BPP's Circuit Complexity

Theorem 77 (Adleman (1978)) *All languages in BPP have polynomial circuits.*

- Our proof will be *nonconstructive* in that only the existence of the desired circuits is shown.
  - Recall our proof of Theorem 14 (p. 164).
  - Something exists if its probability of existence is nonzero.
- It is not known how to efficiently generate circuit C_n.
- If the construction of C_n can be made efficient, then P = BPP, an unlikely result.

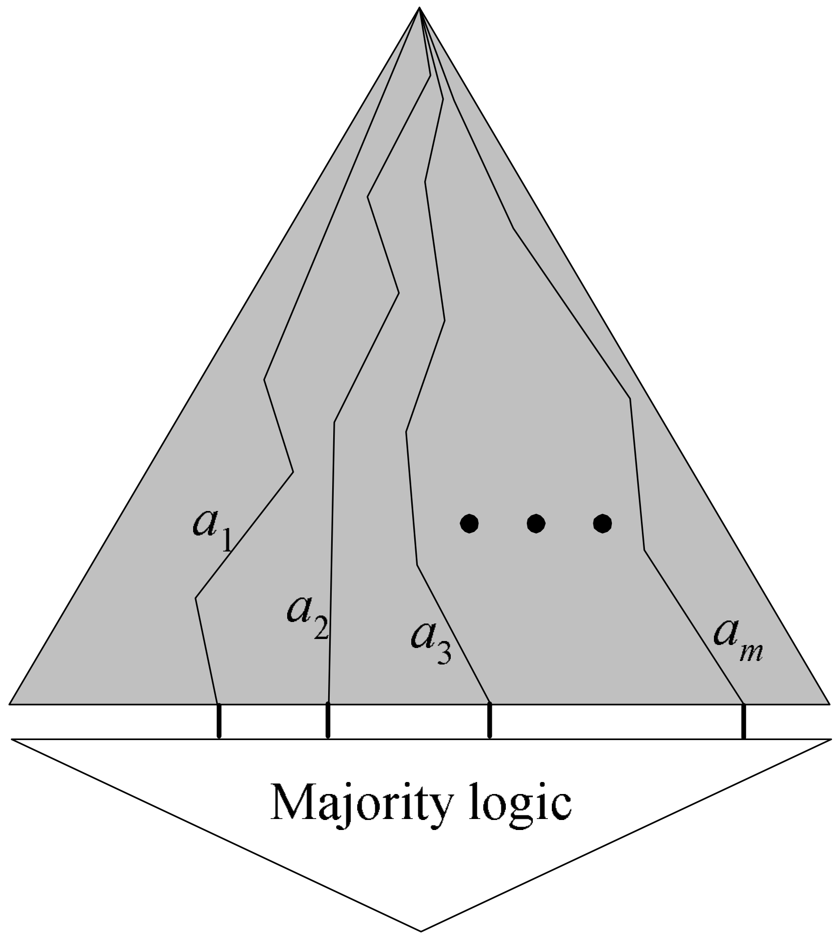
The Proof

- Let $L \in \text{BPP}$ be decided by a precise NTM N by clear majority.
- We shall prove that L has polynomial circuits $C_0, C_1, \ldots$.
- Suppose N runs in time $p(n)$, where $p(n)$ is a polynomial.
- Let $A_n = \{a_1, a_2, \ldots, a_m\}$, where $a_i \in \{0, 1\}^{p(n)}$.
- Pick $m = 12(n + 1)$.
- Each $a_i \in A_n$ represents a sequence of nondeterministic choices (i.e., a computation path) for N.

The Proof (continued)

- Let x be an input with $|x| = n$.
- Circuit C_n simulates N on x with each sequence of choices in A_n and then takes the majority of the m outcomes.
- Because N with a_i is a polynomial-time TM, it can be simulated by polynomial circuits of size $O(p(n)^2)$.
  - See the proof of Proposition 75 (p. 539).
- The size of C_n is therefore $O(mp(n)^2) = O(np(n)^2)$.
  - This is a polynomial.

The Circuit



The Proof (continued)

- We now prove the existence of an A_n making C_n correct on *all* inputs.
- Call a_i **bad** if it leads N to a false positive or a false negative.
- Select A_n *uniformly randomly*.
- For each $x \in \{0,1\}^n$, $1/4$ of the computations of N are erroneous.
- Because the sequences in A_n are chosen randomly and independently, the expected number of bad a_i's is $m/4$.

The Proof (continued)

- By the Chernoff bound (p. 521), the probability that the number of bad a_i's is $m/2$ or more is at most

$$e^{-m/12} < 2^{-(n+1)}.$$

- The error probability is $< 2^{-(n+1)}$ for each $x \in \{0,1\}^n$.
- The probability that there is an x such that A_n results in an incorrect answer is $< 2^n 2^{-(n+1)} = 2^{-1}$.
  - $\text{prob}[\, A \cup B \cup \cdots] \leq \text{prob}[\, A\,] + \text{prob}[\, B\,] + \cdots$.
- Note that each A_n yields a circuit.
- We just showed that at least half of them are correct.

The Proof (concluded)

- So with probability ≥ 0.5, a random A_n produces a correct C_n for *all* inputs of length n.
- Because this probability exceeds 0, an A_n that makes majority vote work for all inputs of length n exists.
- Hence a correct C_n exists.[a]
- We have used the **probabilistic method**.

[a] Quine (1948), "To be is to be the value of a bound variable."

Lengths of Boolean Formulas for the Threshold Function

- Define the boolean function $T_k(x_1, \ldots, x_n)$ to be 1 if at least k of the x_i's are 1s, and 0 otherwise.
- Trivially, a formula of size $O(\binom{n}{k})$ exists.
- Surprisingly, for any k, there exists a constant c_k such that $T_k(x_1, \ldots, x_n)$ has formula size at most $c_k n \log_2 n$.
- The construction is again probabilistic, not constructive.

Lengths of Boolean Formulas for the Threshold Function (continued)

- We will verify the $k = 3$ case below.
- Suppose we construct the formula of the form

$$F = F_1 \vee \cdots \vee F_r .$$

- Each F_i is constructed *randomly* and takes the form:

$$F_i = (\vee \cdots) \wedge (\vee \cdots) \wedge (\vee \cdots).$$

  – By the distribution law,

$$\begin{aligned} & (a_1 \vee a_2 \vee \cdots) \wedge (b_1 \vee b_2 \vee \cdots) \wedge (c_1 \vee c_2 \vee \cdots) \\ = \quad & (a_1 \wedge b_1 \wedge c_1) \vee (a_1 \wedge b_1 \wedge c_2) \vee \cdots . \end{aligned}$$

Lengths of Boolean Formulas for the Threshold Function (continued)

- Each x_j is placed into one of the brackets at random.
- Each F_i has exactly n variables.
- Clearly, all the monomials of F are of the form $x_a \wedge x_b \wedge x_c$ for distinct a, b, c.
- But T_3 has $\binom{n}{3}$ monomials.
- We shall show, if r is large enough, all $\binom{n}{3}$ monomials will appear with high probability.

Lengths of Boolean Formulas for the Threshold Function (continued)

- The probability that any given monomial $x_a \wedge x_b \wedge x_c$ appears in a given F_i is the probability that x_a, x_b, x_c are thrown into *distinct* brackets.
- The probability is hence equal to $(2/3)(1/3) = 2/9$.
- The probability that $x_a \wedge x_b \wedge x_c$ is not a monomial of F_i's is $(7/9)^r$.
- Therefore, the probability that at least one of the $\binom{n}{3} \leq n^3$ monomials is missing from all the F_i's is at most $n^3(7/9)^r$.

Lengths of Boolean Formulas for the Threshold Function (concluded)

- This probability is less than one when $n^3(7/9)^r < 1$.
- When this happens, F includes all $\binom{n}{3}$ monomials, and F has size $< rn$.
- In particular, with $r = -\log_{7/9} 2n^3$, the probability that $F \neq T_3$ is at most $1/2$.
- In other words, the probability of that $F = T_3$ is at least $1/2$.
- Hence a formula of size $O(n \log n)$ exists.

Finding Short Formulas for the Threshold Function

- Our analysis implies an expected polynomial-time randomized algorithm to find such a formula (for T_3).
- Generate F randomly as described.
- In $O(\binom{n}{3}) = O(n^3)$ time, evaluate F with every n-bit truth assignment with three 1's and check if $F = 1$.
- In $O(\binom{n}{2}) = O(n^2)$ time, evaluate F with every n-bit truth assignment with two 1's and check if $F = 0$.
- In $O(n)$ time, evaluate F with every n-bit truth assignment with one 1 and check if $F = 0$.
- Check if $F = 0$ with the all-0 truth assignment.

Finding Short Formulas for the Threshold Function (concluded)

- If F passes all the tests, return F.
  - No need to check if $F = 1$ when the truth assignment contains more than three 1's because F is monotone.[a]
- Otherwise, repeat the experiment.
- Clearly, the expected running time to find a valid formula is proportional to

$$n^3 + (1/2)\, n^3 + (1/2)^2\, n^3 + \cdots = O(n^3).$$

[a] Thanks to a lively class discussion on December 8, 2009.

Cryptography

Whoever wishes to keep a secret
must hide the fact that he possesses one.
— Johann Wolfgang von Goethe (1749–1832)

Cryptography

- **Alice** (A) wants to send a message to **Bob** (B) over a channel monitored by **Eve** (eavesdropper).
- The protocol should be such that the message is known only to Alice and Bob.
- The art and science of keeping messages secure is **cryptography**.

$$\text{Alice} \xrightarrow{\text{Eve}} \text{Bob}$$

Encryption and Decryption

- Alice and Bob agree on two algorithms E and D—the **encryption** and the **decryption algorithms**.
- Both E and D are known to the public in the analysis.
- Alice runs E and wants to send a message x to Bob.
- Bob operates D.
- Privacy is assured in terms of two numbers e, d, the **encryption** and **decryption keys**.
- Alice sends $y = E(e, x)$ to Bob, who then performs $D(d, y) = x$ to recover x.
- x is called **plaintext**, and y is called **ciphertext**.[a]

[a] Both "zero" and "cipher" come from the same Arab word.

Some Requirements

- D should be an inverse of E given e and d.
- D and E must both run in (probabilistic) polynomial time.
- Eve should not be able to recover x from y without knowing d.
  - As D is public, d must be kept secret.
  - e may or may not be a secret.

Degrees of Security

- **Perfect secrecy**: After a ciphertext is intercepted by the enemy, the a posteriori probabilities of the plaintext that this ciphertext represents are identical to the a priori probabilities of the same plaintext before the interception.
  - The probability that plaintext $\mathcal{P}$ occurs is independent of the ciphertext $\mathcal{C}$ being observed.
  - So knowing $\mathcal{C}$ yields no advantage in recovering $\mathcal{P}$.
- Such systems are said to be **informationally secure**.
- A system is **computationally secure** if breaking it is theoretically possible but computationally infeasible.

Conditions for Perfect Secrecy[a]

- Consider a cryptosystem where:
  - The space of ciphertext is as large as that of keys.
  - Every plaintext has a nonzero probability of being used.
- It is perfectly secure if and only if the following hold.
  - A key is chosen with uniform distribution.
  - For each plaintext x and ciphertext y, there exists a unique key e such that $E(e, x) = y$.

---

[a]Shannon (1949).

The One-Time Pad[a]

1: Alice generates a random string r as long as x;

2: Alice sends r to Bob over a secret channel;

3: Alice sends $r \oplus x$ to Bob over a public channel;

4: Bob receives y;

5: Bob recovers $x := y \oplus r$;

---

[a]Mauborgne and Vernam (1917); Shannon (1949). It was allegedly used for the hotline between Russia and U.S.

Analysis

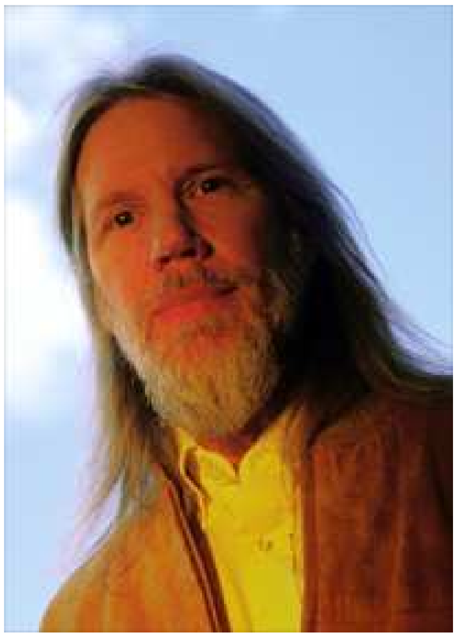
- The one-time pad uses $e = d = r$.
- This is said to be a **private-key cryptosystem**.
- Knowing x and knowing r are equivalent.
- Because r is random and private, the one-time pad achieves perfect secrecy (see also p. 566).
- The random bit string must be new for each round of communication.
  - **Cryptographically strong pseudorandom generators** require exchanging only the seed once.
- The assumption of a private channel is problematic.

Public-Key Cryptography[a]

- Suppose only d is private to Bob, whereas e is public knowledge.
- Bob generates the (e, d) pair and publishes e.
- Anybody like Alice can send $E(e, x)$ to Bob.
- Knowing d, Bob can recover x by $D(d, E(e, x)) = x$.
- The assumptions are complexity-theoretic.
  - It is computationally difficult to compute d from e.
  - It is computationally difficult to compute x from y without knowing d.

[a]Diffie and Hellman (1976).

Whitfield Diffie (1944–)



Martin Hellman (1945–)

Complexity Issues

- Given y and x, it is easy to verify whether $E(e, x) = y$.
- Hence one can always guess an x and verify.
- Cracking a public-key cryptosystem is thus in NP.
- A necessary condition for the existence of secure public-key cryptosystems is P $\neq$ NP.
- But more is needed than P $\neq$ NP.
- For instance, it is not sufficient that D is hard to compute in the worst case.
- It should be hard in "most" or "average" cases.

One-Way Functions

A function f is a **one-way function** if the following hold.[a]

1. f is one-to-one.
2. For all $x \in \Sigma^*$, $|\, x\, |^{1/k} \leq |f(x)| \leq |\, x\, |^k$ for some $k > 0$.
   - f is said to be **honest**.
3. f can be computed in polynomial time.
4. f^{-1} cannot be computed in polynomial time.
   - Exhaustive search works, but it is too slow.

[a] Diffie and Hellman (1976); Boppana and Lagarias (1986); Grollmann and Selman (1988); Ko (1985); Ko, Long, and Du (1986); Watanabe (1985); Young (1983).

Existence of One-Way Functions

- Even if P $\neq$ NP, there is no guarantee that one-way functions exist.
- No functions have been proved to be one-way.
- Is breaking glass a one-way function?

Candidates of One-Way Functions

- Modular exponentiation $f(x) = g^x \bmod p$, where g is a primitive root of p.
  - **Discrete logarithm** is hard.[a]
- The RSA[b] function $f(x) = x^e \bmod pq$ for an odd e relatively prime to $\phi(pq)$.
  - Breaking the RSA function is hard.

---

[a] Conjectured to be 2^{n^ϵ} for some $\epsilon > 0$ in both the worst-case sense and average sense. It is in NP in some sense (Grollmann and Selman (1988)).

[b] Rivest, Shamir, and Adleman (1978).

Candidates of One-Way Functions (concluded)

- Modular squaring $f(x) = x^2 \bmod pq$.
  - Determining if a number with a Jacobi symbol 1 is a quadratic residue is hard—the **quadratic residuacity assumption** (**QRA**).[a]

[a] Due to Gauss.

The RSA Function

- Let p, q be two distinct primes.
- The RSA function is $x^e \bmod pq$ for an odd e relatively prime to $\phi(pq)$.
  - By Lemma 54 (p. 405),

$$\phi(pq) = pq\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) = pq - p - q + 1. \quad (8)$$

- As $\gcd(e, \phi(pq)) = 1$, there is a d such that

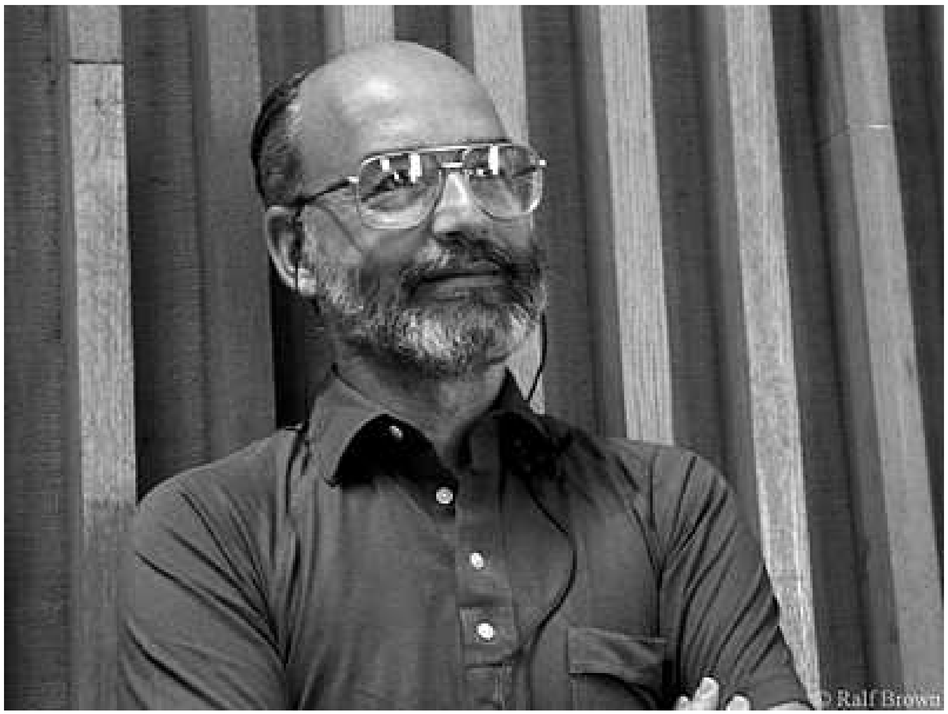
$$ed \equiv 1 \bmod \phi(pq),$$

which can be found by the Euclidean algorithm.

Adi Shamir, Ron Rivest, and Leonard Adleman

Ron Rivest (1947–)

Adi Shamir (1952–)



Leonard Adleman (1945–)

A Public-Key Cryptosystem Based on RSA

- Bob generates p and q.
- Bob publishes pq and the encryption key e, a number relatively prime to $\phi(pq)$.
  - The encryption function is $y = x^e \bmod pq$.
  - Bob calculates $\phi(pq)$ by Eq. (8) (p. 577).
  - Bob then calculates d such that $ed = 1 + k\phi(pq)$ for some $k \in \mathbb{Z}$.
- The decryption function is $y^d \bmod pq$.
- It works because $y^d = x^{ed} = x^{1+k\phi(pq)} = x \bmod pq$ by the Fermat-Euler theorem when $\gcd(x, pq) = 1$ (p. 413).

The "Security" of the RSA Function

- Factoring pq or calculating d from (e, pq) seems hard.
  - See also p. 409.
- Breaking the last bit of RSA is as hard as breaking the RSA.[a]
- Recommended RSA key sizes:[b]
  - 1024 bits up to 2010.
  - 2048 bits up to 2030.
  - 3072 bits up to 2031 and beyond.

---

[a] Alexi, Chor, Goldreich, and Schnorr (1988).
[b] RSA (2003).

The "Security" of the RSA Function (concluded)

- Recall that problem A is "harder than" problem B if solving A results in solving B.
  - Factorization is "harder than" breaking the RSA.
  - Calculating Euler's phi function is "harder than" breaking the RSA.
  - Factorization is "harder than" calculating Euler's phi function (see Lemma 54 on p. 405).
  - So factorization is hardest, followed by calculating Euler's phi function, followed by breaking the RSA.
- Factorization cannot be NP-hard unless NP = coNP.[a]
- So breaking the RSA is unlikely to imply P = NP.

[a] Brassard (1979).