Logic Colloquium '88
Ferro, Bonotto, Valentini and Zanardo (Editors)

Some Problems and Results in the Theory of Actually Computable Functions (preliminary abstract)

WOLFGANG MAASS* AND THEODORE A. SLAMAN**

We consider subsets A of $\{0,1\}^*$ (the set of all finite binary strings x).

According to Church's thesis A is computable (meaning that there is an algorithm for deciding "$x \in A$?") if and only if A is recursive. Because of the advance of computing machinery it has also become of interest to find a precise mathematical definition which captures the intuitive notion that A is *actually computable* (meaning that there is an algorithm for deciding "$x \in A$?" so that this computation can actually be carried out if $|x|$, the length of x, is not "too large"). It has become common in computational complexity theory to consider the class

$$P := \bigcup_{k \in \mathbf{N}} DTIME(n^k)$$

as an adequate mathematical model for the class of "actually computable" sets (to be more precise: if one considers not only deterministic algorithms but also randomized algorithms whose output may be incorrect with an exponentially small probability, then one may replace P by the potentially larger class BPP; e.g. $PRIMES$ is known to be in BPP but it is open whether $PRIMES \in P$).

The complexity classes $DTIME(t(n))$ that occur in the definition of P are somewhat dependent on the considered computational model (although P isn't). For the

*Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, Chicago, IL. 60680. Written under partial support by NSF-Grant CCR 8703889, while visiting the Department of Computer Science at the University of Chicago.
**Department of Mathematics, University of Chicago, Chicago, IL. 60637. Written under partial support by Presidential Young Investigator Award DMS-8451748 and NSF-Grant DMS-8601856.

random access machine (RAM), the model on which we focus in this paper, one defines for arbitrary functions $t : \mathbf{N} \to \mathbf{N}^+$:

$A \in DTIME_{RAM}(t(n)) :\Leftrightarrow$ there exists a (deterministic) RAM R that decides "$x \in A$?" so that R is of time complexity $O(t(n))$ (i.e. there is a constant $K > 0$ s.t., for every input $x \in \{0,1\}^*$, R executes at most $K \cdot t(|x|)$ instructions).

The RAM is the model of choice for most algorithm designers because its instructions and its computation time (= number of instructions that are executed) correspond quite well to those of a real machine (see e.g. Aho, Hopcroft, Ullman [AHU]). Turing machines are somewhat less realistic because they are confined to specific data structures (the tapes) which have artificially large access time (because in order to read a bit far away on a tape the respective head has to travel over all cells in between). Very roughly one can view a RAM as a stronger version of a Turing machine where the "head" may jump in one step to an arbitrary "cell" (called *register* on a RAM) whose address has been specified. Furthermore in order to allow that such address be stored in a single register, one allows that a register may hold an arbitrary integer (however one linits the allowed arithmetical operations on register contents so that after t computation steps all integers stored in registers of the RAM have bit length $\leq t$). For a more detailed definition of a RAM we refer to [CR], [AHU], [MY], [P]. A reader who is more familiar with Turing machines should note that the only deviating feature of RAM's that we will exploit in the subsequent analysis is the fact that a RAM allows more sophisticated diagonalization-algorithms with regard to computation time (in particular a fine hierarchy theorem is known for computation time on RAM's (see below), whereas the corresponding statement for multi-tape Turing machines is still open - because for $k \geq 3$ it is not known whether every k-tape Turing machine can be simulated by a Turing machine with $k-1$ tapes in linear time.

As *supporting evidence for this thesis* (that P is the class of actually computable sets) we would like to point to

- the mathematical simplicity of this definition
- its robustness with regard to changes of the computational model (e.g. Turing machines, random access machines, uniform circuit families)

– the "test of time" (about 20 years of intensive work in computational complexity theory and algorithm design have so far not provided strong arguments against this thesis).

Among the *problematic points of this* thesis we would like to mention:

– constant factors in front of the considered polynomial time bounds (as well as the size of the programs) are ignored (consequently theory and praxis disagree about the feasibility of certain algorithms; however it appears to be very hard to take these aspects into account and still preserve the simplicity and robustness of the mathematical definition).

– its focus on worst case time bounds, as opposed to the practically more important average case analysis (this point can be addressed in a mathematical model if one can specify which input distribution is "realistic").

Let us assume then that P is an adequate model for the class of "actually computable sets". There are several aspects of P that can then be studied: we can compare it to other complexity classes (e.g. NP, EXPTIME, PSPACE); we can examine specific problems to see if they are in P; we can look for internal structure in P; we can compare the structure of P with familiar recursion theoretic classes (recursive sets, r.e. sets, arithmetical sets, sets recursive in objects of higher types, α- and β-recursive sets). Our intention is to look for the fine structure of P, which may shed light on its other aspects as well. In this abstract we focus on one facet of the fine structure of P, the classification of sets in P according to their exact time complexity. Of course quite a bit is known already about the fine structure of P. Thus before we describe our results we would like to give for non-experts in complexity theory an idea of the type of results that have been achieved so far.

i) Immerman and Moschovakis have shown that

$$A \in P \Leftrightarrow A \text{ is uniformly inductively definable over finite structures with linear order.}$$

This equivalence leads to results about the fine structure of P from the point of view of definability ([F], [I], [M], [R]).

ii) Hierarchy Theorems.

The first hierarchy theorem was proven by Hartmanis and Stearns [HS] for Turing

machines. For random access machines (RAM's), the model on which we will focus in this paper, Cook and Reckhow [CR] have shown: Let $f, g : \mathbf{N} \to \mathbf{N}^+$ be functions such that $f(n), g(n) \geq n$, $\sup_n \frac{f(n)}{g(n)} = \infty$, and f is time constructible on a RAM (i.e. the map $1^n \leftrightarrow 1^{f(n)}$, where both the input n and the output $f(n)$ are given in unary representation, can be computed in time $O(f(n))$ on some RAM). Then

$$DTIME_{RAM}(f(n)) \not\subseteq DTIME_{RAM}(g(n)).$$

It turns out that all explicitly defined functions that are actually used to estimate the computation time of algorithms are in fact time constructible on a RAM. For example the hierarchy theorem implies that

$$\begin{aligned}
&DTIME_{RAM}(n) \underset{\neq}{\subseteq} DTIME_{RAM}(n \cdot \log^* n) \underset{\neq}{\subseteq} \dots \\
&\underset{\neq}{\subseteq} DTIME_{RAM}(n \cdot \log\log n) \underset{\neq}{\subseteq} DTIME_{RAM}(n \cdot \log n) \\
&\underset{\neq}{\subseteq} \dots \underset{\neq}{\subseteq} DTIME_{RAM}(n^2) \underset{\neq}{\subseteq} \dots \underset{\neq}{\subseteq} DTIME_{RAM}(n^3) \\
&\underset{\neq}{\subseteq} \dots \underset{\neq}{\subseteq} P.
\end{aligned}$$

NOTE: It has been shown that the hierarchy theorem does not hold for arbitrary recursive f, i.e. the time-constructability-assumption for f is essential [HH].

iii) Concrete positive results, e.g.

$$ADDITION \in DTIME(n),$$
$$MULTIPLICATION \in DTIME_{RAM}(n \cdot \log\log n) \text{ [SS]},$$
$$LINEAR\ PROGRAMMING \in P \text{ [K]}$$

Of course, there exist an enormous number of results of this type.

iv) Comparisons between computational models, e.g.

$$DTIME_{\text{Turing machine}}(t(n)) \subseteq$$
$$DTIME_{RAM}(t(n)/\log t(n)), \text{ for } t(n) \geq n \cdot \log n \text{ [P]}.$$

On the other hand one has not been able to show that $A \notin DTIME_{RAM}(n)$ for any "natural" set $A \in P$ (in fact not even for any "natural" set $A \in NP$), which is perhaps the most prominent gap in our knowledge about the fine structure of P ("natural" means here that A should have some independent mathematical significance (MULTIPLICATION is a good candidate), in particular A should not just be the result of a diagonalization-argument.)

Remark. In the following we will only consider time complexity on RAM's and we write $DTIME(f)$ instead of $DTIME_{RAM}(f)$.

The preceding points indicate that an important structural property of P (which has no analogue in the more traditional structures of recursion theory) is the fact that we have a natural hierarchy for sets $A \in P$, given by the complexity classes $DTIME(f)$ for f bounded by a polynomial. We can classify the sets $A \in P$ according to their time complexity, i.e. according to their associated set of time bounds

$$\{f \mid A \in DTIME(f)\}.$$

Thus we define $A \leq_C B$ ("A has time complexity less or equal to the time complexity of B") by:

$$A \leq_C B :\Leftrightarrow \forall f \in T(B \in DTIME(f) \Rightarrow A \in DTIME(f))$$

and $A =_C B$ ("A has the same time complexity as B") is the induced equivalence relation:

$$A =_C B :\Leftrightarrow \forall f \in T(A \in DTIME(f) \Leftrightarrow B \in DTIME(f)).$$

We will refer to the equivalence classes of $=_C$ as *complexity types.*

The set T of considered time bounds is defined as follows:

$$\begin{aligned} T := \{f : \mathbf{N} \to \mathbf{N}^+ \mid\ & f(n) \geq n, f \text{ is time constructible on a RAM,} \\ & f(c \cdot n) = 0(f(n)) \text{ for every } c \in \mathbf{N}^+, \\ & \text{except for finitely many arguments} \\ & f \text{ agrees with a function } g \text{ that is concave} \\ & \text{i.e. } \forall n > m(g(n) \geq \frac{n}{m} g(m))\}. \end{aligned}$$

Remark. This definition of T has been chosen so that it is large enough to contain all functions that are actually used to estimate the time complexity of algorithms, on the other hand it is not too large in order to avoid some pathological phenomena (e.g. gap theorems [HU], [HH] and non-closure of time complexity classes under linear time reduction). Note that T is closed under variation on finitely many arguments, i.e. $f \in T$, $f =^* g$ and $g(n) \geq n$ implies $g \in T$.

Definition. We say that A is linear time reducible to B ($A \leq_{\text{lin}} B$) if there is a RAM with oracle B that decides "$x \in A$?" in time $O(|x|)$.

We assume here that a RAM with oracle B has a separate infinite array of query-registers into which it can write any string $y \in \{0,1\}^*$ (one bit per register) for which it wants to find out whether $y \in B$. After each execution of the additional instruction ORACLE QUERY the answer to this question can be found in the first register (and all query registers are erased).

Note that $\leq_{\text{lin}}$ is transitive. One calls the equivalence class of $=_{\text{lin}}$ linear (time) degrees.

Remark. The main interest in linear time reductions arised from the fact that $A \leq_{\text{lin}} B$ implies that $A \leq_C B$ (i.e. the time complexity of A is less or equal to that of B). This implication has provided the only successful means to compare the time complexity of sets A and B. We will also use it in this way, for example to show that every complexity type $\mathcal{C}$ contains a sparse set (Corollary 8). (On the other hand we are forced to provide in Theorem 1 a different method to show that $A =_C B$ when we given an example of sets A and B of the same time complexity but incomparable linear time degree.)

In this way one can show for example that the time complexity of deciding whether an *undirected* graph has a path from a distinguished vertex s to a distinguished vertex t is less or equal to that of the corresponding problem for *directed* graphs. Similarly one can show that the circuit value problem (CVP) with the requirement that the gate records in the input are sorted according to gate numbers has exactly the same time complexity as CVP without this requirement (the reduction from the latter to the former version exploits that a RAM can sort in linear time).

Less obvious linear time reductions have been constructed for example by Dewdney [D] (he shows in particular that SATISFIABILITY $=_{\text{lin}}$ $3-$COLORABILITY $=_{\text{lin}}$ SET-SPLITTING and that BIPARTITE MATCHING $=_{\text{lin}}$ VERTEX CONNECTIVITY ("are there $\geq k$ disjoint uv-paths in G, for u, v, k given"); note that the latter problems are known to be in P) and Grandjean [G1], [G2] (he shows that some natural NP-complete problems B are not in $DTIME_{\text{Turing machine}}(n)$ by showing that $A \leq_{\text{lin}} B$ for every $A \in NTIME(n)$).

Theorem 1. Every complexity type $\mathcal{C} \neq 0$ contains incomparable linear time degrees.

In order to *prove* this theorem we construct for an arbitrary given complexity type

$\mathcal{C} \neq 0$ two sets $A, B \in \mathcal{C}$ with $A \mid_{\text{lin}} B$ (i.e. $A \not\leq_{\text{lin}} B$ and $B \not\leq_{\text{lin}} A$). The only nontrivial part of this construction is to find suitable requirements whose satisfaction implies that the constructed sets A, B are of the given complexity type $\mathcal{C}$. It is quite easy to find such requirements if $\mathcal{C}$ is a *principal complexity type*, where we call a complexity type $\mathcal{C}$ *principal* if there exists a time bound $f_{\mathcal{C}} \in T$ s.t. for any $A \in \mathcal{C}$

$$\{f \in T | A \in DTIME(f)\} = \{f \in T | (f = \Omega(f_{\mathcal{C}})\}$$

(i.e. $f_{\mathcal{C}}$ is an "optimal time bound" for sets $A \in \mathcal{C}$).

The Cook-Reckhow construction of a diagonalizing RAM [CR] provides, for every $f \in T$, a set A whose complexity type $\mathcal{C}$ is principal and where $f_{\mathcal{C}} := f$ is an "optimal time bound" for $\mathcal{C}$.

On the other hand Blum's speed-up theorem ([B], [MY]) implies that there are sets $A \in P$ whose complexity type $\mathcal{C}$ is *non-principal*, e.g. one can construct a set A for which

$$\{f | A \in DTIME(f)\} = \left\{ f | \exists i \in \mathbf{N} \left(f = \Omega\left(\frac{n^2}{(\log n)^i} \right) \right) \right\}.$$

In order to construct sets A of an arbitrary given complexity type $\mathcal{C}$ (principal or non-principal) we use as a tool characteristic T-sequences (which are certain effective sequences of indices). We will show in Lemma 2 that one can associate with every complexity type $\mathcal{C}$ a characteristic T-sequence $(t_i)_{i \in \mathbf{N}}$ s.t.

$$\{f \in T | A \in DTIME(f)\} = \{f \in T | \exists i \in \mathbf{N} (f(n) = \Omega(\{t_i\}(n)))\}$$

for every $A \in \mathcal{C}$ (we say then that "$(t_i)_{i \in \mathbf{N}}$ is characteristic for $\mathcal{C}$"). This fact is complemented by Lemma 3, where we show that for any characteristic T-sequence $(t_i)_{i \in \mathbf{N}}$ there is a complexity type $\mathcal{C}$ s.t. $(t_i)_{i \in \mathbf{N}}$ is characteristic for $\mathcal{C}$. Furthermore the construction method of Lemma 3 reappears in the proof of Theorem 1 to ensure that the constructed sets A, B are of the given complexity type $\mathcal{C}$. We arrange that their sets of time complexity bounds are characterized by the same T-sequence that is characteristic for $\mathcal{C}$. Thus, we avoid the usual device of achieving $A =_{\mathcal{C}} B$ via showing that $A =_{\text{lin}} B$.

Definition. $(t_i)_{i \in \mathbf{N}} \subseteq \mathbf{N}$ is called a characteristic T-sequence if $t : i \mapsto t_i$ is recursive and

a) $\forall i \in \mathbf{N}(\{t_i\} \in T$ and program t_i is a witness for the time-constructibility of $\{t_i\})$

b) $\forall i, n \in \mathbf{N}(\{t_{i+1}\}(n) \leq \{t_i\}(n))$.

Lemma 2. ("inverse of the speed-up theorem"). For every recursive set A there exists a characteristic T-sequence $(t_i)_{i\in\mathbf{N}}$ such that

$$\forall f \in T(A \in DTIME(f) \Leftrightarrow \exists i \in \mathbf{N}(f(n) = \Omega(\{t_i\}(n)))).$$

Lemma 3. ("refinement of the speed-up theorem"). For every characteristic T-sequence $(t_i)_{i\in\mathbf{N}}$ there exists a recursive set A such that

$$\forall f \in T(A \in DTIME(f) \Leftrightarrow \exists i \in \mathbf{N}(f(n) = \Omega(\{t_i\}(n)))).$$

Remark. The proof of Lemma 3 is somewhat more delicate than the usual proof of the speed-up theorem because $\{t_{i+1}\}$ need not be a strictly slower growing function than $\{t_i\}$ (in fact we have in general that for many i both functions are the same, and $\{t_{i+1}\}$ may even have a larger time-constructibility-factor than $\{t_i\}$). In both lemmata we exploit that our concepts are rather concrete (in particular that we consider only time constructible functions and a specific machine model).

Idea of the proof of Theorem 1. Let $(t_i)_{i\in\mathbf{N}}$ be a characteristic T-sequence for the given complexity type $\mathcal{C}$. Construct sets A, B s.t. both $A|_{\text{lin}}B$ and for all $f \in T$ the set A (or the set B) is in $DTIME(f)$ iff $f(n) = \Omega(\{t_i\}(n))$ for some $i \in \mathbf{N}$. Use the priority method to handle conflicts between the resulting two types of requirements. □

Remark. Although it is the main goal of our approach to study the fine structure of P, it also yields some new results (and open problems) about polynomial time reductions. In particular it allows to make the "upper bounds" of the constructed sets more precise (instead of making the sets just recursive). As a sample we give the following result, which is proved with the same arguments as Theorem 1.

Theorem 4. A complexity type $\mathcal{C}$ contains sets A, B that are incomparable w.r.t. polynomial time reductions if and only if $\mathcal{C} \not\subseteq P$.

One may ask whether it can happen that two sets A, B of the same complexity type have "no common information", i.e. that they form a minimal pair w.r.t. linear time (resp. polynomial time) reductions.

Theorem 5. There are recursive sets A, B of the same complexity type $\mathcal{C}$ such that A, B form a minimal pair w.r.t. linear time (resp. polynomial time) reductions.

The proof of this result combines methods from Cohen forcing and from the proof of Lemma 3 in a finite injury priority argument. It is not known whether such a pair of sets exists in every complexity type.

It would be of some interest to characterize the structure of the linear time degrees inside a given complexity type. The following result shows that these structures are not all isomorphic (it also shows that linear time degrees play a significant role in the analysis of complexity types).

Theorem 6. A complexity type $\mathcal{C}$ has a largest linear degree if and only if $\mathcal{C}$ is principal (in fact if $\mathcal{C}$ is non-principal then it does not even contain a maximal linear degree).

Finally we would like to mention briefly two other concepts that are of interest for the fine structure of P (although they have no direct analogue in classical recursion theory): sparse sets and the padding operator.

Definition. $A \subseteq \{0,1\}^*$ is sparse if $n \mapsto \mathrm{card}(A \cap \{0,1\}^n)$ is bounded by a polynomial.

Definition. For arbitrary functions $s \in T$ we associate with every set $A \subseteq \{0,1\}^*$ the s-padding $P_s(A) := \{x \# O^{s(|x|)} | x \in A\}$ of A.

Note that P_s is in fact well-defined on complexity types (i.e. $A =_C B$ implies $P_s(A) =_C P_s(B)$) and that P_s projects complexity classes downwards:

$$A \in DTIME(f) \Rightarrow P_s(A) \in DTIME(f \circ s^{-1}).$$

Theorem 7. ("inversion of the padding operator"). Fix any $s \in T$. Then the map

$$\mathcal{C} \ni A \mapsto P_s(A) \in \mathcal{C}'$$

from the complexity types into the complexity types is onto.

Corollary 8. Every complexity type $\mathcal{C}$ contains a sparse set.

In order to derive this corollary from Theorem 7 one considers the padding function $s(n) = 2^n$ and exploits that there exists for every set $A \subseteq \{0,1\}^*$ some set $B \in \{1\}^*$ (thus B is sparse) with $P_{2^n}(A) =_{\mathrm{lin}} B$.

Remark. It is interesting to compare Corollary 8 with Mahaney's result [Ma] that $P \neq NP$ implies that there is no NP-complete sparse set.

Finally we would like to point out that there are of course many other interesting aspects of the fine structure of P which we have not even mentioned here (some of the other aspects will be discussed in a forthcoming paper).

Some open problems about the fine structure of P:

1. Is there any "natural" $A \in P$ s.t. $A \notin DTIME(n)$?
2. Are there relationships between the mathematical structure of a set A (e.g. combinatorial properties of a set A, or other "extensional" properties) and its time complexity?
3. What is the structure of linear time degrees inside a complexity type $\mathcal{C}$? In particular: are those structures isomorphic for all principal (resp. non-principal) complexity types $\mathcal{C}$?
4. What is the structure of linear time degrees inside a complexity *class* $DTIME(f)$? In particular: does this structure depend on f (for f superlinear)?
5. Are there besides the padding operators any other interesting operators on P? In particular: is there some interesting notion of a "jump"?
6. Are there sets A, B that are in the same complexity type (i.e. A and B have the same deterministic time complexity) but which have different nondeterministic time complexity (or: different space complexity)?

REFERENCES

[AHU] A.V. AHO, J.E. HOPCROFT, J.D. ULLMAN, The Design and Analysis of Computer Algorithms, Addison-Wesley (Reading, 1974).

[B] M. BLUM, A machine-independent theory of the complexity of recursive functions, *J. ACM* **14**(1967), 322-336.

[CR] S.A. COOK, R.A. RECKHOW, Time-bounded random access machines, *J. Comp. Syst. Sc.* **7**(1973), 354-375.

[D] A.K. DEWDNEY, Linear time transformations between combinatorial problems, *Internat. J. Computer Math.* **11**(1982), 91-110.

[F] R. FAGIN, Monadic generalized spectra, *Zeitschrift f. Math. Logik* **21**(1975), 89-96.

[G1] E. GRANDJEAN, A natural NP-complete problem with a nontrivial lower bound, *SIAM J. Comp.* 1988 (to appear).

[G2] E. GRANDJEAN, A nontrivial lower bound for an NP problem to automata, preprint (1987).

[HS] J. HARTMANIS, R.E. STEARNS, On the computational complexity of algorithms, *Trans. AMS* **117**(1965), 285-306.

[HH] J. HARTMANIS, J.E. HOPCROFT, An overview of the theory of computational complexity, *J. ACM* **18**(1971), 444-475.

[I] N. IMMERMAN, Languages which capture complexity classes, *Proc. of the 15th ACM Symp. on the theory of computing,* 1983, 347-354.

[K] L.G. KHACHIAN, A polynomial algorithm for linear programming, *Soviet Math. Doklady.* **20**(1979), 191-194.

[MY] M. MACHTEY, P. YOUNG, An Introduction to the General Theory of Algorithms, North-Holland (Amsterdam, 1978).

[Ma] S. MAHANEY, Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis, *J. Comp. Syst. Sc.* **25**(1982), 130-143.

[M] Y.N. MOSCHOVAKIS, Abstract recursion as a foundation for the theory of algorithms, preprint (1983).

[P] W.J. PAUL, Komplexitätstheorie, Teubner (Stuttgart, 1978).

[R] M. DE ROUGEMONT, Uniform definability on finite structures with successor, *Proc. of the 16th ACM Symp. on the theory of computing,* 1984, 409-417.

[SS] A. SCHÖNHAGE, V. STRASSEN, Schnelle Multiplikation grosser Zahlen, *Computing* **7**(1971), 281-292.