On Reducibility to Complex or Sparse Sets

NANCY LYNCH

University of Southern California, Los Angeles, California

ABSTRACT. Sets which are efficiently reducible (in Karp's sense) to arbitrarily complex sets are shown to be polynomial computable. Analogously, sets efficiently reducible to arbitrarily sparse sets are polynomial computable. A key lemma for both proofs shows that any set which is not polynomial computable has an infinite recursive subset of its domain, on which every algorithm runs slowly on almost all arguments.

KEY WORDS AND PHRASES: polynomial time reducibility, a.e. complexity, many-one reducibility, complexity core

CR CATEGORIES: 5.25, 5.26

1. *Introduction*

In [3], a difference is noted between the polynomial-time-bounded reducibilities of Cook [1] and Karp [2] (whose definitions appear in Section 2): Any recursive set is reducible in Cook's sense to arbitrarily complex (on almost all arguments) sets, but the same is not true for Karp's reducibility. When sparseness is considered rather than complexity, it is discovered that not every recursive set is reducible to arbitrarily sparse sets, according to either definition. This indicates that reducibility arguments of the type used in [6, 7] to prove inherent complexity do not apply to sets which are complex almost everywhere, or to sets which are complex only on a sparse subset of their arguments.

In this paper we strengthen the results in [3] which pertain to Karp's reducibility. In each case, where previously we only knew that a result did not hold for *all* recursive sets, we now show that the only sets for which it holds are those whose characteristic functions are computable in polynomial time. Thus, no inherent complexity higher than polynomial can be proved by efficient reducibility techniques, for sets which are complex almost everywhere, or very sparse. One way of interpreting the first of these results is as evidence for the unnaturality of the condition of complexity on almost all arguments.

Section 2 contains notation and definitions. Section 3 is devoted to proving a lemma which is very similar to a result of abstract complexity theory in [4]; namely, if the characteristic function of a recursive set A is not computable in polynomial time, then there is an infinite recursive subset X of the domain on which *all* algorithms for A run slower than *all* polynomials, for almost all members of X. This set X may be considered to comprise the "core" of the complexity of A.

We use this lemma in Section 4 to show that the only recursive sets which are reducible in Karp's sense to arbitrarily complex (on almost all arguments) sets are those which are polynomial computable. The proof is nonconstructive, an example of an argument which proves that a problem is easy to compute, but which does not exhibit a fast algorithm for its computation.

Author's address: Department of Mathematics, University of Southern California, University Park, Los Angeles, CA 90007.

Similarly, in Section 5, we use the lemma to show that the only sets which are reducible in Karp's sense to sets complex on arbitrarily sparse subsets of their domains are the polynomial computable sets. Again, the proof is nonconstructive.

The obvious questions of constructivity, as well as extension of the sparseness result to Cook's reducibility, are discussed in Section 6.

2. *Notation and Definitions*

All sets will be sets of finite strings over $\Sigma = \{0, 1\}$. If $x \in \Sigma^*$, $| x |$ will represent the length of string x. For any string x, we will write $\hat{x}$ for the integer whose binary representation is $1x$.

For a set A, $| A |$ will represent the cardinality and C_A the characteristic function.

λ represents the empty string. (λ will also be used as in Church's lambda notation.)

We write $\exists^\infty x$ or i.o. (x) to denote "for infinitely many x," and $\forall^\infty x$ or a.e. (x) to denote "for all except possibly finitely many x." When no confusion is likely, we write simply i.o. or a.e.

If A is a set, $t : \Sigma^* \to N$ a recursive function, we write comp $A \leq t$ if there is a Turing machine computing C_A which uses not more than $t(x)$ steps ("time $t(x)$") on any input string x. (The machine is not restricted as to number of tapes, number of worktape symbols, or (standard) input-output conventions.) Similarly, we write comp $A \leq t$ a.e. if a Turing machine exists which computes C_A on all arguments, and which uses not more than $t(x)$ steps a.e. (x). We write comp $A \leq t$ i.o. if a Turing machine exists which computes C_A on all arguments, and which uses not more than $t(x)$ steps i.o. (x). comp $A > t$ a.e. will denote $\sim$comp $A \leq t$ i.o., and comp $A > t$ i.o. will denote $\sim$comp $A \leq t$ a.e.

Also, we write comp $A \leq t$ on X for t, A as above, and set X if there is a Turing machine computing C_A on *all* arguments, using not more than $t(x)$ steps on any $x \in X$. The other definitions in the preceding paragraph are extended analogously.

Following Karp, we write $A \in \mathcal{P}$ if for some polynomial p, comp $A \leq \lambda x[p(| x |)]$.

We write $A \leq_T^P B$ (A is polynomial-time Turing reducible to B) for Cook's reducibility; namely, $A \leq_T^P B$ iff there is an oracle Turing machine M and a polynomial p such that $x \in A \Leftrightarrow M$ with input x and oracle B accepts within $p(| x |)$ steps.

We write $A \leq_m^P B$ (A is polynomial-time many-one reducible to B) for Karp's reducibility. Namely, $A \leq_m^P B$ iff there is a polynomial-time-computable function f such that $x \in A \Leftrightarrow f(x) \in B$. We say $A \leq_m^P B$ *via* f in this case.

3. *A Polynomial Complexity Core*

The following lemma is very similar to one proved in [4] in an axiomatic setting. Since we require the specific result for polynomial-time-bounded computation, we state and prove the new version here. The lemma isolates a "complexity core" for any set not in $\mathcal{P}$.

LEMMA 1. *If A is any recursive set with $A \notin \mathcal{P}$, then there exists an infinite recursive set X such that*

$$(\forall p, \textit{a polynomial})[\textit{comp } A > \lambda x[p(| x |)] \textit{ a.e. on } X].$$

PROOF. Let $\{p_i\}$ be an effective enumeration of a set of polynomials such that $(\forall p, \text{polynomial})(\exists i)(\forall n)[p(n) \leq p_i(n)]$, and $(\forall i, j, n)[i \leq j \Rightarrow p_i(n) \leq p_j(n)]$.

Let $\{\phi_i\}$ be a standard enumeration of functions computed by Turing machines of some fixed type, $\{T_i\}$ the associated running times. X will be constructed in successive stages, $1, 2, 3, \cdots$, with one element x_n being added to X at the completion of each stage n.

We start by setting $y = \lambda$.

Stage n:

(a) For each i, $1 \leq i \leq n$, such that i is not yet *canceled*, see if

$$T_i(y) \leq p_n(| y |) \textit{ and } \phi_i(y) \neq C_A(y).$$

Cancel all i for which both are true. Go to substage (b).

(b) See if for all *uncanceled* i, $1 \leq i \leq n$,

$$T_i(y) > p_n(|\, y \,|).$$

(1) If so, let $x_n = y$, let y be the string having $\hat{y} = \hat{y} + 1$, and go on to stage $n + 1$.

(2) If not, let y be the string having $\hat{y} = \hat{y} + 1$, and return to substage (a).

END

The set $X = \{x_i\}$ constructed in this way is surely recursive. We claim it is infinite, because every stage must eventually terminate. For if not, then let n be the number of the stage that is reached but fails to terminate. Then for sufficiently long strings y, there must exist $i \leq n$, *uncanceled* at substage (b), such that $T_i(y) \leq p_n(|\, y \,|)$. Furthermore, for any $i \leq n$, *uncanceled* at substage (b), we have $T_i(y) \leq p_n(|\, y \,|) \Rightarrow \phi_i(y) = C_A(y)$, for otherwise i would have been *canceled* during substage (a). Thus, if we dovetail the computations of all functions ϕ_i, $1 \leq i \leq n$, for which i is *never canceled*, we obtain an algorithm for C_A on sufficiently long strings y, which runs in polynomial time. A patch for the shorter strings shows $A \in \mathcal{P}$, contrary to our assumption.

Now consider any polynomial p and any $\phi_i = C_A$. i can never be *canceled*. Let n be such that $n \geq i$ and $p_n \geq p$. Then the construction guarantees $T_i(x_n) > p(|\, x_n \,|)$, as required. □

4. *Reducibility to Complex Sets*

It is shown in [5] that all recursive sets are $\leq_T^P$-reducible to arbitrarily complex sets. (A single membership question may be encoded as the mod 2 sum of two questions of membership in the complex set.) The following theorem shows, in a strong way, that this result fails for $\leq_m^P$-reducibility:

THEOREM 2. $(\forall s, \textit{recursive})(\exists B, \textit{recursive})$ [*comp* $B > s$ *a.e.* and $A \leq_m^P B$] *iff* $A \in \mathcal{P}$.

PROOF. $\Leftarrow$ Trivial.

$\Rightarrow$ Assume the hypothesis, and assume $A \notin \mathcal{P}$. Obtain X by Lemma 1.

Let r be some recursive function, monotone nondecreasing in its argument, with $r \geq \lambda x[2^{|x|}]$, and comp $A \leq r$. We assume without loss of generality that for all y, $x \in X$ with $\hat{y} < \hat{x}$, we have $r(y) \leq |\, x \,|$. (For X may be replaced, if necessary, by some infinite subset whose elements are sufficiently separated.)

We now require the following lemma from [3]. It states that for A as in the hypothesis, we may choose a complex set B and a reducibility function f in such a way as to insure that f is very length-decreasing:

LEMMA 3. *Assume A is such that for all recursive s there is some recursive B with $A \leq_m^P B$ and comp $B > s$ a.e. Then for any recursive t,*

$$(\exists B, \textit{recursive})(\exists f)[A \leq_m^P B \textit{ via } f \text{ and } |\, x \,| > t(f(x)) \textit{ a.e.}].$$

PROOF. Detailed verification is given in [3]. The basic idea is that, if the inequality fails to hold, then C_B could be computed quickly i.o. by using the inverse of the function f and a fixed program for A. □

Using this lemma, we will be able to insure that f is very far from 1-1 on X, so that for infinitely many x in X there will be a much smaller y with $f(y) = f(x)$. But this will allow us to obtain a polynomial-time (i.o. on X) algorithm for C_A, as follows: Given x, we will search for a string y, with $\hat{y} < \hat{x}$ and $f(y) = f(x)$, and say $x \in A$ or not, according to whether $y \in A$ or not.

Assuming that $\{x_i\}$ is an enumeration of X in increasing order, we now define a function $t : \Sigma^* \to N$ by $t(y) = |\, x_{2\hat{y}} |$.

By Lemma 3, we obtain B and f with $A \leq_m^P B$ via f and $|\, x \,| > t(f(x))$ a.e. Since t is monotone nondecreasing in its argument, we can easily show:

$$(\forall^\infty(x, y))[|\, x \,| \leq t(y) \Rightarrow \hat{f}(x) < \hat{y}].$$

Since t is defined in such a way as to bound the lengths of elements of X, we see that

$$(\forall w \in \Sigma^*)(\exists C_w \subseteq X, |C_w| = 2\hat{w})(\forall x \in C_w)[|x| \leq t(w)].$$

Combining the last two formulas, we obtain an integer k such that

$$(\forall w \in \Sigma^*)(\exists C_w \subseteq X, |C_w| = 2\hat{w})[|\{x \in C_w : \hat{f}(x) < \hat{w}\}| \geq 2\hat{w} - k].$$

Then since f on so many arguments can take on so few values, the Pigeonhole principle yields:

$$(\forall w)[|\{(x, y) : x, y \in C_w \text{ and } \hat{y} < \hat{x} \text{ and } f(x) = f(y)\}| \geq \hat{w} - k + 1].$$

Thus,

$$(\exists^\infty x \in X)(\exists y \in X)[\hat{y} < \hat{x} \text{ and } f(x) = f(y)]. \tag{1}$$

We now use this function f and a fixed algorithm for C_A to define the following Algorithm $\mathfrak{A}$ for computing C_A:

Algorithm $\mathfrak{A}$. Given input x, compute $f(x)$. Then compute, in order, $f(\lambda), f(0), f(1), f(00), \cdots$. If for any y with $\hat{y} < \hat{x}$, we discover that $f(x) = f(y)$, we compute and output $C_A(y)$ by the fixed algorithm. Otherwise, we compute $C_A(x)$ by the fixed algorithm.

END

By (1), there is an infinite set of arguments $x \in X$ for which the first alternative in Algorithm $\mathfrak{A}$ will hold, and for which the y found will satisfy $\hat{y} \leq \hat{z} < \hat{x}$ for some $z \in X$. For these arguments x, the amount of time used by Algorithm $\mathfrak{A}$ may be bounded by $p(|x|, r(y))$, for some polynomial p, where y represents the first argument found by the search in Algorithm $\mathfrak{A}$. The lower bound on r is used here, as well as the condition comp $A \leq r$. But then by the monotonicity of r and the sparseness of X this number of steps is bounded by $p_1(|x|)$ for some polynomial p_1.

In other words, comp $A \leq \lambda x[p_1(|x|)]$ i.o. on X. But this contradicts the choice of X. □

We note that Theorem 2 seems counterintuitive; this indicates that the condition of large complexity on almost all arguments is an unnatural one to consider.

5. *Reducibility to Sparse Sets*

More in accord with intuition is the result of this section, similar to Theorem 2 but for sparseness rather than complexity. Here we note that "sparseness" refers not to the elements of the set but rather to the arguments on which the characteristic function has more than polynomial complexity.

Definition 4. If $s : \Sigma^* \to N$ is recursive, and B is recursive, we say B *is s-sparse* if there is a Turing machine M computing C_B, and a polynomial p such that for any string x, M runs in time greater than $\lambda y[p(|y|)]$ for at most $\hat{x}$ strings of length less than or equal to $s(x)$.

THEOREM 5. $(\forall s, \text{recursive})(\exists B, \text{recursive})[B \text{ is } s\text{-sparse and } A \leq_m^p B]$ *iff* $A \in \mathcal{P}$.

PROOF. $\Leftarrow$ Trivial.

$\Rightarrow$ Assume the hypothesis, and assume $A \notin \mathcal{P}$. Obtain X by Lemma 1. As for Theorem 2, let r be an upper bound for A's complexity, with the same properties as before, and assume the elements of X are separated by r, as before.

We claim that for any recursive s,

$(\exists B, \text{recursive})(\exists f)[A \leq_m^p B \text{ via } f \text{ and } B \text{ is } s\text{-sparse and}$

$$(\forall i)[\phi_i = C_B \Rightarrow (\forall p, \text{polynomial})(\forall^\infty x)[x \in X \Rightarrow T_i(f(x)) > p(|f(x)|)]]]. \tag{2}$$

(That is, f may be chosen to map most elements of X into the sparse set on which B is complex.) This is so, since the hypothesis on A yields recursive B and f such that $A \leq_m^p B$ via f and B is s-sparse. If some i, p exist with $\phi_i = C_B$ and $(\exists^\infty x \in X)[T_i(f(x))$

$\leq p(|f(x)|)]$, then C_A is polynomial computable for infinitely many elements of X, contradicting the condition on X.

Now let s be defined by: $s(y) = 2^{|x_{\hat{z}}\hat{y}|}$, where $\{x_i\}$ is an enumeration of X in increasing order. We obtain B and f satisfying (2) for this function s.

Now clearly:

$$(\forall w \in \Sigma^*)(\exists C_w \subseteq X, |C_w| = 2\hat{w})(\forall x \in C_w)[|x| \leq |x_{\hat{w}}|].$$

Since B is s-sparse, there exists an integer i and a polynomial p such that $\phi_i = C_B$ and

$$(\forall x)[T_i > \lambda y[p(|y|)] \text{ for at most } \hat{x} \text{ strings of length} \leq s(x)].$$

For this i, p, (2) yields:

$$(\forall^\infty x)[x \in X \Rightarrow T_i(f(x)) > p(|f(x)|)].$$

But then for some constant k,

$$(\forall w)(\forall x)[|\{x \in C_w : T_i(f(x)) > p(|f(x)|)\}| \geq 2\hat{w} - k].$$

By the polynomial computability of f,

$$(\forall^\infty w)(\forall x \in C_w)[|f(x)| \leq 2^{|x_{\hat{z}}\hat{w}|} = s(w)].$$

Combining the last two lines, by the s-sparseness of B and the Pigeonhole principle we obtain:

$$(\forall^\infty w)[|\{(x, y) \mid x, y \in C_w \text{ and } \hat{y} < \hat{x} \text{ and } f(x) = f(y)\} \geq \hat{w} - k].$$

Thus, $(\exists^\infty x \in X)(\exists y \in X)[\hat{y} < \hat{x} \text{ and } f(x) = f(y)]$. The remainder of the proof is completed exactly as for Theorem 2. □

6. *Related Questions*

Theorem 2, as noted, fails to hold for $\leq_T^P$ in place of $\leq_m^P$. The extension of Theorem 5 to $\leq_T^P$ has been announced by R. Solovay [8] and will appear in a future paper.

Since the proofs of both theorems of this paper are based on Lemma 1, which is proved by contradiction, they are both proofs that something is polynomial computable which do not explicitly produce polynomial-bounded algorithms. We wonder if it is possible to provide more constructive proofs which exhibit specific polynomial-bounded algorithms.

REFERENCES

1. Cook, S. A. The complexity of theorem-proving procedures. Conf. Rec. Third ACM Symp. on Theory of Computing, 1971, pp. 151–158.
2. Karp, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., Plenum Press, New York, 1972, pp. 85–103.
3. Lynch, N. Complexity-class encoding sets. Submitted for publication.
4. Lynch, N. Helping: Several formalizations. *J. Symbolic Logic* (to appear).
5. Lynch, N., Meyer, A., and Fischer, M. Relativization of the theory of computational complexity. *Trans. AMS* (to appear).
6. Meyer, A. Weak monadic second order theory of successor is not elementary-recursive. Manuscript, M.I.T., Cambridge, Mass., 1972.
7. Stockmeyer, L. Ph.D. Th., Dep. of Elec. Eng., M.I.T., Cambridge, Mass., June 1974.
8. Solovay, R. Private communication.

RECEIVED AUGUST 1974; REVISED DECEMBER 1974