The Achievements and Challenges of Computational Complexity

Stephen A. Cook

University of Toronto

September 30, 1999

Computational Complexity

What is the minimum time required to solve a given problem, considering all possible algorithms?

Can replace "time" by memory, circuit size, etc.

Computer model can be:

- Sequential
- Parallel
- Boolean Circuit family

Alan Turing:
"The Father of Computer Science"

Introduced the "Turing Machine" in 1936.
Argued it can simulate any (human) computer

Other Models:

Allow several tapes
Allow a random access memory

All these are roughly equivalent (can simulate each other by squaring or cubing the running time.)

Proving **upper** bounds on computation time

This is done by presenting algorithms

Rich, well-developed methodology:

- Linear Programming
- Dynamic Programming
- Greedy Algorithms
- Divide and Conquer Algorithms,...

Taught in every undergraduate CS curriculum.

Proving **lower** bounds on computation time

Methods:

Diagonalization

Reduction (reduce hard problem to your problem)

Counting (for Boolean circuit size)

Good success for very hard problems (super exponential complexity)

No success for common practical problems which appear difficult

Scheduling and other NP-complete problems

Factoring integers ...

Diagonalization (Cantor)

Consider functions

$$f : N \mapsto N \cup \{U\}$$

U = "Undefined"
i.e. Turing Machine does not halt

Let $M_1, M_2, ...$ be a computable enumeration of all TM's.

f_n is the function computed by M_n
$f_1, f_2, ...$ lists all computable functions

$$D(n) = \begin{cases} 0 & \text{if } f_n(n) = U \\ U & \text{if } f(n) \neq U \end{cases}$$

$D(n) \neq f_n(n)$
D is not computable

Reduction (Turing)

The halting function is not computable.

$$HALT(m,n) = \begin{cases} 1 & \text{if } M_m \text{ halts on input } n \\ 0 & \text{otherwise} \end{cases}$$

D is reducible to HALT:

$$D(n) = \begin{cases} 0 & \text{if } HALT(n,n) = 0 \\ U & \text{if } HALT(n,n) = 1 \end{cases}$$

Since D is not computable, it follows that HALT is not computable.

Complexity Classes

Hartmanis and Stearns (1965)

Study "languages" $L \subseteq \{0,1\}^*$

Let $T : N \mapsto N$ be a time bound.

$DTIME(T(n)) =$

$\{L \mid$ some TM M recognizes L within time $T(n)\}$

Here n is the *length* of the input.

The diagonal language $D_T \notin DTIME(T(n))$

$D_T = \{w \mid M_w$ does not accept w in time $T(|w|)\}$

Therefore the **universal language**
$UNIV_T \notin DTIME(T(n))$

$UNIV_T = \{\langle w, v\rangle \mid M_w$ accepts v within time $T(|v|)\}$

But if T is "constructible" then
$UNIV_T \in DTIME(T^2(n))$

A Provably Hard Problem

Star-free Regular Expressions

Allow three operations on languages $L \subseteq \{0,1\}^*$:

$\cup$ (union) $\cdot$ (concatenation) $\neg$ (complement)

$$L_1 \cdot L_2 = \{w_1 w_2 \mid w_i \in L_i, i = 1,2\}$$

A SFRE is an expression built from symbols $0, 1, \cup, \cdot, \neg$ where 0,1 stand for the languages $\{0\}, \{1\}$, respectively.

Theorem (Stockmeyer, 1974) The equivalence problem for SFRE's is not in $DTIME(E_k(n))$, for any k, where

$E_1(n) = n$
$E_{k+1}(n) = 2^{E_k(n)}$

Proof: Reduction from $UNIV_{E_k}$.

Meyer and Stockmeyer gave other hard examples

Boolean Circuit Complexity (Shannon)

$B_n = \{f : \{0,1\}^n \mapsto \{0,1\}\}$ (Boolean functions)

A Boolean Circuit C_n has n inputs, 1 output, computes f in B_n. Allow all functions in B_2 as gates.
NO FEEDBACK (No loops).

$BC(f)$ is the minimum number of gates required to compute f.

Theorem (Shannon 1949) Almost all f in B_n have

$$BC(f) \geq 2^n/n$$

Proof: (Counting): $|B_n| = 2^{2^n}$, but at most $(b+n+1)^{2b}16^b b/b!$ circuits with b gates and n inputs.

Theorem (Lupanov 1958) All f in B_n have

$$BC(f) \leq (1+o(1))2^n/n$$

Astronomical Lower Bounds for Specific Functions

For $L \subseteq \{0,1\}^*$, let $f_{L,n}$ be the characteristic function of $L \cap \{0,1\}^n$.

Thus $f_{L,n} \in B_n$

Theorem: There is $L \in DTIME(2^{O(2^n)})$ with

$$BF(f_{L,n}) \geq 2^n/n$$

Proof: Choose L so $f^{L,n}$ is the lexicographically first Boolean function with complexity $\geq 2^n/n$.

Theorem (Stockmeyer 1974) The language EWS1S on inputs of 616 characters requires a Boolean circuit with at least 10^{123} gates.

(The language alphabet has 63 characters.)

The Complexity Class **P**
Polynomial Time

$$\mathbf{P} = \bigcup_{k=1}^{\infty} DTIME(n^k)$$

Suggested independently by Cobham, Edmonds, ca. 1965.

Robust definition

Polytime Thesis: A "natural problem" has a feasible algorithm iff it has a polytime algorithm.

The Complexity Class **NP** Nondeterministic Polynomial Time

$L \in \mathbf{NP} \Longleftrightarrow \exists$ polytime relation $R(x,y) \exists k$:
$L = \{x \mid \exists y(|y| \leq |x|^k \wedge R(x,y)\}$

Example: Scheduling problems

Given a set of classes, times, rooms, constraints: Can the classes be assigned rooms and times satisfying the constraints?

Satisfiability: The Standard Example in **NP**:

SAT is the set of all (strings coding) satisfiable propositional formulas in CNF.

$$(p \vee \bar{q} \vee r) \wedge (q \vee \bar{r} \vee \bar{s}) \wedge ...$$

Then $R(x,y) \Longleftrightarrow y$ is (codes a) truth assignment satisfying the formula (coded by) x.

NP-Completeness

Definition: L is **NP**-complete iff L is in **NP** and every problem in **NP** is p-reducible to L.

Definition of p-reducibility:
$L_1 \leq_p L_2$ iff there is a polytime function f such that for all $w \in \{0,1\}^*$

$$w \in L_1 \Longleftrightarrow f(w) \in L_2$$

Theorem (C71, Levin): SAT is **NP**-complete.

If L is in **NP**, then to show L is **NP**-complete it suffices to show SAT $\leq_p L$.

Thousands of **NP**-complete problems have been identified, including many practical problems (scheduling). (Karp, 1972...)

A polytime algorithm for any one of these problems would yield polytime algorithms for all.

SAT has additional importance because in practice **NP** problems can be efficiently reduced to SAT.

$\mathbf{P} = \mathbf{NP}$??

Obviously $\mathbf{P} \subseteq \mathbf{NP}$.

$\mathbf{P} = \mathbf{NP}$ iff SAT (or any other **NP**-complete problem) is in **P**.

Steve Smale lists the $\mathbf{P} = \mathbf{NP}$ question in the top three

"Mathematical Problems for the Next Century"

(Math. Intelligencer 20, 1998)

Cryptography

Complexity theory has had an enormous impact on modern cryptography.

Public-key cryptosystems (RSA)
Private-key cryptosystems
authentication
signature schemes

Major Application: Financial transactions on the internet.

Theorem: If **P** = **NP** then cryptography is impossible (except for a one-time pad.)

An efficient algorithm for SAT would crack every cryptographic scheme, and our banking system would collapse.

Example: DES (Data Encryption Standard) Used (among other things) for UNIX password security.

DES: Password $W \mapsto$ Encrypted form E

E is public. DES is presumed computationally intractable to invert.

Each E can be mapped to a CNF formula F_E with about 25,000 three-literal clauses. Any satisfying assignment to F_E yields a corresponding password W.

An n^2 algorithm for SAT would (at 10^9 operations/second) crack DES in 10 seconds.

AES (in the works) won't remedy this danger.

Why do Complexity Theorists Believe P $\neq$ NP?

Suppose **P** = **NP**.

How would you prove it?
Just exhibit an efficient algorithm for SAT (or Traveling Salesman Problem...)

Vast numbers of people working to find efficient algorithms for **NP** problems

Example: Integer factorization (Gauss)

Large and sophisticated tool kit available.

Strong motivation (cryptography, industry)

Best known deterministic algorithm for 3-SAT: Time 1.505^n for n variables. (Kullmann)
(Compare with 2^n)

Suppose **P** $\neq$ **NP**.

How would you prove it?

Method 1) Diagonalize and Reduce

But these arguments usually relativize, and

$$\mathbf{P}^A = \mathbf{NP}^A$$

if A is **PSPACE**-complete.

We cannot even disprove SAT decidable in time $O(n)$ on a multitape Turing Machine.

Method 2) Show SAT requires large Boolean circuits. (Every problem in **P** can be solved by polynomial size Boolean circuits.)

Result: Total Failure.
Cannot disprove BC(SAT)$\leq 4n$.
Best upper bound: Exponential

"Natural Proofs" (Razborov/Rudich): Standard methods for proving circuit lower bounds unlikely to succeed.

We know that we have failed to prove true simple proper inclusions:

Example 1:

LOGSPACE $\subseteq$ **P** $\subseteq$ **NP** $\subseteq$ **PSPACE**

We know **LOGSPACE** $\neq$ **PSPACE**, but we cannot prove any of the three inclusions above is proper.

Example 2:

LINEAR-SIZE is the class of problems solvable by Boolean circuits of size $O(n)$.

Open Question: **NP** $\subseteq$ **LINEAR-SIZE** ?

Theorem: Either **P** $\not\subseteq$ **LINEAR-SIZE** or **P** $\neq$ **NP**.

Proof: **PH** $\not\subseteq$ **LINEAR-SIZE**, but
P $=$ **NP** $\Rightarrow$ **PH** $=$ **P**

Randomized Algorithms

A source of random bits seems to make some decision problems easier.

Example 1: Prime recognition (Solovay/Strassen; Miller/Rabin)

Fermat test: Choose a random $c, 1 < c < p$. If

$$c^{p-1} \bmod p \neq 1$$

then p is not prime.

Example 2: (Schwartz) Test whether an $n \times n$ matrix with multivariate polynomial entries is singular.

Randomly set variables to small integer values and evaluate the determinant.

Neither of these problems is known to be in **P**.

Do random bits really help?

Definition: $L \in \mathbf{BPP}$ iff L can be recognized by some randomized polynomial time algorithm, with exponentially small error probability.

Primes and nonsingular multivariate matrices are in **BPP**.

Fundamental Question : **BPP = P**?

In practice, **BPP** algorithms are executed by deterministic pseudo-random number generators (with a "random" seed).

Definition: **E** = $DTIME(2^{O(n)})$.

Assertion A: Some language in **E** requires exponential Boolean circuit size:

$$BC(f_{L,n}) \geq 2^{\epsilon n}, \text{ for some } \epsilon > 0$$

Theorem:

If **A** then **BPP = P**. (Imp/Wig)

If ¬**A**, then **P** ≠ **NP**. (Kabanets)

Propositional Proof Systems

Abstract Def'n: A proof system is a polytime map

$$f : \{0,1\}^* \stackrel{\text{onto}}{\longrightarrow} \{\text{tautologies}\}$$

If $f(x) = A$, then x is a *proof* of A.

Def'n: The system is *polybounded* iff for some polynomial $p(n)$, every tautology of length n has a proof of length at most $p(n)$.

Observation: **NP = co-NP** iff there exists a polybounded proof system.

Weak Conjecture: **NP** $\neq$ **co-NP**

But possibly **NP= co-NP** and still
P $\neq$ **NP**

Try to prove specific proof systems are not polybounded.

Resolution

Resolution is a refutation system for CNF formulas, but it can be turned into a proof system for tautologies.

Theorem: (Haken 1985) Resolution is not polybounded.

In fact "Pigeon Hole Formulas" require exponentially many resolvents to refute.

Corollary The Davis-Putnam procedure (and many variations) for SAT requires exponential time in the worst case.

Theorem: (Urq;Chv/Sze, 1987) For an appropriate distribution, a random 3-CNF formula is almost surely unsatisfiable, but almost surely requires exponentially many resolvents to refute.

Other proof systems known NOT polybounded:

Bounded Depth Frege Systems
Cutting Plane Systems
Polynomial Calculus

Major Open Question: Are Frege Systems Polybounded??

Progress(?) in showing SAT $\notin$ P

Fortnow recently proved that SAT cannot be solved simultaneously in time $n^{1+o(1)}$ and space $n^{1-\epsilon}$ on a TM, for any $\epsilon > 0$.