1978

A Note on Structure and Looking Back Applied to the Relative Complexity of Computable Function

Paul Chew

Michale Machtey

Report Number:
79-312

A Note on Structure and Looking Back Applied to the Relative Complexity of Computable Functions

Paul Chew
and
Michael Machtey

Purdue University
Department of Computer Sciences
West Lafayette, Indiana 47907

July, 1979

CSD TR 312

ABSTRACT

A strong connection is established between the structural and the looking back techniques for manipulating the relative complexity of computable functions and exploring the nature of subrecursive reducibilities. Looking back serves as a basis for a simple and general structural result which can be used to derive many fundamental properties of subrecursive degrees and complexity classes. For example, as has been shown by Landweber, Lipton, and Robertson, there is a minimal pair of polynomial time degrees below any nonzero computable degree.

In addition, the structural method is used to settle a problem concerning the enumeration properties of classes of computable functions. NP-P cannot be effectively presented by domain (i.e. by r.e. indices). However, it can be effectively presented by Δ_2 indices.

This research was supported by the National Science Foundation under Grant MCS-76-09212A1.

This note is a preliminary report of continuing research. Its purpose is limited but timely dissemination to interested experts, and it should be regarded and treated accordingly.

A Note on Structure and Looking Back Applied to the Relative Complexity of Computable Functions

Paul Chew
and
Michael Machtey

Purdue University
Department of Computer Sciences
West Lafayette, Indiana 47907

July, 1979

CSD TR 312

Introduction and Preliminaries

Recent work by Landweber, Lipton, and Robertson [Univ. Wisc. CSTR#342] has shown how to take a highly structural approach to manipulating the relative complexity of computable sets. Previous work of this type has used diagonal constructions employing a looking back technique to keep complexities under control. The structural approach is an attractive alternative, and in some situations it is perhaps preferable to looking back. The looking back method (sometimes somewhat inappropriately called "delayed diagonalization") has been introduced independently in the past decade by several authors, including the second author of this note.

This note has three purposes: The first is to show an intimate connection between the structural approach and the looking back method; the structural approach can be viewed as

"precomputing" the information which looking back would find "on line". The second is to give a conceptually simpler and technically stronger proof of the central structural result in Landweber *et al*. The third is to settle an open problem concerning the recursive presentation of NP-P posed by Landweber *et al*.

The methods and results in this note are extremely general, and they will be presented in a suitably general context. In the interests of brevity and of not obscuring our main points, little or no space will be devoted to carefully explaining this general context. Instead, for those readers with doubts or who simply prefer to navigate in a more specific and concrete environment, we shall provide parenthetical pointers to such an environment [in this manner].

We consider computable functions over the natural numbers, *N*. If *f* is a function from N into N then f{n} stands for the restriction of f to the domain {0,...,n}; *warning*, this nonstandard notation will come back to haunt you! [Functions used in contexts such as reducibilities in which the reader may customarily encounter sets (i.e. characteristic functions) will be denoted in upper case. We shall use c to denote the generic integer constant].

We assume the reader is familiar with the basics of computability and complexity theory, and has at least glanced at Landweber *et al*. Our model of computation is any programming system (general or subrecursive) and accompanying computational

complexity measure which together satisfy some simple "manipulation" conditions (e.g. "succinct composition"). Only the most important of these manipulation conditions will be made explicit. [Turing machines and Turing machine time are a suitable concrete example].

Structure and Looking Back

In this section we establish the connection between the looking back method and the structural approach to diagonalization results in complexity theory. For any computable A, we use looking back to "precompute" an honest witness function which bounds how far we have to go in order to have witnesses that A is not computed by short, cheap programs. Combining this witness function with functions bounding some simple operations on programs, we get a conceptually simpler and technically tighter proof of a basic result of Landweber, Lipton, and Robertson:

> For every computable A¢P there is a total recursive function int_A such that for all B, if A is polynomial time reducible to B then no polynomial time algorithm can compute infinitely many int_A size segments of B.

We assume we are given a recursively enumerable list {i} of total programs, and we let P_i denote the total function computed by program i. P will stand for the set $\{P_i\}$, and further properties of P will be specified below. [E.g. let P_i be the set

accepted by the i-th "clocked" polynomial time Turing machine].

Let small be any unbounded, honest function; for convenience we also assume small is nondecreasing. [E.g. small(x)=log|x|]. For any computable A we define a looking back for witnesses function as follows:

$$lbw_A(x) = \text{SPEND UP TO small}(x) \text{ COST FINDING}$$
$$\max j \; \forall i \leq j \; \exists w \leq x \; (\; P_i(w) \neq A(w) \;).$$

This definition assumes that our programming system has some reasonable "conservation" facility allowing a program to limit its use of resources; also, it actually depends on some specific program for A, a fact which we have deliberately suppressed in our notation. If A is not in P then lbw_A will be unbounded, and in any case it is nondecreasing and very "cheaply" computable. Intuitively, the lbw functions capture the essence of the looking back diagonalization technique.

From lbw_A we define its inverse witness function as follows

$$wit_A(j) = \min x \; (\; lbw_A(x) \geq j \;).$$

For A not in P, wit_A is well defined and honest; thus the range of wit_A is a very easy set to recognize. Intuitively, wit_A "precomputes" easy to find bounds on the size of initial segments of A which can be computed by short, cheap programs. This connection between the looking back method and the structural approach can be summarized in the following property:

$$(1) \qquad \forall i \leq j \; \exists w \; (\; w \leq wit_A(j) \;\; \& \;\; P_i(w) \neq A(w) \;).$$

To extend the connection between the looking back method and the structural approach further we need some simple properties of the class P. We need that P is "succinctly" closed under finite variants; that is, patching in finite tables works roughly as one would expect.

Specifically, let i be any program in the list {i} and let t be any function from {0,...,x} into {0,...,b(x)}. That is, t is a "table" of outputs $\leq b(x)$ on inputs $\leq x$ and b is a function bounding the "width" of the table in terms of its "length". (In the context of arbitrary functions, it is convenient to assume that b majorizes $\{P_i\}$). We assume there is some program j in the list {i} such that j agrees with t up to x and agrees with i thereafter; that is, $P_j\{x\}=t\{x\}$ and for $y>x$, $P_j(y)=P_i(y)$. Such a j can generally be found effectively from i and t, but we require only that we be able to bound its "size" effectively. Thus, given b we let $\underline{ta}$ and $\underline{tab}$ be honest monotone functions such that for any i, j, and t as above

$$j \leq ta(i,x) \quad \text{and} \quad \left(\max_{i \leq x} ta(i,x) \right) \leq tab(x).$$

[For Turing machines and $b(x)=2^x$, tab(x) can be of the form 2^{cx^2}; for b(x)=1 (i.e. for tables corresponding to sets), tab(x) can be of the form 2^{cx}].

Using the function tab to patch in tables for A up to x, we have proved the following extension of property (1):

Theorem 1: Let P and tab be as specified above. For any computable $A \in P$, the following holds for all x:

(2) $\forall i \leq x \; \exists w \; (\; x < w \leq wit_A \bullet tab(x) \;\; \& \;\; P_i(w) \neq A(w) \;).$

Property (2) expresses the fact that $wit_A \bullet tab$ precomputes easy to find bounds on the size of segments of A beginning at x which can be computed by short, quick programs. In the terminology of Landweber, Lipton, and Robertson, A cannot be $wit_A \bullet tab$ interval easy.

Theorem 1 supplies sufficient structural information to begin proving nice results. As an example, we shall use it to reprove the following from Landweber, Lipton, and Robertson:

Theorem 2: [Landweber et al] Let A be a set decidable in exponential but not polynomial time. There is a minimal pair of polynomial time degrees below A.

Proof: (Note: the assumption that A is in EXPTIME is purely for convenience). Since this is essentially the same proof as given by Landweber et al , we shall be very sketchy. Define the function big_A as follows:

$$big_A(x) = \max \{ \; wit_A \bullet tab(x), \; 2^{2^x} \; \}.$$

Define the sets D and E as follows:

$$D = \{ \; x \mid \exists n \; (\; big_A^{(4n)}(0) \leq x < big_A^{(4n+1)}(0) \;) \; \};$$

$$E = \{ \; x \mid \exists n \; (\; big_A^{(4n+2)}(0) \leq x < big_A^{(4n+3)}(0) \;) \; \}.$$

If we let $B=D \cap A$ and $C=E \cap A$ then, as we shall see, B and C form the required minimal pair.

Since big_A is honest, D and E are certainly in P. Thus B and C are each polynomial time reducible to A. Neither B nor C is in P by property (2) above and the definition of big_A; property (2) has ensured that segments of B and C mimic A long enough to look back and see additional diagonalizations against P. Suppose that F is a set which is polynomial time reducible to both B and C. It follows that F is in P by the same argument of Ladner's [JACM, 1/75] as used by Landweber et al , which exploits the double exponential gaps between the end of one section of B and the beginning of the next section of C. [7]

We now extend the connection between looking back and structure yet further by considering reducibilities. Let $\{ \underline{P}_i[] \}$ be a recursively enumerable list of general recursive operators (i.e. "transducers"), and assume that exam is an honest, monotone function which bounds $\{P_i[]\}$'s "examination" of arguments as follows: for all B and for all $i \leq x$, $B\{exam(x)\}$ completely determines $P_i[B]\{x\}$. (This assumption of the existence of exam puts some restrictions on $\{P_i[]\}$ when applied to unbounded functions). [E.g. $P_i[]$ can be given by the i-th polynomial time oracle Turing machine, in which case $exam(x)=2^x$].

We also assume that P and $\{P_i[]\}$ are related by a succinct composition property. Let com be an honest, monotone function such that for all i and j there is a k with $P_i[P_j]=P_k$ and $k \leq com(i,j)$. [For Turing machine time, com(i,j) can be of the

form c*i*j].

The fundamental idea behind reducibilities is that they transfer (hypothetical) fast algorithms from one function to another. Using the function com we can strengthen property (2) and see that this idea also applies to short, quick algorithms for initial segments. Suppose A is computable and $A=P_i[B]$; then for all x,

(3) $\forall j \underline{\leq} x \ \exists v \ (\ x \leq v \leq exam \bullet wit_A \bullet com(i,tab(x)) \ \ \& \ \ P_j(v) \neq B(v) \ )$.

In order to summarize this structural property we define the function $\underline{int_A}$ by

$$int_A(x) = exam \bullet wit_A(\ max_{i \underline{\leq} x} \ com(i,tab(x)) \ ).$$

int_A is an honest, monotone function, and we have proved our main result of this section:

> Theorem 3: Let P, tab, $\{P_i[]\}$, exam, and com be as specified above. For any computable $A \notin P$, the following holds for all B and x:
>
> (*) $\forall i,j \underline{\leq} x \ [\ \exists v \ (\ x \leq v \leq int_A(x) \ \ \& \ \ B(v) \neq P_j(v) \ ) \ \ or$
> $\exists w \ (\ w \underline{\leq} exam^{-1} \bullet int_A(x) \ \ \& \ \ A(w) \neq P_i[B](w) \ )]$.

Note that if $A=P_i[B]$ then the v's in (*) are very easy to find as well. In the terminology of Landweber, Lipton, and Robertson, property (*) expresses the fact that if A is reducible to B then B cannot be int_A interval easy. We point out that int_A is far smaller than the bound given by Landweber et al , and is also

stated in a far broader context.

Enumeration and Looking Back

In this section we present the answer to an open problem posed by Landweber, Lipton, and Robertson. First, we use a variation on the method of the previous section to show the following:

> If P≠NP then there is no recursive presentation of NP-P by domain (i.e. by r.e. indices).

Finally, we sketch a proof that NP-P can be recursively presented by Δ_2 indices. Thus the previous result is essentially the strongest possible.

The first result is a consequence of the following:

> Theorem 4: Let $\{A_i\}$ be a recursively presented list of infinite, recursively enumerable sets, let $\{P_i\}$ be a recursively presented list of recursive sets, and let B be a set not in $\{P_i\}$. There is an easily recognized set C such that B∩C is in neither $\{P_i\}$ nor $\{A_i\}$.

Proof: The functions lbw_A and wit_A in the previous section depended on having a total program for A. If A is an infinite r.e. set, we can still define a function fin which precomputes witnesses to A's being infinite as follows:

$$fin_A(x) = \min y \; (\text{IN small}(y) \text{ COST WE CAN FIND } z \text{ in } A \text{ with } x < z \leq y).$$

If A is an infinite r.e. set then fin_A is an honest, monotone function.

Let $\{A_i\}$, $\{P_i\}$, and B be as stated in the Theorem, and let r be an honest, monotone function which majorizes each fin_{A_i} as well as $wit_B \circ tab$. Define the set C as follows:

$$C = \{ x \mid \exists n \; (r^{(2n)}(0) \leq x < r^{(2n+1)}(0)) \}.$$

Since r is honest, C is certainly easily recognized [e.g., C is in P]. Since C contains all strings in infinitely many r-segments and r majorizes $wit_B \circ tab$, $B \cap C$ is not in $\{P_i\}$. Since C has infinitely many r-gaps and r majorizes each fin_{A_i}, $B \cap C$ is not in $\{A_i\}$ (for any set B). //

The previous proof is a good example of a situation in which either the structural approach or the looking back method seem equally useful: intuitively, C is constructed by alternately looking back for witnesses to the fact that A_i is infinite or that $B \neq P_i$, for successive values of i.

The following answer to the open problem posed by Landweber et al is now immediate:

Corollary 5: If P≠NP then there is no recursive presentation of NP-P by domain (i.e. by r.e. indices).

To conclude this note, we sketch a proof that Theorem 4 is

essentially as strong as possible. Theorem 4 rules out the enumeration of certain classes by Σ_1 indices. The next theorem shows that classes such as NP-P can be enumerated by Δ_2 indices.

Recall that the Δ_2 functions are those functions which are recursive in the halting problem. Thus, a Δ_2 index has the power to determine whether two total recursive functions are equal (by asking its oracle whether the search for an argument on which they differ will ever halt).

Theorem 6: Let $\{P_i\}$ and $\{Q_i\}$ be recursively presented lists of total recursive functions such that $\{Q_i\}-\{P_i\}$ is closed under finite variants. Then $\{Q_i\}-\{P_i\}$ is recursively presentable by Δ_2 indices.

Proof: If $\{Q_i\}-\{P_i\}=\emptyset$ then the result is trivial; therefore, assume Q is in $\{Q_i\}-\{P_i\}$. Define the Δ_2 function D_i as follows:

$$D_i(x) = \text{IF } \forall j \leq x (Q_i \neq P_j)$$
$$\text{THEN } Q_i(x)$$
$$\text{ELSE } Q(x).$$

If Q_i is not in $\{P_i\}$ then $D_i=Q_i$; otherwise, D_i is a finite variant of Q. //

As an immediate corollary, we get the following:

Corollary 7: NP-P is recursively presentable by Δ_2 indices.