The Complexity Types of Computable Sets (extended abstract)

WOLFGANG MAASS* AND THEODORE A. SLAMAN**

Abstract. We analyze the fine structure of time complexity classes for RAM's, in particular the equivalence relation $A =_C B$ ("A and B have the same time complexity") $\Leftrightarrow$ (for all time constructible $f : A \in DTIME_{RAM}(f) \Leftrightarrow B \in DTIME_{RAM}(f)$). The $=_C$-equivalence class of A is called its *complexity type*. We prove that every set X can be partitioned into two sets A and B such that $X =_C A =_C B$, that a complexity type $\mathcal{C}$ contains sets A, B which are incomparable with respect to polynomial time reductions if and only if $\mathcal{C} \nsubseteq P$, and that there is a complexity type $\mathcal{C}$ that contains a minimal pair with respect to polynomial time reductions. Furthermore we analyze the fine structure of P with respect to linear time reductions: We show that each complexity type $\mathcal{C} \nsubseteq DTIME(n)$ contains a rich structure of linear time degrees, and that these degree structures are not all isomorphic (in particular we characterize those $\mathcal{C}$ that have a maximal linear time degree). Finally we show that every complexity type contains a sparse set. Our proofs employ finite injury priority arguments, together with a new technique for constructing sets of a given time complexity type.

*Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, Chicago, IL. 60680. Written under partial support by NSF-Grant CCR 8703889.
**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.

We consider the following set of time bounds:

$$T := \{f : \mathbf{N} \to \mathbf{N} \mid f(n) \geq n \text{ and } f \text{ is time constructible on a RAM}\}.$$

where f is called time constructible on a RAM if some RAM can compute the function $1^n \mapsto 1^{f(n)}$ in $O(f(n))$ steps. We do not allow arbitrary recursive functions as time bounds in our approach in order to avoid pathological phenomena (e.g. gap theorems [HU], [HH]). In this way we can focus on those aspects of complexity classes that are relevant for concrete complexity (note that all functions that are actually used as time bounds in the analysis of algorithms are time constructible). We use the random access machine (RAM) with uniform cost criterion as machine model (see [CR], [AHU], [MY]) because this is the most frequently considered model in algorithm design, and because a RAM allows more sophisticated diagonalization - constructions then a Turing machine. We write $DTIME(f)$ for $DTIME_{RAM}(f)$ in the following.

For sets $A, B \in \{0,1\}^*$ we define

$$\begin{aligned} &A =_C B (\text{“}A \text{ has the same det. time complexity as } B\text{”}) \\ &:\Leftrightarrow \forall f \in T(A \in DTIME(f) \Leftrightarrow B \in DTIME(f)). \end{aligned}$$

A *complexity type* is an equivalence class of this equivalence relation $=_C$.

Theorem 1. Every set X can be split into two sets A, B of the same complexity type as X (i.e. $X = A \cup B$, $A \cap B = \emptyset$, $X =_C A =_C B$).

In order to *prove* this result one needs a technique for controlling the complexity type of the constructed sets A, B. This is less difficult if X has an "optimal" time bound $f_X \in T$ for which $\{f \in T \mid X \in DTIME(f)\} = \{f \in T \mid f = \Omega(f_X)\}$ (in this case we say that X is of *principal* complexity type). However Blum's speed-up theorem [B] asserts that there are for example sets $X \in P$ such that

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

Note that this effect occurs even if one is only interested in time constructible time bounds (and sets X of "low" complexity).

In order to prove Theorem 1 also for sets X whose complexity type is non-principal, we show that in some sense the situation of Blum's speed up theorem (where we can characterize the functions f with $X \in DTIME(f)$ with the help of a "cofinal" sequence of functions) is already the worst that can happen (unfortunately this is not quite true, since we cannot always get a cofinal sequence of functions f_i where $f_{i+1}(n) = O(\frac{f_i(n)}{g(n)})$ for a fixed function g with $g(n) \rightarrow \infty$ for $n \rightarrow \infty$, as required for the proof of the speed-up theorem).

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 1. ("inverse of the speed-up theorem"). For every recursive set A there exists a characteristic T-sequence $(t_i)_{n \in \mathbf{N}}$ such that $(t_i)_{i \in \mathbf{N}}$ is characteristic for A (i.e. $(\forall f \in T(A \in DTIME(f) \Leftrightarrow \exists i \in \mathbf{N}(f(n)) = \Omega(\{t_i\}(n)))))$.

Remark. In a similar fashion, Kolomogorov and Levin [L] had captured the *space* complexity of a recursive function by a sequence of space constructible functions.

Idea of the proof of Theorem 1. Associate with the given set X a characteristic T-sequence $(t_i)_{i \in \mathbf{N}}$ as in Lemma 1. For every $e, n \in \mathbf{N}$ and $x \in \{0,1\}^*$ define

$$TIME(e, x) := (\text{number of steps in the computation of } \{e\} \text{ on input } x)$$

and

$$MAXTIME(e, n) := \max\{TIME(e, x) \mid |x| = n\}.$$

It is sufficient to partition X into sets A and B in such a way that for every $e \in \mathbf{N}$ the following requirements $R_e^A, R_e^B, S_e^A, S_e^B$ are satisfied:

$$R_e^A :\Leftrightarrow (A = \{e\} \Rightarrow \forall f \in T(\forall n(MAXTIME(e, n) \leq f(n)) \Rightarrow \exists j \in \mathbf{N}(f(n) = \Omega(\{t_j\}(n)))))$$

$$S_e^A :\Leftrightarrow A \in DTIME(\{t_e\}(n)).$$

R_e^B, S_e^B are defined analogously.

Note that it is not possible to satisfy R_e^A by simply setting $A(x) := 1 - \{e\}(x)$ for some x: in order to achieve that $A \subseteq X$ we can only place x into A if $x \in X$.

Instead, we adopt the following strategy to satisfy R_e^A (the strategy for R_e^B is analogous): For input $x \in \{0,1\}^*$ compute $\{e\}(x)$.

Case I. If $\{e\}(x) = 0$, then this strategy issues the constraint "$x \in A \Leftrightarrow x \in X$".

Case II. If $\{e\}(x) = 1$, then this strategy issues the constraint "$x \notin A$" (which forces x into B if $x \in X$).

In the case of a conflict for some input x between strategies for different requirements one lets the requirement with the highest priority (i.e. the smallest index e) succeed (this causes in general an "injury" to the other competing requirements).

The interaction between the described strategies is further complicated by the fact that in the case where R_e^A is never satisfied via Case II, or via Case I for some $x \in X$, we have to be sure that Case I issues a constraint *for almost every input* x (provided that the simulation of $\{e\}(x)$ is not prematurely halted by some requirement S_i^A with $i \leq e$, see below). Consequently the number of requirements whose strategies act on the same input x grows with $|x|$ (only those R_i^A, R_i^B with $i < |x|$ can be ignored where one can see by "looking back" for $|x|$ steps that they are already satisfied).

The strategy for requirement S_e^A (S_e^B) is as follows: it issues the constraint that for all inputs x with $|x| \geq e$ the *sum* of all steps that are spent on simulations for the sake of requirements $R_i^A, R_i^B, S_i^A, S_i^B$ with $i \geq e$ has to be bounded by $O(\{t_e\}(|x|))$. One can prove that in this way $S_e^A(S_e^B)$ becomes satisfied (because only finitely many inputs are placed into A or B for the sake of requirements of higher priority). One also has to prove that the constraint of S_e^A does not hamper the requirements of lower priority in a serious manner.

This part of the construction is more difficult than its counterpart in Blum's speed-up-theorem [B], because it need not be the case that $\{t_{i+1}\} = o(\{t_i\})$. A further complication is caused by the fact that although there are constants K_i, K_{i+1} such that $\{t_i\}(n)$ converges in $\leq K_i \cdot \{t_i\}(n)$ steps and $\{t_{i+1}\}(n)$ converges in $\leq K_{i+1} \cdot \{t_{i+1}\}(n)$ steps, we may have that $K_i \ll K_{i+1}$ (and therefore $K_i \cdot \{t_i\}(n) \ll K_{i+1} \cdot \{t_{i+1}\}(n)$). Therefore the requirements S_j^A with $j > i$ are not able to "take over" the job of S_i^A, and *all* computations $\{t_i\}(|x|)$, $i \leq |x|$, have to be simulated simultaneously for each input x.

In order to show that a single RAM R can carry out simultaneously all of the described startegies, one exploits in particular that a RAM can dovetail an unbounded number of simulations in such a way that the number n_e of steps that it has to spend in order to simulate a single step of a simulated program $\{e\}$

does not grow with the number of simulated programs (the precise construction of R is rather complex).

In order to verify that this construction succeeds, one has to show that each requirement R_e^A, R_e^B is "injured" at most finitely often. This is not obvious, because we may have for example that R_{e-1}^B (which has higher priority) issues overriding constraints for infinitely many arguments x according to Case I. However in this case we know that only finitely many of these x are elements of X (otherwise R_{e-1}^B would have been seen to be satisfied from some point of the construction on), and all of its other constraints are "compatible" with the strategies of lower priority (since we make $A, B \subseteq X$).

Finally we verify that each requirement $R_e^A(R_e^B)$ is satisfied. This is obvious if Case II occurs in the strategy for R_e^A for some input x where R_e^A is no longer injured; or if Case I occurs for such input x with $x \in X$ (in both cases we can make $A \neq \{e\}$). However it is also possible that $x \notin X$ for each such x (and that $\{e\} = A$), in which case R_e^A becomes satisfied for a different reason. In this case we have $\{e\}(x) = 0 = X(x)$ for each such x. Therefore we can use $\{e\}$ to design a new algorithm for X that is (for every input) at least as fast as the algorithm $\{e\}$ for A (it uses $\{e\}$ for those inputs where $\{e\}$ is faster than the "old" algorithm for X of time complexity $\{t_e\}$). Therefore one can prove that $X \in DTIME(f)$ for every $f \in T$ that bounds the running time of algorithm $\{e\}$ for A. This implies that $f(n) = \Omega(\{tj\}(n))$ for some $j \in \mathbf{N}$ (by construction of the characteristic T-sequence $(t_i)_{i\in\mathbf{N}}$). $\square$

The question arises, whether all sets in a given complexity type contain essentially the same information (for example whether they are all "universal" for that complexity type). The following two results show that this is not the case (it applies to any reasonable definition of polynomial time reduction).

Theorem 2. A complexity type $\mathcal{C}$ contains sets A, B that are incomparable with regard to polynomial time reductions if and only if $\mathcal{C} \not\subseteq P$.

Theorem 3. There is a complexity type $\mathcal{C}$ which contains sets A, B that form a minimal pair with regard to polynomial time reduction (i.e. $A, B \notin P$, but $D \leq_p A$ and $D \leq_p B$ implies $D \in P$ for every set $D \subseteq \{0,1\}^*$).

The proofs of these theorems require nontrivial finite injury priority arguments. Note that "delayed diagonalization" [LA] does not suffice in a situation where the constructed sets A,B have to be of the same complexity type. For the same reason the customary gap-technique for the construction of minimal pairs

[LLR] cannot be applied (we use a constructive version of Cohen forcing for the proof of Theorem 3).

In contrast to many results in structural complexity theory which only apply to sets outside of P, the investigation of complexity types also sheds some new light on the structure of P. Note that those time bounds f that are used in the analysis of algorithms for problems in P have the property that $DTIME(f)$ is closed under linear time reductions. For the subsequent analysis of the structure of linear time degrees we therefore replace the considered set T of time bounds by

$$T_L := \Big\{ f \in T \mid f(c \cdot n) = O(f(n)) \text{ for every } c \in \mathbf{N} \text{ and } f \text{ agrees almost everywhere with some function } g \text{ that is concave, i.e.} \forall n > m \Big(g(n) \geq \frac{n}{m} g(m) \Big) \Big\}$$

(Theorem 7 holds only for T_L, all other results in this paper hold both for T and T_L as class of time bounds.) Obviously every complexity type is closed under linear time reductions if it is defined with T_L instead of T. Note that linear time reductions have in fact provided the only successful means to show that certain concrete sets have exactly the same time complexity (e.g. Dewdney [D] proved that BIPARTITE MATCHING $=_{\text{lin}}$ VERTEX CONNECTIVITY ("are there $\geq k$ disjoints uv-paths in G, for u, v, k given")). The following result implies that this method is not general.

Theorem 5. Every complexity type $\mathcal{C} \not\subseteq DTIME(n)$ contains sets of incomparable linear time degree, furthermore the linear time degrees in $\mathcal{C}$ have no minimal element and they are dense.

It is tempting to conjecture that for every recursive set $A \notin DTIME(n)$ there is a set B of the same complexity type with $A \not\leq_{\text{lin}} B$ and $B \not\leq_{\text{lin}} A$, furthermore that the structure of linear time degrees of sets in a complexity type is the same for every complexity type $\mathcal{C} \not\subseteq DTIME(n)$. The following result implies that both conjectures are false (see the proof of Theorem 1 for the definition of a principal complexity type).

Theorem 6. A complexity type $\mathcal{C}$ has a largest linear time 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).

In the proof of this result one uses a finite injury priority argument to construct in the case where $\mathcal{C}$ is non-principal for every given set $X \in \mathcal{C}$ a set $A \in \mathcal{C}$ with $X <_{\text{lin}} A$. In this construction one has to use the existing infinitely many $i \in \mathrm{N}$ with $\{t_i\}|_{S_i} = o(\{t_{i-1}\}|_{S_i})$ (for some infinite set $S_i \in \mathrm{N}$) as "windows" through which at least one more requirement can be satisfied. □

Very little is known so far about the relationship between the mathematical structure of a set and its time complexity. The following theorem gives a somewhat unexpected result of this type (compare with [Ma]).

Theorem 7. Every complexity type contains a sparse set.

The proof of this result follows from:

Theorem 8. (padding-inversion-theorem) Consider an arbitrary function $s \in T$. The padding operator

$$A \mapsto P_s(A) := \{x\#0^{s(|x|)} \mid x \in A\}$$

is well-defined on complexity types (i.e. $A =_C B \Rightarrow P_s(A) =_C P_s(B)$). Furthermore it is invertible: for every set $D \subseteq \{0,1\}^*$ there is a set $A \subseteq \{0,1\}^*$ with $P_s(A) =_C D$.

In order to derive Theorem 7 from Theorem 8 one exploits the fact that for $s(n) := 2^n$ every set $P_s(A)$ is of the same linear time degree as some tally set.□

Conclusion. Complexity types provide a useful conceptual basis for the investigation of the structure of complexity classes, in particular also of classes in low-level complexity theory. They not only give rise to a number of interesting new problems (some of which are even solvable), but they provide the m$\overset{e}{\vee}$ans to sharpen some traditional questions in complexity theory (e.g. compare Theorems 2 and 3 with the customary weaker versions [LA], [LLR]) in such a way that they pose a more serious challenge to our construction techniques (in particular to the best of our knowledge this is the first occasion where nontrivial priority arguments with real "injuries" are needed to answer questions in low-level complexity theory).

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.

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

[LA] R. Ladner, On the structure of polynomial-time reducibility, *J. ACM,* **22**(1975), 155-171.

[LLR] L. Landweber, R. Lipton, and E. Robertson, On the structure of sets in *NP* and other classes, *TCS* **15**(1981), 181-200.

[L] L.A. Levin, On storage capacity for algorithms, *Soviet Math. Dokl.,* **14**(1973), 1464-1466.

[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.