Circuits versus Trees in Algebraic Complexity

Pascal Koiran

LIP, Ecole Normale Supérieure de Lyon
46 allée d'Italie, 69364 Lyon Cedex 07, France
http://www.ens-lyon.fr/~koiran

Abstract. This survey is devoted to some aspects of the "P = NP ?" problem over the real numbers and more general algebraic structures. We argue that given a structure M, it is important to find out whether NP_M problems can be solved by polynomial depth computation trees, and if so whether these trees can be efficiently simulated by circuits. Point location, a problem of computational geometry, comes into play in the study of these questions for several structures of interest.

1 Introduction

In algebraic complexity one measures the complexity of an algorithm by the number of basic operations performed during a computation. The basic operations are usually arithmetic operations and comparisons, but sometimes transcendental functions are also allowed [21–23, 26]. Even when the set of basic operations has been fixed, the complexity of a problem depends on the particular model of computation considered. The two main categories of interest for this paper are circuits and trees. In section 2 and 3 we present a general framework for studying these questions, in the spirit of Poizat's theory of computation over arbitrary structures [16, 35]. The focus is therefore on superpolynomial lower bounds for decision problems, and in particular for NP_M-complete problems. This line of research was initiated by Blum, Shub and Smale [5]; the main emphasis of their paper was on the case where M is a ring (see [4] for a recent account).

We will ignore completely the large body of work on (sub)polynomial lower bounds, which has been an active area of research for decades ([7] is a comprehensive text on this topic). A consequence of our higher ambition is that we actually have very few lower bounds to present. The main result of this type, presented in Theorem 2, was obtained by Meer [29] for the reals with addition and equality. The transfer theorems of section 4 show that there may be good reasons for this relative scarcity of definitive results. The bright side of this state of affairs is that there are plenty of difficult open problems to capture the attention of present and future researchers.

As explained in section 5, point location, a problem of computational geometry, plays an important role in the proofs of two transfer theorems. The branching complexity of point location in arrangements of real or complex hypersurfaces is also discussed in that section. The upper bound of Theorem 8 on point location in arrangements of complex hypersurfaces seems to be new; in the real case, this is a recent result of Grigoriev [20]. Finally, we show in section 6 that a solution to the "computation tree alternative" would lead to a much better understanding of the "P = NP ?" question over the real and complex numbers.

2 Computation Models

2.1 Arbitrary Structures

We first recall some elementary definitions from logic, which should be familiar to most of our readers. By "structure", we mean a set M equipped with a finite set of functions $f_i : M^{n_i} \to M$ and relations $r_i \subseteq M^{m_i}$. A function of arity 0 is called a *constant*. We always assume that our structure contains the equality relation.

Terms are built from the basic functions of M by composition. More precisely, we have the following inductive definition:

(i) Variables and elements of M are terms of depth 0;
(ii) A term of depth $d \geq 1$ is of the form $f_i(t_1, \ldots, t_{n_i})$, where f_i is a function of M and $t_1, \ldots, t_{n_i}$ are terms of maximal depth $d-1$.

A term in which n distinct variables $x_1, \ldots, x_n$ occur computes a function from M^n to M.

An *atomic formula* is of the form $r_i(t_1, \ldots, t_{m_i})$ where $t_1, \ldots, t_{m_i}$ are terms. A quantifier-free formula is a boolean combination of atomic formulas. To be completely precise, one can give an inductive definition:

(i) Atomic formulas are formulas of depth 0.
(ii) If F is a formula of depth $d-1$, $\neg F$ is a formula of depth d.
(iii) If $\max(\text{depth}(F), \text{depth}(G)) = d-1$, $F \vee G$ and $F \wedge G$ are formulas of depth d.

A formula in which n distinct variables $x_1, \ldots, x_n$ occur defines a subset of M^n. The elements of M occurring in terms or formulas (or in trees, circuits, etc...) are called *parameters*.

Example 1. If M is a field, terms of $(M, +, \times, =)$ represent polynomials; the sets defined by quantifier-free formulas are called *constructible*. If M is an ordered field, the definable sets of $(M, +, \times, \leq)$ are by definition the semi-algebraic sets.

An existential formula is of the form $\exists x_1\, \exists x_2 \cdots \exists x_n\, F$, where F is a quantifier-free formula. M is said to admit quantifier elimination if every existential formula is equivalent to a quantifier-free formula (of course, as long as complexity issues are not taken into account, it is sufficient to consider the case $n = 1$). As explained in section 4, this notion plays an important role in the study of the "$\mathrm{P}_M = \mathrm{NP}_M$?" problem.

2.2 Trees

The simplest and most powerful computation model that we shall consider is the *branching tree.* A branching tree with n input variables $x_1, \ldots, x_n$ recognizes a subset of M^n in the following way. Each internal node g is labeled by some atomic formula $F_g(x_1, \ldots, x_n)$ of M, and has two children. If the input satisfies F_g we go left, otherwise we go right. Leaves are labeled *accept* or *reject.* Alternatively, leaves could be labeled by terms of M. The tree would then compute a function from M^n to M. As with other tree models, complexity is measured by depth, i.e., the branching complexity of a subset of M^n is the smallest depth of a branching tree that recognizes it.

The branching tree model is obviously not very realistic since any term, no matter how complex, can be evaluated for free. Consequently, this model is perhaps more suitable for proving lower bounds than upper bounds (some upper bounds are nevertheless presented in section 5.2). Lower bounds for approximating roots of complex polynomials in the structure $(\mathbb{R}, +, -, \times, /, \leq)$ can be found in [39, 41]. In this context, the terms *topological complexity* and *topological decision tree* have been used instead of *branching complexity* and *branching tree.*

The *computation tree* is a more realistic model in which the complexity of terms is taken into account. In addition to branching nodes, a computation tree has unary computation nodes. A computation node g computes a term of the form $f_i(t_1, \ldots, t_{n_i})$ where f_i is a function of M and $t_1, \ldots, t_{n_i}$ are variables, parameters from M or terms computed by computation nodes located between the current node g and the root of the tree. The branching nodes of a computation tree are labeled by atomic formulas of the form $r_i(t_1, \ldots, t_{n_i})$ where r_i is a relation of M and again $t_1, \ldots, t_{n_i}$ are variables, parameters from M or terms computed by computation nodes located between the current node and the root of the tree.

Note that a computation tree is nothing but a special way of representing a formula, and therefore the subsets of M^n accepted by computation trees (or by branching trees, or by circuits as defined in section 2.3) are simply the definable sets. One may again argue that the computation tree model is mostly suitable for lower bounds as it is still too powerful. Note for instance than in the standard structure $M = \{0, 1\}$ any subset of M^n can be recognized in depth n. As far as we are concerned, the "fully realistic" model is the circuit model of section 2.3. Computation trees are nevertheless suitable for upper bounds if preprocessing is allowed. That is, if we have to recognize very quickly a subset X of M^n for some fixed value of n, we might first spend a lot of time (and space) to construct a computation tree recognizing X. The depth of this tree is a reasonable measure of the number of elementary operations needed to decide whether an element of M^n belongs to X. Preprocessing is a quite common technique in computational geometry, see for instance [17].

2.3 Circuits

The input gates of a circuit are labeled by variables or parameters from M. There are several types of computation gates in a circuit over M:

1. For each function f_i of M, gates of type f_i apply this function to their n_i inputs.
2. For each relation r_i of M, gates of type r_i apply the characteristic function of r_i to their inputs. For this definition to make sense, we assume that M contains two distinguished elements called 0 and 1.
3. Finally, selection gates compute a function $s(x, y, z)$ of their three inputs such that $s(0, y, z) = y$ and $s(1, y, z) = z$. The behaviour of s on an input (x, y, z) with $x \notin \{0, 1\}$ is not important. We shall assume that $s(x, y, z)$ is equal in this case to some fixed term $t(x, y, z)$ of M.

A circuit is *parameter-free* if it uses only the parameters 0 and 1. Strangely enough, selection gates already appear in [42], in a context where they are not really needed. Indeed, in any field the term $t(x, y, z) = xz + (1-x)y$ is a selection function.

The number of computation gates in a circuit is its *size*. A circuit C with n input variables and m output gates computes a function from M^n to M^m. We are mostly interested in the case $m = 1$. For such a circuit, the set of accepted inputs is by definition the set of inputs $(x_1, \ldots, x_n) \in M^n$ such that $C(x_1, \ldots, x_n) = 1$. As mentioned before, circuits with n input variables accept exactly the definable subsets of M^n.

Computation trees are at least as powerful as circuits since a circuit of size s can be simulated by a computation tree of depth $O(s)$. As we shall see in section 4, the converse is not so clear.

3 Complexity Classes

Our main complexity classes, such as P and NP, will be defined in terms of circuits. Formally, a problem is simply a subset of $M^\infty = \bigcup_{n \geq 1} M^n$. We first define the class $\mathbb{P}_M$ of non-uniform polynomial-time problems. A problem $X \subseteq M^\infty$ is in $\mathbb{P}_M$ if there exists parameters $\alpha_1, \ldots, \alpha_p \in M$, a polynomial $p(n)$ and a family of parameter-free circuits $(C_n)_{n \geq 1}$ where C_n has $n + p$ inputs, is of size at most $p(n)$ and satisfies the following condition:

$$\forall x \in M^n \; x \in X \Leftrightarrow C_n(\alpha_1 \ldots \alpha_p, x_1, \ldots, x_n) = 1 \tag{1}$$

$\mathrm{N}\mathbb{P}_M$ is the non-deterministic version of $\mathbb{P}_M$. That is, a problem X is in $\mathrm{N}\mathbb{P}_M$ if there exists a polynomial $q(n)$ and a problem $Y \in \mathbb{P}_M$ such that for any $n \geq 1$ and any $x \in M^n$,

$$x \in X \Leftrightarrow \exists y \in M^{q(n)} \; \langle x, y \rangle \in Y. \tag{2}$$

Several equivalent choices for the pairing function $\langle ., . \rangle : M^\infty \times M^\infty \to M^\infty$ are possible, for instance:

$$\langle (x_1, \ldots, x_n), (y_1, \ldots, y_m) \rangle = (0, x_1, 0, x_2, \ldots, 0, x_n, 1, y_1, 1, y_2, \ldots, 1, y_m).$$

One could also choose q so that the map $n \mapsto n + q(n)$ is injective, and simply concatenate x and y.

A problem X is in the class P_M of polynomial-time problems if $X \in \mathbb{P}_M$ and the corresponding circuit family (C_n) in (1) is uniform in the following sense: there exists a (classical) Turing machine which on input n constructs C_n in time polynomial in n. This definition makes sense since a parameter-free circuit is a purely boolean object. To be completely precise one should specify how circuits are encoded in binary words [35]; there is no significant difference with the classical case [1]. This complexity class can also be defined with Turing machines over M instead of circuits [16, 35]. In the case where M is a ring, these Turing machines are equivalent to Blum-Shub-Smale machines [5].

The class NP_M of non-deterministic polynomial time problems is obtained from P_M in the same way as $\mathbb{NP}_M$ is obtained from $\mathbb{P}_M$: just replace the condition $Y \in \mathbb{P}_M$ in (2) by $Y \in \mathrm{P}_M$.

Example 2. For the standard structure $M = \{0, 1\}$ the usual complexity classes are recovered: $\mathrm{P}_M = \mathrm{P}$, $\mathrm{NP}_M = \mathrm{NP}$, $\mathbb{P}_M = \mathrm{P/poly}$, $\mathbb{NP}_M = \mathrm{NP/poly}$.

For $M = (\mathbb{R}, +, -, \times, \leq)$ or $M = (\mathbb{R}, +, -, \leq)$, $\mathrm{P}_M = \mathbb{P}_M$ and $\mathrm{NP}_M = \mathbb{NP}_M$ since circuit families can be encoded in parameters.

One can also define the class $\mathbb{P}^0_M$ of parameter-free non-uniform polynomial time problems (set $p = 0$ in (1)), and from this class we obtain the classes $\mathbb{NP}^0_M$, P^0_M and NP^0_M, which are the parameter-free versions of $\mathbb{NP}_M$, P_M and NP_M. These parameter-free classes appear in section 6.

The problems "$\mathbb{P}_M = \mathbb{NP}_M$?" and "$\mathrm{P}_M = \mathrm{NP}_M$?" are open for most structures of interest (but for any structure M, the latter equality implies the former). The main open problem in this general theory is whether there exists a structure M satisfying $\mathrm{P}_M = \mathrm{NP}_M$.

There is more than P and NP in the world of structural complexity, and lots of classical complexity classes can be redefined in our general framework. For instance, the reader can easily imagine how the k-th levels Σ^k_M and Π^k_M of the uniform polynomial hierarchy are defined [4, 35] (replace the existential quantifiers in (2) by k alternating quantifier blocks). Of course, there is also a non-uniform polynomial hierarchy.

One of the main differences with the classical theory comes from the role of space: as shown by Michaux [33], a Turing machine over $(\mathbb{R}, +, -, \times, \leq)$ can perform any computation in constant work space. The classical definition of PSPACE is therefore of little use in our general framework. We will instead work with the class PAR_M of parallel polynomial time problems. It is again easier to first define as in [8] a non-uniform version of this complexity class: a problem X is $\mathbb{PAR}_M$ if there exist parameters $\alpha_1, \ldots, \alpha_p \in M$, a polynomial $p(n)$ and a family of parameter-free circuits $(C_n(x_1, \ldots, x_n, y_1, \ldots, y_p))$ such that $C_n(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_p)$ solves X for inputs in M^n, and is of depth at most $p(n)$. We say that X is PAR_M if there exists a Turing machine which, on input n, construct C_n in work space polynomially bounded in n.

Example 3. For $M = \{0, 1\}$, $\mathrm{PAR}_M = \mathrm{PSPACE}$; for $M = (\mathbb{R}, +, -, \times, \leq)$, our definition is equivalent to those of [4] and [11].

4 Is P equal to NP ?

If one wishes to investigate the "$\mathrm{P}_M = \mathrm{NP}_M$?" in some structure M, the first thing to do is to find out whether M admits quantifier elimination. Indeed, if M does not admit quantifier elimination NP_M problems cannot be solved by circuits (or equivalently by quantifier-free formulas) of *any* size. One should therefore consult one's favourite model theory book (e.g. [24, 34]) for a list of structures that admit or do not admit quantifier elimination. For instance, the following structures admit quantifier elimination:

(i) vector spaces over $\mathbb{Q}$, e.g., $\mathbb{R}_{vs} = (\mathbb{R}, +, -, =)$;
(ii) ordered vector spaces over $\mathbb{Q}$, e.g., $\mathbb{R}_{ovs} = (\mathbb{R}, +, -, \leq)$;
(iii) algebraically closed fields of any characteristic, e.g., $\mathbb{C} = (\mathbb{C}, +, -, \times, =)$;
(iv) real closed fields, e.g., $\mathbb{R} = (\mathbb{R}, +, -, \times, \leq)$;
(v) differentially closed fields of characteristic zero (of which we can give no natural example).

The reals with exponentiation $(\mathbb{R}, +, -, \times, \exp, \leq)$ and the integers $(\mathbb{N}, +, -, \times, \leq)$ do not admit quantifier elimination. However, the "$\mathrm{NP}_M = \mathrm{coNP}_M$?" question makes sense in the former structure since it is model-complete (i.e., every existential formula is equivalent to a universal formula).

For structures (i) through (iv), elimination of quantifiers can be performed in a relatively efficient manner in the sense that $\mathrm{NP}_M \subseteq \mathrm{PAR}_M$. Proofs of this fact for $M = \mathbb{R}_{vs}$ and $M = \mathbb{R}_{ovs}$ can be found in [25] (these two structures satisfy the stronger property that $\mathrm{NP}_M = \mathrm{BNP}_M$, i.e., existential quantification over M is polynomially equivalent to existential quantification over $\{0, 1\}$). For algebraically closed fields one may consult [10, 13] and for real closed fields [2, 18, 37]. The complexity theory of differentially closed fields is not so well understood. One may cite a triply exponential quantifier elimination algorithm [19] and a study of the "P = NP ?" problem for these structures in [36].

Quantifier elimination is even more closely connected to the "$\mathrm{P}_M = \mathrm{NP}_M$?" question than explained at the beginning of this section. Roughly speaking, $\mathrm{P}_M = \mathrm{NP}_M$ means that existential quantifiers can be eliminated in polynomial time, if we represent the eliminating formula by a circuit:

Theorem 1. $\mathrm{P}_M = \mathrm{NP}_M$ *if and only if there exist parameters* $\alpha_1, \ldots, \alpha_k$ *of* M *and a polynomial time algorithm (in the classical sense) which, given a parameter-free existential formula* $\exists y_1 \cdots \exists y_p\ F(x_1, \cdots, x_n, y_1, \ldots, y_p)$*, outputs a circuit* C *over* M *such that* $C(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_k)$ *is equivalent to the existential formula.*

The "$\mathrm{P}_M = \mathrm{NP}_M$?" question can therefore be viewed as a purely classical question, that is, a question about algorithms over the structure $\{0, 1\}$. This theorem remains true if we replace the classical elimination algorithm by an algorithm over M, which is the way it is stated in [35].

We have precious few examples of structures that admit quantifier elimination but where P_M is provably different from NP_M. The main example is due to Klaus Meer [29]:

Theorem 2. $\mathrm{P}_{\mathbb{R}_{vs}} \neq \mathrm{NP}_{\mathbb{R}_{vs}}$.

Another example is the structure of arborescent dictionaries constructed in [16] (see also [35]). Meer's proof of Theorem 2 is based on a multidimensional version of the knapsack problem. The proof given below is based on Shub and Smale's [38] Twenty Questions (TQ) problem. An input $(x_1, \ldots, x_n)$ is in TQ if x_1 is an integer between 0 and $2^n - 1$ (x_1 is therefore the only truly "numerical" input; the only role of $x_2, \ldots, x_n$ is to specify the value of n).

Twenty Questions is in $\mathrm{NP}_{\mathbb{R}_{vs}}$ since $(x_1, \ldots, x_n) \in \mathrm{TQ}$ if and only if

$$\exists u_1, \ldots, u_n \in \{0,1\} \; x_1 = \sum_{i=1}^{n} 2^{i-1} u_i.$$

However, TQ is not only outside $\mathrm{P}_{\mathbb{R}_{vs}}$ but its branching complexity is exactly equal to 2^n. Indeed, this problem can obviously be solved in depth 2^n (just perform the 2^n tests "$x_1 = 0$?", …,"$x_n = 2^n - 1$?" sequentially). For the converse, let T_n be a branching tree of depth d which solves Twenty Questions for inputs of the form $x_2 = 0, \ldots, x_n = 0$. This tree has a single real-valued input x_1 and recognizes a finite subset of $\mathbb{R}$, of cardinality 2^n. We claim that $d \geq 2^n$. Each internal node of T_n performs a test of the form "$l(x_1) = 0$?" where l is an affine function; we may assume without loss of generality that l is not identically 0. The *generic path* of T_n is obtained by answering *no* to all these tests. Observe that inputs which follow the generic path are rejected since T_n recognizes a finite subset of $\mathbb{R}$. Accepted inputs must therefore satisfy one of the test performed along the canonical path, which proves the claim, and Theorem 2.

One might be led by the simplicity of this proof to the belief that other results of this type should not be too difficult to obtain. For instance, one could try to:

(i) separate higher levels of the polynomial hierarchy over $\mathbb{R}_{vs}$;
(ii) obtain separation results for richer structures than $\mathbb{R}_{vs}$. For instance, is $\mathrm{P}_{\mathbb{R}_{ovs}}$ different from $\mathrm{NP}_{\mathbb{R}_{ovs}}$?

It turns out that both questions are quite difficult, however. In the first direction, it is actually possible to separate the first few levels of the polynomial hierarchy [15]: a variation on the proof of Theorem 2 shows that Twenty Questions is not in $\mathrm{coNP}_{\mathbb{R}_{vs}}$, which separates $\mathrm{NP}_{\mathbb{R}_{vs}}$ from $\mathrm{coNP}_{\mathbb{R}_{vs}}$; and a similar problem can be used to separate $\Sigma^2_{\mathbb{R}_{vs}} \cap \Pi^2_{\mathbb{R}_{vs}}$ from $\Sigma^1_{\mathbb{R}_{vs}} \cup \Pi^1_{\mathbb{R}_{vs}}$. Separating higher levels is essentially "impossible", as shown by a transfer theorem of [15]:

Theorem 3. *For all $k \geq 0$:*

$$\mathrm{PH} = \Sigma^k \Rightarrow \mathrm{PH}_{\mathbb{R}_{vs}} = \Sigma^{k+2}_{\mathbb{R}_{vs}}.$$

We showed in the same paper that parallel polynomial time cannot be separated from the polynomial hierarchy:

Theorem 4. $\mathrm{P} = \mathrm{PSPACE} \Rightarrow \mathrm{PAR}_{\mathbb{R}_{vs}} = \Sigma^2_{\mathbb{R}_{vs}} \cap \Pi^2_{\mathbb{R}_{vs}}$

and that P cannot be separated from NP $\cap$ coNP:

Theorem 5. $\mathrm{P} = \mathrm{NP} \Rightarrow \mathrm{P}_{\mathbb{R}_{vs}} = \Sigma^1_{\mathbb{R}_{vs}} \cap \Pi^1_{\mathbb{R}_{vs}}$.

Bourgade has extended theorems 3 and 4 to infinite Abelian groups of prime exponent [6].

Twenty Questions cannot be used to separate P from NP in $\mathbb{R}_{ovs}$ since this problem is in $\mathrm{P}_{\mathbb{R}_{ovs}}$ by binary search. In fact, obtaining a proof of this separation is all but hopeless by another transfer theorem from [15]:

Theorem 6. $\mathrm{P} = \mathrm{NP} \Rightarrow \mathrm{P}_{\mathbb{R}_{ovs}} = \mathrm{NP}_{\mathbb{R}_{ovs}}$.

In a previous paper [14] we obtained a similar result for the "P = PAR ?" problem:

Theorem 7. $\mathrm{P} = \mathrm{PSPACE} \Rightarrow \mathrm{P}_{\mathbb{R}_{ovs}} = \mathrm{PAR}_{\mathbb{R}_{ovs}}$.

Theorems 3 through 7 show that a number of separations between real complexity classes are (at least) as hard to prove as outstanding conjectures from discrete complexity theory. Yet we know that these separations must hold true since an equality of two real complexity classes would imply the equality of the corresponding discrete complexity classes. This follows from the "boolean parts" results of [25, 12]. For instance, $\Sigma^k_{\mathbb{R}_{vs}} = \Pi^k_{\mathbb{R}_{vs}}$ would imply $\Sigma^k = \Pi^k$, and $\mathrm{P}_{\mathbb{R}_{ovs}} = \mathrm{NP}_{\mathbb{R}_{ovs}}$ would imply P/poly = NP/poly. Non-uniformity comes into play in the latter statement only because arbitrary real parameters are allowed: P/poly can be replaced by P and NP/poly by NP if we work with parameter-free machines, or in the structure $\mathbb{Q}_{ovs}$ instead of $\mathbb{R}_{ovs}$.

We will see in the next section that Theorems 6 and 7 hinge on the following question:

Question 1. Are computation trees more powerful than circuits ?

If this deliberately fuzzy question is interpreted in its broadest sense, it is possible to give a positive answer for many structures of interest. For instance, as pointed out before, in the standard structure $M = \{0, 1\}$ any boolean function of n variables can be computed by a tree of depth n, but a simple counting argument shows that most boolean functions have exponential circuit complexity [1]. Bounds on the number of consistent sign conditions (à la Thom-Milnor, see end of section 5.1) show that most boolean functions also have exponential circuit complexity over $\mathbb{R}$ and $\mathbb{C}$, even if arbitrary real or complex parameters are allowed [28]. It is nevertheless possible to construct a special-purpose structure in which polynomial size computation trees are equivalent to polynomial size circuits (hint: try to encode a computation tree in a circuit's parameter; this can be done efficiently with Poizat's arborescent dictionaries).

This question becomes more interesting if we compare the power of trees and circuits on a restricted class of problems, for instance on NP_M problems. In this case the situation is dramatically different since, as pointed out by Poizat (personal communication), we do not have a single example of a structure M where NP_M problems can be solved by polynomial depth computation trees, but

$P_M \neq NP_M$. For instance, computation trees over $\mathbb{R}_{ovs}$ are likely to be more powerful than circuits on $NP_{\mathbb{R}_{ovs}}$ problems (in fact, as explained in the next section any such problem can be solved polynomial depth computation trees) but by Theorem 6 proving this is essentially "impossible."

Shub and Smale's invention of Twenty Questions was motivated by an attempt to separate $P_{\mathbb{C}}$ from $NP_{\mathbb{C}}$. The plausible number-theoretic conjecture that "$k!$ is ultimately hard to compute" was put forward in [38], and was shown to imply that Twenty Questions is not $P_{\mathbb{C}}$. Since Twenty Questions is $NP_{\mathbb{C}}$, a proof of this conjecture would indeed separate $P_{\mathbb{C}}$ from $NP_{\mathbb{C}}$. We point out however that Shub and Smale's proof of this implication is based on a canonical path argument similar to the argument of Theorem 2 (but more involved). As a result, the conjecture in fact implies that Twenty Questions cannot be solved by polynomial depth computation trees. Although of great interest, its proof would therefore shed no light on Question 1.

5 Complexity and Point Location

5.1 Point Location by Computation Trees

There are two main steps in the proofs of Theorems 6 and 7:

(i) Show that any $NP_{\mathbb{R}_{ovs}}$ (or $PAR_{\mathbb{R}_{ovs}}$) problem can be solved by polynomial depth computation trees.
(ii) Show that under an appropriate complexity-theoretic assumption (P = NP or P = PSPACE), these trees can be "transformed" into polynomial size circuits.

Two proofs of (i) are known. The first one is essentially due to Meyer auf der Heide [31, 32]. In fact this author showed that $PAR_{\mathbb{R}_{ovs}}$ problems can be solved by $\mathbb{R}_{ovs}$-branching trees of polynomial depth. In order to obtain polynomial depth computation trees, we just have to check that the coefficients of affine functions labeling the nodes of these branching trees are integers of polynomial size [15]. Meyer auf der Heide's proof exploits the geometrical structure of a $PAR_{\mathbb{R}_{ovs}}$ problem: its restriction to inputs in $\mathbb{R}^n$ is a union of faces of an arrangement of $2^{n^{O(1)}}$ hyperplanes. We recall that any finite set of m hyperplanes of equations $h_1(x) = 0, \ldots, h_m(x) = 0$ partitions $\mathbb{R}^n$ in a finite number of faces, where each face is the set of points satisfying a system of equations of the form

$$\mathrm{sign}(h_1(x)) = \epsilon_1, \ldots, \mathrm{sign}(h_m(x)) = \epsilon_m \tag{3}$$

for some fixed sign vector $(\epsilon_1, \ldots, \epsilon_m) \in \{-1, 0, 1\}^m$. In order to decide whether an input point should be accepted we just need to locate it in the arrangement, that is, to determine to which face it belongs. Meyer auf der Heide [32] asked whether any union of m hyperplanes of $\mathbb{R}^n$ can be recognized by $\mathbb{R}_{ovs}$-branching trees of depth $(n \log m)^{O(1)}$. His construction does not quite yield that result because it uses certain bounds on the size of the hyperplanes' equations. In [14] we used a construction of Meiser [30] to give a second proof of (i); this construction also yields a positive answer to Meyer auf der Heide's question. In fact Meiser's construction almost answers that question, except that he has a non-degeneracy assumption on the arrangement, and that he allows multiplications in his computation model (that is, he works with $\mathbb{R}$-branching tree instead of $\mathbb{R}_{ovs}$-branching trees). These problems can be fixed as explained in [14]. To complete the second proof of (i) we also need to analyze the size of coefficients in the resulting $\mathbb{R}_{ovs}$-branching trees; this is done in the same paper.

The transformation of polynomial depth trees into polynomial size circuits under the assumption P = PSPACE is based on an exhaustive search procedure among all polynomial depth trees. This procedure can be made to run in polynomial space, and thus in polynomial time under the assumption P = PSPACE (see [14] or section 6). In order to use the weaker assumption P = NP, the mere knowledge that $\mathrm{PAR}_{\mathbb{R}_{ovs}}$ can be solved by polynomial depth trees is not sufficient: we also need to know how these trees are constructed. We chose to work with Meyer auf der Heide's construction because it yields a stronger result than Meiser's. Namely, we obtained the unconditional result that $\mathrm{NP}_{\mathbb{R}_{ovs}}$ problems can be solved by $\mathrm{P}_{\mathbb{R}_{ovs}}$ algorithms with the help of a boolean NP oracle [15]; with Meiser's construction one would obtain an oracle in a higher level of the polynomial hierarchy.

Nothing like Theorem 6 or Theorem 7 is known if we replace $\mathbb{R}_{ovs}$ by $\mathbb{R}$ or $\mathbb{C}$. It is not even known whether $\mathrm{NP}_{\mathbb{R}}$ or $\mathrm{NP}_{\mathbb{C}}$ problems can be solved by polynomial depth computation trees (as shown in section 6, this question is very much related to the "P = NP ?" problem over the real and complex numbers). In fact, it is natural to conjecture that Twenty Questions *cannot* be solved by polynomial depth computation trees (as pointed out in section 4, this would follow from the conjecture that $k!$ is ultimately hard to compute). The bold conjecture that all $\mathrm{PAR}_{\mathbb{R}}$ problems *can* be solved by polynomial depth computation tree was put forward in [14]. Like in the $\mathbb{R}_{ovs}$ case, there is an intimate relationship between $\mathrm{PAR}_{\mathbb{R}}$ problems and point location: the restriction to $\mathbb{R}^n$ of such a problem is a union of faces of an arrangement defined by $m = 2^{n^{O(1)}}$ polynomials $f_1, \ldots, f_m$ of degree $2^{n^{O(1)}}$. Faces are defined as in the $\mathbb{R}_{ovs}$ case: just replace $h_1, \ldots, h_m$ in (3) by $f_1, \ldots, f_m$. A similar property is also true of $\mathrm{PAR}_{\mathbb{C}}$ problems. Here we need to redefine the sign function so that it only takes the values 0 (if its input is equal to zero) or 1 (if it is nonzero). As explained in [14], point location in an arrangement defined by m real polynomials of fixed degree in n variables can be performed by a computation tree of depth polynomial in $n \log m$ by reduction to the linear case.

At least we know that $\mathrm{PAR}_{\mathbb{R}}$ and $\mathrm{PAR}_{\mathbb{C}}$ problems have polynomial branching complexity. In the real case, this follows from the well-known result that m polynomials of degree d in n variables define an arrangement with $(md)^{O(n)}$ faces (see [2] for a sharper bound) and from a recent result of Grigoriev [20] who, answering a question of [14], showed that point location in an arrangement with N faces has branching complexity $O(\log N)$. The $(md)^{O(n)}$ bound on the number of faces still holds in the complex case, and point location in an arrangement with N faces has branching complexity $O(n \log N)$ as explained in section 5.2. This shows that $\mathrm{PAR}_{\mathbb{C}}$ problems indeed have polynomial branching complexity.

5.2 Branching Complexity of Point Location

Recall that a tree solves the point location problem for a given arrangement if two input points arriving at the same leaf of the tree always belong to the same cell. In this section we prove the $O(n \log N)$ upper bound on the complexity of point location in the complex case, and give some hints for the proof of Grigoriev's $O(\log N)$ bound. In fact, his proof applies not only to $\mathbb{R}$ but to any ordered field. Likewise, we state and prove our $O(n \log N)$ bound not just for the field of complex numbers, but for an arbitrary field.

We need the following fact for the proof of Theorem 8 (see [9] for a constructive version).

Proposition 1. *Let K be an infinite field and V a variety of K^n defined by polynomials $f_1, \ldots, f_s \in K[X_1, \ldots, X_n]$ of degree at most d. This variety can be defined by $n+1$ polynomials of $K[X_1, \ldots, X_n]$ of degree at most d.*

Proof. We shall see that V can be defined by $n+1$ "generic" linear combinations of the input equations. More precisely, for a matrix $\alpha = (\alpha_{ij})_{1 \le i \le n+1, 1 \le j \le s}$ of elements of K, let us denote by V_α the variety of K^n defined by the polynomials $g_i = \sum_{j=1}^{s} \alpha_{ij} f_j$ $(1 \le i \le n+1)$. Obviously, $V \subseteq V_\alpha$ for any α. It turns out that the converse inclusion holds for "most" α's.

Let p be the characteristic of K and F_p the prime field of characteristic p (i.e., $F_0 = \mathbb{Q}$ and $F_p = \mathbb{Z}/p\mathbb{Z}$ for $p \ge 2$). We shall assume for now that K is algebraically closed, of infinite transcendence degree over F_p.

The coefficients of the f_i's lie in a subfield $k \subseteq K$ of finite transcendence degree. We claim that $V = V_\alpha$ if the entries of α are algebraically independent over k. Assume indeed that $\sum_{j=1}^{s} \alpha_{ij} f_j(x) = 0$ for $i = 1, \ldots, n+1$. Since the tuple $x\hat{}\alpha$ is of transcendence degree at least $s(n+1) - n$ over $k(x)$, any transcendence base of $k(x, \alpha)$ over $k(x)$ which is made up of entries of α must contain at least one row of α. If i is such a row, the equality $\sum_{j=1}^{s} f_j(x) \alpha_{ij} = 0$ implies that $f_j(x) = 0$ for all j since $f_j(x) \in k(x)$ and $\alpha_{i1}, \ldots, \alpha_{is}$ are algebraically independent over that field by choice of i. We therefore conclude that $x \in V$, and that $V_\alpha \subseteq V$ since x was an arbitrary point of V_α.

To complete the proof of the proposition, we just need to remove the assumption on K. In the general case, K can be embedded in an algebraically closed field $\hat{K}$ of infinite transcendence degree. The f_i's define a variety $\hat{V}$ of $\hat{K}^n$, and likewise for any matrix α with entries in $\hat{K}$, the g_i's define a variety $\hat{V}_\alpha$ of $\hat{K}^n$ which contains $\hat{V}$. Let G be the (constructible) set of all $\alpha \in \hat{K}^{s(n+1)}$ such that $\hat{V} = \hat{V}_\alpha$. We have seen that $\alpha \in G$ if its entries are algebraically independent over k. This implies that G is dense in $\hat{K}^{s(n+1)}$. It follows that for any infinite subset E of $\hat{K}$, and in particular for $E = K$, there exists a matrix $\alpha \in G$ with entries from E (this can be proved by induction). For such a matrix, $V = V_\alpha$ since $\hat{V} = \hat{V}_\alpha$.

Theorem 8. *Let $\mathcal{F} = \{f_1, \ldots, f_s\}$ be a family of polynomials of $K[X_1, \ldots, X_n]$, where K is an arbitrary field. The point location problem for $\mathcal{F}$ has branching complexity $O(n \log N)$, where N is the number of cells in the arrangement.*

Proof. We will assume that K is infinite since the result is trivial for finite fields (in that case, any function on K^n can be computed in depth $O(n)$).

For any subset I of $[n] = \{1, \ldots, n\}$, we denote by C_I the sign condition

$$(f_i = 0)_{i \in I} \wedge (f_i \neq 0)_{i \notin I}.$$

We say that C_I is feasible if the set of points satisfying this sign condition is nonempty (in which case it is a cell of the arrangement). We shall also denote by $\leq$ a (fixed) total order on the subsets of $[n]$ which is compatible with inclusion (i.e., $I \subseteq J$ implies $I \leq J$). Our construction is based on these two observations:

1. For any $I \subseteq [n]$, $V_I = \bigcup_{J \geq I} C_J$ is a variety of K^n: it is the union for $J \geq I$ of the varieties $(f_i = 0)_{i \in J}$.
2. Any variety of K^n can be defined by $n+1$ equations, as shown in Proposition 1.

Note that $I \leq J$ if and only if $V_J \subseteq V_I$. Let us say that $I \subseteq [n]$ is feasible if C_I is a feasible sign condition. On input $x \in K^n$, our point location algorithm finds by binary search the largest feasible I such that $x \in V_I$. Since x satisfies C_I, this is the desired sign condition.

There are N satisfiable sign conditions, so that we only need $\log N$ steps of binary search. At each step the algorithm performs a test of the form: "$x \in V_J$?". By the second observation, this test can be performed in depth $n+1$. The overall depth of the corresponding tree is therefore $O(n \log N)$.

Note that $\log N$ is an obvious lower bound on the branching complexity of point location (a binary tree with at least N leaves must have depth at least $\log N$), so that the $O(n \log N)$ of Theorem 8 is not too far from the optimum.

If K is the field of real numbers (or more generally is an ordered field), the $\log N$ lower bound is in fact tight. Indeed, we no longer need $n+1$ polynomials to define a variety as in Proposition 1: by the usual "sum of squares" trick, a single polynomial suffices (see [20] for a more direct proof). The construction of Theorem 8 therefore yields a branching tree of depth $O(\log N)$. As shown in [20], it possible within the same depth not only to find out whether each polynomial f_i is zero or non-zero at the input point, but to find out whether it is positive or negative:

Theorem 9. *Let $\mathcal{F} = \{f_1, \ldots, f_s\}$ be a family of polynomials of $R[X_1, \ldots, X_n]$ where R is an ordered field. The point location problem for $\mathcal{F}$ has branching complexity $O(\log N)$, where N is the number of cells of the arrangement.*

The proof of Theorem 9 relies on a two-stage construction. In the first stage we determine for each f_i whether it vanishes at the input point x. This can be done in depth $O(\log N)$ as explained before Theorem 9. In the second stage, we determine the sign of $f_i(x)$ for each polynomial f_i which does not vanish at x. The proof that this can also be done in depth $O(\log N)$ relies on a nice combinatorial lemma:

Lemma 1. *Let $u_1, \ldots, u_m$ be pairwise distinct vectors of $(\mathbb{Z}/2\mathbb{Z})^k$. If $m \geq 6$ there exists a vector $v \in (\mathbb{Z}/2\mathbb{Z})^k$ such that*

$$m/3 \leq |\{1 \leq i \leq m;\ \langle v, u_i \rangle = 0\}| \leq 2m/3.$$

For the proof of this lemma and its application to Theorem 9, we refer the reader to [20].

6 The Computation Tree Alternative

In this section we show that Question 1 plays a crucial role in the study of the "P = NP?" problem over the real and complex numbers. The proofs are based on techniques from [14].

The $\mathrm{NP}_{\mathbb{C}}$-complete $\mathrm{HN}_{\mathbb{C}}$ appears in the statement of our computation tree alternative for the complex numbers. This is the problem of deciding whether a system of polynomial equations $f_1(x_1, \ldots, x_n) = 0, \ldots, f_s(x_1, \ldots, x_n) = 0$ in n complex variables has a solution.

Theorem 10. *If $\mathrm{HN}_{\mathbb{C}}$ can be solved by a family of parameter-free computation trees of polynomial depth we have the following transfer theorem:* P = PSPACE *implies* $\mathrm{P}_{\mathbb{C}} = \mathrm{NP}_{\mathbb{C}}$. *Otherwise,* $\mathrm{P}_{\mathbb{C}} \neq \mathrm{NP}_{\mathbb{C}}$.

Proof. If $\mathrm{HN}_{\mathbb{C}}$ cannot be solved by a family of parameter-free computation trees of polynomial depth then this problem cannot be solved by a family of parameter-free circuits of polynomial size. By elimination of parameters [3, 4, 27], this implies that $\mathrm{P}_{\mathbb{C}} \neq \mathrm{NP}_{\mathbb{C}}$.

It is well known that the field of complex numbers satisfies the hypothesis of Theorem 12 below [13]. If P = PSPACE and $\mathrm{HN}_{\mathbb{C}}$ can be solved by a family of parameter-free computation trees of polynomial depth, $\mathrm{HN}_{\mathbb{C}}$ is therefore $\mathrm{P}_{\mathbb{C}}$ (even $\mathrm{P}^0_{\mathbb{C}}$) by Theorem 12 and thus $\mathrm{P}_{\mathbb{C}} = \mathrm{NP}_{\mathbb{C}}$.

Theorem 10 can be interpreted as follows. If we can prove (unconditionally) that $\mathrm{HN}_{\mathbb{C}}$ cannot be solved by a family of parameter-free polynomial depth computation trees, we have obtained an unconditional separation of $\mathrm{P}_{\mathbb{C}}$ from $\mathrm{NP}_{\mathbb{C}}$. If $\mathrm{HN}_{\mathbb{C}}$ *can* be solved by a family of parameter-free polynomial depth computation trees, $\mathrm{P}_{\mathbb{C}}$ is still very likely to be different from $\mathrm{NP}_{\mathbb{C}}$ (otherwise, NP would be included in BPP [40, 28]). However, obtaining a proof of this separation would be a hopeless problem, at least in the current state of discrete complexity theory.

For the reals we can replace $\mathrm{HN}_{\mathbb{C}}$ by the $\mathrm{NP}_{\mathbb{R}}$-complete problem $4\mathrm{FEAS}_{\mathbb{R}}$: this is the problem of deciding whether a polynomial of degree at most 4 in n real variables has a root. We can only deal with parameter-free algorithms since elimination of parameters is not known to hold in the real case (see [8] for some results in this direction and more references).

Theorem 11. *If* $4\mathrm{FEAS}_{\mathbb{R}}$ *can be solved by a family of parameter-free computation trees of polynomial depth we have the following transfer theorem:* $\mathrm{P} = \mathrm{PSPACE}$ *implies* $\mathrm{P}^0_{\mathbb{R}} = \mathrm{NP}^0_{\mathbb{R}}$ *(which implies* $\mathrm{P}_{\mathbb{R}} = \mathrm{NP}_{\mathbb{R}}$*). Otherwise,* $\mathrm{P}^0_{\mathbb{R}} \neq \mathrm{NP}^0_{\mathbb{R}}$.

Proof. It is similar to the proof of Theorem 10 (the decision algorithm of [13] can be replaced by the algorithm of [2] or [37]).

In the proof of Theorem 12 it is convenient to work with branching trees of a restricted form instead of computation trees. That is, we will use the observation that a computation tree of depth d can be simulated by a branching tree of depth at most d whose nodes are labeled by atomic formulas $r_i(t_1, \ldots, t_{m_i})$ satisfying the following condition: $t_1, \ldots, t_{m_i}$ can be computed by straight-line programs of size at most d. We recall that a straight-line program over M is a circuit in which all gates are labeled by functions of M (relation and selection gates are not used).

Theorem 12. *Let* M *be a structure whose parameter-free* Σ^2 *formulas can be decided in polynomial space. Let* X *be a* NP^0_M *problem which can be solved by a family of parameter-free computation trees of polynomial depth. If* $\mathrm{P} = \mathrm{PSPACE}$ *this problem is* P^0_M.

For the proof of this theorem we need to associate to X a boolean problem $\tilde{X}$. An instance of $\tilde{X}$ is described by three integers n, L, d (written in unary), and by a conjunction $F(x_1, \ldots, x_n)$ of (parameter free) atomic formulas. For the terms in F we assume a straight-line representation. This conjunction defines a subset S_F of $\mathbb{R}^n$. An instance is positive if there exists a branching tree T satisfying the following properties:

(i) T is of depth at most d and solves X for inputs in S_F.
(ii) The nodes of T are labeled by atomic formulas of the form $r_i(t_1, \ldots, t_{m_i})$ where the terms $t_1, \ldots, t_{m_i}$ are computed by straight-line programs of length at most L.

We need an algorithm to solve $\tilde{X}$, and for positive instances of this problem we also need to compute the label l_r of the root of a corresponding tree T (this tree might not be unique, but any solution will do). Thus l_r is just a boolean value if T is reduced to a leaf, and an atomic formula otherwise.

Lemma 2. *If* $X \in \mathrm{NP}^0_M$ *then* $\tilde{X} \in \mathrm{PSPACE}$. *Moreover, for a positive instance* l_r *can be constructed in polynomial space.*

Proof. We first determine whether T can be of depth 0, i.e., reduced to a leaf. In that case, T recognizes either $\mathbb{R}^n$ or $\emptyset$ depending on the label of that leaf. Label 0 is suitable if the formula

$$\exists x \in \mathbb{R}^n \; F(x) \wedge (x \in X)$$

is false. By hypothesis on M, this Σ^1 formula can be decided in polynomial space (note that we can introduce additional existentially quantified variables in order to move from the straight-line representation of F to the standard representation). Label 1 is suitable if the Σ^2 formula

$$\exists x \in \mathbb{R}^n \ F(x) \wedge (x \notin X)$$

is false. If there is a solution in depth 0, we accept the instance of $\tilde{X}$ and output the corresponding label. Otherwise, for $d > 0$ we look for solutions of depth between 1 and d (for $d = 0$ we exit and reject the instance). To do this we enumerate (e.g. in lexicographic order) all atomic formulas $A(x_1, \ldots, x_n)$ where the terms in A are given by straight-line programs of length at most L. For each such formula we do the following.

1. Decide by a recursive call whether $(n, L, d-1, F(x) \wedge A(x))$ is a positive instance of $\tilde{X}$.
2. Decide by a recursive call whether $(n, L, d-1, F(x) \wedge \neg A(x))$ is a positive instance of $\tilde{X}$.
3. In case of a positive answer to both questions, exit the loop, accept (n, L, d, F) and output $l_r = A$.

The instance is rejected if it is not accepted in the course of this enumeration procedure.

In addition to the space needed to solve the depth 0 case, we just need to maintain a stack to keep track of recursive calls. Hence this algorithm runs in polynomial space, showing that $\tilde{X} \in$ PSPACE. For positive instances, the algorithm also outputs l_r as needed.

Proof (of Theorem 12). For inputs of size n, X can be solved by a computation tree of depth bounded by an^b, where a and b are constants. The idea is to use Lemma 2 to move down that tree. The hypothesis P = PSPACE implies that $\tilde{X} \in$ P. Moreover, for positive instances l_r can be constructed in polynomial time (one should argue that each bit of l_r is in PSPACE, and therefore in P). Thus we set $L = d = an^b$ and $F \equiv$ True. By hypothesis (n, L, d, F) is a positive instance of $\tilde{X}$ and therefore l_r can be computed in polynomial time. If l_r is a boolean value we stop and output that value. Otherwise A is an atomic formula, and we can determine in polynomial time whether the input $x \in \mathbb{R}^n$ to X satisfies A (because straight-line programs can be evaluated efficiently). If so, we set $F' = F \wedge A(x)$. Otherwise, we set $F' = F \wedge \neg A(x)$. In any case, we set $d' = d - 1$, and feed (n, L, d', F') to the algorithm deciding $\tilde{X}$. This process continues until a leaf is reached. This requires at most an^b steps.

Bruno Poizat pointed out that it would be natural to work with families of computation trees using the same parameters for all input sizes, since we use this convention for our polynomial-time algorithms. The reader may try to formulate and prove the counterparts of Theorems 10 and 11 in that setting.

Acknowledgements

Thanks to Dima Grigoriev for the discussions on the branching complexity of point location. The results of section 5.2 were obtained while we were visiting the Mathematical Sciences Research Institute in Berkeley. Useful comments were made by Hervé Fournier, Bruno Poizat, and Natacha Portier.

References

1. J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1988.
2. S. Basu, R. Pollack, and M.-F. Roy. On the combinatorial and algebraic complexity of quantifier elimination. *Journal of the ACM*, 43(6):1002–1045, 1996.
3. L. Blum, F. Cucker, M. Shub, and S. Smale. Algebraic settings for the problem "P≠NP?". In J. Renegar, M. Shub, and S. Smale, editors, *The Mathematics of Numerical Analysis*, volume 32 of *Lectures in Applied Mathematics*, pages 125–144. American Mathematical Society, 1996.
4. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer-Verlag, 1998.
5. L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, July 1989.
6. M. Bourgade. Séparations et transferts dans la hiérarchie polynomiale des groupes abéliens infinis. preprint, 1999.
7. P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory.* Springer, 1997.
8. O. Chapuis and P. Koiran. Saturation and stability in the theory of computation over the reals. *Annals of Pure and Applied Logic*, 99:1–49, 1999.
9. A. Chistov and D. Grigoriev. Subexponential-time solving systems of algebraic equations I, II. Preprints LOMI E-9-83, E-10-83, Leningrad, 1983.
10. A. Chistov and D. Grigoriev. Complexity of quantifier elimination in the theory of algebraically closed fields. In *Proc. MFCS'84*, volume 176 of *Lectures Notes in Computer Science*, pages 17–31. Springer-Verlag, 1984.
11. F. Cucker and D. Grigoriev. On the power of real Turing machines with binary inputs. *SIAM Journal on Computing*, 26(1):243–254, 1997.
12. F. Cucker and P. Koiran. Computing over the reals with addition and order: Higher complexity classes. *Journal of Complexity*, 11:358–376, 1995.
13. N. Fichtas, A. Galligo, and J. Morgenstern. Precise sequential and parallel complexity bounds for quantifier elimination over algebraically closed fields. *Journal of Pure and Applied Algebra*, 67:1–14, 1990.
14. H. Fournier and P. Koiran. Are lower bounds easier over the reals ? In *Proc. 30th ACM Symposium on Theory of Computing*, pages 507–513, 1998.
15. H. Fournier and P. Koiran. Lower bounds are not easier over the reals: Inside PH. LIP Research Report 99-21, Ecole Normale Supérieure de Lyon, 1999.
16. J. B. Goode. Accessible telephone directories. *Journal of Symbolic Logic*, 59(1):92–105, 1994.
17. J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press, 1997.

18. D. Grigoriev. Complexity of deciding Tarksi algebra. *Journal of Symbolic Computation*, 5:65–208, 1988.
19. D. Grigoriev. Complexity of quantifier elimination in the theory of ordinary differential equations. In *Proc. Eurocal'87*, volume 378 of *Lectures Notes in Computer Science*, pages 11–25. Springer-Verlag, 1989.
20. D. Grigoriev. Topological complexity of the range searching. To appear in *Journal of Complexity*.
21. D. Grigoriev, M. Karpinski, and R. Smolensky. Randomization and the computational power of algebraic and analytic decision trees. *Computational Complexity*, 6(4):376–388, 1996/97.
22. D. Grigoriev, M. Singer, and A. Yao. On computing algebraic functions using logarithms and exponentials. *SIAM Journal on Computing*, 24(2):242–246, 1995.
23. D. Grigoriev and N. Vorobjov. Complexity lower bounds for computation trees with elementary transcendental function gates. *Theoretical Computer Science*, 2:185–214, 1996.
24. W. Hodges. *Model Theory.* Encyclopedia of Mathematics and its Applications **42**. Cambridge University Press, 1993.
25. P. Koiran. Computing over the reals with addition and order. *Theoretical Computer Science*, 133(1):35–48, 1994.
26. P. Koiran. VC dimension in circuit complexity. In *Proc. 11th IEEE Conference on Computational Complexity*, pages 81–85, 1996.
27. P. Koiran. Elimination of constants from machines over algebraically closed fields. *Journal of Complexity*, 13(1):65–82, 1997. Erratum on *http://www.ens-lyon.fr/∼koiran.*
28. P. Koiran. A weak version of the Blum, Shub & Smale model. *Journal of Computer and System Sciences*, 54:177–189, 1997.
29. K. Meer. A note on a $P{\neq}NP$ result for a restricted class of real machines. *Journal of Complexity*, 8:451–453, 1992.
30. S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.
31. F. Meyer auf der Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *Journal of the ACM*, 31(3):668–676, 1984.
32. F. Meyer auf der Heide. Fast algorithms for n-dimensional restrictions of hard problems. *Journal of the ACM*, 35(3):740–747, 1988.
33. C Michaux. Une remarque à propos des machines sur $\mathbb{R}$ introduites par Blum, Shub et Smale. *C. R. Acad. Sci. Paris*, 309, Série I:435–437, 1989.
34. B. Poizat. *Cours de Théorie des Modèles.* Nur Al-Mantiq Wal-Ma'rifah **1**. 1985.
35. B. Poizat. *Les Petits Cailloux.* Nur Al-Mantiq Wal-Ma'rifah **3**. Aléas, Lyon, 1995.
36. N. Portier. Stabilité polynomiale des corps différentiels. *Journal of Symbolic Logic*, 64(2):803–816, 1999.
37. J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. parts I, II, III. *Journal of Symbolic Computation*, 13(3):255–352, March 1992.
38. M. Shub and S. Smale. On the intractability of Hilbert's Nullstellensatz and an algebraic version of "P=NP". *Duke Mathematical Journal*, 81(1):47–54, 1996.
39. S. Smale. On the topology of algorithms. I. *Journal of Complexity*, 3:81–89, 1987.
40. S. Smale. On the P=NP problem over the complex numbers. Lecture given at the MSRI workshop on Complexity of Continuous and Algebraic Mathematics, November 1998. Lecture on video at www.msri.org.

41. V. A. Vassiliev. *Complements of Discriminants of Smooth Maps: Topology and Applications.* Translations of Mathematical Monographs **98**. American Mathematical Society, 1994.
42. J. von zur Gathen. Parallel arithmetic computations: a survey. In *Mathematical Foundations of Computer Science*, volume 233 of *Lecture Notes in Computer Science*, pages 93–112. Springer-Verlag, 1986.