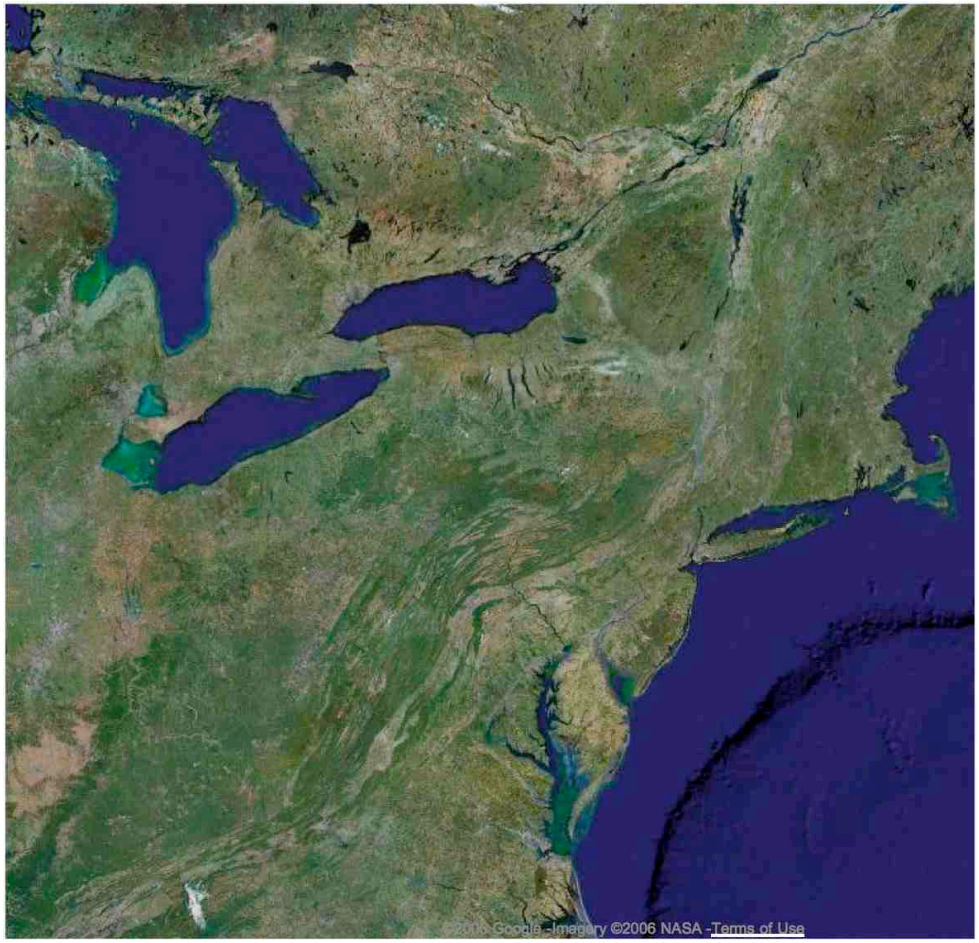
Computational Complexity
An Overview of Open Problems

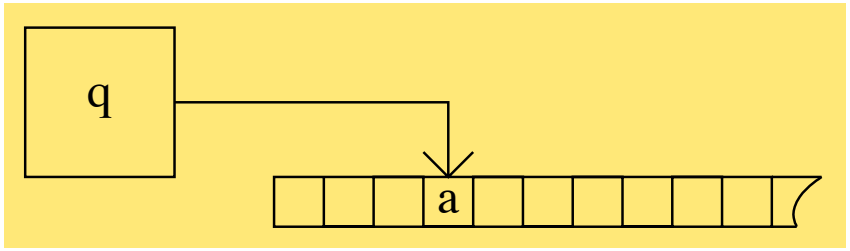
Michael Soltys
McMaster University, Canada


©2006 Google - Imagery ©2006 NASA - Terms of Use

Seminal paper of Hartamnis and Stearns,

On the computational complexity of algorithms.
Transactions of the American Mathematical Society, **1965**.

Definitions of quantified time and space complexity on Turing machines.



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

- Are there efficient(=***polytime***) algorithms for hard problems (traveling salesman, satisfiability, knapsack packing, etc.) ?
- Is ***finding*** a solution intrinsically harder than ***verifying*** a solution ?

One of 7 Millennium Problems (6 left; Poincaré Conjecture solved?).

Clay Mathematical Institute offers 1 million for a solution. (`http://www.claymath.org/millennium/P_vs_NP`)

Most researchers believe $\mathbf{P} \neq \mathbf{NP}$: we are very good at inventing efficient algorithms but really bad at proving algorithms don't exist.

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

- The proof is non-constructive or impractical (yields algorithms in $O(n^{100})$). In this case few practical consequences.
- But, experience shows that for "natural" problems in **P** we can find feasible algorithms.
- Suppose we have a feasible algorithm for SAT.
  - Hundreds of **NP**-complete problems reducible to SAT.
  - Mathematics transformed: we could find short proofs.
  - Fundamental problems in AI: planning, natural language understanding, vision, etc., made easy.
- Negative consequence: security of the Internet depends on hardness of factoring!

Suppose $\mathbf{P} \neq \mathbf{NP}$.

- Complexity theorists are *not* out of work:
  - How large is the time lower bound for SAT ? (super-poly or exponential or in-between ?)
  - Worst-case inputs or convincing average-case lower bounds ?
  - What about lower bounds for **NP** approximation problems?
  - What about integer factorization: it may not be **NP**-hard ?
- In general, proving that security of cryptographic protocols such as RSA or DES is much harder than proving $\mathbf{P} \neq \mathbf{NP}$.
- Today, no convincing program for showing that $\mathbf{P} \neq \mathbf{NP}$. (Standard techniques fail: diagonalization, circuit lower bounds, etc.)

NP = **co-NP** ?

- Are there succinct proofs of tautologies ?
  (i.e., can we do better than truth tables ?)
- Of interest to:
  - Automated theorem proving
  - Lower bounds for algorithms for satisfiability
- Plan of attack: show lower bounds for stronger and stronger propositional proof systems.

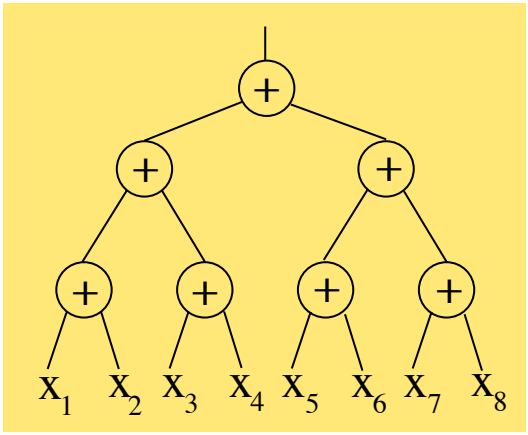
  Resolution, Bounded Depth-Frege, Bounded Depth-Frege with modular gates, Frege, Extended Frege, Quantified Frege.
- **NP** $\neq$ **co-NP** implies **P** $\neq$ **NP**.

Can we construct hard boolean functions ?

Let B_n be the set of boolean functions on n variables.

A **gate** is a binary boolean function (there are 16, e.g., AND, OR).

This circuit computes PARITY $\in B_8$, using one gate type $\oplus$ = XOR.



Almost all $f \in B_n$ require large circuits ($\Omega(2^n/n)$). The hardest ***explicit*** function we know requires $3n - o(n)$ many gates.

C. Shannon. The Synthesis of two terminal networks. Bell Systems Tech. Journal, 28, 1949.

N. Blum, *A boolean function requiring 3n size*, Theoretical Computer Science 28 (1984), 337–345.

Primality is in P; what about factoring ?

Agrawal, Kayal, Saxena. PRIMES is in P. Annals of Math. 160, no. 2 (2004).

PRIMES $\in$ **NP** $\cap$ **co-NP**.

Fact: A number $p > 1$ is prime iff $\exists 1 < r < p$ such that $r^{p-1} = 1 \pmod{p}$, and furthermore $r^{\frac{p-1}{q}} \neq 1 \pmod{p}$ for all prime divisors q of $p - 1$.

A *certificate of primality* : $C(p) := (r; q_1, C(q_1), \ldots, q_k, C(q_k))$.

PRIMES $\in$ **co-RP** (and COMPOSITES in **RP**).

(Rabin Miller) On input p:
if p is even, accept if $p = 2$, otherwise reject
Select $a \in \mathbb{Z}_p^+ = \{1, 2, \ldots, (p-1)\}$ at random
if $a^{p-1} \neq 1 \pmod{p}$ reject
else let $(p-1) = st$ where s is odd & $t = 2^h$
compute $S = \{a^{s \cdot 2^j} \pmod{p}\}_{j=0}^{h}$
if every element in S is 1, accept
else find the last element that is not 1
if that element is -1, accept
else reject

No *false-negatives*, and *false-positives* with $\mathbf{Pr} \leq \frac{1}{2}$, so this is a Monte-Carlo algorithm.

So can we factor in **P** ?

Can we show that we cannot factor in **P** ?

Is **RSA** secure ?

> **Avelino** sets up a mechanism whereby he can receive and decode encoded messages from an arbitrary person — and no one else can read them.

RSA: Public-key encryption scheme

Avelino advertises a function E, and *anyone* can compute $E(M)$ for *any* message M, but only **Avelino** can *efficiently* compute M from $E(M)$ using D, $D(E(M)) = M$.

```
Choose two odd primes p, q, set n = pq
Choose k ∈ Z*_φ(n), k > 1
Advertise E, where E(M) = M^k (mod n)
Compute l = k^{-1} (inverse of k in Z*_φ(n))
Secret D, where D(C) = C^l (mod n).
```

$\phi(n)$ is the *Euler totient function*, nr. of elements co-prime to n.
Euler's Thm. $a^{\phi(n)} = 1 \pmod{n}$.
Observe that $\phi(n) = (p-1)(q-1)$.
Note that $D(E(M)) = M^{kl} \pmod{n} = M$
How to get p, q ?

How secure is RSA ?

Suppose that **Bibiana** is eavesdropping.

If she can compute l (secret) from n, k (public), then she could also factor n in randomized polytime:

Suppose Bibiana successfully computes l.
$\phi(n)|(kl-1)$, so she has $m = (kl-1)$ some multiple of $\phi(n)$.
Further, $\phi(n) = \phi(pq) = (p-1)(q-1)$.
Thus

$$\begin{array}{ll} p+q = n - \phi(n) + 1 & \\ pq = n & \end{array} \quad \Rightarrow \quad \begin{array}{l} p+q = n - (m/a) + 1 \\ pq = n \end{array}$$

From this, she obtains p, q.

Thus, breaking the RSA scheme (by computing l) is at least as hard as factoring ...

What about a clever scheme to infer M from $E(M)$ *without* computing l ?

If RSA can be broken on some small fraction of the inputs, then it is totally insecure.

Let $C(A)$ be the set of all $x \in \mathbb{Z}_n^*$ such that A can compute $x^l \pmod n$ from $\langle n, k \rangle$.

Suppose there exists a (possibly randomized) polytime algorithm A_1 for which $|C(A_1)| \geq \varepsilon |\mathbb{Z}_n^*|$, for some $\varepsilon > 0$.

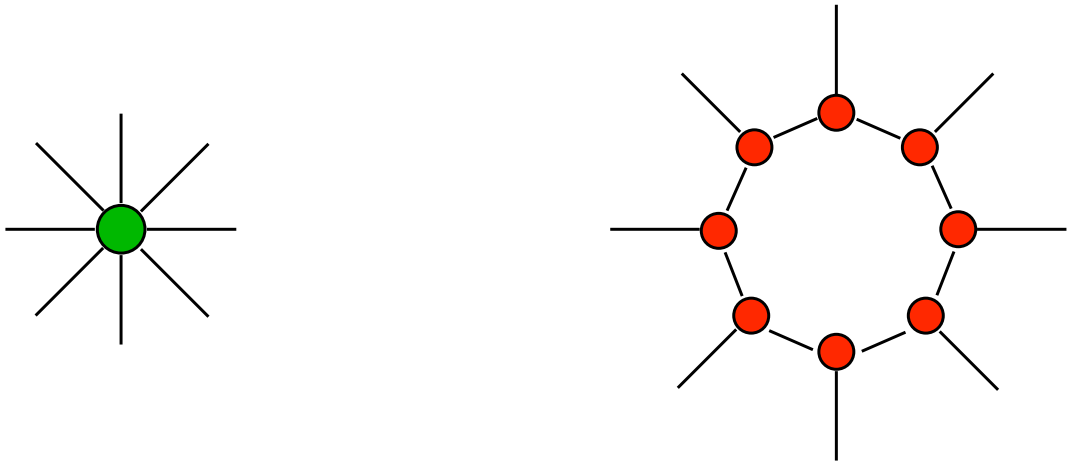
Then, there exists a Las Vegas algorithm A_2 for which $C(A_2) = \mathbb{Z}_n^*$, and the expected running time of A_2 is polynomial in $\frac{1}{\varepsilon} \log n$.

L = SL

Omer Reingold. Undirected ST-connectivity in log-space. STOC'2005.

Main Idea: Depth-first-search to establish if $s \rightsquigarrow t$ requires $O(n \log n)$ space for a graph of n nodes.

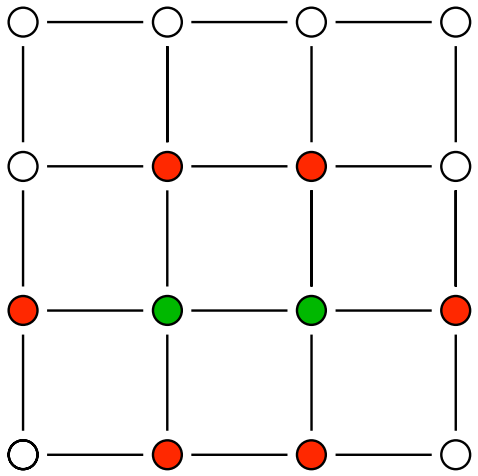
Transform the graph so that it has a bounded degree:



Now depth-first-search: $O(n \log d) = O(n)$.

Need to decrease the diameter from n to $\log n$: convert the graph (in log-space) to an expander graph – preserving constant degree and adding few new nodes.

Expander Graph



There exists an $\varepsilon > 0$ so that for any $S \subseteq V$, $|S| \leq \frac{|V|}{2}$, $\text{nhbd}(S) > (1+\varepsilon)|S|$.

Introduce the z-product on graphs ...

We know $\mathbf{L} = \mathbf{SL} \subseteq \mathbf{RL} \subseteq \mathbf{NL}$.

$\mathbf{L} = \mathbf{NL}$?

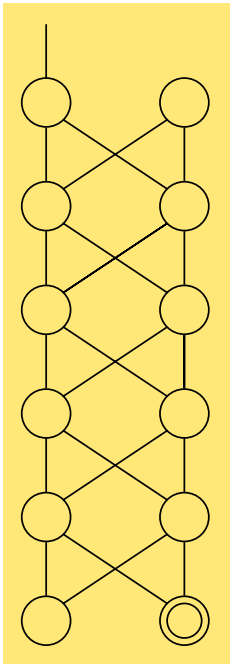
$\mathbf{L} = \mathbf{RL}$?

We know $\mathbf{NL/poly} = \mathbf{UL/poly}$.

$\mathbf{NL} = \mathbf{UL}$?

5-PBP = $\mathbf{NC}^1$; what about 4-PBP ?

D. Barrington. *Bounded-width polynomial-size branching programs recognize exactly those languages in* $\mathbf{NC}^1$. Journal of Comp. and System Sci., 1989.



A 2-PBP computing
$\mathsf{XOR}(x_1, x_2, x_3, x_4, x_5)$

α	(12345)
β	(13542)
γ	(13254)
θ_0	(12534)
θ_1	(14253)
θ_2	(14325)
θ_3	(15324)

$\gamma = [\alpha, \beta] = \beta^{-1} \circ \alpha^{-1} \circ \beta \circ \alpha$
$\theta_0^{-1} \circ \gamma \circ \theta_0 = \alpha$
$\theta_1^{-1} \circ \gamma \circ \theta_1 = \beta$
$\theta_2^{-1} \circ \gamma \circ \theta_2 = \alpha^{-1}$
$\theta_3^{-1} \circ \gamma \circ \theta_3 = \beta^{-1}$

$\mathbf{NC}^1 \rightsquigarrow$ 5-PBP

C_n is a circuit.

Inductively build a program P for each gate in the circuit, starting with the input gates.

For each gate g, P^g is the program that computes g in the following sense: $g(x) = 1 \Rightarrow P^g[x] = e$, and $g(x) = 0 \Rightarrow P^g[x] = \gamma$.

Notice that our program is always e or γ.

For input gates $x_i, \bar{x}_i$ (we assume that we only have negations at the inputs) $P^{x_i} = \{(i, e, \gamma)\}$ and $P^{\bar{x}_i} = \{(i, \gamma, e)\}$.

Suppose we have PBP programs for gates g_1, g_2, call them P^{g_1}, P^{g_2}, respectively.

Assume that $f = g_1 \vee g_2$, and we show how to construct P^f:

$$\underbrace{\theta_0^{-1} P^{g_2} \gamma^{-1} \theta_0}_{S_1} \quad \underbrace{\theta_1^{-1} P^{g_1} \gamma^{-1} \theta_1}_{S_2} \quad \underbrace{\theta_2^{-1} P^{g_2} \gamma^{-1} \theta_2}_{S_3} \quad \underbrace{\theta_3^{-1} P^{g_1} \gamma^{-1} \theta_3 \gamma}_{S_4}$$

P^{g_1}	P^{g_2}	S_1	S_2	S_3	S_4	P^f
e	e	α^{-1}	β^{-1}	α	$\beta\gamma$	e
e	γ	e	β^{-1}	e	$\beta\gamma$	γ
γ	e	α^{-1}	e	α	γ	γ
γ	γ	e	e	e	γ	γ

This works because S_5 is not solvable!

Where is this fact used in the above proof?

The unsolvability of S_5 implies that S_5 has a subgroup $H \neq \{e\}$ such that $H = [H, H]$. (Note that given $h_1, h_2 \in H$, $[h_1, h_2] = h_2^{-1} h_1^{-1} h_2 h_1$, and $[H, H] = \langle [h_1, h_2] : h_1, h_2 \in H \rangle$, that is, the group generated by the commutators.)

The fact that $H = [H, H]$ implies that every element of H can be written as a product of commutators of H, i.e., for every $h \in H$, there exist $a_1, b_1, \ldots, a_k, b_k \in H$ such that $h = [a_1, b_1] \cdots [a_k, b_k]$.

We know k-**PBP** = **5-PBP** for every $k \geq 5$. We also know **1,2,3-PBP**s are too weak.

What about **4-PBP** ?

Many open problems in Complexity Theory.

Interesting mathematically, and relevant from an engineering point of view. (*Computers and Intractability.* Garey and Johnson, 1979.)

Many are very difficult, but a poll of leading researchers says that $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ will be settled by 2100. (The $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ poll. SIGACT news, 33(2):34–47, June 2002.)

There are surprises:

- Savitch: $\mathbf{PSPACE} = \mathbf{NPSPACE}$. [1970]
- Immerman-Szelepscényi result: $\mathbf{NL} = \mathbf{co\text{-}NL}$. [1988]
- Reingold's result $\mathbf{L} = \mathbf{SL}$. [2004]

Future: model theory, algebra, number theory, ... ?