# SPEED-UPS BY CHANGING THE ORDER IN WHICH SETS ARE ENUMERATED (PRELIMINARY VERSION)

Paul R. Young
Purdue University
Lafayette, Indiana

## SUMMARY

In a suitably general context, the following analogue of the Blum Speed-up Theorem is proven: There are some infinite sets which are so difficult to enumerate that, given any order for enumerating the set, there is some other order, and some <u>one</u> method of enumerating the set in this second order which is much faster than <u>any</u> method of enumerating the set in the first ordering. It may be possible to interpret this result as a statement about the relative merits of "hardware" <u>vs</u>. "programming" speed-ups. The proof itself is one of the first nontrivial applications of priority methods to questions of computational complexity. As such, it perhaps represents an advance in bringing the results and techniques of contemporary "pure" recursion theory to bear on questions of computational complexity.

In this paper we shall prove, in a suitably general context, the following analogue of the Blum Speed-up Theorem, [Bl]: There are some infinite sets which are so difficult to enumerate that, given any order for enumerating the set, there is some other order, and some <u>one</u> method of enumerating the set in this second order which is much faster than <u>any</u> method of enumerating the set in the first ordering.

Before proceeding with the details, let us consider a possible interpretation. While we will not vouch for the strict validity of the interpretation, it will perhaps suggest why the result is interesting. In any real computer installation there are two components to the computational procedures. First, there is a physical machine, and second, there are programs for the machine. If, for example the programs are written in a higher level language, there may also be a compiler for translating the higher level language into machine language. As always when doing recursion theory, we assume a potentially unlimited memory, so that there is, e.g., no limit to the number of tapes which a machine may use in executing a program.

We now imagine a program written in a higher level language which is designed to enumerate or generate an infinite list of integers. We ask how we can speed-up the task. An obvious answer is to speed up the basic cycle time of the machine. Indeed, such hardware speed-ups have been an important reason for the increased efficiency of digital computers. Clearly, an increase in the basic cycle time of the machine will not change the order in which a given program will enumerate a fixed set of integers. Although it is possible to imagine hardware improvements utilizing parallel processing which will change the order in which a given higher level program enumerates a given set, there is a good reason for avoiding such changes in the order in which the members of the set are enumerated as output (even though the internal sequence of machine operations may change): A programmer working in a higher level language has a good notion of how he expects the program to operate. If a machine fails to enumerate the answers in the order implicitly assumed by the programmer, the programs are likely to be difficult to "debug". Indeed, to the extent that higher level languages are "machine independent", we may expect that hardware improvements will not change the order in which programs enumerate sets.

Our result now strongly suggests that there are some infinite sets which have the property that, no matter what program is used to enumerate the set, a single reprogramming will result in much greater gains in operational efficiency than can be achieved by any improvement that can be made by speeding up the hardware (without changing the order of enumeration).

We briefly review our notion of an enumeration technique, explained more fully in [Y]. We let $D_o, D_1, D_2, \ldots$ be any canonical enumeration of all finite subsets of N, the set of nonnegative integers: from i, one can effectively list all members of $D_i$ and know when the listing is complete. A total recursive function E is an <u>enumeration technique</u> if for every recursively enumerable (r.e.) set W there is an integer e such that $W = \bigcup_{n \in N} D_{E(e,n)}$. We call e an index of W and write $W_e$ for $\bigcup_n D_{E(e,n)}$. We always assume that the resulting indexings satisfy the Universal Turing Machine Theorem and the $S^n_m$ - Theorem (for sets; for details see [Y]). For convenience, we always assume $D_{E(e,o)} = \emptyset$, and we define $E'(e,n)$ by $D_{E'(e,n)} = \bigcup_{m \leq n} D_{E(e,m)}$. We use the $\lambda$ - notation for functions, e.g., if E is a function of two variables, $\lambda y E(x,y)$ is, for each

Supported by N.S.F. Research Grant No. GP 6120

fixed x, the resulting function of the second variable alone. $\lambda i \emptyset_i$ is a fixed standard enumeration of all partial recursive functions. We assume no special connections between the indexings $\lambda i \emptyset_i$ and $\lambda i W_i$; in particular $W_i$ need not be the range or domain of $\emptyset_i$. $\tau$ is any 1-1 effective map from NXN to N, and we always denote $\tau(x,y)$ by $<x,y>$. $\pi_1 <x,y> = x$ and $\pi_2 <x,y> = y$.

Definition. With every enumeration technique E we associate the partial recursive function $\lambda i n A_i(n)$ defined by

$$A_i(n) = (\mu y)\ [|D_{E'(i,y)}| \geq n],$$

where "$(\mu y)[\ldots]$" means "the least y such that $[\ldots]$" and $|D_{E'(i,n)}|$ is the number of elements in $D_{E'(i,n)}$.

$A_i(n)$ may be thought of, e.g., as the number of steps Turing machine i takes to enumerate at least n elements, or as the number of tape squares machine i takes to enumerate at least n elements, or as the length of the longest derivation which Post-system i uses to enumerate at least n elements, or as the number of instructions the i'th Fortran program takes in order to write at least n elements. Etc. The predicate $A_i(x) \leq y$ is a recursive predicate of three variables.

We now wish to talk about programs i and j which enumerate the same set in the same order. Since this notion cannot be recursively defined, we must content ourselves with a series of recursive approximations which in the limit give us almost, but not quite, the notion we are after.

Definition. We say that i appears to have the same order as j through n steps and write i $\text{app}^n$ j if for all $x \leq n$ and for all y, $z < n$,

$$[D_{E'(i,x)} \subseteq \bar{D}_{E'(j,x)} \text{ or } D_{E'(j,x)} \subseteq D_{E'(i,x)}] \ \& \ [m \in (D_{E'(i,y+1)} - D_{E'(i,y)}) \cap (D_{E'(j,z+1)} - D_{E'(j,z)}) \Rightarrow D_{E'(i,y)} \subseteq D_{E'(j,z+1)} \ \& \ D_{E'(j,z)} \subseteq D_{E'(i,y+1)}].$$

Definition. We say that i appears to have the same order as j and write i app j if for all sufficiently large n, i $\text{app}^n$ j is true.

We leave the reader to verify that this notion is intuitively correct (bearing in mind that $D_{E'(i,y+1)} - D_{E'(i,y)}$ may have more than one element (so we have given the broadest possible interpretation to "have the same order")), that not i $\text{app}^n$ j implies not i $\text{app}^{n+1}$ j, that $\{<i,j> \mid \text{not } i \text{ app } j\}$ is r.e., and that $\{<i,j> \mid i \text{ app } j\}$ is not obviously enumerable.

(1) We also ask the reader to note that i app j iff both $W_i$ and $W_j$ are infinite and i and j enumerate the same set in the same order, or else one of $W_i$ and $W_j$ is a finite subset of the other but the orders cannot be distinguished from the enumeration. (Thus the relation i app j when restricted to those i and j for which $W_i$ and $W_j$ are infinite is an equivalence relation. But if $i_0$ is an index of the empty set, then $i_0$ app j for any j.)

We now define a recursive function $s(j,n)$ with the hope that $\lim_n s(j,n)$ will be the smallest index i for which $W_j = W_i$ and i app j. We do not fully succeed.

Definition. $s(0,n) = 0$ for all n and

$$s(j+1,n) = (\mu i)[i \leq j+1 \ \& \ i \text{ app}^n\ j+1 \ \& \ (\forall z \leq j)\ [\text{not } z \text{ app}^n\ j+1 \Rightarrow i \neq s(z,n)]].$$

Again, we leave the reader to verify simple facts about s:

$s(j,n) \leq j$ for all j and n.

(2) $\lim_n s(j,n)$ always exists (by induction on j).

(3) $\lim_n s(j,n) = \lim_n s(i,n) \Rightarrow$ j app i.

(4) If i is the smallest index for which $W_i = W_j$ and i app j, then $\lim_n s(j,n) \leq i$.

Finally, in order to state our theorem, we recall one final

Definition. An operator $\underset{\sim}{F}$ carrying partial recursive functions to partial recursive functions is effective if there is a total recursive function g such that

$$\underset{\sim}{F}(\emptyset_e) = \emptyset_{g(e)}$$

for all e such that $\emptyset_{\ell} \in$ domain $\underset{\sim}{F}$.

We shall be concerned only with effective operators which carry all total recursive functions to total recursive functions. Examples of such operators include: (i) If $r(x,y)$ is any total recursive function, the operator $\underset{\sim}{F}_1$ defined by $\underset{\sim}{F}_1(\emptyset_i)(x) = r(x, \emptyset_i(x))$, and (ii), the operator $\underset{\sim}{F}_2$ defined by $\underset{\sim}{F}_2(\emptyset_i)(x) = \emptyset_i(\emptyset_i(x))$.

Theorem 1. Let E be any enumeration technique and let $\underset{\sim}{F}$ be any effective operator carrying all total recursive functions to total recursive functions. Then there exists an infinite r.e. set W such that if $W_i = W$, there exists i' such that $W_{i'} = W$ and for any j such that i and j enumerate W in the same order

$$A_j(n) > \underset{\sim}{F}(A_{i'})(n)$$

for all but finitely many n.

The details of our proof are lengthy and tedious, and we will reserve them for later publication. We will however give a brief outline of the proof, explaining how it builds on and differs from the proofs of speed-up theorems in [B1] and [M-F]. We assume the reader is familiar with one or both of these earlier proofs.

As is usual, we first define a total recursive function $t(u,v,\ell)$ satisfying three conditions, which we number to correspond to the numbering of our proof:

(12) Suppose $\emptyset_\ell$ is a total recursive function. Then there is a (possibly noneffective) sequence $0 = v_0, v_1, v_2, v_3, \ldots$ such that

$$W_{t(o,o,\ell)} = W_{t(1,v_1,\ell)} = W_{t(2,v_2,\ell)} = \ldots .$$

(13) There exists a total recursive function $\emptyset_\ell$ for which, for all i and v, and for all t such that $(n-1)^3 < t \le n^3$,

$$\emptyset_\ell(<i,n>) \ge \max\{\underset{\sim}{F}(A_{t(i+1,v,\ell)})(t),\ \emptyset_\ell(<i+1,n>)\}$$

for all but finitely many n.

(14) For $\emptyset_\ell$ given by (13), if $W_i = W_{t(o,o,\ell)}$ and if i is the smallest index for enumerating $W_{t(o,o,\ell)}$ in the order given by i, then for any j for which $W_j = W_{t(o,o,\ell)}$ and i app j, we must have, for t such that $(n-1)^3 < t \le n$,

$A_j(t) > \emptyset_\ell(<i,n>)$ for all but finitely many n.

The proof then follows from (12), (13), and (14) in virtually the same way similar results are used to obtain the speed-up theorems of [B1] and [M-F], so we omit details.

To prove (13), we first define an effective operator, $\underset{\sim}{F}^*$, carrying all total recursive functions to total recursive functions by

$$\underset{\sim}{F}^*(\emptyset_j)(n) = \max\{F(\emptyset_j)(t) \mid (n-1)^3 < t \le n^3\}.$$

(13) is now proven by using the method introduced by Meyer and Fischer for proving their operator speed-up theorem. (Actually, we get by with a slightly simpler version of their proof.) We use the recursion theorem to obtain a function $\emptyset_\ell$ for which

$$\emptyset_\ell(<i,x>) = 0 \text{ if } x \le i \text{ or } (\exists n \le i)[\emptyset_\ell(<0,n>) \text{ is not defined in } \le x \text{ steps}], \text{ otherwise}$$

$$\emptyset_\ell(<i,x>) = \sum_{v \le x}[\underset{\sim}{F}^*(A_{t(i+1,v,\ell)})(x) + \emptyset_\ell(<i+1,v>)].$$

For this $\ell$, it is easily seen that (13) holds if $\emptyset_\ell$ is total. But it is easily seen that $\lambda x \emptyset_\ell(<i,x>)$ total implies $\lambda x \emptyset_\ell(<i+1,x>)$ total, and that if $\lambda x \emptyset_\ell(<0,x>)$ were not total then $\lambda x \emptyset_\ell(<i,x>)$ would be total for all sufficiently large i, say for all $i > M$. This in turn is adequate for showing that $W_{t(M+1,v,\ell)}$ always has at least n elements for $n \ge M+1, v$. This in turn implies the totality of $A_{t(M+1,v,\ell)}$ for all v, and hence the totality of $\lambda x \emptyset_\ell(<M,x>)$. Iterating, we see that $\lambda x \emptyset_\ell(<0,x>)$ is total.

The trick in our construction is to define the recursive function $t(u,v,\ell)$ in such a way that (12) and (14) hold. (14) is not too difficult, for if i and j are as described in (14), we may observe by (4) that $\lim_n s(j,n) \le i$. Let $i' = \lim_n s(j,n)$. The construction is so arranged that once $s(j,n)$ achieves its limit, $i'$, if

$$A_j(t) \le \emptyset_\ell(\langle i',n\rangle) \quad (\ge \emptyset_\ell(\langle i,n\rangle) \text{ a.e.})$$

then we must <u>attack</u> $i'$ by j by withholding a member of $W_j$ from $W_{t(o,o,\ell)}$ unless $i'$ is under attack by some smaller $j'$ with $\lim_n s(j',n) = i'$. Since $\lim_n s(j',n) = \lim_n s(j,n)$, by (3) j app $j'$. Since $W_j$ is infinite this implies by (1) that $W_{j'} \subseteq W_j$. Since $i'$ is under attack by $j'$, some member of $W_{j'}$, (and hence of $W_j (= W_i)$) is withheld from $W_{t(o,o,\ell)}$.

Our proof of (12) is more difficult than the proof of the corresponding results in [B1] and [M-F]. There are several problems to be overcome here, but the most notable is the following: In the corresponding proofs in [B1] and [M-F], each i gets attacked at most finitely often. The variable v in $W_{t(u,v,\ell)}$ is thus used to handle the finitely many attacks on the finitely many i's $\le u$. However, we have been unable to arrange that each i be attacked at most finitely often. Consequently, we have had to arrange that if i gets attacked infinitely often, almost all of the attacks are irrelevant, and v must somehow accomodate only the relevant attacks. This seems to make this portion of our proof considerably more delicate than the corresponding proofs in [B1] and [M-F]. We shall give the details in the published version of this paper.

In conclusion, we might remark that the sort of speed-up described by Theorem 1 of this paper is not the only sort of speed-up for enumerating sets: Suppose E is an enumeration technique for which it is possible, given an intuitive description for enumerating a set, to obtain an index e for which $\lambda n D_{E(e,n)}$ enumerates the set in the same order as given by the intuitive description. Then a straightforward adaptation of the proof of the Blum Speed-up Theorem in [B1] or its generalization in [M-F] yields:

<u>Theorem 2</u>. Let $\underset{\sim}{F}$ be any effective operator carrying all total recursive functions to total recursive functions. Then there exists an infinite r.e. set W and an index i of W for which if $W_j = W$ there exists an i' such that $W_{i'} = W$, i app i', and

$A_j(n) \geq F(A_{i'})(n)$ for all but finitely many n.

Finally, we remark that anytime one has a set W, like those constructed in Theorems 1 and 2, which is difficult to enumerate, W cannot have an infinite subset S which is "much easier" to enumerate than is W itself. The reason is that no set W is "much more difficult" to enumerate than any of its subsets, S. This is because a "simultaneous" enumeration of S and W produces elements "almost" as rapidly as does an enumeration of S alone.

## References

[B1] Blum, Manuel, A machine independent theory of computational complexity, J. Assoc. Comp. Mach., 14 (1967), 322-336.

[B2] Blum, Manuel, On the size of machines, Inf. and Control, 11 (1967), 257-265.

[M-F] Meyer, A. R., and Fischer, P.C., On computational speed-up, to appear.

[R] Rogers, H., Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, (1967).

[Y] Young, P.R., Toward a theory of enumerations, J. Assoc. Comp. Mach. 16, (1969), to appear. (Preliminary version in Proc. 9th annual I.E.E.E. Symposium on Switching and Automata Theory, (1968), 334-350.)