Richard Edwin Stearns

Turing Award Lecture

It's Time to Reconsider Time

When I was a student, there was no such thing as computer science and my interests were strictly mathematics. I graduated from Carlton College in 1958 with a BA in mathematics and went on to Princeton to do graduate work in mathematics. There, under the supervision of Harold W. Kuhn and the mentoring of Robert J. Aumann, I wrote a Ph.D. thesis on game theory entitled "Three-Person Cooperative Games without Side Payments" [8] and graduated in the fall of 1961.

In 1960, I was hired by Richard L. Shuey to work for the summer at General Electric's Research Laboratory in Schenectady, New York. Shuey was manager of a group called the "Information Studies Section" which included Juris Hartmanis and Philip M. Lewis II. This group was part of a larger group called "Electron Physics." During that summer, I worked with Hartmanis continuing some work he had started on the decomposition of sequential machines. As a result, the first Hartmanis-Stearns article [9] was produced. I was very impressed with the quality of the people at GE and the freedom they had to define their own research goals. Thus, when Shuey invited me to return in 1961 as a permanent employee, I jumped at the chance. There were a number of jokes made about my Ph.D. thesis since GE had just settled an antitrust suit for cooperating with two other companies to fix prices (no side payments involved!).

When I started at GE, it was strictly as a mathematician. The research laboratory did not have a computer and I had never used a computer. A few years later, the laboratory obtained a GE300 and installed a copy of the timesharing system developed at Dartmouth. My first programming experiences were using Basic on this machine through a teletype interface. These experiences came after Hartmanis and I had worked out our theory of computational complexity. My transformation into a computer scientist was a matter of evolution rather than a conscious decision. I was pursuing the mathematical problems I found most interesting and found myself in the middle of computer science.

I continue to hold a simultaneous interest in both game theory and computer science. In fact, my first article with Hartmanis preceded my thesis work. You may wonder if these fields have anything in common. I think they do.

One commonality is that they were both started by John von Neumann. The beginning of game theory is clearly marked by the famous book by von Neumann and Morgenstern [13] and von Neumann with his "von Neumann stored program computer model" is also considered a founding father of computer science.

A second thing they have in common is the need to understand something very intangible yet very real. In the case of game theory, it is the nature of competition. In the case of computer science, it is the nature of computation. Both areas have required some completely new mathematics requiring the development of new mathematical models. My attraction to game theory began as an undergraduate one summer when I read [13] for recreation and saw how much attention was given to discussion about the choices of model.

To me, the most interesting mathematics is that which tells us something about the real world. Thus, the first question we as mathematicians and computer scientists should ask is "how well do our models reflect the salient features of the objects or situations we wish to describe?" The significance of a result depends more on the information it conveys rather than on the complexity of its proof. Garbage-in/garbage-out applies to theory as well as software.

Complexity

The complexity work with Hartmanis appeared in both a conference version [5] and a journal version [6].[1] The conference version was given at the Fifth Annual Symposium on Switching Circuit Theory and Logical Design which has undergone a couple of name changes and is now known as FOCS (Foundations of Computer Science). Both versions of the article had the phrase "computational complexity" in their title, the first time this phrase was used. Thus, we were the first to call what we were doing "computational complexity."

Although the conference version appeared before the journal version, the text of the conference version was written later and referred to itself as an "update" on the journal version. One update was inclusion of a reference to Blum's Ph.D. thesis from MIT which was to appear as [1]. This work, developed independently of ours, provided a more abstract view of complexity classes. This was a very significant contribution which should also be credited with inspiring the early interest and rapid growth of the complexity field.

In [6], we developed our theory on input-less multitape Turing machines which produced an infinite sequence of zeros and ones. This was essentially the same model used by Yamada in his work about "real-time countable functions" [14]. However, the now familiar language recognition model was rapidly ascending to the prominent position in automata theory it holds today. Recognizers are a valuable model because, despite their simplistic yes/no outputs, they are already sufficient to discuss most computational issues. Furthermore, they are ideally suited for the study of nondeterminism. In the update, we discussed the application of our methods to this model.

A third update was to discuss the implication of using the Hennie-Stearns two-tape $n \log n$ simulation [7] instead of the Hartmanis-Stearns one-tape n^2 simulation. (Fred Hennie worked at GE from time to time as a visitor.) The implication was a sharper, sufficient condition for distinguishing complexity classes.

The primary contribution of our work was the use of deterministic time to define complexity classes. Using modern notation and the language recognition model, our definition was this:

DEFINITION 1. *DTIME(T(n)) is the set of all languages L for which there is a multitape Turing machine such that the machine*

1. answers the question "does input w belong to language L?," and
2. answers the question in at most T(|w|) moves where |w| is the length of input w.

In other words, a problem can be placed in the class *DTIME(T(n))* by providing a program which answers the corresponding membership question in *T(n)* steps where *n* is the size of the input. The most common objective in the analysis of algorithms is to place an upper bound on the runing time of an algorithm. In terms of the time com-

[1]There was also Research Laboratory report dated April, 1963.

plexity model, this corresponds to placing the problem solved by the algorithm into a complexity class.

In a sense, these complexity classes should be called "easiness classes" since expressing the idea that a problem is inherently hard (ie., establishing a lower bound) is achieved by showing that a problem does *not* belong to a certain time class.

One of the first things we proved was a "speed-up theorem":

THEOREM 1.

$$DTIME(T(n)) = DTIME(c \cdot T(n))$$

for all $c > 0$.[2]

In other words, it is $O(T(n))$ and not $T(n)$ which is important. This is a property we should expect from a proper model since there is no meaningful way to relate the number of transitions of an automaton to actual time measured in seconds and since the time in seconds to perform an algorithm will vary with computer technology. It was nice to see this property as a consequence of our definition. Of course constant factors are important in practice and not all $O(T(n))$ programs are equally good, but we do not expect to discriminate between such programs at an automata theoretic level.

The central result of our work was:

THEOREM 2. *If*

$$\lim_{n\to\infty} \frac{T(n) \cdot \log(T(n))}{U(n)} = 0$$

then $DTIME(U(n))$ *contains a language which is not in* $DTIME(T(n))$.[3]

This result implies that there really is a time hierarchy and that small changes in the time function give different classes. For example, there are problems that can be solved in $O(n^2)$ time that cannot be solved in $O(n^{2-\epsilon})$ time for any $\epsilon > 0$. Therefore, certain problems have an inherent complexity that cannot be circumvented by clever programming.

One property our model did not have was "machine independence." That is, if the complexity classes were defined with respect to some enhanced Turing machine, say with two-dimensional tapes, the classes would be somewhat different. Our article investigated several such enhancements and found that the time differences were related by low-degree polynomials. A complexity concept is now considered "machine independent" if it does not change with models polynomially related in time to Turing machines.

A short time later, Hartmanis and I looked at "tape" complexity (now known as space complexity) with Phil Lewis [10]. This included an innovation at the model level. We defined our space complexity with respect to the amount of scratch tape used (ie., read-only input tapes would not be counted in the measurement). This enabled sublinear complexity classes all the way down to $\log n$ space and in some cases even to $\log \log n$ space.

[2]For purposes of this talk, I am somewhat simplifying theorem statements.

[3]Originally without [7], we had $[T(n)]^2$ instead of $T(n) \log T(n)$. Also $U(n)$ must be "time constructible."

Hardness Concepts

Although the time and space complexity classes form hierarchies and provided concepts for discussing hardness, there were no obvious techniques for taking a particular problem and showing it was not easy. In other words, it was (and still is) hard to show that there is no good method at all for solving a particular problem. Just because I have a clever method of solving a problem in say $O(T(n))$ time, how do I know that there does not exist an $O(T'(n))$ algorithm where $T'(n)$ is much smaller than $T(n)$?

The situation improved considerably when Cook introduced the concepts we now call NP-hardness and NP-completeness [2]. The overall idea was to relate the hardness of a particular problem to the hardness of some set of seemingly difficult problems. This would be done by showing that, if a good algorithm existed for the given problem, a good algorithm would exist for each problem in the set. In Cook's case, the set of seemingly difficult problems was the set of problems now called NP-complete and an algorithm would be considered "good" if it took only polynomial time. The hardness concept itself is now called NP-completeness.

The NP-complete problems seem to be very difficult because, by definition, if any of them can be solved in polynomial time, so can any problem which has a nondeterministic polynomial algorithm. In fact, these problems seem to require exponential time. The standard way of proving a problem X is NP-complete involves taking a problem Y already known to NP-complete and demonstrating that some polynomial time reduction can be used to convert any instance of Y into an instance of X having the same answer. In effect, the reduction implies that any method of solving X can be used to solve Y equally efficiently and so X cannot be easier than Y. Since we believe Y to be hard, we also believe X is hard. Thus, NP-hardness is accepted as good evidence that a problem requires exponential time.

To make this plan work, it is necessary to have some problems known to be NP-complete. Cook's article started things off by showing that the problem known as SAT, namely the satisfiability problem for CNF Boolean formulas, is NP-complete. This was quickly followed by the work by Karp [4] showing that many combinatorial problems of practical interest are also NP-complete. Soon hundreds of practical problems were shown to be NP-complete, many of which are found in [3].

Another hardness concept soon appeared, namely PSPACE-hardness, based on the concept of PSPACE-completeness. PSPACE-hard problems seem very difficult since if any of them can be solved in polynomial time, all problems in polynomial space can be solved in polynomial time. PSPACE-complete problems also seem to require exponential time. As with NP-completeness, the standard method of proving PSPACE-completeness involves polynomial time reductions, this time from PSPACE-complete problems. PSPACE-hardness is stronger evidence of hardness than NP-hardness since PSPACE-hard problems might still require exponential time even if it unexpectedly turns out that the NP-complete problems can be solved in polynomial time. One of the first problems shown to be PSPACE-complete is QSAT, the problem of deciding if a quantified CNF formula is true [12].

Hardness concepts have proven to be very useful in classifying a variety of problems of practical interest and have contributed greatly to our understanding about the real world of computing. The application of these ideas has been so successful that we sometimes overlook their limitations or forget what they really mean. Some of these limitations will be pointed out here and then we will see that, from the viewpoint of deterministic time, there is a lot more to be learned about the time complexity of such problems.

Although PSPACE-completeness is stronger evidence of hardness than NP-completeness, there is no reason to believe that PSPACE-complete problems are harder in the sense that they require more time. Consider for example SAT which is NP-complete and QSAT which is PSPACE-complete. The best algorithms known for these problems are practically identical. Both involve considering all $\Theta(2^n)$ assignments to the variables and both use $\Theta(n)$ space. We are inclined to believe that SAT cannot be solved more quickly and therefore are inclined to believe that both SAT and QSAT have the same complexity.

When we say that NP-complete problems seem to take "exponential time," we do not mean 2^n. "Exponential" in this case means 2^{n^a} for some $a > 0$. Thus 2^{n^2}, $2^{\sqrt{n}}$, $2^{\sqrt[3]{n}}$, and even $2^{\sqrt[100]{n}}$ are exponential. In fact, the classes $DTIME(2^{n^\epsilon})$ contain PSPACE-complete and NP-complete problems for all $\epsilon > 0$. Time $\Theta(2^{n^\epsilon})$ may not be impossibly large when ϵ is small. For example when $n = 31{,}991$, $2^{\sqrt[3]{n}}$ operations can be performed in an hour on a 1 mips computer and $2^{\sqrt[4]{n}}$ is only about 10615.

Thus, it is conceivable that some of these "hard" problems can be easily solved for all inputs of significant size.

A Time-Based Perspective

In order to facilitate further discussion, consider the following time-based concept of *power index* that Harry B. Hunt III and I have developed [11]:

Definition 2. *The power index of a problem L is the greatest lower bound on the set*

$$\{r | L \in DTIME(2^{n^r})\}$$

if the set is nonempty and ∞ if the set is empty.

This concept has several attractive properties:

1. Every problem has a power index because every nonempty set of nonnegative reals has a greatest lower bound.
2. For every rational r, there is a problem with power index r. (This follows from Theorem 2.)
3. The power index concept is "machine independent."

Problems with polynomial algorithms (ie., problems in P) all have power index zero. Problems with an exponential time bound (ie., problems in EXPTIME) all have finite power indices.

If any NP-complete problem has a power index greater than zero, then all do and $P \neq NP$. This is the same for PSPACE-complete problems. Therefore, proving that an NP-complete problem has nonzero power index would prove $P \neq NP$ and proving that a PSPACE-complete problem has nonzero power index would prove $P \neq PSPACE$. Thus we expect difficulties in establishing the power indices of such problems. However, as with "evidence of hardness," we can study "evidence of power index" by using reductions. To do this, we must pay attention to reduction *size* defined as follows:

Definition 3. *A reduction R is of size $s(n)$ if and only if the length of output $R(w)$ is $\Theta(s(n))$.*

Relationships between power indices can be established by the following:

Theorem 3. *If the power index of L_1 is r and L_1 is reducible to L_2 by a polynomial time reduction of size n^s, then the power index of L_2 is at least r/s.*

Notice that the larger the size of the reduction (ie., the larger the degree s) the weaker the lower bound r/s. Just knowing that a reduction is polynomial conveys no information at all about power index.

To put Theorem 3 in perspective, consider SAT and CLIQUE. The standard reductions from SAT to CLIQUE [3, 4] have n^2 size and no better reduction is known. Thus, the strongest conclusion made from the theorem is that the power index of CLIQUE is at least one-half that of SAT. This actually fits the known facts quite well since the best-known algorithm for SAT takes $2^{O(n)}$ time and the best algorithm for CLIQUE uses only $2^{O(\sqrt{n})}$ time.[4] If we could find a linear-sized reduction from SAT to CLIQUE, we would then have a $2^{O(\sqrt{n})}$ time algorithm for SAT!

From the viewpoint of deterministic time, all NP-complete problems are not equally hard. By playing close attention to the size of reductions, sharper comparisons can be made between the complexities of various NP-complete problems. One way to assess our understanding is to ask the following question: Assuming that SAT does require $2^{\Theta(n)}$ time (ie., assuming the power index of SAT is one) what can be inferred about the power indices of other NP-complete problems?

In many cases, we have linear reductions from SAT to other problems solvable in $2^{O(n)}$ time, and the answer to this question is that these also must have power index one. However, there are also many problems where the best known reductions are not linear and these problems are potentially much easier. In most cases, it is an open question if this potential is somehow achievable or if smaller reductions exist.

On a more theoretical level, power indices remind us

[4]The easier algorithm for CLIQUE is due to the facts that a clique of size k has $O(k^2)$ edges and a problem instance of length n has no more than n edges. The number of nodes in the largest clique is therefore at most $O(\sqrt{n})$ and an exhaustive search can be restricted accordingly.

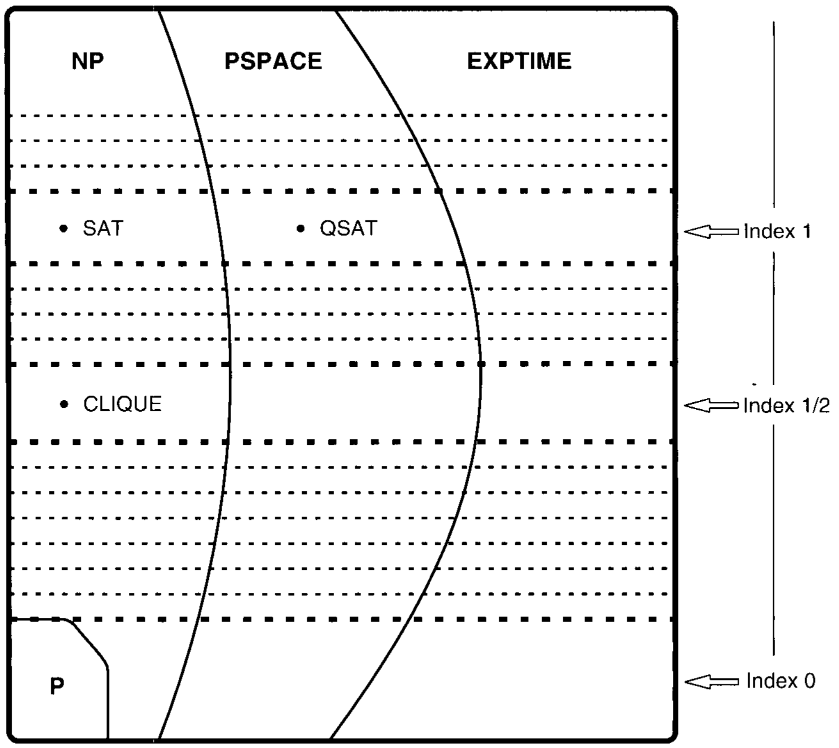

Figure 1. EXPSPACE subdivided by power index

that, to the best of our knowledge, the time hierarchy is largely orthogonal to the classes associated with evidence of hardness. Our best guess is that the world looks similar to the one shown in Figure 1. This figure shows the set EXPTIME carved up into classes of equal power index. These classes also carve up the sets PSPACE and NP. Both SAT and QSAT are shown, as conjectured, having power index one. Below that, CLIQUE is shown as conjectured with power index one-half and P is shown contained in the power index zero problems.

Our observations relating deterministic time to evidence of hardness can be summarized as follows:

- All NP-complete problems are not equally hard.
- All PSPACE-complete problems are not equally hard.
- A PSPACE-complete problem can be easier than an NP-complete problem.
- Even if SAT does require $2^{\Theta(n)}$ time, the possibility remains that many NP-complete problems of practical interest may require a lot less time. ◘

References

1. Blum, M. A machine-independent theory of the complexity of recursive functions. *J. ACM* 14, 4 (Apr. 1967), 322-336.
2. Cook, S.A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing,* Shaker Heights, Ohio, 1971, pp. 151–158.
3. Garey, M.R. and Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.
4. Karp, R.M. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations,* Plenum, N.Y. 1972, pp. 85–103.
5. Hartmanis, J. and Stearns, R.E. Computational complexity of recursive sequences. In *Proceedings of the Fifth Annual IEEE Symposium on Switching Circuit Theory and Logical Design,* Princeton, N.J., 1964, pp. 82–90.
6. Hartmanis, J. and Stearns, R.E. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. 117,* (May 1965), 285–305.
7. Hennie, F.C. and Stearns, R.E. Two-tape simulation of multitape turing machines. *J. of ACM,* 13, 10 (Oct. 1966), pp. 533–546.
8. Stearns, R.E. Three-person cooperative games without side payments. In M. Dresher, L.S. Shapley, and A.W. Tucker, eds., *Advances in Game Theory,* Annals of Mathematics Studies #52, Princeton University Press 1964, pp. 307–406.
9. Stearns, R.E. and Hartmanis, J. On the state assignment problem for sequential machines II.. *IRE Trans. Electr. Comput., EC-10,* (Dec. 1961), 593–603.
10. Stearns, R.E., Hartmanis, J., and Lewis II, P.M. Hierarchies of memory limited computations. In *Proceedings of the Sixth Annual IEEE Symposium on Switching Circuit Theory and Logical Design,* Ann Arbor, Mich. 1965, pp. 179–190.
11. Stearns, R.E. and Hunt III, H.B. Power indices and easier hard problems. *Math. Syst. Theory* 23 (1990), 209–225.
12. Stockmeyer, L.J. and Meyer, A.R. Word problems requiring exponential time. In *Proceedings of Fifth Annual ACM Symposium on the Theory of Computing,* Austin, Tex., 1973, pp. 1–9.
13. von Neumann, J. and Morgenstern, O. *Theory of Games and Economic Behavior.* Princeton, N.J., 1944.
14. Yamada, H. Real-time computation and recursive functions not real-time computable. *IRE Trans. Electr. Comput., EC-11,* (1962), 753–760.

About the Author:
RICHARD EDWIN STEARNS is a professor of computer science at the University at Albany, State University of New York. His current research interests include the structure of problem instances and the implications of such structure for complexity.
Author's Present Address: University at Albany—SUNY, Computer Science Dept., Albany, NY 12222; email: res@cs.albany.edu