Alcom-FT Technical Report Series
ALCOMFT-TR-01-37

Some aspects of studying an optimization or decision problem in different computational models

Klaus Meer[1]* and Gerhard Wilhelm Weber[2]

[1] Department of Mathematics and Computer Science
Syddansk Universitet, Odense, Denmark
e–mail: meer@imada.sdu.dk
[2] Mathematical Institute and Center of Applied Informatics
University Cologne, Weyertal 86-90, 50931 Cologne, Germany
e–mail: gwweber@zpr.uni-koeln.de or weber@mathematik.tu-darmstadt.de

Abstract. In this paper we want to discuss some of the features coming up when analyzing a problem in different complexity theoretic frameworks. the focus will be on two problems. The first is related to mathematical optimization. We consider the quadratic programming problem of minimizing a quadratic polynomial on a polyhedron. We discuss how the complexity of this problem might change if we consider real data together with an algebraic model of computation (the Blum-Shub-Smale model) instead of rational inputs together with the Turing machine model. The results obtained will lead us to the second problem; it deals with the intrinsic structure of complexity classes in models over real- or algebraically closed fields. A classical theorem by Ladner for the Turing model is examined in these different frameworks.
Both examples serve well for working out in how far different approaches to the same problem might shed light upon each other. In some cases this will lead to quite diverse results with respect to the different models. On the other hand, for some problems the more general approach can also give a unifying idea why results hold true in several frameworks.
The paper is of tutorial character in that it collects some results into the above direction obtained previously.

Key-words: Complexity theory, algebraic models of computation, linear and quadratic programming, structure of complexity classes, saturation

AMS-Subject Classification: 90C20 ; 68Q15 ; 68Q25

1 Introduction

A lot of research in optimization is centered around the analysis of linear and quadratic optimization. This refers as well to the design and implementation of algorithms as to the theoretical study both of concrete algorithms and the structure of the problems itself.

A major role here is played by the analysis of interior-point algorithms and their effects on different aspects of optimization: in classical complexity theory (Turing model) they provide polynomial time algorithms for the linear programming and other convex optimization problems; there are relations to semi-infinite optimization and optimal control (cf. [41]); and, of course, there is a tremendous amount on practical implementations of these methods.

* partially supported by EU ESPRIT project NeuroCOLT2 and by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

In this paper we want to focus on complexity theoretic questions. Analyzing optimization problems in algebraic models of computation leads to a lot of interesting questions. We want to treat completeness aspects in the real number model of Blum, Shub, and Smale [4]. It seems to be reasonable to conjecture that the complexity behavior of some optimization problems is quite different in such a setting compared to the Turing model. This gives rise to some further studies concerning the internal structure of complexity classes. Here, model theory comes into play.

1.1 Motivation

Whereas the study of complexity traditionally has mainly been performed in a discrete framework using the Turing machine model, a different approach arose around the subject of computations over continuous domains since they underly, for example, some of the most central algorithms in numerical analysis (such as Newton's method). In recent years, models more appropriate for continuous problems have been considered as well. To mention some of the most important there are algebraic circuits analyzed in algebraic complexity theory; the real Turing resp. Blum-Shub-Smale machines which provide a machine model for uniform algorithms over the real numbers; and the approach followed in Information Based Complexity. For excellent text books on these subjects we refer to [3], [7], [37]. Also the Turing model has been used in recent years to deal with real number problems, for example, in recursive analysis (see [18],[42]) or in interval arithmetics [19].

Dealing with different approaches leads to the question in how far results obtained in one setting can shed some light upon the situation as it appears within another framework. As one highlight of such a possible exchange of results one might see the famous P versus NP question: this major open problem in the classical Turing setting seems to be of similar importance for computational theories over different other structures. There are similar versions of this problem over the real and the complex numbers, which were introduced by Blum, Shub, and Smale in [4]. An "ideal" kind of result under that point of view would look like P = NP over $\{0, 1\}$ if and only if $P_{\mathbb{R}} = NP_{\mathbb{R}}$ over $\mathbb{R}$ (or $P_{\mathbb{C}} = NP_{\mathbb{C}}$ over $\mathbb{C}$). In fact, some very interesting results into this direction have already been obtained, see [3], chapter 6 and [14].

Clearly, there are also some doubts about comparing such different approaches. A computer scientist will probably feel uneasy studying models in which real numbers are considered as entities as it is done, for example, in the Blum-Shub-Smale (henceforth for short: BSS) model. On the other hand, we believe that even in classical complexity theory many important algorithms are developed on a "continuous" level and so to speak only subsequently turn out to behave well on the bit level. A typical example is provided by interior point methods which to a large extent make use of numerical algorithms such as Newton's method.

In this paper we want to discuss some previous work based on [2], [24], [25] in order to compare the Turing and the BSS model. We will focus on two different problems which in our opinion show very well some features of analyzing the same question in different settings. The first problem under consideration is mathematical optimization. Here, it is well known that the linear programming problem LP admits polynomial time algorithms in discrete complexity theory; the quadratic (nonconvex) programming problem QP is NP-complete, thus bearing the entire difficulty of the class NP. As we will see in section 2 both problems are likely to behave very different in a real number setting.

The results obtained in section 2 lead in a straightforward manner to another problem related to the intrinsic structure of NP within the different models; this problem asks whether, assuming that P $\neq$ NP, there are problems in NP $\setminus P$ which are not NP-complete. It was answered in the positive by Ladner for the Turing model. In section 3 this problem will be studied for the complex and the real numbers in relation with the BSS model. The analysis behind also gives a clear understanding why Ladner's result over finite alphabets holds true.

1.2 The computational models

We will mainly deal with two models of computation; when the underlying structure is discrete the classical Turing model is used. We suppose the reader to be familiar with it and just refer to the textbooks [1] or, more recently, [5]. If the data of a problem are real or complex numbers then we use the Blum-Shub-Smale model of computation introduced in [4]. An extensive treatment of it can be found in [3]; a general survey on BSS theory is [26]. A survey specifically devoted to the benefit of using real number algorithms for solving discrete problems is [10].

For sake of completeness we will now briefly describe the BSS model.

Essentially, a (real) Blum-Shub-Smale machine can be considered as a Random Access Machine over $\mathbb{R}$ which is able to perform the basic arithmetic operations at unit cost and which registers can hold arbitrary real numbers.

Definition 1 ([4]) *a) Let $Y \subseteq \mathbb{R}^* := \bigoplus_{k\in\mathbb{N}} \mathbb{R}^k$, i.e. the set of finite sequences of real numbers. A* BSS-machine M over $\mathbb{R}$ with admissible input set Y *is given by a finite set I of instructions labeled by $1,\ldots,N$. A configuration of M is a quadruple $(n,i,j,x) \in I \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}^*$. Here, n denotes the instruction currently executed, i and j are used as addresses (copy-registers) and x is the actual content of the registers of M. The initial configuration of $M's$ computation on input $y \in Y$ is $(1,1,1,y)$. If $n = N$ and the actual configuration is (N,i,j,x), the computation stops with output x .*

We denote by Φ_M the function computed by machine M and write $\Phi_M(y) = x$.

The instructions which M is allowed to perform are of the following types :

Computation: *$n : x_s \leftarrow x_k \circ_n x_l$, where $\circ_n \in \{+,-,*,:\}$ or $n : x_s \leftarrow c$ for some constant $c \in \mathbb{R}$. The register x_s will get the value $x_k \circ_n x_l$ or c, respectively. All other register-entries remain unchanged. The next instruction will be $n+1$; moreover, the copy-register i is either incremented by one, replaced by 0, or remains unchanged. The same applies for the copy-register j.*

We call the finite set $\{c_1,\ldots,c_k\}$ of all real constants introduced in an operation $x_s \leftarrow c_i$ the machine constants *of M.*

Branch: *n:* **if** $x_0 \geq 0$ **goto** $\beta(n)$ **else goto** $n+1$. *According to the answer of the test the next instruction is determined (where $\beta(n) \in I$). All other registers are not changed.*

Copy: *$n : x_i \leftarrow x_j$, i.e. the content of the "read"-register is copied into the "write"-register. The next instruction is $n+1$; all other registers remain unchanged.*

b) The size *of an $x \in \mathbb{R}^k$ is $size_{\mathbb{R}}(x) := k$. The* cost *of any of the above operations is 1. The cost of a computation is the number of operations performed until the machine halts.*

c) For some $B \subseteq \mathbb{R}^*$ we call a function $f : B \to \mathbb{R}^*$ (BSS-) computable *iff it is realized by a BSS machine over admissible input set B, i.e. iff there is a BSS machine M such that $f \equiv \Phi_M$. Similarly, a set $A \subseteq B \subseteq \mathbb{R}^*$ is* decidable *in B iff its characteristic function is computable. (B, A) is called a* decision problem.

d) All the above notions can directly be transformed as well to computations over the complex numbers. Note that in this case the underlying structure is not ordered. Therefore, the branch instructions are changed to be of the form:

$$n\text{: } \textbf{if } x_0 = 0 \textbf{ goto } \beta(n) \textbf{ else goto } n + 1.$$

Remark 1. For sake of completeness we add that (similarly to $\mathbb{R}^*$) for a finite set Σ (a so called finite alphabet) the set Σ^* in discrete complexity theory is defined as the set of all words of finite length with letters from Σ. We will mainly use $\{0, 1\}^*$, i.e. the set of all finite binary strings.

Most of the well known boolean time-complexity classes can now be defined analogously over the reals. Here, we give a precise definition of the two main such classes.

Definition 2 *i) A problem $(B, A), A \subseteq B \subseteq \mathbb{R}^*$ is in class* $\mathrm{P}_{\mathbb{R}}$ *(*decidable in deterministic polynomial time*) iff there exist a polynomial p and a (deterministic) real BSS machine M deciding A (in B) such that $T_M(y) \leq p(size_{\mathbb{R}}(y))\ \forall y \in B$.*
Here, for a BSS machine M and an input y the term $T_M(y)$ denotes the number of instructions (as defined above) performed by M on input y until the machine halts (or ∞ if it will compute forever). The value $T_M(y)$ is called the running time *of M on input y.*

ii) (B, A) is in $\mathrm{NP}_{\mathbb{R}}$ *(*verifiable in non-deterministic polynomial time over $\mathbb{R}$ *) iff there exist a polynomial p and a real BSS machine M working on input space $B \times \mathbb{R}^*$ such that*
- *a) $\Phi_M(y, z) \in \{0, 1\}\ \forall y \in B, z \in \mathbb{R}^*$*
- *b) $\Phi_M(y, z) = 1 \implies y \in A$*
- *c) $\forall y \in A\ \exists z \in \mathbb{R}^*\ \Phi_M(y, z) = 1$ and $T_M(y, z) \leq p(size_{\mathbb{R}}(y))$*

The vector z in complexity theory is usually called a guess *or a* witness.

iii) A problem (B_1, A_1) is called BSS-reducible in polynomial time *to a problem (B_2, A_2) iff there exists a BSS machine M running in polynomial time such that $\Phi_M : B_1 \to B_2$ and $\forall\ y \in B_1\ \Phi_M(y) \in A_2 \Leftrightarrow y \in A_1$.*

iv) A problem in $\mathrm{NP}_{\mathbb{R}}$ *is* $\mathrm{NP}_{\mathbb{R}}$*-complete iff every other problem in* $\mathrm{NP}_{\mathbb{R}}$ *can be reduced to it in polynomial time (in the BSS model).*

v) If we work over the complex numbers, we define the corresponding classes $\mathrm{P}_{\mathbb{C}}$ *and* $\mathrm{NP}_{\mathbb{C}}$ *in a similar manner. The same holds true for the notion of* $\mathrm{NP}_{\mathbb{C}}$*-completeness. A major difference to the real number model is that branches are of the form "is $z = 0$?", i.e. no ordering is available.*

For what follows it will be important to note that the guess z in the above definition of $\mathrm{NP}_{\mathbb{R}}$ is taken from an uncountable space. This is very much different to the classical setting where the guess is a string in $\{0, 1\}^*$. Since the length of the guess is polynomially bounded in the length of the input, over finite alphabets the search space is finite, though exponentially large.

Some remarks might be in order to clarify the concepts. If we consider a data-set of rational numbers the algebraic size measure is bounded from above by the bit-size measure.

On the other hand, the guess space for a problem in $\mathrm{NP}_{\mathbb{R}}$ is much larger. It is not obvious in how far these properties influence the intrinsic difficulty of the P versus NP question in the different models. There are good reasons to believe that the questions are of the same difficulty both for the discrete and the real model, see [14] for interesting results into this direction. The model over $\mathbb{C}$ in its pure form does not include any of the operations $z \to re(z), z \to im(z), z \to \overline{z}$ or $z \to |z|$. The presence of already one of them would imply the real numbers to be decidable in the complex model. Since this is not the case for the original model over $\mathbb{C}$, none of the above functions can be computed in it. However, many variations of the models are possible: We can represent $\mathbb{C}$ as $\mathbb{R}^2$ and allow real operations (that is, we basically consider the real model; this done below in relation with the HQS problem). We might also restrict the real model to equality testing only. Note, however, that the resulting complexity theory is quite different from the one we are dealing with here. It can be shown that, if P = NP holds true over an infinite field and a model of computation which just uses branches on equality, the field has to be algebraically closed (see [23] and [31] for details). Thus, if the reals are just considered with equality branching the P versus NP question is already answered in the negative. Finally, there are also studies concerning the complex model additionally equipped with the map $z \to \overline{z}$; for example in [35] this model is used in relation with an analysis of globally converging algorithms for zero-finding of polynomials.

It is obvious how to define further real or complex complexity classes; for our purposes the above will be sufficient.

2 Mathematical Optimization

The problems of linear and quadratic programming have been studied quite intensively in mathematical optimization. In particular, there is a major interest in the question whether linear programming admits polynomial time algorithms in algebraic models of computation (see for example [27, 33, 36, 39] and the literature cited in there). Also relations between discrete and continuous aspect of optimization problem have become more and more crucial (see [20]).

As decision problems linear and quadratic programming can be defined via

$$\text{(LP)} \quad \begin{cases} \text{INSTANCE: given } f \in \mathbb{R}[x_1, \dots, x_n], \deg(f) = 1 \text{ and } A \in \mathbb{R}^{m \times n}\ ,\ b \in \mathbb{R}^m \\ \\ \text{QUESTION: is } min\{f(x) | x \in \mathbb{R}^n, A \cdot x \le b\} \le 0\ ? \end{cases}$$

and

$$\text{(QP)} \quad \begin{cases} \text{INSTANCE: given } f \in \mathbb{R}[x_1, \dots, x_n], \deg(f) = 2 \text{ and } A \in \mathbb{R}^{m \times n}\ ,\ b \in \mathbb{R}^m \\ \\ \text{QUESTION: is } min\{f(x) | x \in \mathbb{R}^n, A \cdot x \le b\} \le 0\ ? \end{cases} ,$$

respectively.

In the Turing model the complexity behavior of both problems is well analyzed. Here, the inputs (i.e. the coefficients of f and the entries of A and b) are restricted to be rational numbers. The size of an input is the number of bits necessary to represent it as a string over

$\{0,1\}$. And the running time of an algorithm is defined as the number of bit operations it performs. In this model it is known that LP is polynomially time solvable (w.r.t. "input-bit-size") whereas general (nonconvex) QP is NP-complete (cf. [16], [17], [29]).

Let us consider the common LP methods in more detail:

Using the fact that the minimum of a linear objective function on a (bounded) polyhedron $M : \{x \in \mathbb{R}^n | A \cdot x \leq b\}$ (if it exists) is obtained in a vertex, the simplex method tries to decrease the objective value by moving from one vertex of M to another. The number of arithmetic operations $+, -, *, :$ together with the number of tests whether specific intermediately computed values are negative in every step from one vertex to another can be bounded by a function which only depends on the "geometric" size of the problem, i.e. the dimension n of the underlying space and the number m of constraints. However, the number of vertices attained before a solution is obtained in the worst case is exponentially depending on $n \cdot m$; that is the reason why the Simplex method is <u>not</u> a polynomial time method for LP in the Turing model (and in the real number model as well). In 1979, Khachiyan proved (rational) LP to belong to P by using the so-called ellipsoid method; in 1984, Karmarkar invented the first interior point method for LP. His idea of numerically following a specific "central" path in the interior of M has initiated the development of numerous algorithms in mathematical programming. Even though both the ellipsoid and the interior point methods are polynomial time methods for LP, they lack the property of the Simplex algorithm of having a running time which can be bounded by a function solely depending on n and m (even though, of course, this bound for the Simplex method is exponential; cf. however also the remarks w.r.t. [33] below). The polynomiality of ellipsoid and interior-point methods essentially relies on incorporating the bit size L of the problem instance into the size measure of the underlying complexity model. In fact, as was shown by Traub and Woźniakowski [38] for the ellipsoid method, if we only count the number of arithmetic operations and tests with respect to the geometric size $n \cdot m$ then the above mentioned polynomial time algorithms for LP have an unbounded running time for instances of some fixed (geometric) size. Let us briefly sketch the underlying idea of [38]: Consider two Linear Programs in feasibility form

$$(1) \quad \begin{cases} \text{INSTANCE: given } A = [a_{ij}] \in \mathbb{N}_0^{m \times n} \ , \ b = (b_1, \ldots, b_m) \in \mathbb{N}_0^m \\ \text{QUESTION: is there an } x \in \mathbb{R}^n \text{ s.t. } A \cdot x \leq b \ ? \end{cases}$$

and

$$(2) \quad \begin{cases} \text{INSTANCE: given } A = [a_{ij}] \in \mathbb{N}_0^{m \times n} \ , \ b = (b_1, \ldots, b_m) \in \mathbb{N}_0^m \ , \ t \in \mathbb{N} \\ \text{QUESTION: is there an } x \in \mathbb{R}^n \text{ s.t. } t \cdot A \cdot x \leq b \ ? \end{cases}$$

Here, t is considered as a parameter.

The problem sizes in the bit measure are

$$L_1 = \sum_{i,j} \lceil \log(a_{ij}) \rceil + \sum_i \lceil \log(b_i) \rceil + m \cdot n$$

and

$$L_2 = \sum_{i,j} \lceil \log(t \cdot a_{ij}) \rceil + \sum_i \lceil \log(b_i) \rceil + m \cdot n = L_1 + m \cdot n \cdot \log(t) \quad ,$$

respectively. Obviously, x is a solution for (2) if and only if $t \cdot x$ solves (1). Khachiyan's algorithm now constructs families $\{E_i^{(1)}, i \in \mathbb{N}\}$ resp. $\{E_i^{(2)}, i \in \mathbb{N}\}$ of ellipsoids; it stops as soon as ellipsoids $S^{(1)}$ resp. $S^{(2)}$ are obtained such that

$$vol(S^{(1)}) \leq 2^{-(n+1)\cdot L_1}$$

and

$$vol(S^{(2)}) \leq 2^{-(n+1)\cdot L_2} = 2^{-(n+1)\cdot L_1} \cdot \frac{1}{t^{(n+1)\cdot m \cdot n}} \ .$$

Moreover,

$$vol(E_1^{(1)}) = \mu_n \cdot \sqrt{2^{L_1 \cdot n}} < \mu_n \cdot \sqrt{2^{L_2 \cdot n}} = vol(E_1^{(2)})$$

(where μ_n denotes the volume of the n-dimensional unit ball).

The factor by which the volume of two subsequent ellipsoids constructed during the algorithm decreases only depends on the dimension n of the underlying space, but not on L_1 or L_2. Thus, if t tends to infinity then the same holds true for the number of steps (even arithmetic ones) performed by the ellipsoid method (note $vol(E_1^{(2)}) > vol(E_1^{(1)})$ and $\lim\limits_{t\to\infty} vol(S^{(2)}) = 0$).

The bit complexity measure absorbs this effect by increasing the input size of system (2) when the value of t increases. However, considering the geometric problem size $m \cdot n$, the above reasoning shows Khachiyan's algorithm not to work in polynomial time any longer. The same effect can be noted when dealing with interior-point methods.

There arise some interesting questions in relation with a change of the computational model from the Turing machine to real number machines such as

(i) does LP remain in a corresponding class like P in the real number model?
(ii) does a problem like QP maintain its universal difficulty in such a setting?

If we just count arithmetic operations and assume exact computation, there is no need to restrict problem data to be rational. Therefore, the above questions can be perfectly addressed in the BSS model as

(i) does LP belong to $P_{\mathbb{R}}$?
(ii) Is QP $NP_{\mathbb{R}}$-complete?

We have seen above that a positive answer towards (i) would probably ask for a completely new kind of algorithm. There has been very interesting work into this direction (see [27], [36], [39]). Recently, Renegar [33] announced an interior-point method running in simply exponential time in the algebraic size of the problem and thus attaining similar worst-case theoretical bounds as the Simplex method. Problem (i) to our knowledge at the moment is still open.

Let us now consider in detail problem (ii). Whereas the LP problem seems to become more difficult in real number models, next we will substantiate the conjecture that QP is not complete for the real complexity class $NP_{\mathbb{R}}$. Of course, as long as the $P_{\mathbb{R}}$ versus $NP_{\mathbb{R}}$ problem is open we cannot expect a definite statement such as "QP is not $NP_{\mathbb{R}}$-complete".

In order to grasp one of the basic reasons which probably prevent QP from being $NP_{\mathbb{R}}$-complete consider the most natural non-deterministic verification algorithm establishing membership of QP in $\mathrm{NP}_{\mathbb{R}}$: first, guess a vector $x \in \mathbb{R}^n$; now check whether x is feasible, i.e. $A \cdot x \le b$ and if the latter holds, plug x into the objective function f, evaluate $f(x)$ and accept if and only if $f(x) \le 0$. This verification algorithm runs in polynomial (BSS) time with respect to the real size of the QP instance, which is of order $O(n^2 + (n+1) \cdot m)$.

The basic difference between non-determinism as it is modeled in classical and in BSS complexity is the structure of the search space. Above we have used a continuous search space for guessing a feasible point. That is, the guess which is used for the verification in principle can be taken from the space $\mathbb{R}^*$. Hence, there are uncountable many choices for such a vector. In the Turing model a NP verification algorithm just performs a search in a discrete space: there is only a finite number of choices which can be made by the transition relation of the non-deterministic machine at each move. A naturally arising question is in how far a reduced search space would influence the expressiveness of class $\mathrm{NP}_{\mathbb{R}}$; what happens if, for non-deterministic verification, only a discrete search can be performed? The corresponding class of problems was introduced into the BSS theory by Cucker and Matamala [11] and, independently, by Poizat [15].

Definition 3 *A decision problem (B, A) in $\mathrm{NP}_{\mathbb{R}}$ belongs to the class $\mathrm{DNP}_{\mathbb{R}}$ (*digital $\mathrm{NP}_{\mathbb{R}}$*) if and only if the guess space for a verification algorithm can be reduced to $\{0,1\}^*$, i.e. (compare Definition 2) there exist a polynomial p and a BSS machine M working on input space $B \times \{0,1\}^*$ such that*

a) $\Phi_M(y,z) \in \{0,1\}\ \forall y \in B, z \in \{0,1\}^*$
b) $\Phi_M(y,z) = 1 \implies y \in A$
c) $\forall y \in A\ \exists z \in \{0,1\}^*\ \Phi_M(y,z) = 1$ *and* $T_M(y,z) \le p(size_{\mathbb{R}}(y))$

Note that $\mathrm{P}_{\mathbb{R}} \subseteq \mathrm{DNP}_{\mathbb{R}} \subseteq \mathrm{NP}_{\mathbb{R}}$ and none of the inclusions is known to be proper. A typical example problem in $\mathrm{DNP}_{\mathbb{R}}$ is the classical 3-SAT problem (see [1]), now considered as a problem to be solved with a real number machine. Given a Boolean formula $\phi(x_1, \ldots, x_n)$ in conjunctive normal form, a satisfying assignment for the Boolean variables $x_1, \ldots, x_n$ can be guessed in $\{0,1\}^n$ and verified in polynomial time.

On the other hand, for many problems in $\mathrm{NP}_{\mathbb{R}}$ it is not at all obvious whether a verification algorithm in $\mathrm{DNP}_{\mathbb{R}}$ exists. We will now argue that 1.) QP belongs to $\mathrm{DNP}_{\mathbb{R}}$ and 2.) $\mathrm{NP}_{\mathbb{R}}$-completeness of a problem in $\mathrm{DNP}_{\mathbb{R}}$ would imply some consequences widely believed not to hold true.

Theorem 4. *([25])* QP *belongs to the class* $\mathrm{DNP}_{\mathbb{R}}$.

The proof is basically due to the fact that polynomials of degree 2 behave well with respect to their minima on polyhedra. Consider, for example, the polynomial $p(u,t) := (u \cdot t - 1)^2 + t^2$ for $(u,t) \in \mathbb{R}^2$. An easy calculation shows $inf(p) = 0$ whereas $min(p)$ does not exist. It is known since long time by a result of Eaves that such a behavior does not occur among quadratic polynomials.

Lemma 1. *([13]) Let $f \in \mathbb{R}[x_1, \ldots, x_n]$ be of degree at most two, $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$.*

a) If f is bounded from below on $M := \{x \in \mathbb{R}^n | A \cdot x \le b\}$ then it attains its minimum on M.
b) If f is unbounded from below on M then it is unbounded on a half-ray through M.

Proof (of Theorem 4). The lemma can be applied to settle the statement of the theorem. Obviously, given a QP instance (f, A, b), we cannot guess any longer a feasible point since it necessarily consists of real components. However, using the lemma some other discrete information can be taken for coding a feasible point x, which is then computed by a deterministic subprogram and afterwards plugged into f in order to check $f(x) \leq 0$.

We will briefly describe this kind of digital information. The main information a discrete guess in $\{0,1\}^*$ will provide is about boundedness of the objective function respectively some other QP subproblems arising during the verification procedure. Suppose the first component of a guess to be "1" ; it is interpreted as boundedness of f on M, and part a) of the lemma allows us to switch to a LP subproblem. Here, we use the fact that first-order optimality conditions decrease the degree of the objective function by one. If the guess gives the component "0" then we assume to be in situation b) of the lemma. The assertion can be applied to eliminate one of the n variables and to obtain a QP subproblem in one variable less. Continuing iteratively this argument we finally end up in an LP subproblem. It remains to show LP $\in$ $\mathrm{DNP}_{\mathbb{R}}$.

Towards this end the LP problem is transformed into its feasibility form, say $\tilde{A} \cdot y \leq \tilde{b}$. If this problem is feasible at all, there exists an admissible point on a face of the corresponding feasible set which is of lowest dimension. Such a face can be described by some of the inequalities of $\tilde{A} \cdot y \leq \tilde{b}$ which are active in the solution. The guess is used to code those inequalities being active. The corresponding solution x is computed by solving a linear system of (active) equations. If $f(x) \leq 0$, the original QP instance is accepted; and in case the input is a solvable QP instance and all digits of a guess provide the correct information the computed x indeed will satisfy $f(x) \leq 0$. Thus QP $\in$ $\mathrm{DNP}_{\mathbb{R}}$. □

The theorem shows that, although defined as a real number problem, QP has an internal discrete structure. This structure is hidden behind the non-determinism and probably causes the problem to be not of universal difficulty for class $\mathrm{NP}_{\mathbb{R}}$ in the real number model. In order to substantiate this conjecture it is helpful to study a problem related to algebraic geometry. Recall that an algebraic variety V is a subset of an n-dimensional space (either $\mathbb{R}^n$ or $\mathbb{C}^n$) which is given as the set of common zeros of a system of polynomial equations (over $\mathbb{R}$ or $\mathbb{C}$). An algebraic variety is called irreducible if it cannot be decomposed into two proper disjoint varieties.

We now study the following problem over $\mathbb{R}$: The *homogeneous quadratic systems* decision problem HQS is defined as

$$
(\mathrm{HQS}) \quad \begin{cases} \text{INSTANCE: given } n \in \mathbb{N} \text{ and } f_1, \ldots, f_n \in \mathbb{R}[x_1, \ldots, x_n], \\ \qquad\qquad\quad \text{each } f_i \text{ homogeneous of degree 2;} \\ \\ \text{QUESTION: does there exist a common zero } z \in \mathbb{C}^n \setminus \{0\}? \end{cases}
$$

As is shown in Theorem 5 below HQS belongs to $\mathrm{NP}_{\mathbb{R}}$; to the best of our knowledge it seems to be unknown at the moment whether it is $\mathrm{NP}_{\mathbb{R}}$-complete.

It is well known that for every n the set V_n of solvable systems in HQS builds an irreducible algebraic variety. More explicitly, if a homogeneous quadratic system $f = (f_1, \ldots, f_n)$ is represented by its coefficient vector in some space $\mathbb{R}^N$ then V_n corresponds to an algebraic variety in $\mathbb{R}^N$ which is given as zero-set of the so-called resultant polynomial RES_n. Resultant polynomials are an extension of determinants to nonlinear systems of polynomial equations. In the HQS problem above we look for complex zeros in order to follow closely the commonly used introduction of resultants; for more about that see [22].

The problem of computing the above family $\{RES_n\}_{n\in\mathbb{N}}$ of particular resultant polynomials is conjectured to be extremely difficult. This is due to its relations with computing mixed volumes and permanents (see [8], [34]).

In view of the QP problem and its possible $\mathrm{NP}_{\mathbb{R}}$-completeness now resultants come into play. The next theorem states that QP is of universal difficulty in $\mathrm{NP}_{\mathbb{R}}$ only if the problem "computation of resultants" , which is generally assumed to be difficult, is easy (modulo some conditions). It is thus reasonable to conjecture non-completeness of QP for $\mathrm{NP}_{\mathbb{R}}$.

Theorem 5. *([25]) If* QP *(or any other problem in* $\mathrm{DNP}_{\mathbb{R}}$*) would be* $\mathrm{NP}_{\mathbb{R}}$*-complete then particular multiples of the resultant polynomials were computable in (non-uniform) polynomial time on a Zariski-dense subset of* V_n *for all* $n \in \mathbb{N}$.

Proof. First note that HQS is a member of $\mathrm{NP}_{\mathbb{R}}$. The guess encodes real and imaginary part of all components of a potential zero. Assuming QP to be $\mathrm{NP}_{\mathbb{R}}$-complete together with Theorem 4 yields the existence of a $\mathrm{DNP}_{\mathbb{R}}$-verification algorithm M for HQS. The argument now is non-uniform and makes use of the properties of the V_n's; for fixed $n \in \mathbb{N}$ the verification algorithm M terminates its computation within a fixed number of steps. Because solely digital guesses are made there is only a finite number of different guesses used to verify membership in V_n. The irreducibility of V_n guarantees one of these guesses, say g_0, to verify already membership of Zariski-almost all solvable systems in V_n. Fixing the guess g_0 we turn M into a deterministic algorithm, still working in polynomial time (but, of course, solving the HQS problem only on a specific subset of inputs from $\mathbb{R}^N$). A similar argument, combined with an application of Hilbert-Nullstellensatz, establishes this deterministic algorithm to compute a multiple of RES_n on another Zariski-dense subset of the variety V_n. □

A closer information on how the particular multiple looks like is not provided by the Hilbert-Nullstellensatz. The non-uniformity in the statement of the theorem refers to the fact that we argued for a fixed dimension n. The issue of non-uniformity is addressed more precisely again in section 3 below.

The above results substantiate the conjecture that linear and quadratic programming behave quite different if studied under a discrete or a continuous point of view. For LP this is due to the sensitivity of the known polynomial time algorithms towards scaling; for QP it is due to the reduced structure of the problem which allows to perform a discrete search for obtaining a fast verification. Because of this structure QP seems to be too weak to capture the complexity of those other problems for which a continuous search is intrinsic in order to obtain such verification procedures.

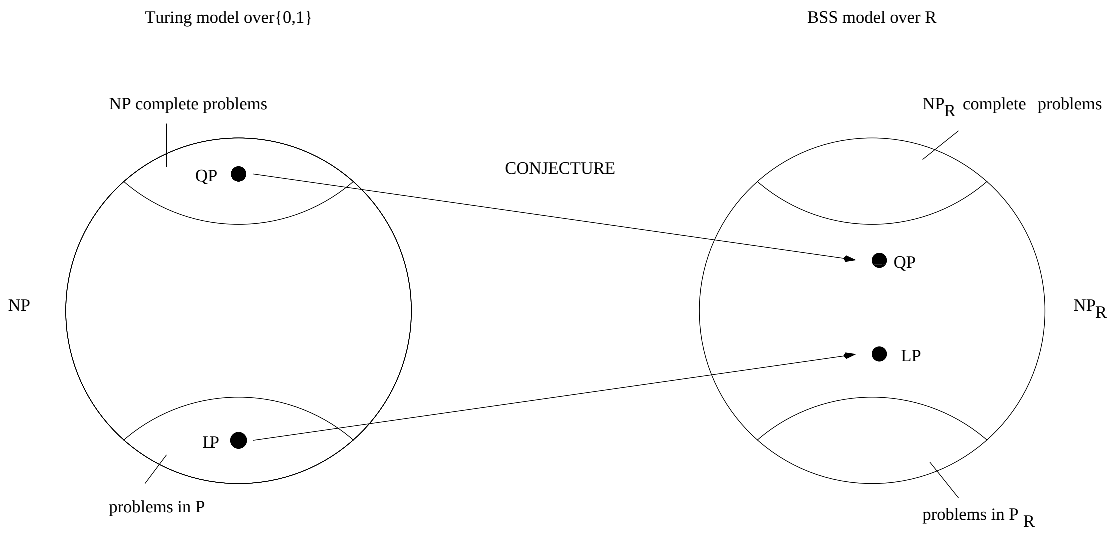
To finish this section another recent approach for studying the complexity of mathematical optimization problems should at least be mentioned briefly. Following the tradition in numerical analysis, Renegar [32] introduces a notion of efficient algorithms for LP based on the conditioning of a particular instance. A fundamental difference with respect to classical and BSS complexity is the information available about a problem instance. A linear program, for example, is specified via rational input data approximating the true (possibly real valued) instance within a given error bound. Thus the information is only partial. Another difference involves the size measure for problems: it incorporates the condition of the problem instance and is not known in advance at the beginning of a solution algorithm.

An efficient algorithm is now defined to be a method which either correctly solves in classical polynomial time the given problem (and thus all possible problems within the error tolerance) or answers that the given accuracy of the data does not suffice to make a correct decision because of the conditioning of the problem. The time the algorithm is given to detect the latter is then depending on that condition number. For more details and interesting results into this direction dealing with LP and QP see [32],[40] and the literature cited there. In [12] related questions are studied for general (nonlinear) polynomial problems.

Let us finally mention that only having partial information available is the typical situation in Information Based Complexity (IBC, see [37]). Again, such a computational approach focuses on some other complexity aspects of problems, for example the number of accesses to specific information operators. Concerning the use of the BSS model in IBC we refer to [30].

3 On the structure of NP in different settings

Comparing the statement of Theorem 5 with the situation for rational mathematical programs in the Turing model we have a situation as shown in the figure.



Of course, the conjectures indicated by the arrows would imply $P_{\mathbb{R}} \neq NP_{\mathbb{R}}$ and thus are probably hard to establish. However, another (easier) question arises from the above figure. It deals with the intrinsic structure of the complexity class $\mathrm{NP}_{\mathbb{R}}$. If we suppose $\mathrm{P}_{\mathbb{R}}$ to be different from $\mathrm{NP}_{\mathbb{R}}$ then the conjecture implies the existence of some "intermediate" problems, i.e. problems being neither $\mathrm{NP}_{\mathbb{R}}$-complete nor polynomial time solvable over $\mathbb{R}$. But do such problems exist at all? The corresponding statement over finite alphabets was shown to be true by Ladner:

Theorem 6. *([21]) If* $\mathrm{P} \neq \mathrm{NP}$ *then there exist problems in* $\mathrm{NP} \setminus \mathrm{P}$ *not being* NP*-complete.*

We now want to discuss this problem within the BSS model. It turns out that the more general viewpoint also gives a clear idea about the intrinsic reason why Ladner's theorem holds true.

Here is a short description of Ladner's proof. Working over a finite alphabet like $\{0,1\}$ the families $\mathcal{P}$ of all polynomial time machines on the one hand side and $\mathcal{R}$ of all polynomial time oracle machines on the other are effectively enumerable. We take such an effective enumeration of these machine classes and assume $\mathrm{P} \neq \mathrm{NP}$. A problem $\mathcal{A}$ of intermediate complexity is constructed as follows: start with a NP-complete problem and change it on specific input sizes to coincide with an easy (i.e. polynomial time solvable) problem. This is done in such a way that, step by step, alternately both the machines from $\mathcal{P}$ and $\mathcal{R}$ fail: keeping the problem difficult on sufficiently many input sizes the machines from $\mathcal{P}$ will believe it to be a difficult one; making it easy on sufficiently many other input sizes the machines from $\mathcal{R}$ will believe it to be too easy to use it as an oracle for solving another NP-complete problem.

It is obvious that this proof heavily relies on the countability of the underlying structure; the machines from $\mathcal{P}$ and $\mathcal{R}$ must be enumerated in order to be fooled one by one. Therefore, it is not at all straightforward what idea behind the proof allows a transformation to uncountable structures like $\mathbb{C}$ or $\mathbb{R}$. We will now work out that it is a model theoretic property, namely saturation, which is responsible for Ladner's proof to work. The structures $\mathbb{C}$ and $\{0,1\}$ are saturated, giving the intended theorem, whereas $\mathbb{R}$ is not. Only a non-uniform version is currently present. Work presented here is based on [2], [24] and [28].

Let us consider a time bounded decisional BSS machine M over $\mathbb{R}$ or $\mathbb{C}$ respectively, i.e. a machine whose running time on inputs of dimension n is bounded by some function $T(n)$ solely depending on n. To M there corresponds a finite set $\underline{c} := \{c_1, \ldots, c_k\}$ of machine constants, either complex or real. Thus, M can be seen as a machine with two different kind of inputs: the set of machine constants $\underline{c}$ together with the problem input x. As soon as these are fixed M can be described in discrete terms. This will be extremely important for what follows.

If an input dimension n is fixed, the computation of M on $\mathbb{C}^n$ (resp. on $\mathbb{R}^n$) can be described by a first-order formula ρ_n over the corresponding structure. This formula again depends on x and $c_1, \ldots, c_k$; moreover, M accepts an input x if and only if $\rho_n(x, c_1, \ldots, c_k)$ holds (see [28]). The elements in the family $\{\rho_n\}_{n \in \mathbb{N}}$ are first-order formulas (for a definition see below). The family itself is uniform in the sense that it is generated by the same BSS machine. In particular, the choice for the constants c_i is the same for all $n \in \mathbb{N}$. Note that especially for each problem in $\mathrm{NP}_{\mathbb{C}}$ (or $\mathrm{NP}_{\mathbb{R}}$) such a family of formulas exists.

The idea to attack Ladner's theorem over $\mathbb{C}$ or $\mathbb{R}$ is to separate the discrete part of a machine (i.e. the description of its instructions) from its uncountable part (i.e. the input and the machine constants). That way, one can enumerate at least the "basic" part of machines, namely the program, without fixing the constants. The problem arising with this idea is that now different sets of constants can correspond to different input dimensions (respectively formulas ρ_n). This introduces non-uniform features and the question is how to move back to a uniform situation. Here saturation comes into play.

We will first define more precisely what saturation means. Towards this end recall the definition of a first-order formula over the real closed field $\mathbb{R}$. Atomic formulas are of the form $p(x) = 0$ or $p(x) \geq 0$ where p is a real polynomial over variables $x \in \mathbb{R}^n$. The language of first-order formulas is now built up from atomic ones by using the connectives $\wedge, \vee$, logical negation and the first-order quantifiers $\forall, \exists$. First-order here means that these quantifiers only run over elements in $\mathbb{R}$, not over subsets etc. First-order formulas over $\mathbb{C}$ (or over $\{0,1\}$) are defined similarly. For example, over the complex numbers atomic formulas are defined to be the form $p(x) = 0$ only.

Next, let a countable family $\{\psi_n(z)\}_{n\in\mathbb{N}}$ of first-order formulas over a structure $\mathcal{S}$ with free variables $z = (z_1, \ldots, z_k)$ be given. Such a family is called *finitely satisfiable* if, for all $n \in \mathbb{N}$, there exists a $\overline{z} \in \mathcal{S}^k$ such that $\psi_s(\overline{z})$ holds for all $1 \leq s \leq n$. And it is *satisfiable* if $\exists \overline{z} \in \mathcal{S}^k$ such that $\psi_n(\overline{z})$ holds for all $n \in \mathbb{N}$. Finally, the structure $\mathcal{S}$ is called *saturated* (more precisely: ω*-saturated*) iff every finitely satisfiable family $\{\psi_n(z)\}_{n\in\mathbb{N}}$ is satisfiable. It is well known from model theory that the structures $\{0,1\}$ and $\mathbb{C}$ are saturated, whereas $\mathbb{R}$ is not. The latter can be seen with the following easy example: for $n \in \mathbb{N}$ and $c \in \mathbb{R}$ choose $\rho_n(c) : c > n$.

Obviously, saturation is related to a kind of uniformity since the existence of one satisfying assignment corresponds to the use of a single uniform set of machine constants as explained above. With respect to our structural questions about the various NP classes saturation now can be used as follows: as already explained before we consider the basic discrete part of polynomial time BSS machines. We get an effectively enumerable family of basic BSS machines $M(\underline{c})$, where the $\underline{c}$ corresponds to the machine constants and are not fixed a priori. Hence, the class of all polynomial time BSS machines is partitioned into countably many subclasses, one for all those machines which coincide on their program and the number of machine constants being used, but not necessarily on the particular value assigned for it on different input dimensions. Moreover, to every such subclass there corresponds a family $\{\rho_n(\underline{c})\}_{n\in\mathbb{N}}$ representing the machine computation.

Let us fix for the moment $\mathbb{C}$ as underlying structure. Our goal is to construct a problem belonging to $\mathrm{NP}_{\mathbb{C}} \setminus \mathrm{P}_{\mathbb{C}}$ not being $\mathrm{NP}_{\mathbb{C}}$-complete. Towards this end consider a $\mathrm{NP}_{\mathbb{C}}$-complete problem, for example the complex Hilbert-Nullstellensatz HNS problem (cf. [4]). This is the problem to decide, whether a system of complex polynomial equations $f = (f_1, \ldots, f_s) = 0$, each f_i of degree 2, has a common complex root. It is easy to see that to this problem there corresponds a family $\{h_n\}_{n\in\mathbb{N}}$ of constant-free first-order formulas (see [28]). Recall that this family is related to the decision problem in the sense that an input f of size n is a solvable instance of the Hilbert-Nullstellensatz problem if and only if $h_n(f)$ holds true over $\mathbb{C}$.

In order to construct a problem not in $\mathrm{P}_{\mathbb{C}}$ a diagonalization argument can be performed as follows. The non-completeness then is obtained by a similar argument. The Hilbert-Nullstellensatz problem is turned into an easier one by making it trivial on specific input dimensions. However, this is done in such a way that, step by step, all polynomial time machines belonging to one of the above introduced subclasses are fooled. More explicitly, we look for an input dimension n_0 such that the formula

$$\Theta_n \equiv \forall \underline{c} \in \mathbb{C}^k \; \exists f \in \text{HNS such that } f \text{ has a common zero and} \\ \neg(\rho_{n_0}(f, \underline{c}) \Longleftrightarrow h_{n_0}(f))$$

holds true. The formula just asserts that the machine M generating the sequence $\{\rho_n\}$ will not give the correct answer for the HNS problem for all inputs f of size n_0 . Thus, M cannot be used to solve an easier subproblem of HNS as long as the latter corresponds to HNS at least on dimension n_0 . This argument works independently of which values are assigned to the constants of M. If such a dimension n_0 exists it can be computed by a quantifier elimination procedure.

Concerning the existence of such a dimension $n_0 \in \mathbb{N}$ (in fact infinitely many) we have to address briefly the issue of non-uniform algorithms. If we assume that our initial $\mathrm{NP}_{\mathbb{C}}$-complete problem cannot be solved by a family of so called non-uniform polynomial time BSS machines then n_0 exists.

Non-uniformity is a central issue in complexity theory. It refers to the solution of a problem by using a family of algorithms, one for each input dimension. In the definition of the classes P and NP (no matter whether over $\{0,1\}, \mathbb{R}$, or $\mathbb{C}$) we have also in principle another algorithm for each input dimension, but they are uniform in the sense that they are all given by one single machine (Turing or BSS). If we weaken the conditions on how the different algorithms can be obtained we end up with a notion of non-uniform complexity. The most important such complexity classes are denoted by $\mathcal{C}/poly$ as non-uniform version of a uniform complexity class $\mathcal{C}$, see [1].

In the above reasoning, the machine M gives rise to a non-uniform family of algorithms; even though the "skeleton" of the algorithms is the same for all dimensions, non-uniformity is caused by the potentially different choices of the machine constants $c_1, \ldots, c_k$ for each dimension. There is no uniform rule which, given an input dimension, computes the correct vector of machine constants.

In saturated structures we can switch from different such vectors of constants to a uniform one. Thus, the above assumption on the non-solvability of HNS results in assuming $\mathrm{P}_{\mathbb{C}} \neq \mathrm{NP}_{\mathbb{C}}$, turning the argument into a uniform statement. This final argument is valid as well over $\{0,1\}$, and thus we obtain a uniform result both over $\mathbb{C}$ and $\{0,1\}$. For the real numbers it is not applicable, the reason why here only a non-uniform version is obtained. Denoting by $\mathrm{P}_{\mathbb{R}}/poly$ the non-uniform version of class $\mathrm{P}_{\mathbb{R}}$ (to be understood in a similar way as explained above) we have to replace the uniform assumption $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NP}_{\mathbb{R}}$ by a non-uniform one $\mathrm{NP}_{\mathbb{R}} \not\subseteq \mathrm{P}_{\mathbb{R}}/poly$. This results in

Theorem 7. *([24],[2])*

a) Assume $\mathrm{P}_{\mathbb{C}} \neq \mathrm{NP}_{\mathbb{C}}$. *Then there exists a non-complete problem in* $\mathrm{NP}_{\mathbb{C}} \setminus \mathrm{P}_{\mathbb{C}}$.
b) Assume $\mathrm{NP}_{\mathbb{R}} \not\subseteq \mathrm{P}_{\mathbb{R}}/poly$. *Then there exists a non-complete problem in* $\mathrm{NP}_{\mathbb{R}} \setminus \mathrm{P}_{\mathbb{R}}/poly$.
□

The situation over arbitrary real closed fields has been further and deeply studied in [9]. A version of Ladner's theorem in terms of Valiant's theory on computing families of real polynomials is given in [6].

4 Conclusions

In the above sections we have seen some various flavors of studying a problem in different models of computation. Optimization in the well known problems LP and QP has been considered in discrete and continuous models of computation. Their complexity seems to be intrinsically different within these models. Nevertheless, we consider it to be important to investigate these problems further since, for example, the discovery of a $\mathrm{P}_{\mathbb{R}}$ algorithm for LP would certainly have tremendous applications also for practical implementations, whereas the disprove of its existence will avoid unnecessary efforts and lead to focus on other questions such as the incorporation of conditioning into the complexity analysis. On the other hand, the investigation on Ladner's Theorem showed the use of studying questions in a broader framework. It can be helpful to get a clearer view of what is really behind a result. Instead of excluding mutually, we believe the research in various settings to be a completion for understanding all complexity aspects of a computational problem.

5 Acknowledgement

Thanks are due to three unknown referees for their valuable remarks which helped to improve the presentation, and to T. Illes and T. Terlaky for their help and encouragement.

References

1. J.L.Balcázar, J.Diaz, J.Gabarró, Structural Complexity I,II. EATCS Monographs of Theoretical Computer Science, vol. 11 and vol. 22, Springer-Verlag, Berlin, (1988) and (1990).
2. S.Ben-David, K.Meer, C.Michaux, A note on non-complete problems in $NP_{\mathbb{R}}$. Journal of Complexity, **16** , 324–332 (2000).
3. L.Blum, F.Cucker, M. Shub, and S. Smale, Complexity and Real Computation. Springer (1998).
4. 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 American Mathematical Society 21, 1–46 (1989).
5. D.P. Bovet, P. Crescenzi, Introduction to the theory of complexity. Prentice Hall, New York (1993).
6. P. Bürgisser, On the structure of Valiant's complexity classes. In: Proc. of the 15th Symposium on Theoretical Aspects of Computer Science STACS '98, Lecture Notes in Computer Science 1373, Springer, 194–204 (1998).
7. P.Bürgisser, M.Clausen, A.Shokrollahi, Algebraic Complexity Theory. Grundlehren der mathematischen Wissenschaften, vol. 315 (1996).
8. J. Canny, I. Emiris, Efficient incremental algorithms for the sparse resultant and the mixed volume. Journal of Symbolic Computation 20, 117–149 (1995).
9. O. Chapuis, P. Koiran, Saturation and stability in the theory of computation over the reals. Annals of Pure and Applied Logic, **99** , 1–49 (1999).
10. F. Cucker, Machines over the reals and non-uniformity. Mathematical Logic Quarterly 43, 143–157 (1997).
11. F. Cucker, M. Matamala, On digital nondeterminism. Mathematical Systems Theory 29, 635–647 (1996).
12. F. Cucker, S. Smale, Complexity estimates depending on condition and round-off error. Journal of the ACM, vol. 46, no. 1, 113–184 (1999).
13. C.B. Eaves, On quadratic programming. Management Science (Theory) 17 no. 11, 698–711 (1971).
14. H. Fournier, P. Koiran, Are lower bounds easier over the reals ? In: Proc. 30th ACM Symposium on Theory of Computing, 507–513 (1998).
15. J.B. Goode, Accessible telephone directories. Journal of Symbolic Logic, vol. 59, 1, 92–105 (1994).
16. N. Karmarkar, A new polynomial time algorithm for linear programming. Combinatorica, vol. 4, 373–395 (1984).
17. L.G. Khachiyan, A polynomial algorithm in linear programming. Dokl. Akad. Nauk USSR, vol. 244, 1093–1096 (1979); english translation in: Soviet Math. Dokl., vol. 20, 191–194 (1979).

18. Ker-I Ko, Complexity of real functions. Birkhäuser, Basel-Berlin-Boston (1991).
19. V. Kreinovich, A.Lakeyev, J.Rohn, P.Kahl, Computational complexity and feasibility of data processing and interval computations. Kluwer Academic Publishers, Boston/London/Dordrecht (1997).
20. E. Kropat, St. Pickl, A. Rößler and G.-W. Weber, On theoretical and practical relations between discrete optimization and nonlinear optimization. To appear in: Journal of Computational Technologies.
21. R.Ladner, On the structure of polynomial time reducibility. Journal of the ACM 22, 155–171 (1975).
22. F.S. Macauley, The algebraic theory of modular systems. Cambridge Tracts in Mathematics and Mathematical Physics 10, Steckart-Hafner Service Agency, New York (1964).
23. A. Macintyre, K. McKenna, L. van den Dries, Elimination of Quantifiers in Algebraic Structures. Advances in Mathematics 47, 74–87 (1983).
24. G. Malajovich, K. Meer, On the Structure of $\mathrm{NP}_{\mathbb{C}}$. SIAM Journal on Computing Vol. 28, 27–35 (1999).
25. K. Meer, On the complexity of quadratic programming in real number models of computation. Theoretical Computer Science 133, 85–94 (1994).
26. K. Meer, C. Michaux, A survey on real structural complexity theory. Bulletin of the Belgian Mathematical Society 4, 113–148 (1996).
27. N.Megiddo, Towards a genuinely polynomial algorithm for linear programming. SIAM Journal on Computing 12, 347–353 (1983).
28. C. Michaux, $P \neq NP$ over the nonstandard reals implies $P \neq NP$ over $\mathbb{R}$. Theoretical Computer Science 133, 95–104 (1994).
29. K.G. Murty, S.N. Kabadi, Some NP-complete problems in quadratic and nonlinear programming. Mathematical Programming 39, 117–129 (1987).
30. E. Novak, The real number model in numerical analysis. Journal of Complexity 11, 57–73 (1995).
31. B. Poizat, Les petits Cailloux. Aléa (1995).
32. J. Renegar, Incorporating condition measures into the complexity theory of linear programming. SIAM Journal of Optimization 5, 506–524 (1995).
33. J. Renegar, Talk at the Smalefest 2000, 13.- 17.7.2000, City University Hong Kong (2000).
34. M. Shub, Some Remarks on Bezout's Theorem and Complexity Theory. In : M.Hirsch, J.Marsden, M.Shub, eds., From Topology to Computation, Proceedings of the Smalefest, Springer, 443–455 (1993).
35. M. Shub, S. Smale, On the Existence of Generally Convergent Algorithms, Journal of Complexity 2, 2–11 (1986).
36. E.Tardos, A strongly polynomial algorithm to solve combinatorial linear programs. Operations Research 34(2), 250–256 (1986).
37. J.F.Traub, G.W.Wasilkowski, H.Woźniakowski, Information-based complexity. Academic Press (1988).
38. J.F.Traub, H.Woźniakowski, Complexity of linear programming. Operations Research Letters, vol.1, no.2, 59–62 (1982).
39. S. Vavasis, Y. Ye, A primal-dual interior-point method whose running time depends only on the constraint matrix. Mathematical Programming 74, 79-120 (1996).
40. J. Vera, On the complexity of linear programming under finite precision arithmetic. Mathematical Programming 80, 91–123 (1998).
41. G.W. Weber, Semi-infinite optimization, optimal control and some relations with interior point methods. Talk given at 2^{nd} Workshop on Interior Point Methods IPM–2000" in Budapest (2000).
42. K. Weihrauch, Computability. EATCS Monographs on Theoretical Computer Science 9, Springer (1987).