varepsilon(f)$ is an ε-approximation of a zero of f for any $f \in G$, is equal to* $\lceil \log_2(\varepsilon^{-1} + 2) \rceil - 1$.

Proof. The lower bound is given by Proposition 3.1. For the upper bound set

$$n_\varepsilon := \lceil \log_2(\varepsilon^{-1} + 2) \rceil - 2 \qquad \text{and} \qquad \delta := \frac{1}{2^{n_\varepsilon + 2} - 2}.$$

Then the computation tree of Proposition 3.2, applied to n_ε, is a CCT with size $\lceil \log_2(\varepsilon^{-1} + 2) \rceil - 1$. It computes a δ-approximation of a zero of f. This is also an ε-approximation because of $\delta \leq \varepsilon$. Now use Theorem 2.7. ■

We summarize the results for real number oracle machines.

THEOREM 3.4. *Let G be a subspace of functions with $F_{\mathrm{nd}} \subseteq G \subseteq F$ and $\varepsilon > 0$. Consider the set of all real number oracle machines that compute an ε-approximation of a zero of f for any $f \in G$.*

1. *Even if such a machine is allowed to use arbitrary continuous arithmetic operations and arbitrary continuous information operations its computation tree must contain at least $\lceil \log_2(\varepsilon^{-1} + 2) \rceil - 2$ branching nodes in total and at least $\lceil \log_2(\lceil \log_2(\varepsilon^{-1} + 2) \rceil - 1) \rceil$ branching nodes on its longest path.*
2. *There is a such a machine which uses only the arithmetic operations* $\mathrm{ARI}_{\mathrm{abs}}$ *and the information operator* eval *and whose computation tree contains $\lceil \log_2(\varepsilon^{-1} + 2) \rceil - 2$ branching nodes in total and at most $\lceil \log_2(\lceil \log_2(\varepsilon^{-1} + 2) \rceil - 1) \rceil$ branching nodes on any path.*

Proof. The lower bound for the total number of tests in the first part follows from Proposition 3.1 and Theorem 2.7. Since a binary tree with n leaves must have at least one path with at least $\lceil \log_2 n \rceil$ branching nodes, the lower bound for the depth follows, too. For the upper bound one observes that the real number oracle machine of Proposition 3.2, applied to $n_\varepsilon := \lceil \log_2(\varepsilon^{-1} + 2) \rceil - 2$, has all the desired properties. ■

Finally we give a formulation in the terminology for real number oracle machines established by Novak and Woźniakowski (1996). For an algorithm A and an input f for A we denote by $\mathrm{cost}^{\mathrm{TOP}}(A, f)$ the number of tests that are executed during the computation of A on input f. For any subspace $G \subseteq F$, for any set of arithmetic operations ARI, and for any $\varepsilon > 0$ we define

$$\mathrm{comp}^{\mathrm{TOP}}(G, \mathrm{ARI}, \varepsilon) := \min_{\text{alg. } A} \sup_{f \in G} \mathrm{cost}^{\mathrm{TOP}}(A, f)$$

where the minimum is taken over all algorithms A that compute an ε-approximation of a zero of f for any $f \in G$ and that use only the arithmetic operations ARI and the information operator eval (besides comparisons). Let us denote by $\mathrm{ARI}_{\mathrm{con}}$ the set of all continuous arithmetic operations.

THEOREM 3.5. *Let G be a subspace with $F_{\mathrm{nd}} \subseteq G \subseteq F$, let* ARI *be a set of arithmetic operations with* $\mathrm{ARI}_{\mathrm{abs}} \subseteq \mathrm{ARI} \subseteq \mathrm{ARI}_{\mathrm{con}}$, *and let $\varepsilon > 0$. Then*

$$\mathrm{comp}^{\mathrm{TOP}}(G, \mathrm{ARI}, \varepsilon) = \lceil \log_2(\lceil \log_2(\varepsilon^{-1} + 2) \rceil - 1) \rceil.$$

Proof. The number $\mathrm{comp}^{\mathrm{TOP}}(G, \mathrm{ARI}, \varepsilon)$ is the minimal possible depth of a computation tree for the problem, with the given restrictions. The assertion follows immediately from Theorem 3.4. ■

Novak and Woźniakowski (1996) have proved

$$\mathrm{comp}^{\mathrm{TOP}}(G, \mathrm{ARI}_{\mathrm{hol}}, \varepsilon) = \lceil \log_2 \varepsilon^{-1} \rceil - 1,$$

where G is any subspace of $F^* := \{f \in F \mid f(0) < 0 \text{ and } f(1) > 0\}$ containing all linear functions and where $\mathrm{ARI}_{\mathrm{hol}}$ denotes the set of all arithmetic operations that satisfy a Hölder condition on any bounded subset of their domain of definition. Thus, among these algorithms the bisection algorithm is already optimal. But, as we have seen, one can perform exponentially better if one drops the Hölder condition. Note that addition, subtraction, and the absolute value satisfy even a Lipschitz condition on their whole domain $\mathbb{R}^2$, resp. $\mathbb{R}$, while multiplication satisfies a Lipschitz condition at least on any bounded subset of $\mathbb{R}^2$. But division does not satisfy a Hölder condition on each bounded subset of its domain. Just adding the division makes an exponentially better algorithm possible.

3.2. *The Lower Bound*

Here we prove Proposition 3.1. It follows immediately from the next lemma and Lemma 2.5(2). For $0 < l < 1$ we define

$$F_l := \{f \in F_{\mathrm{nd}} \mid f^{-1}\{0\} \text{ is an interval of length } l\}.$$

LEMMA 3.6. *Let $\varepsilon > 0$ and $P_\varepsilon : F_{\mathrm{nd}} \to \mathbb{R}$ be a function such that $P_\varepsilon(f)$ is an ε-approximation of a zero of f for any $f \in F_{\mathrm{nd}}$. Then lev$(P_\varepsilon, f) \geq \log_2(l/\varepsilon + 2) - 1$ for all $f \in F_l$, $l \in (0, 1)$.*

Proof. Let $\varepsilon > 0$ be fixed. For $l \in (0, 1)$ we define $n_l := \lceil \log_2(l/\varepsilon + 2) \rceil - 1$. Hence the number n_l is uniquely determined by

$$(2^{n_l+1} - 2) \cdot \varepsilon \geq l > (2^{n_l} - 2) \cdot \varepsilon.$$

It is always greater than zero. The proof of the assertion "lev$(P_\varepsilon, f) \geq n_l$ for all $f \in F_l$" is done by induction over n_l.

Since P_ε is defined on all functions $f \in F_{\mathrm{nd}}$ we have lev$(P_\varepsilon, f) \geq 1$ for all $f \in F_{\mathrm{nd}}$. Hence the assertion is true for $n_l = 1$.

Now fix an $l \in (0, 1)$ with $n_l \geq 2$ and a function $f \in F_l$. We define

$$l' := \left(\frac{l}{2} - \varepsilon + (2^{n_l-1} - 2) \cdot \varepsilon\right)/2.$$

Then

$$\frac{l}{2} - \varepsilon > l' > (2^{n_l - 1} - 2) \cdot \varepsilon$$

and $n_{l'} = n_l - 1$. At least one of the two endpoints of the interval $I := f^{-1}\{0\}$ has distance greater than or equal to $l/2$ from the point $P_\varepsilon(f)$. Hence the interval I contains a closed subinterval J of length l' with $d(P_\varepsilon(f), J) \geq l/2 - l' > \varepsilon$. Arbitrarily close to f there are functions $g \in F_{l'}$ with $g^{-1}\{0\} = J$. For these functions g we have $d(P_\varepsilon(f), P_\varepsilon(g)) \geq l/2 - l' - \varepsilon > 0$. By the induction hypothesis we know $\mathrm{lev}(P_\varepsilon, f) \geq n_l - 1$ and $\mathrm{lev}(P_\varepsilon, g) \geq n_{l'} = n_l - 1$ for these functions g, hence $f, g \in C_{n_l - 2}(P_\varepsilon)$. We conclude that the restricted function $P_\varepsilon|_{C_{n_l-2}(P_\varepsilon)}$ is discontinuous in f. This finally implies $\mathrm{lev}(P_\varepsilon, f) \geq n_l$. ■

3.3. *The Algorithm*

In this section we prove Proposition 3.2. The following lemma shows that with the help of $\mathrm{ARI_{abs}}$ one can compute maximum, minimum, and the sign function except of the point of discontinuity.

LEMMA 3.7. *For the following functions there are arithmetic expressions over* $\mathrm{ARI_{abs}}$.

1. $\max, \min : \mathbb{R}^n \to \mathbb{R}$ *for each* $n \in \mathbb{N}$,
2. $\mathrm{sig} : \mathbb{R} \setminus \{0\} \to \{0, 1\}$ *with* $\mathrm{sig}(x) := 0$ if $x < 0$, 1 if $x > 0$.

Proof. It is sufficient to show the assertion about max and min for $n = 2$. For $x, y \in \mathbb{R}$ one has $\max\{x, y\} = \frac{1}{2} \cdot (x + y + |x - y|)$ and $\min\{x, y\} = \frac{1}{2} \cdot (x + y - |x - y|)$. For all $x \in \mathbb{R} \setminus \{0\}$ one has $\mathrm{sig}(x) = \frac{1}{2} \cdot (1 + x/|x|)$. ■

We remark that vice versa $|x| = \max\{x, -x\}$ for all $x \in \mathbb{R}$. The algorithm constructed in the following proof will use only the arithmetic operations $\{+, -, *, \max, \min, \mathrm{sig}\}$. Division and taking the absolute value will be used only implicitly via Lemma 3.7.

Proof of Proposition 3.2. We shall construct an algorithm that solves the problem for functions in the subspace

$$F^* := \{f \in C[0, 1] \mid f(0) < 0 \text{ and } f(1) > 0\}$$

of F. Given a function $f \in F$, one can apply the algorithm to the function

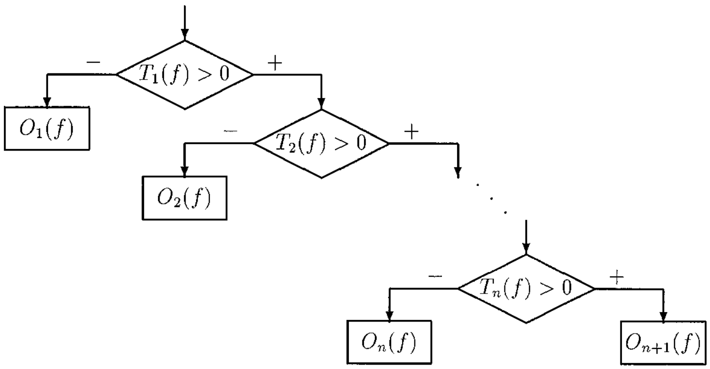

FIG. 3. Nonbalanced algorithm for the approximation of zeros.

$f(1) \cdot f \in F^*$. We construct a computation tree as in Fig. 3. Here $O_1(f), \ldots, O_{n+1}(f)$ are the output values at the end of each computation path. The functions $T_i : F^* \to \mathbb{R}$ for $i = 1, \ldots, n$ are total real-valued functions. The functions $O_i :\subseteq F^* \to (0, 1)$ for $i = 1, \ldots, n + 1$ are partial functions with values in the interval $(0, 1)$. They are defined at least on $\{f \in F^* \mid T_j(f) > 0 \text{ for } 1 \le j < i\}$, for $i = 1, \ldots, n + 1$ respectively. The values $T_i(f)$ as well as the values $O_i(f)$ will be given by arithmetic expressions using only $\mathrm{ARI}_{\mathrm{abs}}$ and the values $f(j/(2^{n+1} - 1))$ for $j = 1, \ldots, 2^{n+1} - 2$.

This computation tree already has size n. In order to obtain a tree with the desired depth $\lceil \log_2(n + 1) \rceil$ one has to construct an equivalent balanced tree of the same size. But the tree can be balanced as in Lemma 2.10 since the functions T_i are total and the resulting logical combinations of tests can also be expressed via $\mathrm{ARI}_{\mathrm{abs}}$ and eval because of

$$a > 0 \wedge b > 0 \Leftrightarrow \min\{a, b\} > 0,$$

$$a > 0 \vee b > 0 \Leftrightarrow \max\{a, b\} > 0,$$

and Lemma 3.7.

Before we give the general definition of the functions T_i and O_i we explain the case $n = 1$. Using one test the bisection algorithm gives a $\frac{1}{4}$-approximation of a zero of a function $f \in F^*$. Our algorithm will give a $\frac{1}{6}$-approximation of a zero. For $n = 1$ the definitions below evaluate to

$$T_1(f) = f(\tfrac{1}{3}) \cdot f(\tfrac{2}{3}),$$

$$O_1(f) = \tfrac{1}{2},$$

$$O_2(f) = \tfrac{1}{6} \cdot \mathrm{sig}(f(\tfrac{2}{3})) + \tfrac{5}{6} \cdot \mathrm{sig}(-f(\tfrac{2}{3})),$$

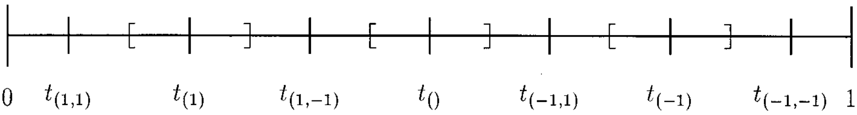

FIG. 4. Subdivision of the interval [0, 1] for $n = 2$.

where sig is the function introduced in Lemma 3.7. If $T_1(f) \leq 0$ then the interval $[\frac{1}{3}, \frac{2}{3}]$ contains a zero and the output $O_1(f) = \frac{1}{2}$ is a $\frac{1}{6}$-approximation of a zero of f. If $T_1(f) > 0$ then $f(\frac{1}{3})$ and $f(\frac{2}{3})$ are either both positive or both negative. If they are positive then the interval $(0, \frac{1}{3})$ contains a zero because of $f(0) < 0$, and the output $O_2(f) = \frac{1}{6}$ is a $\frac{1}{6}$-approximation of a zero. If they are negative then the interval $(\frac{2}{3}, 1)$ contains a zero because of $f(1) > 0$, and the output $O_2(f) = \frac{5}{6}$ is a $\frac{1}{6}$-approximation of a zero. Note that the function sig is evaluated only for nonzero arguments and that it can be computed using only $\mathrm{ARI}_{\mathrm{abs}}$ and no tests.

We come to the definition of the functions T_i and O_i, hence to the precise definition of the algorithm. Let $n \geq 0$ be fixed. One divides the unit interval [0, 1] into $2^{n+1} - 1$ subintervals of the same length. We write this length as $2 \cdot \delta$ with $\delta := 1/(2^{n+2} - 2)$. The midpoints of the intervals are indicated with sign vectors $\overline{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_i) \in \{-1, 1\}^i$ for $i = 0, \ldots, n$,

$$t_{\overline{\varepsilon}} := \frac{1}{2} - \delta \cdot \sum_{j=1}^{i} 2^{n+1-j} \cdot \varepsilon_j,$$

thus, especially $t_{\overline{\varepsilon}} = \frac{1}{2}$ for $\overline{\varepsilon} = (\ ) \in \{-1, 1\}^0$; compare Fig. 4.

The functions $T_i : F^* \to \mathbb{R}$ for $i = 1, \ldots, n$ are defined by

$$T_i(f) := \max\{\min(\mathscr{F}_{\overline{\varepsilon}} \cup \{f(t_{\overline{\varepsilon}} - \delta) \cdot f(t_{\overline{\varepsilon}} + \delta)\}) \mid \overline{\varepsilon} \in \{-1, 1\}^{i-1}\},$$

where

$$\mathscr{F}_{\overline{\varepsilon}} := \{f(t_{(\varepsilon_1, \ldots, \varepsilon_{j-1})} + \delta) \cdot \varepsilon_j \mid 1 \leq j \leq i - 1\}$$

for $\overline{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_{i-1}) \in \{-1, 1\}^{i-1}$, hence especially $\mathscr{F}_{\overline{\varepsilon}} = \emptyset$ for $\overline{\varepsilon} = (\ ) \in \{-1, 1\}^0$. The functions $O_i :\subseteq F^* \to (0, 1)$ for $i = 1, \ldots, n + 1$ are defined by

$$O_i(f) := \sum_{\overline{\varepsilon} \in \{-1,1\}^{i-1}} t_{\overline{\varepsilon}} \cdot \mathrm{sig}(\min(\{1\} \cup \mathscr{F}_{\overline{\varepsilon}})).$$

Here sig : $\mathbb{R} \setminus \{0\} \to \{0, 1\}$ is the function from Lemma 3.7. This completes the description of the algorithm. Note that the functions T_i are total, as needed for the balancing.

Before proving correctness we sketch the idea of the algorithm. The basic idea is that before executing the $(j + 1)$th test the algorithm has implicitly determined a sign vector $\overline{\varepsilon} \in \{-1, 1\}^j$ of length j such that the function f has different signs in the endpoints of the interval with midpoint $t_{\overline{\varepsilon}}$ and of length $2\delta \cdot (2^{n+1-j} - 1)$. Then the $(j + 1)$th test checks whether $f(t_{\overline{\varepsilon}} - \delta) \cdot f(t_{\overline{\varepsilon}} + \delta) \leq 0$ or not. If this is true then the subinterval $[t_{\overline{\varepsilon}} - \delta, t_{\overline{\varepsilon}} + \delta]$ in the middle must contain a zero, and the algorithm produces $t_{\overline{\varepsilon}}$ as output. If this is not true then the function must have different signs either in the endpoints of the remaining left part or in the endpoints of the remaining right part. In fact, such a sign vector has not really been computed by the algorithm, but there is only one sign vector $\overline{\varepsilon} = (\varepsilon_1, . . . , \varepsilon_j)$ such that all subvectors $(\varepsilon_1, . . . , \varepsilon_i)$ for $0 \leq i \leq j$ have the same property. This makes it possible to realize this test by a total function T_{j+1}.

We still have to prove that the algorithm is correct. Fix a function $f \in F^*$. First we prove the following assertion for $0 \leq j \leq n$ by induction:

If $T_k(f) > 0$ for all k with $1 \leq k \leq j$, then

1. there is a unique $\overline{\varepsilon'} \in \{-1, 1\}^j$ with $\min(\{1\} \cup \mathscr{F}_{\overline{\varepsilon'}}) > 0$ and $\min(\{1\} \cup \mathscr{F}_{\overline{\varepsilon}}) < 0$ for all $\overline{\varepsilon} \in \{-1, 1\}^j \setminus \{\overline{\varepsilon'}\}$,
2. this $\overline{\varepsilon'}$ satisfies $f(t_{\overline{\varepsilon'}} - \delta \cdot (2^{n+1-j} - 1)) < 0$ and $f(t_{\overline{\varepsilon'}} + \delta \cdot (2^{n+1-j} - 1)) > 0$.

For $j = 0$ the assertion is correct since by assumption $f(0) < 0$ and $f(1) > 0$. Let us assume $1 \leq j \leq n$ and $T_k(f) > 0$ for $1 \leq k \leq j$. By the induction hypothesis there is a unique sign vector $\overline{\varepsilon''} = (\varepsilon''_1, . . . , \varepsilon''_{j-1}) \in \{-1, 1\}^{j-1}$ with

1. $\min(\{1\} \cup \mathscr{F}_{\overline{\varepsilon''}}) > 0$ and $\min(\{1\} \cup \mathscr{F}_{\overline{\varepsilon}}) < 0$ for all $\overline{\varepsilon} \in \{-1, 1\}^{j-1} \setminus \{\overline{\varepsilon''}\}$, and
2. $f(t_{\overline{\varepsilon''}} - \delta \cdot (2^{n+1-(j-1)} - 1)) < 0$ and $f(t_{\overline{\varepsilon''}} + \delta \cdot (2^{n+1-(j-1)} - 1)) > 0$.

We have

$$\mathscr{F}_{\overline{\varepsilon}} = \mathscr{F}_{(\varepsilon_1,\ldots,\varepsilon_{j-1})} \cup \{f(t_{(\varepsilon_1,\ldots,\varepsilon_{j-1})} + \delta) \cdot \varepsilon_j\}$$

for arbitrary $\overline{\varepsilon} = (\varepsilon_1, . . . , \varepsilon_j) \in \{-1, 1\}^j$. If $(\varepsilon_1, . . . , \varepsilon_{j-1}) \neq \overline{\varepsilon''}$ then the first part of the induction hypothesis implies $\min(\{1\} \cup \mathscr{F}_{\overline{\varepsilon}}) < 0$. Furthermore, because of $T_j(f) > 0$ it implies

$$f(t_{\overline{\varepsilon''}} - \delta) \cdot f(t_{\overline{\varepsilon''}} + \delta) > 0.$$

Using the common sign function sgn: $\mathbb{R} \to \{-1, 0, 1\}$ with $\operatorname{sgn}(x) = -1$ for $x < 0$, $\operatorname{sgn}(0) = 0$, $\operatorname{sgn}(x) = 1$ for $x > 0$, we define

$$\overline{\varepsilon'} = (\varepsilon'_1, \ldots, \varepsilon'_j) := (\varepsilon''_1, \ldots, \varepsilon''_{j-1}, \operatorname{sgn}(f(t_{\overline{\varepsilon''}} + \delta))).$$

This is the unique sign vector demanded in the first part of the inductive assertion. One computes

$$\begin{aligned} t_{\overline{\varepsilon'}} - \delta \cdot (2^{n+1-j} - 1) &= t_{\overline{\varepsilon''}} - \delta \cdot 2^{n+1-j} \cdot \varepsilon'_j - \delta \cdot (2^{n+1-j} - 1) \\ &= \begin{cases} t_{\overline{\varepsilon''}} - \delta \cdot (2^{n+1-(j-1)} - 1) & \text{if } \varepsilon'_j = 1 \\ t_{\overline{\varepsilon''}} + \delta & \text{if } \varepsilon'_j = -1 \end{cases} \end{aligned}$$

and in the same way

$$t_{\overline{\varepsilon'}} + \delta \cdot (2^{n+1-j} - 1) = \begin{cases} t_{\overline{\varepsilon''}} - \delta & \text{if } \varepsilon'_j = 1 \\ t_{\overline{\varepsilon''}} + \delta \cdot (2^{n+1-(j-1)} - 1) & \text{if } \varepsilon'_j = -1. \end{cases}$$

With the second part of the induction hypothesis and with

$$\operatorname{sgn}(f(t_{\overline{\varepsilon''}} - \delta)) = \operatorname{sgn}(f(t_{\overline{\varepsilon''}} + \delta)) = \varepsilon'_j$$

one obtains the second part of the inductive assertion, too. Hence both parts of the inductive assertion are proved.

The first part of this assertion implies that the function O_i is defined on the set $\{f \in F^* \mid (\forall j < i)\ T_j(f) > 0\}$, for $i = 1, \ldots, n + 1$.

Since the functions T_i are total, the number

$$i_f := \max\{i \geq 1 \mid (\forall j < i)\ T_j(f) > 0\}$$

is well defined. By the first part of the assertion above there is a sign vector $\overline{\varepsilon_f} \in \{-1, 1\}^{i_f - 1}$ with $\min(\{1\} \cup \mathscr{F}_{\overline{\varepsilon_f}}) > 0$ and $\min(\{1\} \cup \mathscr{F}_{\overline{\varepsilon}}) < 0$ for all $\overline{\varepsilon} \in \{-1, 1\}^{i_f - 1} \setminus \{\overline{\varepsilon_f}\}$. Hence, on input $f \in F^*$ the output of the algorithm is the point

$$O_{i_f}(f) = t_{\overline{\varepsilon_f}}.$$

In the case $i_f = n + 1$ the second part of the assertion above, applied to $j = n$, gives

$$f(t_{\overline{\varepsilon}_f} - \delta) < 0 \qquad \text{and} \qquad f(t_{\overline{\varepsilon}_f} + \delta) > 0.$$

Hence, in this case the point $t_{\overline{\varepsilon}_f}$ must be a δ-approximation of a zero of f. In the case $i_f \leq n$ we have $T_{i_f}(f) \leq 0$. Because of $\min(\{1\} \cup \mathscr{F}_{\overline{\varepsilon}_f}) > 0$ we obtain

$$f(t_{\overline{\varepsilon}_f} - \delta) \cdot f(t_{\overline{\varepsilon}_f} + \delta) \leq 0.$$

Hence, the point $t_{\overline{\varepsilon}_f}$ is a δ-approximation of a zero of f in this case, too. This proves the correctness of the algorithm and finishes the proof of Proposition 3.2. ■

For numbers n of the form $n = 2^m - 1$ the computation tree constructed in the proof is perfectly balanced and has m branching nodes on each path. This means that using not more than m comparisons one can already compute a $1/(2^{2^m+1} - 2)$-approximation of a zero of an arbitrary function $f \in F$, using only $\mathrm{ARI}_{\mathrm{abs}}$ and function values. This is exponentially better than bisection. A practical drawback of the algorithm is certainly the large number of function evaluations. During one computation it needs $\Theta(\varepsilon^{-1})$ function evaluations in the worst case while the bisection algorithm requires only $\Theta(\log_2(\varepsilon^{-1}))$ function values per computation.

3.4. *Increasing Functions*

We have seen that the topological complexity of zero-finding is the same for the full class F and for the subclass F_{nd} of nondecreasing functions, whether one allows arbitrary continuous operations or just the function evaluation eval and the set $\mathrm{ARI}_{\mathrm{abs}}$. It is not zero. But for the slightly smaller class of increasing functions

$$F_{\mathrm{inc}} := \{f \in C[0,1] \mid f \text{ is increasing and } f(0) < 0, f(1) > 0\}$$

the topological complexity drops to zero. These functions have a unique zero and the function

$$P : F_{\mathrm{inc}} \to (0,1), \qquad P(f) \text{ is the zero of } f,$$

is obviously continuous. Novak and Woźniakowski (1996) have given an algorithm without branching nodes which computes an ε-approximation of the zero of f for any $f \in F_{\mathrm{inc}}$ using only eval and the extended set of arithmetic operations $\mathrm{ARI}_{\mathrm{ext}} := \{+, -, *, /, \exp, \log\}$. This proves $\mathrm{comp}^{\mathrm{TOP}}(F_{\mathrm{inc}}, \mathrm{ARI}_{\mathrm{ext}}, \varepsilon) = \mathrm{comp}^{\mathrm{TOP}}(F_{\mathrm{inc}}, \mathrm{ARI}_{\mathrm{con}}, \varepsilon) = 0$. We show that there is a simple algorithm for this problem that uses the absolute value instead of the functions exp and log.

THEOREM 3.8. *For any $\varepsilon > 0$,*

$$\text{comp}^{\text{TOP}}(F_{\text{inc}}, \text{ARI}_{\text{abs}}, \varepsilon) = 0.$$

Hence, also in this case the topological complexity is the same for arbitrary continuous operations and for eval and ARI_{abs}.

Proof. Set $n := \max\{2, \lceil \varepsilon^{-1} \rceil\}$. We define an algorithm P_ε which computes an ε-approximation of a zero of f for any $f \in F_{\text{inc}}$ without comparisons as follows. Compute

$$x_i := i/n \qquad \text{and} \qquad z_i := f(x_i)$$

for $i = 0, \ldots, n$ and

$$r_i := \max\{0, -z_{i-1} \cdot z_{i+1}\}$$

for $i = 1, \ldots, n-1$. Then set

$$P_\varepsilon(f) := \frac{\sum_{i=1}^{n-1} x_i \cdot r_i}{\sum_{i=1}^{n-1} r_i}.$$

Note that by Lemma 3.7 one can compute the maximum using ARI_{abs}.

The function f is increasing. Hence it has a unique zero $x^*(f)$. We distinguish four cases. If $x^*(f) \leq 1/n$ then $r_1 > 0$ and $r_j = 0$ for all $j > 1$, hence $P_\varepsilon(f) = x_1 = 1/n$. If $x^*(f) \geq (n-1)/n$ then $r_{n-1} > 0$ and $r_j = 0$ for all $j < n-1$, hence $P_\varepsilon(f) = x_{n-1} = (n-1)/n$. If $x^*(f) = x_i$ for some $i \in \{1, \ldots, n-1\}$ then $r_i > 0$ and $r_j = 0$ for all $j \neq i$, hence $P_\varepsilon(f) = x_i = x^*(f)$. Finally, if $x^*(f) \in (x_i, x_{i+1})$ for some $i \in \{1, \ldots, n-2\}$ then $r_i > 0$ and $r_{i+1} > 0$ but $r_j = 0$ for all $j \notin \{i, i+1\}$, hence $P_\varepsilon(f) = (x_i \cdot r_i + x_{i+1} \cdot r_{i+1})/(r_i + r_{i+1}) \in (x_i, x_{i+1})$. In each case we have $|P_\varepsilon(f) - x^*(f)| < 1/n \leq \varepsilon$. ■

The algorithm in the proof uses $\mathcal{O}(\varepsilon^{-1})$ nonadaptive function evaluations, i.e., the points at which the function f is evaluated are fixed and do not depend on f. There is also an algorithm which needs only $\mathcal{O}(\log_2(\varepsilon^{-1}))$ adaptive function evaluations (and no tests). It is described in the second proof of Theorem 3.8.

Second Proof of Theorem 3.8. The idea is to use the algorithm above for $n = 3$ ("trisection") and to iterate it sufficiently often with variable input interval and output interval.

Set $k := \max\{0, \lceil -\log_{3/2}(2\varepsilon) \rceil\}$. For $f \in F_{\text{inc}}$ the algorithm iterates the following loop k times, starting with $(a, b) := (0, 1)$. Finally the midpoint of the last interval (a, b) is the output.

```
begin {loop}
    $c := (b - a)/3$;
    for $j = 0, \ldots, 3$ do begin $x_j := a + j \cdot c$;
                                   $z_j := f(x_j)$ end;
    for $j = 1, 2$ do $r_j := \max\{0, -z_{j+1} \cdot z_{j-1}\}$;
    $x := (x_1 r_1 + x_2 r_2)/(r_1 + r_2)$;
    $a := x - c$; $b := x + c$
end {loop}
```

At the beginning and after each loop the interval (a, b) contains the zero of f. Hence at the end, after k loops, the midpoint x of the last interval (a, b) has distance less than $\frac{1}{2} \cdot (\frac{2}{3})^k \leq \varepsilon$ from the zero of f. ■

We have seen that the topological complexity of zero-finding for continuous functions on the unit interval depends on the set of considered functions. Similarly, in computable analysis different topological obstacles occur in the determination of zeros for different classes of functions, see Weihrauch (1995).

Acknowledgments

The second part of the paper was inspired by the recent paper of Novak and Woźniakowski (1996). I thank them for sending me an early version. Furthermore, I thank E. Novak for useful comments.

References

Blum, L., Shub, M., and Smale, S. (1989), On a theory of computation and complexity over the real numbers: NP completeness, recursive functions and universal machines, *Bull. Amer. Math. Soc.* (*N.S.*) **21,** 1–46.

Hausdorff, F. (1935), "Mengenlehre," 3rd ed., Academic Press, New York.

Hertling, P. (1996), Unstetigkeitsgrade von Funktionen in der effektiven Analysis, preprint.

Hertling, P., and Weihrauch, K. (1994), Levels of degeneracy and exact lower complexity bounds for geometric algorithms, *in* "Proceedings of the Sixth Canadian Conference on Computational Geometry, Saskatoon," pp. 237–242.

Novak, E. (1995), The real number model in numerical analysis, *J. Complexity* **11,** 57–73.

Novak, E., and Woźniakowski, H. (1996), Topological complexity of zero-finding, *J. Complexity* **12,** 380–400.

Smale, S. (1987), On the topology of algorithms, I, *J. Complexity* **3,** 81–89.

VASSILIEV, V. A. (1992), "Complements of Discriminants of Smooth Maps: Topology and Applications," Transl of Math. Monographs, Vol. 98, Amer. Math. Soc., Providence, RI.

WEIHRAUCH, K. (1995), "A Simple Introduction to Computable Analysis," Informatik Bericht, Vol. 171, 2nd ed., FernUniversität Hagen.

YAP, C.-K. (1990), Symbolic treatment of geometric degeneracies, *J. Symbolic Comput.* **10,** 349–370.