ON GENUINELY TIME BOUNDED COMPUTATIONS

Friedhelm Meyer auf der Heide *
Informatik II, Universität Dortmund,
D - 4600 Dortmund 50, Fed. Rep. of Germany

Abstract: This survey paper presents a complexity theoretical approach to genuinely time bounded computations. Such computations are executed by random access machines with given set $S \subseteq \{+, -, *, DIV, \ldots\}$ of arithmetic operations. The uniform cost measure is assumed, and the input is given integer by integer, not bit by bit. "Genuinely" (also called "strongly" in the literature) means that we measure the time complexity $T(n)$ as the worst case runtime over all inputs consisting of n *integers* (not of n *bits*). Computability and complexity now heavily depend on the operation set S. In this paper genuine complexity classes for given operation sets S are defined, following ideas due to Karpinski and the author from [KaM 88]. Furthermore, results on genuine computability, lower bound methods, as well as examples for complexity gaps and separated complexity classes are surveyed.

1. Introduction

The classical theory of computability and complexity considers uniform computation models like Turing machines, random access machines (RAMs) and their nondeterministic, alternating, parallel... versions, as well as nonuniform models like Boolean circuits, decision trees, or branching programs. Such computation models compute functions $f : A^* \to A^*$ for some finite alphabet A, e.g. $A = \{0, 1\}$, and the runtime of an algorithm is measured by a function $T : I\!N \to I\!N$, where $T(n)$ denotes its worst case runtime taken over all inputs of length n.

On the other hand the efficiency of algorithms considered in the theory (and practice) of the design and analysis of efficient algorithms is usually measured in a different way. Here one has in mind a RAM with given set $S \subseteq \{+, -, *, DIV, \ldots\}$ of arithmetic operations, i.e. an S-RAM. The functions that are computed are typically defined on the integers, i.e. $f : \mathbb{Z}^* \to \mathbb{Z}^*$, and the complexity is measured by a function $T : I\!N \to I\!N$ where $T(n)$ is the worst case runtime taken over all inputs consisting of n *integers*. One assumes the uniform cost measure, i.e. an elementary step (storage access, branching, arithmetic operation from S) takes one unit of time. Examples are sorting (e.g. "Heapsort needs $O(n \log(n))$ steps"), searching, network flow, linear programming, knapsack,

* Supported in part by DFG-Grants ME 872/1-2 and WE 1066/2-1

travelling salesman, problems from computational geometry, etc. Following notions due to Megiddo from [Meg 83], algorithms that are $T(n)$-time bounded in this sense are called *genuinely $T(n)$-time bounded* (or strongly $T(n)$-time bounded).

In this paper we want to describe the foundations of a complexity theory of genuinely time bounded computations. For this purpose we shall define our computation models and corresponding complexity classes. Computability and complexity now critically depends on the set S of arithmetic operations allowed. We shall define "genuinely computable" via the existence of a $T(n)$-time bounded RAM for some $T : I\!N \to I\!N$, i.e. it is not sufficient that the algorithm halts for each input, but there must even be a finite upper bound $T(n)$ on the runtimes for inputs of length n.

With this strong notion of genuine computability it is immediate that we have to take the operation set S into account. Thus we consider *genuine S-computability.*

All known bounds for genuine computations are proved for a nonuniform computation model, namely for *computation trees* as defined e.g. by Ben-Or in [BO 83]. Therefore we shall also define *nonuniform complexity classes.*

The paper is organized as follows:
The complexity classes are introduced in Chapter 2. In Chapter 3 we list some computational problems in order to compare genuine and classical complexity measures. In Chapter 4 we try to characterize the S-computable languages. Chapter 5 presents general lower bound methods like the *component counting lower bound* due to Dobkin/Lipton from [DL 75] and to Ben-Or from [BO 83]. Chapter 6 shows examples of genuine complexity gaps for nonuniform complexity classes. For example, the results from [MadH 84], [MadH 85] imply that, for operations $\{+,-\}$, the nonuniform classes P and PARALLEL (with $2^{poly(n)}$ processors) are identical. In Chapter 7 we shall present separation results due to Karpinski and the author from [KaM 88]. E.g., the above nonuniform gap does not exist if the operations $\{+,-,*\}$ are allowed. Furthermore, we separate the uniform classes P and NP for operations $\{+,-,DIC_c\}$, where DIV_c denotes integer division by a value that only depends on the *number* of input variables but not on their *values*. Finally, in Chapter 8, we conclude with a list of open problems.

This paper is intended to give an introduction to and a survey of the current state of the complexity theory of genuinely time bounded computations. The sketches of proofs are not meant to be selfcontained. Instead they should be looked upon as guided tours through the proof. Afterwards the reader is invited to go back to the most exciting places (i.e. papers) to study them in more detail.

Related Work

There are some related approaches to develope a complexity theory which is not based on bit-manipulations.

Schönhage and Simon have investigated complexity classes of RAMs where the input consists of bits, but where arithmetic operations can be executed at unit cost. It is shown that $PSPACE$ (in the classical sense) is contained in their complexity class P for the operation set $\{+,-,*,2^x,DIV\}$. (Compare [Schö 79],[Sim 79].)

Blum, Shub, and Smale have recently presented a complexity theoretical approach to computations with real inputs. (Compare [BSS 88]).

2. Computation Models and Complexity Classes

Our basic computation model is the random access machine (*RAM*) with fixed arithmetic operation set $S \subseteq \{+, -, *, DIV, \ldots\}$. We call it *S-RAM*. An *S-RAM* M has an unbounded number of registers $R(0)$, $R(1), \ldots$, each of which can store an integer. The computation is directed by a finite program that consists of instructions of the following types: direct and indirect storage accesses, conditional branchings (IF $R(0) > 0$ THEN ... ELSE ...), and arithmetic operations from S. Each of them can be executed at unit cost. For simplicity, the only constants allowed in the program are 0 and 1.
An S-RAM M computes a partial function $f_M : \mathbb{Z}^* \to \mathbb{Z}^*$ as follows. Initially, started with input $(x_1, \ldots, x_n) \in \mathbb{Z}^*$, $n, x_1, \ldots, x_n$ are stored in $R(1), \ldots, R(n+1)$, all other registers contain 0. Finally, if M stops, $(y_1, \ldots, y_m) = f(x_1 \ldots x_n)$ is stored in $R(1), \ldots, R(m)$. M recognizes $L \subseteq \mathbb{Z}^*$ if M computes the characteristic function of L.
M is (genuinely) $T(n)$-time bounded if M executes at most $T(n)$ steps started with any input from $\mathbb{Z}^n$ (uniform cost criterion).
Analogously we can define *probabilistic, nondeterministic, alternating S-RAM*s. Furthermore we can define a *parallel S-RAM* as a *PRAM* consisting of an unbounded shared memory and an unbounded number of processors which are *S-RAM*s with the additional capability of direct and indirect access to the shared memory. The input and output now appear in the shared memory. The local memory of the i-th processor P_i contains i. Furthermore, we assume that P_1 can "wake up" a number $p(n)$ of processors (the active processors for the computation) in unit time (or in time $O(\log(p(n)))$, that would not change our results).
A *parallel S-RAM* is *(genuinely) $T(n)$-time bounded and $p(n)$-processor bounded*, if it executes at most $T(n)$ steps using at most $p(n)$ active processors, if started with any input from $\mathbb{Z}^n$.

With these notions we can define *S-complexity classes:*
S-P :=$\{L \subseteq \mathbb{Z}^*$: there is a polynomially time bounded *S-RAM* M recognizing $L\}$.
Analogously, we define *S-RP* (probabilistic *S-RAM*, probability of error $\leq \alpha < \frac{1}{2}$), *S-NP, S-ALTERNATION*. (We could define genuine analogues to the whole polynomial time hierarchy.)
Furthermore, *S-PARALLEL* :=$\{L \subseteq \mathbb{Z}^*$: there is a parallel $T(n)$-time bounded, $2^{p(n)}$-processor bounded *S-RAM* recognizing L, where T and p are polynomials$\}$.
The *class of S-computable languages* is *S-REC* := $\{L \subseteq \mathbb{Z}^*$: there is a $T(n)$-time bounded *S-RAM* recognizing L, for some total function $T : \mathbb{N} \to \mathbb{N}\}$.

A *nonuniform S-RAM* is an *S-RAM* M that, started with $x_1, \ldots, x_n$, is allowed to execute an arbitrarily complex precomputation only dependent on n before starting

a computation also dependent on the input variables $x_1, \ldots, x_n$. This precomputation yields a program of an *S-RAM* M_n. If, for each n, M_n needs at most $T(n)$ steps for any input from $\mathbb{Z}^n$, then M is $T(n)$-time bounded. M computes $f : \mathbb{Z}^* \to \mathbb{Z}^*$ if M_n computes $f|_{\mathbb{Z}^n}$.
Analogously we get nonuniform versions of probabilistic, nondeterministic, alternating, and parallel *S-RAM*s.
The corresponding *nonuniform complexity classes* are denoted by the prefix *NU* as *NU-S-P*, etc.
All known lower bounds even hold for nonuniform *S-RAM*s, in other words, there is no method known to deduce lower bounds by exploiting uniformity. The nonuniform models normally used for lower bounds are the *S-computation trees, S-CTs*, which are abstractions of nonuniform *S-RAM*s where storage access is not taken into account.

An *S-CT* for n inputs is a rooted binary tree. Nodes v with outdegree 1 are labelled with a function $p_v : \mathbb{Z}^n \to \mathbb{Z}$, $p_v = p_1 \circ p_2$, with $\circ \in S$, p_1, p_2 either previously computed on the path to v, or a constant from $\{0, 1, n\}$, or an input variable x_i. Nodes v with outdegree 2, the branching nodes, are labelled with predicates "$p(x_1, \ldots, x_n) > 0$" where p is computed on the path to v. Leaves are labelled with "accept" or "reject". An input $(x_1, \ldots, x_n) \in \mathbb{Z}^n$ follows the path from the root to a leaf, choosing the left or right branch of a branching node according to whether $(x_1, \ldots, x_n)$ fulfils the predicate at the node or not. $(x_1, \ldots, x_n)$ is accepted if it reaches an accepting leaf.

It is easily seen that a language $L \subseteq \mathbb{Z}^n$ (n fixed) is S-computable iff it can be recognized by an *S-CT*. Even more, the following can be shown.

Lemma 1:
a) Each S-CT of depth t can be simulated by an S-RAM in time $O(t)$.
b) Each S-RAM recognizing some $L \subseteq \mathbb{Z}^n$ in t steps can be simulated by an S-CT of depth $0(t \log(t))$. □

The additional factor $\log(t)$ in b) appears because we have to simulate indirect storage accesses. This can be organized as a binary search in the set of the up to t values currently stored by the *RAM*. The search keys are the addresses of the registers in use. For details see [MadH 85].

If an *S-RAM* or *S-CT* can recognize $L \subseteq \mathbb{Z}^n$, then we call L *nonuniformly S-computable*. Clearly, if $L \subseteq \mathbb{Z}^*$ is genuinely S-computable, then $L \cap \mathbb{Z}^n$ is nonuniformly S-computable for each $n \in \mathbb{N}$.

Finally, we can introduce some further arithmetic operations. DIV_c denotes the integer division with the following restriction: a DIV_c b is only allowed to be executed if b depends only on n, the *number* of input variables, but not on their *values*. Thus b can be looked upon as a number that is computed by an *S-RAM* with input n. Analogously, we can define $\circ_c$ for any other binary operation.

3. Genuinely Time Bounded Algorithms for Problems in P

We shall list three problems in P (in the Turing sense), together with some facts concerning their genuine complexity.

1) **Network Flow** (compare e.g. [PaSt 82])
The original algorithm due to Ford and Fulkerson is not genuinely time bounded, but the value of the maximum flow is a factor in the runtime, even if we assume the uniform cost criterion. In the sequel, many genuine algorithms for $S = \{+, -\}$ have been developed.

2) **Linear Programming** (compare e.g. [PaSt 82])
This problem was shown to be in P by Khachiyan [Kha 79]. (See also [Karm 84].) The runtimes (assuming the uniform cost criterion) of these algorithms contain the binary input length as a factor.
The best known genuine algorithm ($S = \{+, -, *\}$) is the Simplex Algorithm, which is known to have superpolynomial runtime. It is a challenging open problem to design (or disprove the existence of) a genuinely polynomial algorithm for Linear Programming. Steps in this direction can be found in [Meg 83] and [Tar 86].

3) **Linear Diophantine Equations** (compare e.g. [PaSt 82])
Given $\bar{a} \in \mathbb{Z}^n, b \in \mathbb{Z}$, decide whether there is an $\bar{\alpha} \in \mathbb{Z}^n$ with $\bar{a}\bar{\alpha} = b$. (This problem is in P, but it becomes NP-complete, if $\bar{\alpha} \in \mathbb{N}^n$ is demanded, see [SpSt 76].) We shall see in the next chapter that this problem is not $\{+, -, *\}$-computable, i.e. it is not in $\{+, -, *\}$-REC. If we consider computation trees with real inputs, then the problem is not even computable if S contains all analytic functions and DIV, i.e. it is not genuinely computable for any reasonable set of operations (see [BaJuM 88]).

4. Genuine Computability

(i) Nonuniform S-Computability

As observed in Chapter 1, it suffices to consider S-CTs in order to characterize the nonuniformly S-computable languages in $\mathbb{Z}^n$.
For this purpose, we first have to determine which functions can be computed with operations from S if we do not consider branchings.
An S-CT that does not contain branchings is called a *straight line program over S (S-SLP)*. (Note: Usually S-SLPs are allowed to process real-valued inputs, and arbitrary constants are allowed, not only those from $\{0, 1, n\}$ (compare [BoMu 75])).

Let $F(S, n)$ ($F(S, n, t)$) denote the set of all functions computable by a finite S-SLP (of length t) for n variables.

An (S,n)-*system* ((S,n,t)-*system*) is a Boolean combination of a finite number of (at most 2^t) inequalities "$p(x_1,\ldots,x_n) > 0$", where $p \in F(S,n)$ ($p \in F(S,n,t)$).

Examples:

- $S = \{+,-\} : F(S,n)(F(S,n,t))$ is (contained in) the set of all linear functions with integer coefficients (from $\{-n2^t,\ldots,n2^t\}$).
- $S = \{+,-,*\} : F(S,n)(F(S,n,t))$ is (contained in) the set of all polynomials with (degree at most 2^t and) integer coefficients (from $\{-n^{2^t},\ldots,n^{2^t}\}$).

Now consider an *S-CT* D with depth t.
All functions computed at any node of D are by definition from $F(S,n,t)$. The set $c(v)$ of inputs arriving at a node v is defined by the inequalities at the branching nodes on the path to v. Thus $c(v)$ is the solution set of an (S,n,t)-system. As the accepted language is the union over the sets $c(v)$, v accepting leaf, we obtain:

Theorem 1: *a) $L \subseteq \mathbb{Z}^n$ can be recognized by an S-CT iff L is the solution set of an (S,n)-system.*
b) If $L \subseteq \mathbb{Z}^n$ can be recognized by an S-CT of depth t, then L is the solution set of an (S,n,t)-system. □

Thus, for $S = \{+,-\}$ ($S = \{+,-,*\}$), $L \subseteq \mathbb{Z}^n$ is S-computable, iff L is the solution set of a Boolean combination of linear (polynomial) inequalities with integer coefficients.

Corollary 1: *The problem "Linear Diophantine equations" (i.e. recognize $L := \{(\bar{a},b),\ \bar{a} \in \mathbb{Z}^n, b \in \mathbb{Z}$, there is $\bar{\alpha} \in \mathbb{Z}^n$ with $\bar{a}\bar{\alpha} = b\}$ is not $\{+,-,*\}$-computable.*

Proof (sketch): In a $\{+,-,*\}$-system with solution set L, the linear term $T_{\bar{\alpha}} = \bar{a}\bar{\alpha} - b$ would have to appear as a factor in at least one of the polynomials defining the system, for each $\bar{\alpha} \in \mathbb{Z}^n$. As each polynomial only has finitely many such linear factors, we would need infinitely many polynomials. □

The classes $F(\{+,-\},n)$ and $F(\{+,-,*\},n)$ are well understood because of their algebraic structure. Exploring properties of these linear or polynomial functions is a classical discipline in algebra.

This changes completely if we also allow the DIV-operation. Until now characterizations of S-computability are not known for $S = \{+,-,DIV\}$ or $S = \{+,-,*,DIV\}$ for languages in $\mathbb{Z}^n$ with $n > 1$.

(ii) Nonuniform S-Computability for Languages in $\mathbb{Z}$

For languages $L \subseteq \mathbb{Z}$, S-computability is easily characterized for $S = \{+,-\}$ and $S = \{+,-,*\}$, using the observations from (i).

Let $L \subseteq \mathbb{Z}$. The following three statements are equivalent:

- *$L \subseteq \mathbb{Z}$ is $\{+,-\}$-computable.*
- *L is $\{+,-,*\}$-computable.*

- *One of the sets* L^+ *and* $\mathbb{N} \setminus L^+$ *is finite, and one of the sets* L^- *and* $\mathbb{Z} \setminus (L^- \cup \mathbb{N})$ *is finite, where* $L^+ = L \cap \mathbb{N}$, $L^- = L \setminus \mathbb{N}$.

This characterization is no longer true for $S = \{+, -, DIV\}$ or $S = \{+, -, *, DIV\}$. For example, for fixed $d, a \in \mathbb{Z}$, the *arithmetic progression* $\{a + \lambda d, \lambda \in \mathbb{Z}\}$ can be recognized easily by the test $d \cdot ((x - a)DIV\ d) = x - a$. This test can even be executed with $S = \{+, -, DIV\}$, because the multiplication with the constant d can be simulated by additions.
Note that, for $d > 1$, the above arithmetic progression is not $\{+, -, *\}$-computable. Clearly, all finite languages are $\{+, -, DIV\}$-computable.

It turns out that these two types of languages, arithmetic progressions and finite languages, suffice to characterize the $\{+, -, *, DIV\}$-computable languages in $\mathbb{Z}$:
We call L an *AP-language* (*AP* stands for arithmetic progression) if there are finite sets $A_1, A_2, B \subseteq \mathbb{Z}$, and $d_1, d_2 \in \mathbb{N}$, such that
$L = B \cup \{a + \lambda d_1 : a \in A_1, \lambda \in \mathbb{N}\} \cup \{a - \lambda d_2 : a \in A_2, \lambda \in \mathbb{N}\}$.
In other words, L is the finite union of arithmetic progressions and a finite set. The following characterization has been obtained by Just et al. (recall the definition of DIV_c from Chapter 2):

Theorem 2 [JMW 88]: *Let* $L \subseteq \mathbb{Z}$. *The following three statements are equivalent.*
- L *is* $\{+, -, DIV_c\}$*-computable.*
- L *is* $\{+, -, *, DIV\}$*-computable.*
- L *is an AP-language.* □

For $L \subseteq \mathbb{Z}^n$, $n > 1$, no such characterizations are known as yet.

(iii) Uniform Computability

It turns out that, unless $S = \{+, -\}$, we can only prove characterizations of *S-REC* if we further restrict our notion of S-computability: We need that the *S-RAM*s recognizing a language L have a *computable* time bound (in the Turing sense). Thus we define:
S-REC$_T$:=$\{L \subseteq \mathbb{Z}^n$: there is a $T(n)$-time bounded *S-RAM* for L such that $T(n)$ is (Turing-)computable$\}$.

Theorem 3: *Let* $\{+, -\} \subseteq S$. *Then* $L \in$*S-REC*$_T$ *iff there is a sequence* $B_1, B_2, \ldots$, *where* B_n *is an* (S, n)*-system with solution set* $L \cap \mathbb{Z}^n$, *such that* $n \mapsto B_n$ *is computable.*

Proof (sketch): "⇐"
An *S-RAM* can compute any computable function $f : \mathbb{Z} \to \mathbb{Z}$ (in the Turing sense), because $\{+, -\} \subseteq S$.
Thus, in order to recognize L, the *S-RAM* first computes B_n, and then evaluates B_n with input $x_1, \ldots, x_n$.
"⇒"
Let M be an *S-RAM* recognizing L. Then, for fixed input length n, M can "unroll" its program to an *S-CT*. There are some extra considerations necessary when dealing with indirect storage access, for details compare e.g. [MadH 85].
In order to find out the depth to which the *S-CT* has to be constructed, M computes

$T(n)$. From this *S-CT* of depth $T(n)$, M can compute the corresponding (S,n)-system B_n. □

In certain cases the assumption that $T(n)$ is computable can be avoided. This could be done as follows: For each node v of the *S-CT*, M checks whether there are inputs arriving at that node, i.e. whether the (S,n)-system corresponding to this node has a solution in $\mathbb{Z}^n$. Unfortunately, this test is not possible if $\{+,-,*\} \subseteq S$, because of Matijasevic's result from [Mat 70] about the undecidability of the solvability of polynomial Diophantine equations. On the other hand, for $S = \{+,-\}$ such a test is possible. (The Presburger Arithmetic is decidable, cf. [Opp 73]. In fact we only have to solve the integer programming problem.) This observation yields:

Theorem 4: $\{+,-\}\text{-}REC = \{+,-\}\text{-}REC_T$.

□

It is not known whether *S-REC=S-REC*$_T$ holds for other sets S.

5. General Lower Bound Methods

In this chapter we give an overview of the basic lower bound methods known for *S-CT*s and *S-RAM*s. All these methods even work for nonuniform computations, because no proof methods are known that take uniformity into account.
The most famous lower bound method is the so-called component counting lower bound, proved for $\{+,-,*_c\}$-*CT*s ($*_c$ means multiplication where at least one operand is constant) by Dobkin/Lipton in [DL 75], and for $\{+,-,*,/,\sqrt{\cdot}\}$-*CTs* by Ben-Or in [BO 83]. In both cases it is assumed that inputs are real-valued and that arbitrary constants are allowed. We shall refer to such computation trees as *real-valued S-CTs*.

Theorem 5 [DL 75],[BO 83]: *Let $L \subseteq \mathbb{R}^n$ consist of q connected components. Then each real-valued $\{+,-,*_c\}$-CT (real-valued $\{+,-,*,/,\sqrt{\cdot}\}$-CT) recognizing L has depth at least* $\log(q)\,(0.38\log(q) - 0.61n)$.

Proof (idea): Consider a node v of a real-valued *S-CT* D of depth t, for inputs from $\mathbb{R}^n$. Let $\tilde{F}(S,n,t)$ denote the set of functions $f : \mathbb{R}^n \to \mathbb{R}$ that can be computed by real-valued straight line programs over S of depth t.
For example

- $\tilde{F}(\{+,-,*_c\},n,t)$ is the set of all linear functions $f : \mathbb{R}^n \to \mathbb{R}$, if $t \geq 2n$.
- $\tilde{F}(\{+,-,*\},n,t\})$ is contained in the set of all polynomials $f : \mathbb{R}^n \to \mathbb{R}$ with degree at most 2^t.

For $\{+,-,*_c\}$, the arguments from the last chapter show that for each node v the set $c(v)$ of inputs arriving at v forms a convex polytope, in particular $c(v)$ is connected. As D has at most 2^t leaves, the accepted language can have at most t connected components. This yields the lower bound for $\{+,-,*_c\}$-CTs.
For the other case let us only consider $S = \{+,-,*\}$ for simplicity. Then $c(v)$ is the solution set of a system of at most t polynomial inequalities, each with degree at most 2^t.
In order to bound the number of connected components of this system, Steele and Yao have applied results due to Milnor [Mil 64] from algebraic topology. A consequence of these results is the following:

Lemma 2 [Mil 64]: *For each polynomial $p : \mathbb{R}^n \to \mathbb{R}$ of degree d, the set $\{\bar{x} \in \mathbb{R}^n : p(\bar{x}) \neq 0\}$ has at most $(d+2)(d+1)^{n-1}$ connected components.* □

In [StYa 82], Yao and Steele applied this result to the case where only polynomials of degree at most d are allowed to be computed in the computation tree. A direct application to our computation model only yields a $\Omega(\log(q)/n)$ lower bound.
This approach does not take into consideration that the polynomials appearing in D are not arbitrary polynomials with degree at most 2^t, but that they have a lot of additional structure, because they are even *computed in t steps*. In [BO 83] Ben-Or used this additional structure in a very nice way, also based on Milnor's result. He could show that each set $c(v)$ consists of at most 3^{n+t} connected components. This immediately yields the lower bound. □

Thus proving lower bounds is reduced to proving lower bounds on the number of connected components of a language or its complement.

Examples:

- Let $L(n,k) := \{(\bar{a},b) : \bar{a} \in \mathbb{R}^n, b \in \mathbb{R}$, there is $\bar{\alpha} \in \{0,\ldots,k\}^n$ with $\bar{a}\bar{\alpha} = b\}$.
  For $L(n,k)$ we get the lower bound $\Omega(n^2 \log(k+1))$ for the above CTs as shown in[DL 75] for $k = 1$ and in [MadH 85] for arbitrary k. In particular, for $k = 1$, we obtain the $\Omega(n^2)$ lower bound from [DL 75] for the knapsack problem.

- Let $ED(n) := \{(x_1,\ldots,x_n) \in \mathbb{R}^n : x_i \neq x_j$ for all $i \neq j\}$. For $ED(n)$ we get the lower bound $\Omega(\log(n!)) = \Omega(n\log(n))$ from [Rein 72].

A list of further applications can be found in [BO 83].

In order to obtain lower bounds for S-RAMs from the above lower bounds for real-valued S-CTs we have to handle two problems:
(i) Prove the lower bounds for (integer-valued) S-CTs.
(ii) Handle indirect storage access.

If we restrict ourselves to languages $L = \bigcup_{i=1}^{m} H_i$, where $H_i = \{\bar{x} \in \mathbb{R}^n : \bar{a}\bar{x} = 0\}$ for some $\bar{a} \in \mathbb{Z}^n$, then problem (i) can easily be handled. Thus Lemma 1 yields:

Theorem 6: *Let $L = \bigcup_{i=1}^{m} H_i$ have q connected components. Then each $\{+,-,*\}$-RAM recognizing $L \cap \mathbb{Z}^n$ has complexity $\Omega(\log(q)/\log\log(q))$.* □

Thus we e.g. only get an $\Omega(n^2/\log(n))$ lower bound for the knapsack problem. In [MadH 85] the above result is improved for $S = \{+,-,*_c\}$.

Theorem 7 [MadH 85]: *Let $L = \bigcup_{i=1}^{m} H_i$ have q connected components. Then each $\{+,-,*_c\}$-RAM recognizing $L \cap \mathbb{Z}^n$ has complexity at least $\log(q) - n\log\log(q)$.* □

This yields for example an $\Omega(n^2)$ lower bound for the knapsack problem.
On the other hand, for $ED(n)$ we only get the trivial $\Omega(n)$ lower bound. It can be shown that this bound is tight. By clever use of indirect storage access, $ED := \bigcup_{n\in\mathbb{N}} ED_n$ can be recognized by a $\{+,-\}$-*RAM* in time $O(n)$.
If we now also take integer division into account, the above results become wrong, because there are simple languages like the set of even integers that can be recognized in constant time by a $\{+,-,DIV_c\}$-*CT* although it has infinitely many connected components. In [JMW 88], Just et al. have shown a property of languages $L \subseteq \mathbb{Z}$ which makes them hard to be computed by $\{+,-,DIV_c\}$-*CT*s, even if arbitrary constants are allowed. (DIV_c is defined at the end of Chapter 2.)

Theorem 8 [JMW 88]: *Let $L \subseteq \mathbb{Z}, |L| = n$. If L contains no arithmetic progression of length $k+1$, then each $\{+,-,DIV_c\}$-CT (that may use arbitrary constants) recognizing L has depth $\Omega((\log(n)/\log\log(n))$ if $k \leq \log(n)$, and $\Omega((\log(n)/\log(k))$ otherwise.*

□

Thus, e.g. recognizing $\{2^i, i = 1,\ldots,n\}$ needs $\Omega(\log(n)/\log\log(n))$ steps on $\{+,-,DIV_c\}$-*CT*s, because this language contains no arithmetic progression of length 3.
For $\{+,-,DIV_c\}$-*RAM*s the above lower bounds are worse by a factor of $\log\log(n)$ factor because we have to handle indirect storage access. (compare Lemma 1).

6. Gaps between Genuine Complexity Classes

For nonuniform computations, we show in this chapter that nondeterminism, alternation, and even parallelism does not significantly increase the genuine computation power of nonuniform *S-RAM*s, if $S = \{+,-\}$, $S = \{+,-,*_c\}$, or $S = \{+,-,*_c\} \cup S_c$. Here S_c may contain arbitrary binary operations where both operands only depend on n. Thus, it is still true that only linear functions can be computed, but there is no bound on the size of the coefficients. The following theorem can be deduced from [MadH 84],[MadH 88] for $S = \{+,-\}$. The generalization to $S = \{+,-,*_c\} \cup S'$ is based on a result due to Meiser from [Mei 88].

Theorem 9 [MadH 84],[MadH 88],[Mei 88]: *Let $S_1 = \{+,-\}$, $S_2 = \{+,-,*_c\}$, $S_3 = S_2 \cup S_c$. Then, for $i = 1,2,3$, NU-S_i-P=NU-S_i-PARALLEL.*

Proof (idea): Lemma 1 shows that it suffices to consider *S-CT*s. Let us consider the operation set S_3. First we characterize the languages that can be computed by *parallel* S_3-*RAM*s.
An *(n,m)-language* is the solution set of a Boolean combination of at most m inequalities "$f(x) > 0$", for $f \in A(n) := \{f : \mathbb{Z}^n \to \mathbb{Z} : f(\bar{x}) = \bar{a}\bar{x} - b$ for some $\bar{a} \in \mathbb{Z}^n,\ b \in \mathbb{Z}\}$.
If $f_1, \ldots, f_m$ are the functions from $A(n)$ used to define L, then the *basic language* for L is $B(L) := \bigcup_{i=1}^{m} H_i$, where H_i denotes the set of integer points on the hyperplane $\{\bar{x} \in \mathbb{R}^n : f_i(\bar{x}) = 0\}$.
Now, analogously to the characterization of nonuniform S-computability in Chapter 4, we can prove structural properties of languages recognized by *parallel CT*s.

Lemma 3: *Let $L \subseteq \mathbb{Z}^n$ be recognized by a parallel S_3-CT D with p processors in t steps. Then L is an $(n,(p+2)^{tn+1})$-language.*

Proof (sketch): By the observations from Chapter 4 it is clear that L is an (n,m)-language for some m. The bound for m can be shown as follows.
Each branching node v of D a priori has 2^p children v', one for each possible combination of the outcomes of its p predicates.
But as these predicates are linear inequalities, we can easily conclude from Milnor's result from [Mil 64] (compare Lemma 2) that at most $(p+2)^n$ of them fulfil $c(v') \neq \emptyset$. Thus we can consider D as a *parallel CT* with outdegree at most $(p+2)^n$, therefore with at most $(p+2)^{nt}$ nodes. As each node only contains p functions, at most $(p+2)^{nt} \cdot p \leq (p+2)^{nt+1}$ functions are computed in D. Only these contribute to the system of inequalities defining L. □

In [MadH 88] it is shown that it suffices to consider the basic language of L.

Lemma 4 [MadH 88]: *Let L be an (n,m)-language, $B(L)$ its basic language. For $i = 1,2,3$, if an S_i-CT can recognize $B(L)$ in t steps, then there is an S_i-CT that recognizes L in $2t$ steps.* □

Now, in order to complete the proof of Theorem 9, we still have to show how to recognize the union of m hyperplanes in $\mathbb{R}^n$ in time polynomial in n and $\log(m)$ by an S_3-*CT*. We do not present any ideas of the following very nice theorem.

Theorem 10 [Mei 88]: *Let L be the union of m hyperplanes in $\mathbb{R}^n$. Then there is a (real-valued) S_3-CT for L of depth $O(n^6 \log(m))$. (We assume that constants are free, but evaluating a linear function needs $0(n)$ steps.)* □

In order to prove Theorem 9 for S_1 and S_2 we note that the computations of the constants in the *CT* from Theorem 10 are easy. For S_1, even a weaker version of the above result from Meiser is sufficient. In [MadH 84] it is shown that a union of hyperplanes can be recognized in time polynomial in n and $\log(k)$, where k is an upper bound on the absolute values of the (integer) coefficients of the hyperplanes. □

We finally quote a second gap result. Recall that *S-RP* denotes the class defined via randomized, polynomially time bounded *S-RAMs* with error probability $\leq \alpha < \frac{1}{2}$.

Theorem 11 [MadH 85a]: *NU-*$\{+,-,*\}$*-P=NU-*$\{+,-,*\}$*-RP.*

7. Separating Genuine Complexity Classes

In this chapter we show two separation results. Detailed proofs and further related results can be found in [KaM 88].
The first result demonstrates that the nonuniform complexity classes *NU-S-P* and *NU-S-PARALLEL*, which collapse for $S = \{+,-\}$ (compare Theorem 9), are different for $S = \{+,-,*\}$.

Theorem 12 [KaM 88]: *NU-*$\{+,-,*\}$*-P* $\neq$ *NU-*$\{+,-,*\}$*-PARALLEL.*

The next result even separates uniform classes *S-P* and *S-NP*, but a somewhat artificial operation set S is considered.

Theorem 13 [KaM 88]: $\{+,-,DIV_c\}$*-P*$\neq$ $\{+,-,DIV_c\}$*-NP.*

Proof of Theorem 12 (idea): We shall define a sequence $p_1, p_2, \ldots$ of polynomials, where $p_n : \mathbb{R}^2 \to \mathbb{R}$ has degree 2^n and coefficients from $\{-1,0,1\}$, and $n \mapsto p_n$ is computable (in the Turing sense). The language that separates *NU-*$\{+,-,*\}$*-P* and *NU-*$\{+,-,*\}$*-PARALLEL* then will be
$L := \{(x_1,\ldots,x_n) \in \mathbb{Z}^*, n \in \mathbb{N}, p_n(x_1,x_2) > 0\}$. First we show:

Lemma 5 [KaM 88]: *Each language L as above is in NU-*$\{+,-,*\}$*-PARALLEL.*

Proof (sketch): Given n, the preprocessing first computes the $0(2^{2n})$ many coefficients of p_n. (This can be done because $n \mapsto p_n$ is computable.) From this it computes a program for a *parallel* $\{+,-,*\}$*-RAM* which uses $0(2^{2n})$ processors. The i-th processor computes the i-th summand of $p_n(x_1,x_2)$, these summands are added up in parallel, and the result is finally compared to 0. Clearly this *parallel S-RAM* is nonuniformly $O(n)$-time bounded. □

Now we have to define the polynomials $p_1, p_2, \ldots$. They have to be hard to evaluate, and even the test "$p_n(x) > 0$" has to be hard. Such problems are studied in the field of algebraic complexity theory, where SLPs are considered, and constants and values for variables may be from $\mathbb{R}$ or other fields (compare [BoMu 75] and [Stra 84]).
Our problem looks harder for several reasons:
(i) We only allow integer inputs.
(ii) We have to deal with branchings.
(iii) We have to consider the test "$p(x) > 0$" instead of the evaluation of $p(x)$.

On the other hand, we can apply easy counting arguments for proving the existence of polynomials with high complexity, because we only allow constants from $\{0, 1, n\}$.
(i) and (ii) are not hard to handle. A solution for problem (ii) has been pointed out in a more general context by Strassen in [Stra 73]. For (iii) we have to be a little more careful. For example, if p is a polynomial with one input variable and degree d, then "$p(x) > 0$" can nonuniformly be tested very easily: Precompute the (at most d) roots of p, and, given x, perform binary search among the roots. Thus p does not have to be evaluated at all in order to test "$p(x) > 0$". This changes if p has two variables. In this case, the variety of p, $Var(p) := \{(x_1, x_2) \in \mathbb{R}^2 : p(x_1, x_2) = 0\}$, characterizes p up to a constant factor, if p is irreducible, and $Var(p)$ is infinite.

From these observations one can conclude (for a proof see [KaM 88]):

Lemma 6 [KaM 88]: *Let $p : \mathbb{R}^2 \to \mathbb{R}$ be irreducible and $Var(p)$ unbounded. Let M be a $\{+, -, *\}$-CT recognizing $\{(x_1, x_2) \in \mathbb{Z}^2 : p(x_1, x_2) > 0\}$. Then there is a node v in M where a polynomial q is computed such that $\alpha p + \beta$ is a factor of q, for some $\alpha, \beta \in \mathbb{Z}$, $\alpha \neq 0$.* □

Remark *(i) Note that we even have to demand that $Var(p)$ is unbounded in order to be able to handle branchings.*
(ii) Maybe not p but only $\alpha p + \beta$ appears as a factor of q, because we only consider integer inputs.

Finally, a counting argument and some algebraic considerations yield (for a proof see [KaM 88]):

Lemma 7 [KaM 88]: *For each d there is a polynomial $p : \mathbb{Z}^2 \to \mathbb{Z}$ with degree d and coefficients from $\{-1, 0, 1\}$ with the properties:*
(i) p is irreducible (over $\mathbb{R}$).
(ii) $Var(p)$ is unbounded.
(iii) Each $\{+, -, *\}$-SLP computing a polynomial $q : \mathbb{Z}^2 \to \mathbb{Z}$ such that q has a factor $\alpha p + \beta$ for some $\alpha, \beta \in \mathbb{Z}$, $\alpha \neq 0$, has depth $\Omega(d/\log(d))$. □

Now let p_n be that polynomial with degree 2^n with the properties from Lemma 7 that has the lexicographically first vector of coefficients. By analyzing all (finitely many) $\{+,-,*\}$-*SLP*s of length $\gamma 2^n/n$ (γ is the constant in the Ω-notation from Lemma 7) we can compute p_n from n. Lemmas 6 and 7 imply that testing "$p(x_1,x_2) > 0$" needs $\Omega(2^n/n)$ steps on a $\{+,-,*\}$-CT.
Thus, by Lemma 1, $L := \{\{x_1,\ldots,x_n\} \in \mathbb{Z}^* : n \in \mathbb{N}, p_n(x_1,x_2) > 0\} \notin NU\text{-}\{+,-,*\}\text{-}P$. Theorem 11 now follows with Lemma 5.

Proof of Theorem 13 (idea): We do not give details, but restrict ourselves to defining a language that separates $\{+,-,DIV_c\}$-P and $\{+,-,DIV_c\}$-NP, namely
$L := \{(x_1,\ldots,x_n) \in \mathbb{Z}^* : n \in \mathbb{N}, x_1$ can be divided by all $j \in \mathbb{N}, 1 \le j \le 2^n\}$.
In [KaM 88] it is shown that $\mathbb{Z}^* \backslash L \in \{+,-,DIV_c\}$-$NP$, and $L \notin \{+,-,DIV_c\}$-P. □

8. Concluding Remarks and Open Problems

In this paper we have tried to lay the foundation of a complexity theory of genuinely time bounded computations. We have seen that many problems that are still open in classical complexity theory (lower bounds, separations) can sometimes be solved in our context because of the algebraic properties of the computations. Our new approach offers a lot of interesting problems, some of which are quoted below. I am sure that most of them are much easier than their analogues in classical complexity theory.

(i) Does the Linear Programming Problem belong to $\{+,-,*\}$-P (compare Chapter 3)?

(ii) Are there languages in $\{+,-,*\}$-REC with a complexity that grows faster than any computable function, i.e. $\{+,-,*\}$-$REC \neq \{+,-,*\}$-REC_T (compare Chapter 4)?

(iii) Prove hierarchy-theorems.

(iv) Is it true that Theorem 11 even holds uniformly, i.e. $\{+,-,*\}$-$P \neq \{+,-,*\}$-*PARALLEL*? It can be shown, using the lower bounds for languages $L(n,k)$ from Chapter 5, that $\{+,-\}$-*DTIME(T(n))* $\neq$ $\{+,-\}$-*NTIME(T(n))* and $\{+,-,*\}$-*DTIME(T(n))* $\neq$ $\{+,-,*\}$-*NTIME(T(n))*, if $T(n)$ is S-time constructable. The notions used above are direct analogues to the classical notions.

(v) Compare different operation sets. Clearly, we have to restrict our attention to languages that are computable with both operation sets. For example: Is it true that $\{+,-,*_c\}$-$P = \{+,-,*\}$-$P \cap \{+,-,*_c\}$-REC holds?

References

[BaJuM 88] L.Babai,B.Just,F.Meyer auf der Heide, On the limits of computations with the floor function, Information and Computation 78(2), 99-107, 1988.

[BO 83] M.Ben-Or, Lower bounds for algebraic computation trees, 15th ACM-STOC, 80-86, 1983.

[BoMu 75] A. Borodin,I.Munro, The computational complexity of algebraic and numeric problems, Elsevier Computer Science Library, 1975.

[BSS 88] L.Blum,M.Shub,S.Smale, On a theory of computation over the real numbers; NP-completeness, recursive functions and universal machines, to appear, FOCS, 1988.

[DL 75] D.Dobkin,R.L.Lipton, A lower bound of $\frac{1}{2}n^2$ on linear search programs for the knapsack problem, JCSS 16, 417-421, 1975.

[JMW 88] B.Just,F.Meyer auf der Heide,A.Wigderson, On computations with integer division, 5th STACS, 29-37, 1988.

[KaM 88] M.Karpinski,F.Meyer auf der Heide, On the complexity of genuinely polynomial computations, in preparation, 1988.

[Karm 84] N.Karmakar, A new polynomial time algorithm for linear programming, 16th ACM-STOC, 302-311, 1984.

[Kha 79] L.G.Khachiyan, A polynomial algorithm for linear programming, Soviet Mathematics Declady 20(1), 191-194, 1979.

[MadH 84] F.Meyer auf der Heide, A polynomial linear search algorithm for the n-dimensional knapsack problem, J.ACM 31(3), 668-676, 1984.

[MadH 85] F.Meyer auf der Heide, Lower bounds for solving linear Diophantine equations on random access machines, J. ACM 32(4), 929-937, 1985.

[MadH 85a] F.Meyer auf der Heide, Simulating probabilistic by deterministic algebraic computation trees, TCS 41, 325-330, 1985.

[MadH 88] F.Meyer auf der Heide, Fast algorithms for n-dimensional restrictions of hard problems, J.ACM 35(3), 740-747, 1988.

[Mat 70] Y.V.Matijasevic, Enumerable Sets are Diophantine, Soviet Mathematical Dok lady II, 354-357, 1970.

[Meg 83] N.Megiddo, Towards a strongly polynomial algorithm for linear programming, SIAM J.Comp. 12, 347-353, 1983.

[Mei 88] S.Meiser, Suche in einem Arrangement von Hyperebenen, Diplomarbeit, Universität des Saarlandes, Saarbrücken, 1988.

[Mil 64] J.Milnor, On the Betti numbers of real varieties, Proc. Amer. Math. Soc. 15, 275-280, 1964.

[Opp 73] D.C.Oppen, Elementary bounds for Presburger arithmetic, 5th ACM-STOC, 34- 37, 1973.

[PaSt 82] C.H.Papadimitriou,K.Steiglitz, Combinatorial optimization: algorithms and complexity, Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[Rein 72] E.Reingold, On the optimality of some set algorithms, J.ACM 19, 649-659, 1972.

[Schö 79] A.Schönhage, On the power of random access machines, 6th ICALP, 520-529, 1979.

[Sim 79] J.Simon, Division is good, 20th IEEE-FOCS, 411-420, 1979.

[SpSt 76] E.Specker,V.Strassen, Komplexität von Entscheidungsproblemen, Springer Lecture Notes 43, 1976.

[Stra 73] V.Strassen, Berechnung und Programm II, Acta Informatica 2, 64-79, 1973.

[Stra 84] V.Strassen, Algebraische Berechnungskomplexität, Perspectives in Mathematics, Anniversary of Oberwolfach 1984, Birkhäuser Verlag, Basel, 1984.

[StYa 82] J.M.Steele, A.C.Yao, Lower bounds for algebraic decision trees, J. of Algorithms 3, 1-8, 1982.

[Tar 86] E.Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, Oper. Res. 34, 250-256, 1986.