

Deterministic versus Nondeterministic Time and Lower Bound Problems

RICHARD E. STEARNS

University at Albany

Because many problems of general interest have natural nondeterministic algorithms and because computers act deterministically, it is important to understand the relationship between deterministic and nondeterministic time. Specifically, it is important to understand how quickly a deterministic computing device can determine the outcome of a nondeterministic calculation. So far, we have no general techniques which work any better than trying all step-by-step simulations, an exponential method.

The most famous question concerning determinism versus nondeterminism is the $P = NP$ question. However, this is a different question than “what is the relationship?” and it is possible that significant progress about the relationship can be achieved without answering the $P = NP$ question. Thanks to efficient reductions from Turing machine simulation to SAT, the relationship question can be posed as a question about SAT. The problem of proving a nontrivial lower bound on the time required to solve SAT is just an instance of the larger problem of proving lower bounds for any natural problem. It is suggested that a study of generic problems might be a fruitful approach toward insights on such problems.

Categories and Subject Descriptors: F.1.1 [**Theory of Computation**]: Computation by Abstract Devices—*Models of Computation*

General Terms: Theory, Algorithms

Additional Key Words and Phrases: computational complexity, generic algorithms, generic problems, nondeterminism, NP-completeness, power index, SAT, time complexity

A central unanswered question about the relationship between determinism and nondeterminism is the complexity of the following computational problem:

Given a description of a nondeterministic Turing recognizer M , an input sequence w for M , and an integer n where $n \geq |w|$, determine if M accepts w in n or fewer steps.

By a “nondeterministic Turing recognizer”, we mean a multi-tape nondeterministic Turing machine where certain states have been designated as “accepting states”. The machine “accepts” if and only if some computation sequence leads to an accepting state. By $n \geq |w|$, we mean that the length of input sequence w is no more than n . We shall refer to this problem as the “**Simulation Problem**”.

The complexity theoretic question associated with the Simulation Problem is the following:

Department of Computer Science
University at Albany
Albany, NY 12222
res@cs.albany.edu

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0004-5411/20YY/0100-0001 \$5.00

For a fixed description M , how much time (as a function of n) is required to solve the problem on a deterministic multi-tape Turing machine?

By the “time required”, we mean the maximum number of Turing machine steps taken for any input w , $|w| \leq n$. Although we have used multi-tape Turing machines to pose this question, the answer for other bit-by-bit computation models is, for the reason given below, essentially the same.

The obvious solution method is to systematically explore the possible nondeterministic computation sequences of length n until one is found that accepts or until all sequences have been tested and found not to accept. In the worst case, all computation sequences must be tried so the computation time could be $\Theta(c^n)$ where constant $c > 1$ reflects the number of choices available to the Turing machine at each move. By $\Theta(c^n)$, we mean the number is between k_1c^n and k_2c^n for all large n for some constants k_1 and k_2 .

In the context of the above discussion, the $P = NP$ question can be phrased as follows:

Can the Simulation Problem be solved by a Turing machine which, for fixed M , uses only polynomial time?

Progress on the complexity of simulation can be achieved without answering $P = NP$. For example, if one could find an algorithm which operated in $O(c^{\sqrt{n}})$ time, such an algorithm (assuming small constants) would be a substantial improvement over current $2^{\Theta(n)}$ methods and would have significant practical importance.

As a first step toward quantifying the complexity of simulation, we suggest the following question:

What is the greatest lower bound on the set of numbers a such that the Simulation Problem can be solved (for fixed M) in $O(2^{n^a})$ time?

Using the terminology of [Stearns and Hunt III 1990] and [Stearns 1994], we call this lower bound the “**power index**” of the problem. The reason we say “greatest lower bound” rather than “smallest a ” is that the set may not have a smallest a . This would happen, for example, if the best method took time $\Theta(n^{\sqrt{n}})$ in which case the power index would be $1/2$ but no $O(2^{\sqrt{n}})$ algorithm would exist. The power index of other problems in EXPTIME are similarly defined.

What do we know about the power index of the simulation problem? The answer is “very little”. We know that the index is at most one since that is the complexity of exhaustive simulation. However, we do not know if the answer is greater than zero. The index is zero if $P = NP$ but might be anywhere between zero and one inclusively if $P \neq NP$.

A proof that the power index of the simulation problem is greater than zero would imply $P \neq NP$. Therefore, such a proof must, at best, be difficult to find. However, an improved upper bound on power index could be established by providing a power index a algorithm for some $a < 1$. The existence of such an algorithm would not imply an answer to the $P = NP$ question. The power index concept can be considered “model independent” in that the power index of a problem is the same for any computer model that can be simulated in polynomial time on a Turing machine and that can simulate Turing machines in polynomial time.

Although it is unknown if any NP-complete problem has a non-zero power index, we know that there are problems which have power index a for all time-constructible functions 2^{n^a} . This is because one can construct a set recognizable in $O(2^{n^a})$ time which is not recognizable in $O(2^{n^{a-\epsilon}})$ time for any $\epsilon > 0$ [Hartmanis and Stearns 1965]. The function 2^{n^a} is time-constructible for any rational a so there is a proven hierarchy on EXPTIME based on power index. Also, if some NP-complete problem does have power index a for some $a > 0$, a simple padding argument proves there are NP-complete problems for all smaller rational positive a .

It is also known that there are natural NP-complete problems which have power index less than one. For example CLIQUE, PARTITION, and PLANAR SAT all have power index at most $1/2$ since we have $O(2^{\sqrt{n}})$ time algorithms for these problems [Stearns and Hunt III 1990].

The complexity of Turing machine simulation was attached to SAT (satisfiability for CNF Boolean formulas) in [Cook 1971] where the $P = NP$ problem was posed. This was done using a polynomial time reduction from simulation to SAT. Since then, it has been shown that there are such reductions which are also of size $O(n \log n)$. One such reduction was given in [Schnorr 1978] and another in [Robson 1979]. Further discussion is given in [Cook 1988]. A consequence of such reductions is that the power index of the Simulation Problem is the same as the power index of SAT. Thus our question about the power index of simulation can also be posed as follows:

What is the power index of the satisfiability problem for CNF Boolean formulas?

We know from decidability theory that Turing machine computations are difficult to deal with. It would not be surprising, therefore, if substantial progress on the power index of the Simulation Problem (and therefore of SAT) is impossible. If there is to be a breakthrough in solving these problems, the initial step is more likely to be in the form of an improved $O(2^{n^a})$ algorithm rather than in a polynomial algorithm.

It should be mentioned that, if the power index of SAT is one and thus current algorithms cannot be significantly improved upon, there are important implications for other NP-complete problems. Under the assumption that the power index of SAT is one, certain efficient reductions enable us to establish the power indices of many other NP-complete problems and prove that our algorithms for these problems also can not be improved upon. This includes all the problems in [Karp 1972]. However, there are other NP-complete problems where the best known algorithms can be improved without contradicting the possibility that the power index of SAT is one. This is a topic of [Stearns and Hunt III 1990].

Our failure to prove a non-trivial lower bound for SAT is a symptom of our limited success in proving non-trivial lower time bounds for any natural problem. The successes have mainly been with very hard problems that seem to be outside PSPACE. When we say “natural problem”, we exclude our main source of lower bounds, namely problems constructed by diagonalization. This leads to a second open problem related to, but different from $P = NP$:

Develop techniques for proving non-trivial lower bounds on the time

complexity of natural problems.

One possible attack on the lower bound problem is to look for lower bounds on the complexity of solving generic problems with generic algorithms. By “generic problem”, we mean a problem involving uninterpreted operations and functions. By “generic algorithm”, we mean a program that works for all interpretations of the input. One such generic problem is the following:

Given a set of finite domain variables V and a set of terms T which map variable assignments into elements of a commutative semi-ring $R = (S, +, \cdot, 0, 1)$, find the value of

$$\sum_{\gamma \in \Gamma(V)} \prod_{t \in T} t[\gamma]$$

where $\Gamma(V)$ is the set of assignments to the variables in V and $t[\gamma]$ is the value of term t for assignment γ .

For this particular problem, a generic algorithm would be a program which processes semi-ring elements only with the semi-ring operators, uses only the semi-ring constants 0 and 1, and only tests semi-ring elements for equality. We studied this sum-of-products problem in [Stearns and Hunt III 1996] and a generalization in [Stearns and Hunt III 2002], but we were only looking at how the structure of a set of terms could make a problem instance easier. The problem has also been studied in other guises and at other levels of abstraction, especially in constraint satisfaction as in [Bistarelli et al. 1997], but usually with the goal of restricting the permissible terms so as to make the problem polynomial.

SAT is a sum-of-products problem over the semi-ring $(\{0, 1\}, \vee, \wedge, 0, 1)$, counting the number of satisfying assignments is a sum-of-products problem using the ring of integers, and MAXSAT is a sum-of-products problem using max as plus and integer addition as times. Many more examples of problems that can be interpreted as sum-of-product problems can be found in [Stearns and Hunt III 1996]. Because a generic algorithm for sums-of-products must work for all semi-rings, its complexity must reflect the most difficult interpretations, and thus the generic problem is likely to be inherently hard.

The point of analyzing generic problems in general is that there are limited options available for processing data, hence fewer available algorithms, and hence a better chance of establishing a lower bound. When analyzing sums-of-products in particular, it is important to keep the following in mind. It is best to restrict the variable domains to a fixed set, say $\{0, 1\}$, since we don't want the cardinality of domains to be the cause of complexity. There should be a bound on the number of variables in a term, say three, since we don't want a program to require 2^n time just because it must try all assignments to the variables of some n variable term. Finally, in order to maintain interpretation independence, the complexity should be measured by the number of operations performed. Methods of representing semi-ring elements and the time required to perform semi-ring operations are important but are not generic issues.

REFERENCES

- BISTARELLI, S., MONTANARI, U., AND ROSSI, F. 1997. Semiring-based constraint satisfaction and optimization. *Journal of the ACM* *44*, 2 (Mar.), 201–236.
- COOK, S. 1971. The complexity of theorem-proving procedures. In *Proc. Third Annual ACM Symp. on Theory of Computing*. 151–158.
- COOK, S. 1988. Short propositional formulas represent nondeterministic computations. *Information Processing Letters* *26*, 269–270.
- HARTMANIS, J. AND STEARNS, R. 1965. On the computational complexity of algorithms. *Trans. of the Amer. Math. Society* *117*, 5 (May), 285–306.
- KARP, R. 1972. Reducibility among combinatorial problems. In *Complexity and Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum, New York, 85–103.
- ROBSON, J. 1979. A new proof of the NP-completeness of satisfiability. In *Proc. 2nd Australian Computer Science Conf.* 62–69.
- SCHNORR, C. 1978. Satisfiability is quasi-linear complete in NQL. *Journal of the ACM* *25*, 1 (Jan.), 136–145.
- STEARNS, R. 1994. It's time to reconsider time. *Communications of the ACM* *37*, 11 (Nov.), 95–99.
- STEARNS, R. AND HUNT III, H. 1990. Power indices and easier hard problems. *Math. Systems Theory* *23*, 209–225.
- STEARNS, R. AND HUNT III, H. 1996. An algebraic model for combinatorial problems. *SIAM Journal on Computing* *25*, 2 (Apr.), 448–476.
- STEARNS, R. AND HUNT III, H. 2002. Exploiting structure in quantified formulas. *Journal of Algorithms* *43*, 220–263.