# ON THE STRUCTURE OF SETS IN NP AND OTHER COMPLEXITY CLASSES

by

L. H. Landweber
R. J. Lipton
and
E. L. Robertson

Computer Sciences Technical Report #342
December 1978

# ON THE STRUCTURE OF SETS IN NP AND OTHER COMPLEXITY CLASSES

L. H. Landweber**
Computer Sciences Department
University of Wisconsin-Madison

R. J. Lipton*
Department of Computer Science
University of California-Berkeley
and
Department of Computer Science
Yale University

E. L. Robertson**
Computer Science Department
Indiana University

---

*Supported in part by NSF Grant number MC578-81486 and the Army Mathematics Research Center.

**Supported in part by NSF Grant number MCS76-17323.

This research was also facilitated by the use of Theory Net (NSF Grant number MCS78-01689).

ABSTRACT

A simple technique is developed for manipulating the relative complexity of sets with respect to polynomial time reducibility. One application is the definition of a minimal pair (with respect to polynomial time reducibility) of sets in NP-P. The last section proves that the NP-complete are effectively enumerable while NP-P is not.

# ON THE STRUCTURE OF SETS IN NP AND OTHER COMPLEXITY CLASSES

## 1. INTRODUCTION

The relationship between resource bounded deterministic and nondeterministic complexity classes has been extensively studied. For polynomial time, the associated question P = NP? is particularly important because of the large number of problems of practical interest that can be solved by nondeterministic polynomial time bounded devices. If P = NP, then these problems all have deterministic polynomial time solutions whereas otherwise only exponential time solutions exist. Furthermore, the class NP contains complete problems to which all other members of NP can be reduced [4,6,8] and P = NP if and only if one of these complete problems is in P. With few exceptions, most intuitively appealing members of NP have been shown to be complete.

In this paper, we study the structure of sets in NP. We develop a number of simple tools which facilitate the study of the relative complexity of sets with respect to polynomial time bounded reducibility. The methods also yield rather elegant proofs for results obtained earlier by Ladner [9] and Lynch [11]. In the former case, the theorems involved are significantly strengthened.

The main idea involves an analysis of those input regions on which set membership questions can be decided in polynomial time. Because of the restrictions of polynomial time reducibility, a

bound may then be obtained on the size of such intervals in sets to which a given set can be reduced. This yields the existence of sets in NP which are not complete and which have a variety of other properties. The existence of sets $A,B \in NP-P$ such that any set reducible to both is in P is also an immediate corollary.

Recently a number of papers [1,2,5,7,11] have dealt with such properties as density and sparseness for sets in NP. A set is p-sparse [2] if there is a polynomial q such that for all n at most q(n) strings of length n are in the set. It is not known whether NP-complete sets can be sparse. The relationship between these results and ours is not clear. By introducing gaps (and thereby easy intervals) into a set a sparser or less dense set is obtained (e.g., if one considers distance between set members). Hence our results can be interpreted as imposing limits on how sets can differ with respect to degree of sparseness and still be related by polynomial time reducibility. The interplay between the various definitions and results is an area for future research.

We believe that our results and those referenced above are important because they provide intuition with respect to the structural properties of NP-complete sets. They represent an attempt to look more closely at why sets are NP-complete as opposed to which sets are NP-complete. While demonstrating that a large number of problems are NP-complete has served to increase the importance of the P = NP question, it does not appear to have brought us closer to a solution to this problem. Hopefully,

further study of the relationship between structure and complexity will help in this regard.

The remainder of the paper investigates the existence of effective enumerations of devices for recognizing all and only members of certain classes of sets. A class admitting such an enumeration is said to be recursively presentable. See [10] for earlier work in this area for abstract complexity measures. In particular, it is proved that the NP-complete sets are recursively presentable and moreover the devices all operate in nondeterministic polynomial time. Similar results are obtained for other complexity classes (PSPACE, EXP-TIME, etc). On the other hand if $P \neq NP$, the class NP-P is not recursively presentable by any devices which compute characteristic functions. It is open whether NP-P can be presented in terms of devices whose domains are the sets in questions. In contrast to NP-complete sets, we show that the complete (with respect to Turing reducibility) recursively enumerable sets are not recursively presentable.

Section 2 presents definitions and notation used in the paper. The basic techniques for studying reducibility relationships are obtained in Section 3. As indicated above these results involve analyzing the size of intervals for which easy (polynomial time) recognition algorithms exist. They are conceptually simple and appear to have wide applicability to the study of structural properties of sets in NP. Section 4 uses these methods to generalize results due to Ladner [9] on the existence of non complete sets in NP-P and the definition of a

minimal pair in NP. The last section studies the question of recurisve presentability for various classes of sets.

## 2. DEFINITIONS

Notation : $\Sigma^*$ denotes the finite length strings over the finite alphabet $\Sigma$. We use $|x|$ for the length of a string in $\Sigma^*$. By convention, n and m denote string lengths while letters at the end of the alphabet denote strings.

Assume a fixed bijection between $\Sigma^*$ for finite alphabet $\Sigma$ and the natural numbers, N(e.g., the p-adic notation of [3]). If x is a string and n a natural number, then notation such as $x<n$ means that the number represented by x is less than n.

Our basic model of computation is the multitape Turing Machine (TM). All machines are assumed to be deterministic unless otherwise specified. A (possibly nondeterministic) Turing Machine M has time complexity (space complexity) f(n) or runs in time (space) f(n) if for all inputs of length n, M uses at most f(n) moves (tape squares) on all computation paths. If M runs in time (space) p(n) where p is a polynomial, then we say that M runs in polynomial time (space). Let

$$\text{NTIME}[f(n)] = \{A \mid A \subseteq \Sigma^*, \text{ A recognized by a nondeterministic TM in time } f(n)\}$$

$$\text{DTIME}[f(n)] = \{A \mid A \subseteq \Sigma^*, \text{ A recognized by a deterministic TM in time } f(n)\}$$

NSPACE[f(n)] and DSPACE[f(n)] are defined similarly. Then

$$P = \bigcup_{i>0} \text{DTIME}(n^i)$$

$$NP = \bigcup_{i>0} NTIME(n^i)$$

$$PSPACE = \bigcup_{i>0} DSPACE(n^i) = \bigcup_{i>0} NSPACE(n^i)$$

$$DEXP\text{-}TIME = \bigcup_{i>0} TIME(2^{in}).$$

NEXP-TIME, DEXP-SPACE and NEXP-SPACE are defined similarly.

We consider two notions of polynomial time reducibility due to Cook [4] and Karp [8]. A set A is many-one reducible to a set B in polynomial time ($A \leq_p B$) if there is a function f computable by a Turing Machine which runs in polynomial time such that $x \in A$ if and only if $f(x) \in B$. The function f of the above definition is often referred to as a transducer.

An oracle Turing Machine (with oracle $B \subseteq \Sigma^*$) is a multitape TM with a special tape called the oracle tape. The machine may periodically write a string on the oracle tape and then in one step branch depending on whether the string is in the oracle set B. Of course, if B is not effectively computable, then the computation could not actually be performed. An oracle machine runs in time f if it uses at most f(n) moves on inputs of length n. A set A is Turing reducible to a set B in polynomial time ($A \leq_{pT} B$) if there is an oracle Turing Machine M which runs in polynomial time such that x is in A if and only if M halts in an accepting state on input x with oracle B.

Write $A <_p B$, if $A \leq_p B$ but not $B \leq_p A$. Also $A \equiv_p B$ if $A \leq_p B$

and $B \leq_p A$. If $A \equiv_p B$, we say A and B are in the same degree with respect to $\leq_p$ or are $\leq_p$-equivalent. Similarly define $<_{pT}$ and $\equiv_{pT}$.

Many-one and Turing polynomial reducibility are useful tools for classifying the relative complexity of problems. For example, if $A \leq_p B$, then a polynomial time recognition algorithm for B yields a polynomial time recognition algorithm for A. If $A \equiv_p B$, then modulo a polynomial the complexity of B imposes an upper bound on the complexity of A.

A set A is X-m-complete (X-T-complete) for some complexity classs of sets X if $A \in X$ and $B \leq_p A$ ($B \leq_{pT} A$) for all $B \in X$.

We will omit the prefix when the meaning is clear. Complete sets are in some sense the hardest sets in a complexity class. For X = NP, Cook [4] showed that the set of satisfiable propositionl calculus formulas is complete. Karp [8] and many others have since demonstrated that a very large number of interesting problems are NP-complete (see Garey and Johnson [6]).

A function f is <u>time constructible</u> if there is a Turing machine which on inputs of length n halts in exactly f(n) steps. Polynomials, $2^n$ and n! are examples of time constructible functions. It is easy to see that for any recursive r there is a recursive $r' > r$ such that r' is time constructible (On input x , $|x| = n$, compute r(n) and halt. Then $r'(n) > r(n)$ is just the number of steps required to compute r(n)).

A set of functions $\{f_i\}$ is said to be <u>recursively</u>

presentable if there is an effective enumeration $M_0, M_1, \ldots$ of algorithms (Turing Machines) such that 1) each $f_i$ is computed by some $M_j$ and 2) each $M_j$ computes a member of $\{f_i\}$. Sets are identified with their characteristic functions yielding a definition of recursively presentable for a collection of sets. When the context is clear, we will use the same symbol to denote a set, its characteristic function and an algorithm for the characteristic function. In particular SAT will denote the set of satisfiable propositional calculus formulas encoded suitably in an alphabet.

In the following, $Q_0, Q_1, \ldots$ denotes a fixed recursive presentation of P. The enumeration is in terms of Turing Machines which accept all and only members of P and which, in addition, operate in polynomial time. The required enumeration is obtained by attaching polynomial time clocks to Turing Machines. For each TM M and each i, there is a copy of M with an $n^i$ clock in $\{Q_i\}$. The modified devices operate in polynomial time because polynomials are time constructible.

For any function f, define $f^{\#}$, the exponentiation [3] of f, by

$$f^{\#}(x,0) = f(x)$$
$$f^{\#}(x,y+1) = f(f^{\#}(x,y)).$$

For r increasing, define G[r] by

$$G[r] = \{x \mid r^{\#}(0,n) \leq |x| < r^{\#}(0,n+1) \quad \text{for} \quad n \quad \text{even}\}.$$

G[r] will be used when we wish to introduce r sized gaps into

sets.

For $n \in N$, the r-interval at $n$, $I[r,n]$ is given by

$$I[r,n] = \{y \in \Sigma^* \mid n \leq |y| < r(n)\}$$

A set $B$ has r-gaps if there are arbitrarily large $n$ such that $y \in I[r,n]$ implies $y \notin B$.

Let $Q$ be a polynomial time algorithm and let $B \subseteq \Sigma^*$. The algorithm $Q$ is correct for $B$ on arbitrarily large r-intervals if there exist arbitrarily large $n$ such that for all $y \in I[r,n]$, $Q(y) = B(y)$.

A recursive set $B \subseteq \Sigma^*$ is r-interval easy for some recursive $r$, if there is a polynomial $q$ and an algorithm having time complexity $f$ which recognizes $B$ such that for infinitely many $n$, $y \in I[r,n]$ implies $f(|y|) \leq q(|y|)$.

## 3. STRUCTURAL PROPERTIES

Previous studies have revealed a rich degree structure with respect to polynomial time reducibility . In this section we investigate structural properties of sets which result from their position in the reducibility hierarchy. The strict constraints associated with polynomial time reducibility allow us to develop a fairly simple tool which has a number of applications. Analogous questions in classical recursion theory include the relationship between simple and hypersimple sets and various reducibility notions.

The first theorem imposes a maximum gap size on sets to which a given set is many-one polynomial time reducible.

Theorem 1. Let A be a recursive set. If $A \notin P$, then there is a recursive r such that B has maximum gap size r for each B such that $A \leq_p B$.

Proof. Let $A \notin P$ be a recursive set. The function

$$d(n) = \max_{i \leq n} |[(\mu y)(|z|>n \text{ and } Q_i(z) \neq A(z))]|$$

is therefore total and at each n, d(n)-n provides an upper bound on the interval for which A can agree with the set accepted by the polynomial time algorithm $Q_i$ (for $i \leq n$).

The proof uses the function d to help provide an upper bound for gaps in B where $A \leq_p B$. Intuitively, if B has very large gaps, then the polynomial time transducer from A to B must map A members not near the gap boundary to much smaller

strings. If the shrinkage is sufficient, then we can use a polynomial time algorithm to decide A membership for a portion of the gap which exceeds the bound given by d. This intuitively appealing idea yields an upper bound on the possible gap size in B.

Assume A is recognized by an algorithm which operates in time $g_A$. Choose $g_A$ to be time constructible and increasing (by padding the output of $g_A$). Define $g'_A$ by

$$g'_A(n) = \min [(\mu m)\ (g_A(m) \geq n) \dot{-} 1, n]$$

The value of $g'_A(n)$ is the smaller of n and one less than the length of the shortest input whose time complexity is at least n. Then

$$m \leq g'_A(n) \quad \text{implies} \quad g_A(m) < n.$$

Now choose B such that $A \leq_p B$ via transduction t. We define a polynomial time algorithm Q which agrees with parts of A, dependent on the size of gaps in B:

$Q(x)$: <u>if</u> $(\exists y \leq g'_A(|x|))[t(y) = t(x)]$
<u>then</u> $A(y)$
<u>else</u> reject x

The algorithm Q is polynomial because at most $|x|$ y's are examined, t is polynomial-time computable, $g_A$ is time constructible, and $A(y)$, if evaluated, requires less than $|x|$ steps by the choice of y.

Consider a gap in B beginning at some length n. Pick an

n which is large enough (I.e., there are indices for A and the identically zero function which are $\leq n$.) to insure that

a. A can not agree with Q

and

b. A can not agree with the identically zero function on the interval [n,d(n)].

The second condition guarantees that some member of A is in the interval [n,d(n)]. We must determine the largest possible gap in B at n. That is, what is the largest n' such that

$$n \leq |y| < n' \text{ implies } y \notin B.$$

The idea is to build a gap on which A agrees with Q but which contradicts the definition of d. Observe that if $n \leq |t(x)| < n'$, then $x \notin A$ and Q is correct on x. Also $|t(x)| > n'$ is only possible if $|x|$ is within a polynomial distance from n' since t is a polynomial time transducer.

In the following, we will consider the sub-interval [n,n"] where [n",n'] is the portion of the total interval which is near the boundary. All members of A in [n,n"] must be mapped via t to strings which are shorter than n.

If $|x| > g_A(x_\emptyset)$, for some $x, x_\emptyset$, we can infer that $x_\emptyset \leq g'_A(|x|)$ (Note that $x_\emptyset > g'_A(|x|)$ implies there is a y, $|y| \leq x_\emptyset$ such that $g_A(|y|) \geq |x|$. But then $g_A(x_\emptyset) \geq |x|$ contradicting the assumption.) Thus for x satisfying $|x| > g_A(d(n)) \geq g_A(x_\emptyset)$ and $t(x) = t(x_\emptyset)$, the algorithm Q is correct. The maximum possible gap is then obtained by sucessively iterating $g_A \circ d$. In each such extension there must be at least one A member by the

choice of d because d bounds easy A intervals and if no string in an interval is in A, then A is easy on that interval. Furthermore, by the pigeonhole principle, within $2^n+1$ such extensions there must be an extension such that if x is in the extension and $|t(x)|<n$, then $t(x)=t(y)$ for some y in an earlier extension. The algorithm Q is then correct on this last extension which contradicts the definition of the function d and the choice of n (since the extension is longer than d). Hence the maximum gap size in B is bounded by

$$(\lambda n)\ [\exp\circ(g_A\circ d)^{\#}(n,2^n+1)]$$

where the function exp is added to provide the gap from n" to n. ∎

If we restrict the above result to recursive B such that $A\leq_p B\leq_p C$ for given recursive $A,C\notin P$, the proof becomes considerably easier. Let $g_C$ be the time complexity of C. Then $g_C$ provides a uniform bound on the time complexity of all such sets B. If $g'_C$ is defined as in the previous theorem, then Q(x) is given by

q(x): <u>if</u> $t(x)\leq g'_C(|x|)$
<u>then</u> B(t(x))
<u>else</u> reject x

The maximum gap size in B is then $\exp\circ(d\circ g_C)$.

Corollary 2. If $A \notin P$, then there is a maximum gap size for the $\equiv_p$ class of A.

The fact that a set has r-gaps for large r does not imply that is "easy". By standard complexity theory techniques, recursive sets of arbitrary complexity may be constructed with r-gaps (although the construction requires that the set be more complex than r). This together with Theorem 1 means that given any set A, there are arbitrarily complex sets with arbitrarily large gaps which are incomparable (with respect to many-one polynomial time reducibility) with A. Conversely, the introduction of large enough (polynomial time recognizable) gaps in a set A guarantees that the resulting set A' satisfies $A' <_p A$. (Theorem 8 of Section 4 provides a more significant application of this technique because the A' above may be in P whereas $A \notin P$.)

In contrast to the above, note that polynomial size gaps can be introduced in SAT with the resulting set $SAT \cap G[p]$ remaining NP-complete. Show $SAT \leq_p SAT \cap G[p]$ as follows:

Given x:

if $x \in G[p]$

then compute SAT(x)

else pad x to an equivalent $x' \in G[p]$

and compute SAT(x')

By methods similar to the above one may show:

Theorem 3. Let B have r-gaps for some recursive r. Then there exists an $r' \geq \log \cdot g'$ (g' as in the proof of Theorem 1) such that for all $A \leq_p B$, $A = A' \cup A''$ where $A'' \in P$ and A' has r'-gaps.

The following results deal with the existence of gaps for NP-complete sets. Related questions were considered in [1,2,5,7]. Studies of the gap-density properties of NP-complete sets will hopefully increase our intuition with respect to the difficult open problems for this class.

Theorem 4. Assume that a polynomial time algorithm is correct for SAT for arbitrarily large r-intervals and some recursive r. Then there is a polynomial time computable set A such that $SAT \cap A$ is NP-complete but has r-gaps.

Proof. Assume SAT has r-intervals for which it agrees with the set recognized by the polynomial time algorithm Q. That is, for arbitrarily large n, $y \in I[r,n]$ implies SAT(y) = Q(y). We define a set $A \in P$ such that $SAT \leq_p SAT \cap A$ via a polynomial time transduction t. The set A, will not include any strings y such that Q(y) = SAT(y). Since Q and SAT agree for arbitrarily large r-intervals, this will mean that $SAT \cap A$ has r-gaps.

Let b be a fixed member of SAT. Also put b in A. Define $\overline{A}$ to be the set of inputs (except b) for which the transducer t, defined below, outputs b.

Transducer t:

Input:y

$w \Leftarrow y$

while

w has variables with no value assigned

pick a variable of w and assign 0 and 1

to obtain $w_0$ and $w_1$ - do not simplify

so $|w| = |w_1| = |w_0|$ (so $w, w_0, w_1$ are

in the same r-interval).

case

(choose first true alternative)

$Q(w_0)=1$ : $w \Leftarrow w_0$ and record assignment

$Q(w_1)=1$ : $w \Leftarrow w_1$, and record assignment

else : output y, halt

end case

end while

if variable values found satisfy y

then output b (Q is correct on y and $y \in SAT$,

$b \in SAT \cap A$)

else output y (so $y \in A$)

Then $SAT \leq_p SAT \cap A$ via t because:

a. $y \in SAT$ implies $t(y)=b \in SAT \cap A$ or $t(y)=y \in SAT \cap A$

b. $y \notin SAT$ implies $t(y)=y \notin SAT \cap A$.

The transducer clearly runs in polynomial time. It remains to show that $SAT \cap A$ has r-gaps. Assume $y \in SAT$ is in an interval $I[r,n]$ for which Q is correct. Then the output of t on y

is b so $y \notin A$ by the definition of A. Hence $y \notin SAT \cap A$ and $SAT \cap A$ has r-gaps corresponding to the r-intervals on which Q is correct for SAT. ∎

Theorem 5. Let B be an NP-complete set, r a recursive function and Q a polynomial time algorithm such that Q is correct for B on arbitrarily large r-intervals. Then there exist NP-complete sets with f r log-gaps where f is any function satisfying $f^{-1}(n) > n^k$ a.e. for all k.

Proof. We show that SAT has f∘r∘log-gaps. Let $SAT \leq_p B$ via t. Assume Q is correct for B on arbitrarily large r-intervals. Consider such an r-interval, $n \leq |y| < r(n)$ where n is chosen large enough to allow f to compensate for possible growth in |y| resulting from the application of t. Consider a string y such that |y| is in the subinterval $[2^n, f \circ r(n)]$:

a. If $|t(y)| \in [n, r(n)]$, and $|t(y)| \geq \log(|y|)$ then the algorithm Q applied to t(y) is corect.

b. If $|t(y)| < \log|y|$, then t(y)∈B and y∈SAT can be decided in polynomial (in |y|) time using an exponential time recognizer for B(B is in NP).

Hence there is a polynomial time agorithm which is correct for SAT for intervals $[2^n, f \circ r(n)]$ for arbitarily large n. The result follows by substituting log(n) for n and applying Theorem 4. ∎

The next theorem provides a bound for each $A \notin P$ on the size of intervals for which A can be easy, i.e., on which it can be recognized in polynomial time. Notice that this also yields an

upper bound on permissible gaps in non-polynomial sets since recognition is trivial on gaps. This theorem and the one following enable us to strengthen earlier results on the structure of sets in NP and, in addition, provide an elegant and consistent approach for obtaining such results.

Theorem 6. Let $A \notin P$ be recursive. Then there is a recursive r such that A is not r-interval easy.

Proof. Since $A \notin P$, the witness function

$$w(i,n) = (\mu z)[|x| \geq n \text{ and } A(z) \neq Q_i(z)]$$

is total. The required r is

$$r(n) = \max_{i \leq n} |\{w(i,n)\}| + 1.$$

Assume A is r-interval easy, with an algorithm M having time compexlity $f_A$ and polynomial q. Then the algorithm

if $f_A(|x|) \leq q(|x|)$

then A(x) (using M)

else 0

is polynomial time (Check $f_A(|x|) \leq q(|x|)$ in polynomial time by attaching a clock for q to M) and is therefore computed by $Q_j$ for some j. Furthermore, $f_A(|y|) \leq q(|y|)$ implies $Q_j(y) = A(y)$. But then there is an $n > j$ such that $y \in I[r,n]$ implies $A(y) = Q_j(y)$ but this contradicts the definition of r because then

$$w(j,n) \geq r(n) > w(j,n)$$

Hence A is not r-interval easy. ∎

Theorem 1 is based on the fact that if B has large gaps and $A \leq_p B$, then the transduction from A to B must produce short strings on inputs in A which are also in a B-gap. A more complicated argument must be used for polynomial time Turing reductions, where more than one question may be asked during a computation and where answers must actually be computed (or somehow simulated) for the computation to proceed. For Turing reducibility, the upper bound on gap sizes is not as sharp as for many-one polynomial-time reducibility(by one exponential).

Theorem 7. Let $A \notin P$ be recursive. There is a recursive f such that if $A \leq_{pT} B$ and B is r-interval easy, then $r \leq f$ a.e.

Proof. The proof is similar to that of Theorem 1 in its strong use of the pigeon-hole principle. Assume $A \leq_{pT} B$ via some algorithm T and B is r-interval easy with algorithm having complexity $f_B$ and polynomial q. Let $g_A$ be the complexity of a recognition algorithm for A where $g_A$ is increasing and time constructible. Define $g'_A$ by

$$g'_A(n) = \min[(\mu m)(g_A(m) > n) \dot{-} 1, n].$$

Then $m \leq g'_A(n)$ implies $g_A(m) < n$.

The function d given by

$$d(n) = \max_{i \leq n} |[(\mu z)(|z| > n \text{ and } Q_i(z) \neq A(z))]|$$

for each n provides an upper bound for $i \leq n$ on the interval for which A can agree with the set accepted by the polynomial time algorithm $Q_i$.

We now give an algorithm Q for A. If B is easy on intervals which exceed a function f, to be defined, then Q will be easy on intervals which are too large, i.e., which contradicts the defintion of d.

Algorithm Q

input: $x$

1. Simulate T on input x. If T queries the oracle for some y, test whether

$$f_B(|y|) \leq q(|y|)$$

If yes, the oracle answer can be obtained directly in polynomial time. If not, simulate branches in the computation for $y \in B$ and $y \notin B$. All paths eventually terminate. For each path define a triple $\langle Y_t, Y_f, a \rangle$.

where

$Y_t = \{u \mid u \in B$ assumed on path$\}$

$Y_f = \{u \mid u \notin B$ assumed on path$\}$

$$a = \begin{cases} \text{yes if path accepts } x \\ \\ \text{no if path rejects } x \end{cases}$$

Call the set of triples S(x).

2. If there is a y such that

$$|y| \leq \min(g'_A(|x|), \log \circ \log(|x|))$$

and $S(y) = S(x)$,

then output $A(y)$ - $(g_A(|y|)$ steps

else output $A(x)$ - $(g_A(|x|)$ steps

Assume $x \in I[r', n]$ for $r'$ recursive such that $f_B \leq q$ on that interval and

$2^{2^n} < |x| < \log \circ r'(n)$ (see Figure 1). (*)

The condition (*) insures that oracle questions are either easy on are for strings much shorter than x($<\log \circ \log(|x|)$).

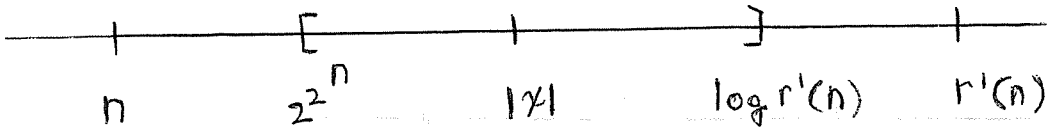


Figure 1

Consider algorithm Q on input x. Because B is easy on [n,r'(n)], only B-oracle questions for y such that $|y|<n$ are included in the sets $y_t$ and $y_f$. Hence each $y_t, y_f$ contains at most $2^n$ members and the cardinality of S(x) is bounded by $2^{2^{(2n+1)}}$. But $|x|>2^{2^n}$ so part 1 of Q can be completed in polynomial time for the given x. Similarly, (since $|y| \leq \min(g'_A(|x|), \log \circ \log(|x|))$, part 2 of Q must compute S(y) for at most log(|x|) strings y. The $Y_t$ and $Y_f$ sets of each such S(y) will only contain strings of length less than n (since B oracle questions for any z in I[r',n] are evaluated directly). Hence as above, the determination of S(y) can be done in polynomial time. Finally, note that if a y such that S(y) = S(x) is found, then $x \in A$ iff $y \in A$ and the condition $|y| < g'_A(|x|)$ insures that $g_A(|y|) < |x|$ so the algorithm for A can be applied to y to decide $x \in A$ in polynomial time. Thus for $2^{2^n} < |x| < \log \circ r'(n)$, Q operates in polynomial time.

We now use the pigeonhole principle for the sets S(x) to define an upper bound f for easy B intervals. For x and n as above there are $2^{(2n+1)}$ triples and hence $2^{2^{(2n+1)}}$ sets of triples. Assume B has easy intervals of size r' where r'(n) exceeds

$$f(n)=\exp\circ d\circ(g_A\circ d)^{\#}(n\ ,\ 2^{2^{(2n+1)}})$$

for infinitely many n. Since B is easy on [n,f(n)], Q is easy for A on at least one subinterval [m,d(m)], because every set of triples occurring for $x\in[m,d(m)]$ have occurred for sufficiently small y ($g_A\circ d$ is iterated $2^{2^{(2n+1)}}$ times = number of sets of triples). This contradicts the definition of d so r' cannot exceed f. Notice that the function exp provides a buffer preventing B oracle questions above the interval on which B is easy. ∎

## 4. APPLICATIONS

In Section 3, we introduced the notion of (easy) gaps and developed a few relationships between gaps and complexity structures. We now apply this machinery to study the existence of noncomplete sets in NP having certain properties. Unless otherwise specified, we assume throughout that $P \neq NP$.

Theorem 8. [Ladner] Let $B \notin P$ be a recursive set. Then there exists a recursive set A satisfying:

a. $A \notin P$

b. $A \leq_p B$

c. $B \not\leq_{pT} A$

Proof. Let $B \notin P$ be a recursive set. Choose r recursive such that r is time constructible and C is not r-interval easy for any C such that $B \leq_{pT} C$. Let $D = G[r]$. Clearly $D \in P$. Let $A = B \cup D$. Since $B \cup D$ is r-interval easy, $B \not\leq_{pT} B \cap D$ by Theorem 7. Furthermore, $B \cap D \leq_p B$ by the trivial transduction

$$\text{if } x \in D \text{ then } x \text{ else } x_0$$

where $x_0$ is a fixed member of $\overline{B}$.

Finally, $B \cap D \notin P$ because B is not r-interval easy $(D = G[r])$. ∎

Corollary 9. [Ladner] There are sets in NP-P which are not NP-complete.

Proof. Choose B in the theorem to be NP-complete. Then $B \cap D \in NP-P$ but is not complete.

Observe that by introducing sufficiently large gaps, any NP-complete set can be transformed into a member of NP-P which is not complete. Furthermore, by choice of sufficiently large r, we can diagonalize out of any r.e. class of infinite recursive sets.

Corollary 10. Let $C_0, C_1, \ldots$ be a recursive presentation of a class of infinite recursive sets. Let $B \notin P$ be recursive. Then there exists a recursive set A satisfying

1. $A \notin P$
2. $A \leq_p B$
3. $B \not\leq_{pT} A$
4. $C_i \not\subseteq A$ for any i.

Hence, the set A of Theorem 8 can be chosen so as not to have an infinite regular or context free subset. ∎

By Theorem 1, the introduction of sufficiently large gaps into a set A results in a set A' satisfying $A' <_p A$. Together with a corollary to the following lemma this yields a strong version of Theorem 8.

Lemma 11. For any recursive A,B, if $A \cap \overline{B} \in P$, then $A \equiv_p A \cap B$.

Proof. Assume A,B are recursive sets and $A \cap \overline{B} \in P$. If $A \cap B$ is empty, then A and $A \cap B$ are in P.

1. $A \leq_p A \cap B$: Let d be a fixed member of $A \cap B$. The required transducer is

```
Given x
    if x∈A∩B̄
        then output d
        else output x
```

2. $A \cap B \leq_p A$: Let d be a fixed member of $\overline{A}$. The required transducer is

```
Given x
    if x∈A∩B̄
        then output d
        else output x
```

∎

Corollary 12. For any recursive B and NP-complete A, if $A \cap B$ is not NP-complete, then $A \cap \overline{B} \notin P$.

Corollary 13. Any NP-complete set A can be decomposed into sets B,C such that

1. $A = B \cup C$
2. $B, C \in NP-P$
3. B,C are not NP-complete

Proof. Let A be NP-complete. Let $B = A \cap G[r]$, $C = A \cap G[r]$ for r

sufficiently large to force $B \leq_p A$, $C \leq_p A$ (Theorem 1). Then by Corollary 13, neither B nor C is in P. ∎

A similar result can be proved for other complete degrees (e.g., PSPACE).

Two recursive sets $A_1$ and $A_2$ form a minimal pair for P if neither is in P but $D \leq_{pT} A_0$ and $D \leq_{pT} A_1$ implies $D \in P$ for all D. Ladner [9] proved the existence of minimal pairs for P but did not give an upper bound of the complexity of the pair. We are able to use the methods of Section 3 to obtain minimal pairs of non NP-complete sets in NP.

Theorem 14. Let $B \in P$ be recursive. Then there are recursive $A_0$ and $A_1$ such that

a. $A_i \leq_p B$ $i = 0,1$

b. $A_i \notin P$ $i = 0,1$

c. $D \leq_{pT} A_i$ $i = 0,1$ implies $D \in P$.

Proof. Let $B \notin P$. Pick r time constructible such that B is not r-interval easy and r(n) is greater than exp(f(n)) where f is the computation time of some algorithm which recognizes B. Define

$$S_0 = \{y \mid r^{\#}(0,4i+1) \leq |y| < r^{\#}(0,4i+2),\ i \geq 0\}$$
$$S_1 = \{y \mid r^{\#}(0,4i+3) \leq |y| < r^{\#}(0,4i+4),\ i \geq 0\}$$

and let $S_2 = \overline{S_0 \cup S_1}$. Figure 2 shows how this partitions N.

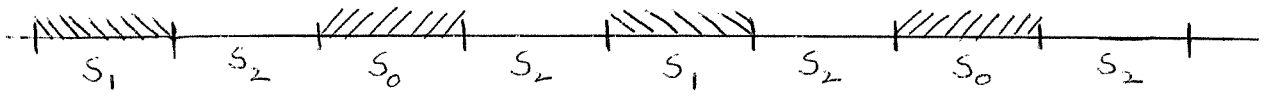


Figure 2

Let $A_i = B \cap S_i$, $i = \emptyset, 1$. The sets $S_\emptyset, S_1, S_2$ partition N with $S_2$ providing a buffer between $S_\emptyset$ and $S_1$. Because r is time constructible, each $S_i$ is in P. Now part (a) follows from the definition of $A_i$ ($A_i \leq_p B$) and because B is not r-interval easy (If $B \leq_p A_i$, then B is easy on $S_{1-i}$). If $B \leq_p A_i$ via transducer t, then for input x in $S_{1-i}$, t(x) either is in $S_{1-i} \cup S_2$ in which case x is not in B or t(x) is in $S_i$. In the latter case, $f(|t(x)|) < |x|$. Hence in either case, $x \in B$ can be decided in polynomial time. Part (b) is true because B is not r-interval easy.

To show (c) assume $D \leq_{pT} A_i = B \cap S_i$, $i = \emptyset, 1$. The main idea of the algorithm for D is that oracle queries are either trivial or involve B computations on strings much shorter than the input. This is accomplished by forcing each B question into an $S_\emptyset$ or $S_1$ interval preceding that of the input (see Figure 3).

The algorithm for D is as follows where B queries use the algorithm having complexity f.

INPUT x:

case

$x \in S_i$ $i = \emptyset, 1$: use $D \leq_{pT} B \cap S_{1-i}$ computation

for query $y \notin S_{1-i}$ answer no

for query $y \in S_{1-i}$ compute $y \in B$

$x \in I[r,n] \subseteq S_2$, $r(n) \in S_i$

if $|x| > \log \circ r(n)$

then use $x \in S_i$ case above

else use $x \in S_{1-i}$ case above

end if

end case

end

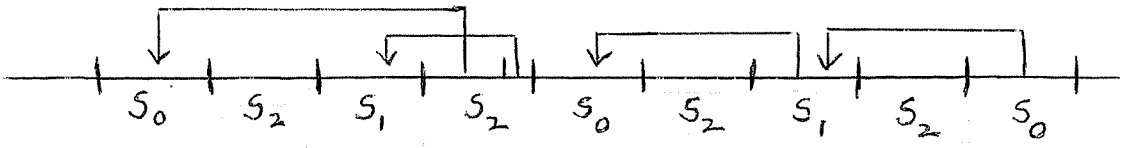


Figure 3

The above algorithm for D is polynomial because all B queries y satisfy $f(|y|) < |x|$ where x is the initial input. This follows because $r(n) > \exp f(n)$. In particular, if $x \in S_\emptyset (x \in S_i)$ and y is a non-trivial B query, then y is in an earlier $S_1 (S_\emptyset)$ interval. If $x \in I[r,n] \subseteq S_2$, then a non-trivial B query is either in the immediately preceding interval (if $|x| > \log r(n)$) so $f(|y|) < f(n) < \log(r(n)) < |x|$) or in a still earlier $S_\emptyset$ or $S_1$ interval. Since there are at most a

polynomial number of queries each of which requires at most a polynomial number of steps, the algorithm runs in polynomial time. ∎

Corollary 15. There exist $A,B \in NP-P$ which are not NP-complete such that $D \leq_p A$ and $D \leq_p B$ implies $D \in P$.

Corollary 16. For any $B \in NP-P$ there exists a minimal pair $A_0, A_1 \notin P$ such that $A_i <_p B$ $i = 0,1$.

By similar methods to the above we may prove a strong minimal pair theorem.

Theorem 17. Let $A,B \notin P$ and $A <_p B$. Then there are $C_1$ and $C_2$ such that

a. $A <_p C_i <_p B$ $i=1,2$

b. $D <_p C_i$ $i=1,2$ implies $D \leq_p A$

Lynch [11], Berman [1] and Hartmanis and Berman [7] have studied the density of sets with respect to a number of different definitions. We can use Theorem 1 to obtain a result due to Lynch which states that if a set is reducible to arbitrarily sparse sets, then the set is in P.

Definition [Lynch]. A set A is s-sparse if there is a polynomial p and an algorithm for recognizing A which runs in time p(n) for all but at most n strings of length less than or equal to s(n).

We use Theorem 1 to show that sets reducible to arbitrarily s-sparse sets must be in P.

Theorem 18 [Lynch]. For any recursive A

$(\forall$ recursive s$)(\exists$ recursive B$)$[B is s-sparse and $A \leq_p B$] iff $A \in P$

Proof.

<= trivial

=> Let $A \notin P$ and assume that A is reducible to the s-sparse set B. Let p be the polynomial for B given by the definition of s-sparse. Construct B' by removing from B all strings, except the shortest one, which are recognized in time p by the B algorithm. Then $A \leq_p B'$ and B' has s(n)/n gaps. Since such sets B exist for arbitrary s, this contradicts Theorem 1. ∎

Theorem 19. For any recursive A

$(\forall$ recursive S$)(\exists$ recursive B$)$[B has s-gaps and $A \leq_p B$] iff $A \in P$

## 5. ENUMERATION PROPERTIES

If a class of sets is recursively presentable, then there is an effective enumeration of devices which recognize all and only sets in the class[10]. The existence of such an enumeration is useful when studying properties of the class. In this section, we show that various important complete complexity classes are recursively presentable. Furthermore, the presentations can be in terms of efficient devices. Whereas the NP-complete sets are recursively presentable, this is not true of either NP-P or the non complete sets in NP-P. This may be interpreted as suggesting why so few intuitively appealing problems in NP ,which are not complete, have been found, i.e., perhaps most 'natural' problems are complete.

Theorem 20. If $P \neq NP$ the class of NP-complete sets is recursively presentable. The presentation is an enumeration of algorithms which run in non-deterministic polynomial time and which accept all and only NP-complete sets.

Proof. Let $A_0, A_1, \ldots$ and $t_0, t_1, \ldots$ be recursive presentations of the non-deterministic polynomial time recognizable sets (by non-deterministic polynomial time devices) and the polynomial time transducers respectively. We describe the algorithm $A_i^j$ which accepts $L(A_i)$ in case $t_j$ is a correct transduction from SAT to $L(A_i)$, i.e., if for all x, $x \in SAT$ if and only if $t_j(x) \in L(A_i)$. If $t_j$ is not a correct transduction, then $L(A_i^j)$ is SAT almost everywhere, that is, for sufficiently long inputs.

Given input $x$, $A_i^j$ is defined by:

1. For each $y$, $|y| < \log \cdot \log(|x|)$, check if $y \in SAT$ iff $t_j(y) \in L(A_i)$
2. If (1) returns true for all $y$
   then $A_i^j(x) \Leftarrow A_i(x)$
   else $A_i^j(x) \Leftarrow SAT(x)$

$A_i^j$ operates in non-deterministic polynomial time because part (1) only checks small strings for which the checking can be done in deterministic polynomial time. As long as (1) does not discover that $t_j$ is not a correct transduction from SAT to $L(A_i)$, $A_i^j$ accepts $L(A_i)$. Once an error is discovered, it will occur in part (1) for all longer inputs so $A_i^j$ accepts an NP-complete set. It remains to note that for every NP-complete set $B$ there are $i,j$ such that $L(A_i) = B$ and $t_j$ is a transduction from SAT to $B$. ∎

The method of proof of the previous theorem can be used to obtain similar results for the complete sets of any time or space complexity class admitting an effective enumeration of the sets of the class by efficient devices (I.e., by devices whose complexity is bounded by the functions which define the class.)

Corollary 21. If $P \neq PSPACE$, then the PSPACE-complete sets are recursively presentable by devices which operate in polynomial space.

Corollary 22. The classes of exponential (deterministic or nondeterministic) time or space complete sets are recursively

presentable by devices which operate in (deterministic and nondeterministic) exponential time or space respectively.

The enumeration of Theorem 20 can be modified so that each NP-complete set is accepted by exactly one device in the enumeration. The proof involves a finite injury priority method whereby each device monitors the enumeration and may be forced to modify the set it is accepting in case a device which is earlier in the enumeration appears to be accepting the same set.

In contrast to the above result, we now show that NP-P is not recursively presentable even if the only requirement on the devices is that they always halt, i.e., that they compute the characteristic functions of sets in NP-P. For NP-complete sets, the enumeration was in terms of non-deterministic polynomial time devices so that an enumeration exists in terms of deterministic exponential time devices for the characteristic functions.

Theorem 23. Unless P = NP, there is no recursive presentation of NP-P by characteristic function. In addition, the non complete sets in NP-P can not be recursively presented.

Proof. Assume $B_0, B_1, \ldots$ is a recursive presentation of NP-P in terms of devices for computing characteristic functions. We will obtain a contradiction by defining a set $A \in NP-P$ such that $B_i \not\leq_{pT} A$ for any i. Let $r_1, r_2, \ldots$ be a recursive presentation of functions such that

a. each $r_i$ is time constructible and the device for computing $r_i$ operates in time $r_i$.

b. C is not $r_i$-interval easy for any C such that $B_i \leq_{pT} C$.

The recursive presentation of $\{r_i\}$ exists because: 1) $B_0, B_1, \ldots$ is a recursive presentation of NP-P by devices computing the required characteristic functions and 2) a real time algorithm for a time constructible $r_i$ can be obtained from $B_i$ by the proof of Theorem 7.

Define r' by

$$r'(n) = \max_{i \leq n} \{r_i(n)\}$$

and let $r \geq r'$ be time constructible. Then by Theorem 7, each $B_i$ satisfies, C is not r-interval easy for any C such that $B_i \leq_{pT} C$. The required set in NP-P which differs from each $B_i$ is $A = B_1 \cap G[r]$. First note that $A \notin P$ because $B_1$ is not r-interval easy and $A \in NP$ because $B_1 \in NP$, $G[r] \in P$ (because r is time constructible). Since A is r-interval easy we have

$$B_i \not\leq_{pT} A \text{ for any } i.$$

But this implies $A \in NP-P$ since $\{B_i\}$ contains all sets in NP-P, a contradiction. Hence there is no recursive presentation of NP-P in terms of devices for computing characteristic functions. ∎

It is important to notice that NP-P can not be recursively presented by characteristic function even if inefficient devices (e.g., non-primitive recursive) are permited. This result may be contrasted with our earlier result for the NP-complete sets.

In Theorem 23, we required a recursive presentation by characteristic function in order to effectively obtain algorithms for the $r_i$ functions.

Open Problem. Assuming $P \neq NP$, does there exist a recursive presentation of NP-P by domain, i.e., in terms of devices whose domains are all and only the sets in NP-P.

The next theorem shows that the recursively enumerable complete sets are not recursively presentable. While not surprising, this result does provide an indication that care should be exercised when considering definitional analogies between NP and the r.e. sets. The proof uses a recursion theoretic hierarchy argument.

Theorem 24. There is no recursive presentation of the recursively enumerable sets(as domains of partial recursive functions) which are complete with respect to Turing reducibility.

Proof. Assume that the complete r.e. sets with respect to Turing reducibility are recursively presentable. Then there is a recursive relation R such that

$$\{y \mid (\exists x) R(y,x)\}$$

is such a recursive presentation. Then the set of all indices of complete r.e. sets (i.e., all indices of partial recursive functions whose domains are complete) can be defined by

$$S = \{e \mid (\exists y)[(\exists x) R(y,x) \text{ and } W_e = W_y]\}$$

where $W_u$ is the domain of the partial recursive function whose index is u[13]. Equivalence of r.e. sets is in $\Pi_2$ in the arithmetic hierarchy. Hence, by the above, S is in $\Sigma_3$. However it can be proved by standard recursion theoretic methods that S is $\Sigma_4$-complete[13]. This is a contradiction and therefore there is no recursive presentation of the r.e. complete sets. ∎

REFERENCES

1. Berman,L. On the structure of complete sets: almost everywhere complexity and infinitely often speedup. Proc. Seventeenth IEEE Symp. on the Found. of Comp. Sci.,1975,76-80.

2. Berman,P., Relationship between density and deterministic complexity of NP-complete languages. Fifth Int. Colloq. on Automata Languages and Programming,1978,63-71.

3. Brainerd,W.S. and Landweber,L.H. Theory of Computation. Wiley, New York,1974.

4. Cook,S.A. The complexity of theorem proving procedures. Proc. Third Annual ACM Symp. on Theory of Computing,1971,151-158.

5. Fortune,S. A note on sparse complete sets.

6. Garey,M. and Johnson,D. Computers and Intractibility: A Guide to the Theory of NP-Completeness. W.H.Freeman and Co., San Francisco,1979.

7. Hartmanis,J. and Berman,L. On isomorphisms and density of NP and other complete sets. SIAM J. Comp. 6(1977),305-322.

8. Karp,R.M. Reducibility among combinatorial problems. In Complexity of Computer Computations, R.E.Miller and J.M.Thatcher, Eds.,Plenum,New York,1975,85-103.

9. Ladner,R. On the structure of polynomial time reducibility. JACM 22(1975),155-171.

10. Landweber,L.H. and Robertson,E.L. Recursive properties of abstract complexity classes. JACM 19(1972),296-308.

11. Lynch,N. On reducibility to complex or sparse sets. JACM 22(1975),341-345.

12. Machtey,M. The honest subrecursive classes are a lattice. Infor. and Contr. 24(1974)247-263.

13. Rogers Jr.,H. Theory of Recursive Functions and Effective Computability. McGraw-Hill,New York,1967.