On the Complexity of Genuinely Polynomial Computation

Marek Karpinski *
Dept. of Computer Science
University of Bonn, 5300 Bonn 1

Friedhelm Meyer auf der Heide †
Dept. of Computer Science
University of Paderborn, 4790 Paderborn

Abstract

We present separation results on genuinely (or strongly) time bounded sequential, parallel and non-deterministic complexity classes defined by RAMs with fixed set of arithmetic operations. In particular, we separate non-uniform polynomial time from non-uniform parallel polynomial time for the set of operations $\{+, -, *\}$ (answering a question of [M 88]), and uniform deterministic polynomial time from uniform non-deterministic polynomial time for the set of operations $\{+, -, DIV_c\}$, where DIV_c denotes a restricted integer division operation.

*Research supported by the International Computer Science Institute, Berkeley, California, by the DFG Grant KA 673/4-1 and by the SERC Grant GR-E 68297.
†Research supported by the DFG Grant ME 872/1-3.

1. Introduction

Classical complexity theory deals with uniform computation models as Turing Machines and Random Access Machines or non-uniform models as Boolean circuits and branching programs. The usual time complexity measure is defined as a function $T(n)$ denoting the worst case runtime taken over all inputs consisting of n bits.

On the other hand, the efficiency of most of the algorithms or data structures considered e.g. in courses on efficient algorithms is measured differently. Here one has in mind a Random Access Machine with uniform cost measure and arithmetic operation set $S \subset \{+, -, *, DIV, MOD, \ldots\}$ (S-RAM), and one looks at the worst case runtime $T(n)$ taken over all inputs consisting of n integers, not n bits (*genuinely* (or *strongly*) $T(n)$-time bounded computation). Examples are sorting, searching, weighted matching, knapsack, travelling salesman, etc. Two very interesting problems in this context are the following:

a) *Linear programming*

The known polynomial algorithms see e. g. [K 84] are not genuinely polynomial, i.e. their uniform runtine depends on the binary input length, not just on the number of integer inputs. The best known genuine time bound comes from the simplex algorithm and is exponential. It is a challenging open problem to find a genuinely polynomial algorithm. Steps in this direction can be found in [M 83] or [T 86].

b) *Integer programming*

It is shown in [BJM 88] that for this problem no genuine algorithm exists, i.e. every algorithm has a uniform runtime which grows to infinity with the binary input length, even if the number of integer input variables is fixed. This is true even for a very powerful operation set: evaluating any analytic function or applying integer division is allowed.

There are a lot more results concerning computability and lower bounds for S-RAMs (see e. g. [M 89]). All these results have in common that they are even true for non-uniform S-RAMs.

There are also results that show how the computation power explodes if we allow non-uniformity:

- For operations $\{+, -\}$, non-uniform polynomial time = non-uniform parallel polynomial time where $2^{poly(n)}$ processors are allowed ([M 84], [M 88]).
- For operations $\{+, -, *\}$, non-uniform polynomial time = non-uniform random polynomial time [M 85].

In this paper we try to lay the foundations of a complexity theory of genuinely polynomial computations (for the survey paper on the topic see [M 89]) and prove some basic separation results. For a given operation set S, we define the complexity classes S-*NP*, S-*P*, S-PARALLEL, S-*NC*. We introduce also a non-uniform S-*RAM* which starting with n variables is allowed to perform

an arbitrarily complex precomputation yielding a program of a S-RAM M_n over arbitrary inputs of n variables. The non-uniform versions of the classes S-NP, S-P and S-PARALLEL will be denoted by NU-S-NP, NU-S-P and NU-S-PARALLEL, respectively.

We prove the following three separation results:

(i) NU-$\{+,-,*\}$-$P \neq NU$-$\{+,-,*\}$-PARALLEL,

(ii) $\{+,-,*\}$-$NC \neq \{+,-,*\}$-P,

(iii) $\{+,-,DIV_c\}$-$P \neq \{+,-,DIV_c\}$-NP, where DIV_c denotes integer division by values that are only dependent on the *number* of input variables, not on their *values*.

Note that for (i) equality holds for the operation set $\{+,-\}$ (see above). (i) answers a question posed in [M 88].

2. Computation Models and Complexity Classes

A Random Access Machine with the arithmetic operation set $S \subset \{+,-,*,DIV,DIV_c,MOD\}$ consists of a finite program, an input tape, an output tape and infinitely many registers numbered 0, 1, 2 .... Each cell of the input or output tape and each register is able to store an integer of arbitrary size. The program consists of direct or indirect storage accesses, operations from S applied to contents of two registers, jumps (**goto** j), and branchings (**if** $Reg(0) > 0$ **then** ... **else** ...), read only (write only) access to the input (output) tape.

The execution of one instruction is counted as one step (uniform cost measure). The program starts with an input $(n, x_1, \ldots, x_n) \in I\!N^*$ on the input tape. At the end of the computation, the output is on the output tape. This defines the computed function $f : I\!N^* \to I\!N^*$. The runtime of a RAM M started with input x is $T_M(x)$, and $T(n) := \max\{T_M(x), x \in I\!N^n\}$ is the complexity of M. M accepts $L \subset I\!N^*$, if it computes its characteristic function. Non-deterministic and parallel RAMs (cf. [KR 88]) are defined as usual. We shall consider the following complexity classes: S-P, S-NP, S-PARALLEL, S-NC. In what follows "polynomially time-bounded" will mean *genuinely polynomially time-bounded*.

$$
\begin{aligned}
S\text{-}P &:= \{L \subset I\!N^*, \text{ there is a polynomially time-bounded } S\text{-RAM accepting } L\}.\\
S\text{-}NP &:= \{L \subset I\!N^*, \text{ there is a polynomially time-bounded non-deterministic } S\text{-RAM accepting } L\}.\\
S\text{-PARALLEL} &:= \{L \subset I\!N^*, \text{ there is a polynomially time bounded parallel } S\text{-RAM that uses } O(2^{poly(n)}) \text{ processors for inputs from } I\!N^n\}.\\
S\text{-}NC &:= \{L \subset I\!N^*, \text{ there is a polylogarithmically time bounded parallel } S\text{-RAM that uses } poly(n) \text{ processors for inputs from } I\!N^n\}.
\end{aligned}
$$

Non-uniformity of computations means that one allows to use a new program for each new input length. The nonuniform versions of the above complexity classes are marked with the prefix NU, e. g.

$$NU\text{-}S\text{-}P \;:=\; \{L \in I\!N^*, \text{ there is a polynomial } p \text{ and } S\text{-RAMs } M_1, M_2, \ldots \text{ such that } M_n \text{ accepts } L \cap I\!N^n \text{ in time } p(n)\}.$$

Such a family $M_1, M_2, \ldots$ of S-RAMs we shall call non-uniform S-RAM for short.

In order to prove lower bounds, we shall, as usual, consider a non-uniform, simplified computation model, the *S-computation tree* (S-CT). It gets inputs from $I\!N^n$, for fixed n. This is a rooted tree with outdegrees 0, 1, 2. A node with outdegree 0 is a leaf. It is labelled *accept* or *reject*.

A node v with outdegree 1 is a computation node, here a function $g = g_1 \circ g_2$ of the input $x_1, \ldots, x_n \in I\!N^n$ is computed, $g, g_1, g_2 : I\!N^n \to I\!N$, where $\circ \in S$ and g_1, g_2 are constants from $\{0, 1, n\}$, input variables, or functions computed on the path from the root to v. A node v with outdegree 2 is a branching node. It tests whether $f(x) > 0$, where f is computed on the path to v. An input $x \in I\!N^n$ now follows a fixed path from the root to a leaf, always going right at a branching if its test is fulfilled, and left else. The inputs arriving at accepting leaves form the accepted language $L \subset I\!N^n$. The complexity of the CT is its depth.

An S-CT that does not contain branchings is called a *straight line program* over S (S-SLP).

(Note that, in contrast to the classical notion of CTs and SLPs, we charge CTs and SLPs for computing large constants.)

3. Relating CTs and RAMs

In what follows, we shall prove lower bounds only for S-CTs. In this chapter we show that they also hold for S-RAMs with a logarithmic time loss at most.

Theorem 1: *Let $\{+, -\} \subset S$. Each* (uniform or non-uniform) *S-RAM recognizing $L \subset I\!N^*$ in $T(n)$ steps can be simulated by a family of S-CTs $D_1, D_2, \ldots$, where D_n recognizes $L \cap I\!N^n$ in $O(T(n) \log T(n))$ steps.*

PROOF: Essentially, an S-RAM can be "unrolled" to an S-CT if all indirect addresses used depend only on the number n of input variables, not on the depth. As n is constant, if we build up the CT D_n, we only have constant addresses. (**If**-questions become branching nodes, arithmetic operations become computation nodes, storage accesses are now implicit by the choices of the operands at a computation node.) Thus we have to show how to simulate an S-RAM by one that only uses addresses computed in special index registers, where only computations dependent on n are performed.

For the purpose of this paper, we maintain the memory of the S-RAM in a data structure D that supports insertion and lookup. If register a_i contains b_i, $i = 1, \ldots, d$, then D has stored the pairs $(a_1, b_1), \ldots, (a_d, b_d)$, where a_i are the keys.

Now reading in register a_i is simulated by looking up (a_i, b_i), writing b to register a is done by searching for key a. If it is already in D, $a = a_i$, then replace b_i by b. If not, insert the pair (a, b).

As D never contains more than $T(n)$ entries, each storage access is simulated in $O(\log(T(n))$ steps. One easily verifies that, if one carefully uses a 2-3-tree or AVL-tree for D, one only needs operations $\{+, -\}$ to maintain D, and only the restricted version of indirect addressing described above is used.
□

This result shows that lower bounds for S-CTs translate to lower bounds for S-RAMs with only logarithmic time loss.

4. Separation Results for $S = \{+, -, *\}$

We prove the following two separation results.

Theorem 2: $\{+, -, *\}$-$NC \neq \{+, -, *\}$-P

Theorem 3: $NU - \{+, -, *\}$-$P \neq NU - \{+, -, *\}$-PARALLEL

For the proofs we consider polynomials from the class $R(d, D) := \{p : I\!R^2 \to I\!R,\ p(x, y) = \tilde{p}(x) - y$, where $\tilde{p} : I\!R \to I\!R$ is a polynomial of degree d, leading coefficient 1, and coefficients from $\{-D, \ldots, D\}\}$, for $d, D \in I\!N$. Note that each $p \in R(d, D)$ is irreducible.

To each $p \in R(d, D)$ we associate the language $L_p := \{(x, y) \in I\!N^2, p(x, y) = 0\}$.

First we show that each $\{+, -, *\}$-PCT for L_p computes a polynomial which is closely related to p.

Lemma 1: *Let $p \in R(d, D)$. If a $\{+, -, *\}$-PCT T (with arbitrarily many processors) recognizes L_p in t steps, then there is a $\{+, -, *\}$-SLP of length t that computes a polynomial $q : I\!N^2 \to I\!N$ with $(p + \delta)|q$ (i. e. $p + \delta$ is a factor of q over $I\!R$) for some $\delta \in I\!R$. q is computed at some node of T.*

PROOF: Let T be a $\{+, -, *\}$-PCT for L_p of depth t. We show that a polynomial q as in the lemma is computed at some node of T. The path to this node can be looked upon as the desired $\{+, -, *\}$-SLP.

First we note: For $p \in R(d, D), \#L_p = \infty$; because, for $x \in I\!N$, $\tilde{p}(x) \in I\!N$, and therefore $L_p = \{(x, \tilde{p}(x)),\ x \in I\!N\}$.

Therefore there is an accepting leaf v of T such that $c(v)$, the set of inputs from $I\!R^2$ following the path to v, is unbounded. For $\alpha \in I\!R$ let

$$C_\alpha := c(v) \cap \{(x, y) \in I\!R^2, ||(x, y)|| > \alpha\}.$$

$c(v)$ is defined by a system of polynomial inequalities, for polynomials defining the branchings nodes on the path to v. Elementary properties of varieties of polynomials show that, for sufficiently large α, $C_\alpha = \{(x,y) \in I\!R^2,\ ||(x,y)|| > \alpha,\ q_1(x,y) > (\text{or } \geq)0,\ q_2(x,y) <(\text{or } \geq)0\}$, i. e. C_α is the strip between the varieties of polynomials q_1 and q_2. q_1 and q_2 can be chosen as irreducible factors of polynomials defining $c(v)$, i. e. of polynomials computed in T.

If $q_1 \equiv q_2$, then clearly $q_1 \equiv q_2 \equiv p$, and the lemma follows.

If $q_1 \not\equiv q_2$, then q_1 (and q_2) has to be of the form $p + \delta$ for some $\delta \in I\!R$. Otherwise, the varieties of q_1 and p would have an unbounded distance in the y-coordinate when x tends to infinity. Thus $c(v)$ would also contain points $(x, \tilde{p}(x) + 1)$ or $(x, \tilde{p}(x) - 1)$ for sufficiently large integers x. But these points do not belong to L_p. □

Proof of Theorem 2:
Let $L_d := \{(x,y) \in I\!N^2,\ x^d = y\}$, $L = \{(x_1, \ldots, x_n) \in I\!N^*,\ (x_1, x_2) \in L_{2^n}\}$.

Obviously, $L \in \{+,-,*\}$-P. (Compute x^{2^n} by iterated squaring.)

But by Lemma 1, a $\{+,-,*\}$-PCT T for L_d computes a polynomial q with $(x^d - y + \delta)|q$ for some δ. Thus q has degree at least d. Therefore, as a $\{+,-,*\}$-PCT can only compute polynomials with degree at most 2^t in t steps, the depth of T is $\Omega(\log(d))$. Thus each $\{+,-,*\}$-PCT for L needs time $\Omega(n)$, i. e. $L \notin \{+,-,*\}$-NC. □

Proof of Theorem 3:
The proof is more involved because we need polynomials from $R(d,D)$ which are much harder to compute than in time $O(\log(d))$. The existence of such polynomials is shown in the following lemma.

Lemma 2: *There are polynomials $p \in R(d,D)$ such that each polynomial $q : I\!R^2 \to I\!R$ with $(p + \delta)|q$ for some $\delta \in I\!R$ needs time $\Omega(d \log(D+1)/\log(d \log(D+1))$ to be evaluated by a $\{+,-,*\}$-SLP.*

Proof: Let $Q_t := \{p : I\!R^2 \to I\!R,\ p$ is a factor (with leading coefficient normalized to 1) of a polynomial computed by some $\{+,-,*\}$-SLP of length $t\}$.

Claim 1: $\#Q_t \leq (3(t+4)^2)^t \cdot 2^t$

Proof: There are at most $(3(t+4)^2)^t$ $\{+,-,*\}$-SLPs of length t, because each of the t steps has 3 choices for the operation ($+$, $-$, or $*$) and at most $t+4$ choices for each of the two operands (at most $t-1$ possible previously computed values, 2 input variables (x or y), 3 constants ($0, 1$, or n)). Further, each polynomial computed by a $\{+,-,*\}$-SLP of length t has degree at most 2^t, thus at most 2^t factors. Thus, $\#Q_t \leq (3(t+4)^2)^t \cdot 2^t$. □

Claim 2: $\#R(d,D) = (2D+1)^d$.

Proof: Each of the d coefficients $a_{d-1}, \ldots, a_0$ of the polynomial $\tilde{p}$ defining $p \in R(d,D)$ can be chosen from $\{-D, \ldots, D\}$. (Note that $a_d = 1$ by definition.) □

Thus, as long as $(3(t+4)^2)^t \cdot 2^t < (2D+1)^d$, there is a polynomial $p \in R(d,D)$ left such that, for no δ, $p+\delta \in Q_t$. This implies lemma 2. □

Next we note that polynomials from $R(d,D)$ can be computed fast in parallel.

Lemma 3: *Each $p \in R(d,D)$ can be computed by a $\{+,-,*\}$-PRAM with $d+1$ processors in time $O(\log(d)+\log(D))$.*

PROOF: The i'th processor, $i = 0,\ldots d$, computes the i'th summand $a_i x^i$ of $\tilde{p}$. As $|a_i| \leq D$, $i \leq d$, this needs time $O(\log(d)+\log(D))$. Computing $\tilde{p}(x)$, i. e. adding up the $d+1$ summands, needs time $O(\log(d))$. Constant time is needed to compute $p(x,y)$ from $\tilde{p}(x)$ and y. □

Now we can easily conclude Theorem 3. Let $p_d \in R(d,1)$ be a polynomial with the properties from lemma 2. Consider $L := \{(x_1,\ldots,x_n) \in \mathbb{N}^*,\ p_{2^n}(x_1,x_2) = 0\}$.

By Lemmas 1 and 2, L has complexity $\Omega(\frac{2^n}{n})$ on $\{+,-,*\}$-CTs, i. e. $L \notin NU$-$\{+,-,*\}$-P. On the other hand, by Lemma 3, $L \in NU$-$\{+,-,*\}$-PARALLEL. □

5. Separation Results for $S = \{+,-,DIV_c\}$

We prove the following separation result.

Theorem 4: *$\{+,-,DIV_c\}$-$P \neq \{+,-,DIV_c\}$-NP.*

PROOF: Let $L = \{(x_1,\ldots,x_n) \in \mathbb{N}^*, n \in \mathbb{N},\ x_1$ can be divided by all j, $1 \leq j \leq 2^n\}$. The following two lemmas imply the theorem.

Lemma 1: *$\overline{L} \in S$-NP.*

Lemma 2: *$L \notin S$-P.*

PROOF OF LEMMA 1: The following non-deterministic algorithm recognizes $\overline{L}$ in $O(n)$ steps.

1) Guess j, $1 \leq j \leq 2^n$.

2) Test whether j divides x_1 by testing $\cdot\, j$. $(x_1 DIV_c\, j) = x_1$. If not, accept.

Step 1 takes $O(n)$ steps (Guess the binary representation $\{a_0,\ldots,a_{n-1}\}$ of j (n bits), and compute j from it).

Step 2 takes $O(n)$ time:

– compute $(x_1 DIV_c\, j) := y$.

– compute $j \cdot y$ (without multiplication) with the help of $a_0, \ldots, a_{n-1}$ in $O(n)$ steps. □

PROOF OF LEMMA 2: Let $L_n := \{x, \text{ each } j, 1 \leq j \leq 2^n, \text{ divides } x\}$. The following claim, together with Theorem 1, implies now Lemma 2.

Claim 3: *Each $\{+, -, DIV_c\}$-CT for L_n has depth $\Omega(2^{\frac{1}{2}n})$.*

PROOF: We show that we can eliminate all DIV_c-operations on a given computation path of a $\{+, -, DIV_c\}$-CT if we restrict the input set $I\!N$ to an arithmetic progression $N(l) := \{l \cdot y, y \in I\!N\}$.

Proposition: *For each $\{+, -, DIV_c\}$-SLP M of length t with one input variable there is an integer $l \geq 0$, $l \leq n^t 2^{t^2}$, such that each function f computed in M fulfills:*

For $x \in N(l)$, $f(x)$ can be written as $f(x) = ax + b$ with
(i) $a \cdot l \in Z\!\!Z$, $b \in Z\!\!Z$,
(ii) $|a|, |b| \leq n \cdot 2^t$.

PROOF: by induction on t.

<u>$t = 0$:</u>
Before the computation starts, only the identity and the constants $0, 1, n$ are computed. They fulfill (i) and (ii) with $l = 1$.

<u>$t > 0$:</u>
Let f be computed in step t. By induction hypothesis, $f(x) = (ax + b) \circ (cx + d)$ for $x \in N(l')$ with $l' \leq n^{t-1} \cdot 2^{(t-1)^2}$, where $a \cdot l' \in Z\!\!Z$, $c \cdot l' \in Z\!\!Z$, $|a|, |b|, |c|, |d| \leq n \cdot 2^{t-1}$, and $\circ \in \{+, -, DIV_c\}$.

If $\circ \in \{+, -\}$, the proposition obviously holds with $l = l'$.

If $\circ = DIV_c$, we distinguish between two cases.

CASE 1: $a = 0$, $c = 0$.
In this case $f(x) = b \, DIV \, d$ for $x \in N(l')$. As $|b \, DIV \, d| \leq |b| \leq n \cdot 2^{t-1} \leq n \cdot 2^t$ by induction hypothesis, the proposition is fulfilled with $l = l'$.

CASE 2: $a \neq 0$, $c = 0$.
Let $l := l' \cdot d$. As, by induction hypothesis, $l' \leq n^{t-1} \cdot 2^{(t-1)^2}$ and $d \leq n \cdot 2^{t-1}$, it follows that $l \leq n^t \cdot 2^{t^2}$.

Now, for $x \in N(l)$ it holds that $\frac{a}{d}x \in Z\!\!Z$. Thus, for $x \in N(l)$, $f(x) = (ax + b) DIV_c \, d = \frac{a}{d}x + b \, DIV_c \, d$. As shown above, $|b \, DIV_c \, d| \leq n \cdot 2^t$. Further $\frac{a}{d} \cdot l = a \cdot l' \in Z\!\!Z$ and $|\frac{a}{d}| \leq |a| \leq n \cdot 2^t$. Thus f fulfills (i) and (ii).

As we have changed l, we have to make sure that (i) and (ii) still hold for the previously computed functions g. Each such g can by induction hypothesis be written as $g(x) = a'x + b'$ for $x \in N(l)$, as $N(l) \subseteq N(l')$. Further, by induction hypothesis, (ii) is fulfilled, and $a' \cdot l' \in Z\!\!Z$. As $l' | l$, also $a' \cdot l \in Z\!\!Z$ and the property (i) also follows.

This finishes the proof of the proposition. □

REMARK: *We do not have to consider the two cases with $c \neq 0$, because we only allow DIV_c, not DIV. The case $a = 0$, $c \neq 0$ would not cause much trouble, we only would have to restrict $N(l)$ to sufficiently large numbers. But the case $a \neq 0$, $c \neq 0$ would substantially damage the proposition. Our method would only guarantee a doubly-exponential bound for l.*

It is now easy to conclude Claim 3.

Let T be a $\{+, -, DIV_c\}$-CT for L_n of depth t. T has an accepting leaf v with $\#c(v) = \infty$. Consider the path to v. This path can be looked upon as a $\{+, -, DIV_c\}$-SLP of length t. By the proposition, we may assume that, for inputs from $N(l)$ for some $l \leq n^t 2^{t^2}$, all functions computed in this *SLP* are linear. Thus there is $z > 0$ such

that $N(l) \cap \{x \in I\!N, x > z\} \subseteq c(v)$, because branchings with linear functions can only partition the set $I\!N$ of inputs into a finite and an infinite interval. As $\#c(v) = \infty$, the path to v always belongs to the infinite interval.

As v is accepting, $c(v) \subseteq N(s)$, where s is the smallest common multiple of $\{1, \ldots, 2^n\}$.

Elementary number theory shows that $s = 2^{\Omega(2^n)}$. Thus, as T recognizes L_n, $n^t 2^{t^2} \geq l \geq s = 2^{\Omega(2^n)}$ must be fulfilled. This implies the desired lower bound $t = \Omega(2^{\frac{1}{2}n})$. □

6. Further Research

It would be very interesting to shed some light on the question whether our Theorem 2 holds also uniformly.

Acknowledgement

We are thankful to Avi Wigderson for a number of interesting discussions.

References

[BJM 88] Babai, L., Just, B., and Meyer auf der Heide, F., *On the Limits of Computation with the Floor Function*, Information and Computation **78** (1988), pp. 99-107.

[BO 83] Ben-Or, M., *Lower Bounds for Algebraic Computation Trees*, Proc. 15^{th} ACM STOC (1983), pp. 80-86.

[BM 75] Borodin, A., and Munro, I., *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier Computer Science Library, 1975.

[C 85] Cook, S. A., *A Taxonomy of Problems with Fast Parallel Algorithms*, Information and Control **64** (1985), pp. 2-22.

[JMW 88] Just, B., Meyer auf der Heide, F., and Widgerson, A., *On Computations with Integer Division*, Rairo Theoretical Informatics and Applications 23(1) (1989), pp. 101-111.

[K 84] Karmakar, N., *A New Polynomial Time Algorithm for Linear Programming*, Proc. 16^{th} ACM STOC (1984), pp. 302-311.

[KR 88] Karp, R. M., and Remachandran, V., *A Survey of Parallel Algorithms for Shared-Memory Machines*, Research Report No. UCB/CSD 88/407, University of California, Berkeley (1988); to appear in: *Handbook of Theoretical Computer Science*, North Holland.

[M 83] Meggido, N., *Towards a Genuinely Polynomial Algorithm for Linear Programming*, SIAM J. Comp. **12** (1983), pp. 347-353.

[M 84] Meyer auf der Heide, F., *A Polynomial Linear Search Algorithm for the n-Dimensional Knapsack Problem*, J. ACM **31** (1984), pp. 668-676.

[M 85] Meyer auf der Heide, F., *Simulating Probabilistic by Deterministic Algebraic Computation Trees*, Theoretical Computer Science **41** (1985), pp. 325-330.

[M 88] Meyer auf der Heide, F. *Fast Algorithms for n-Dimensional Restrictions of Hard Problems*, J. ACM **35** (1988), pp. 740-747.

[M 89] Meyer auf der Heide, F., *On Genuinely Time Bounded Computations*, Proc. 6^{th} STACS (1988), pp. 1-16.

[S 79] Schönhage, A., *On the Power of Random Access Machines*, Proc. 6^{th} ICALP (1979), pp. 520-529.

[SY 82] Steele, J. M., and Yao, A. C., *Lower Bounds for Algebraic Decision Trees*, J. of Algorithms **3** (1982), pp. 1-8.

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

[T 86] Tardos, E., *A Strongly Polynomial Algorithm to Solve Combinatorial Linear Programs*, Operations Research **34** (1986), pp. 250-256.