Is It As Easy To "Discover" As To "Verify"?

Sam Buss
Department of Mathematics
U.C. San Diego

UCSD Math Circle
May 22, 2004

A "discovered" identity

The Bailey-Borwein-Plouffe formula for π (1997):

$$\pi \;=\; \sum_{n=0}^{\infty} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \left(\frac{1}{16} \right)^n$$

The proof

Proof. This identity is equivalent to:

$$\pi = \int_0^{1/\sqrt{2}} \frac{4\sqrt{2} - 8x^3 - 4\sqrt{2}x^4 - 8x^5}{1 - x^8} dx, \tag{1.4}$$

which on substituting $y := \sqrt{2}x$ becomes

$$\pi = \int_0^1 \frac{16\,y - 16}{y^4 - 2\,y^3 + 4\,y - 4} dy\,.$$

The equivalence of (1.2) and (1.4) is straightforward. It follows from the identity

$$\begin{aligned} \int_0^{1/\sqrt{2}} \frac{x^{k-1}}{1 - x^8}\, dx &= \int_0^{1/\sqrt{2}} \sum_{i=0}^{\infty} x^{k-1+8i}\, dx \\ &= \frac{1}{\sqrt{2}^k} \sum_{i=0}^{\infty} \frac{1}{16^i(8i + k)} \end{aligned}$$

That the integral (1.4) evaluates to π is an exercise in partial fractions most easily done in Maple or Mathematica. □

Prove a number is composite?

RSA-129 Problem: Prove the following number is composite:

$$1143816257578888676692357799761466120102182\cdots$$
$$\cdots 967212423625625618429357069352457338978305\cdots$$
$$\cdots 971235639587050589890751475992900268795435 41$$

Prove a number is composite!

RSA-129 Problem: Prove the following number is composite:

$$1143816257578888676692357799761466120102182\cdots$$
$$\cdots 967212423625625618429357069352457338978305\cdots$$
$$\cdots 971235639587050589890751475992900268795435 41$$

Proof: It equals the product

$$349052951084765094914784961990 38\cdots$$
$$\cdots 9813341776463849338784399082 0577$$
$$* 3276913299326670954996198819 0834\cdots$$
$$\cdots 4614131776429679929425397982 88533.$$

.

The RSA-129 number was issued as a challenge problem in 1977 in cryptography. It was used to encode the message

"THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE"

Decoding the message required factoring RSA-129 and was done in 1994 by a team of researchers using computers networked over the internet. It required over 5000 MIPS years of computation, using 750 10MHz computers, for eight months to find the factorization.

"Obviously" easier to verify the factorization than it is to discover it!
((???—Open Question!!!))

Computational complexity of factorization

Consider the problem of factoring an integer x. Assume x written in base two notation. Then the length of the input is

$$n \approx \log_2 x.$$

Brute force search for the smallest factor of x needs to search up to $\sqrt{x}$, that is, for a factor of length up to $n/2$ bits. The computational complexity (=run time) for this is approximately

$$2^{n/2} \cdot (\text{time to check if factor}).$$

SUMMARY: Discovery time $\approx 2^{n/2}$ "Infeasible"

Verification time $\approx n^2$ "Feasible"

"Feasible" versus "Infeasible": Informal definitions.

A problem is *feasible* if, a computer algorithm can solve instances of the problem in a reasonable amount of time — at least, for inputs of reasonable size.

A problem is *infeasible* if it is not feasible.

Usual mathematical formalization of "feasible" is in terms of "polynomial time": A problem is "*polynomial time computable*" if there is an algorithm A and a constant $c \geq 1$ such that for all inputs x of length n, the algorithm A can solve the problem for x in time $O(n^c)$.

P - "Polynomial Time"

For example, if you are given two integers a and b, checking whether a is a factor of b can be done in time $O(n^2)$ where n is the sum of the lengths of a and b. The integers a and b are together the input x.

Thus verifying a divisor is in **P** and is feasible.

On the other hand, given x as input, to find a factor of x takes $2^{n/2}$ trials in the worst case. (If x is not prime, it must have a factor $< \sqrt{x}$.) This is *exponential time*, not *polynomial time*, and is considered infeasible.

The best known algorithms for finding a factorization have runtime $\approx 2^{\sqrt[3]{n}}$. There are more efficient algorithms for determining whether x is prime.

Polynomial-time = Good; Exponential-time = Bad

Input Size	Run Time in Microseconds				
	n	n^2	n^3	$2^{\sqrt{n}}$	2^n
10	10 μs	100 μs	1 ms	9 μs	1 ms
100	100 μs	10 ms	1 sec.	1 μs	10^{14} years
1000	1 ms	1 sec.	16.6 min.	55 min.	10^{288} years
10,000	10 ms	100 sec.	11.6 days	10^{14} years	$\cdots$
100,000	100 ms	16.6 min.	317 years	10^{82} years	$\cdots$

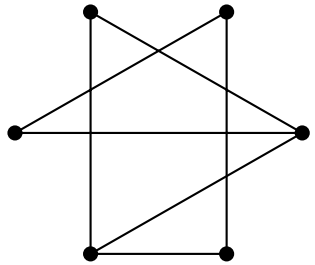
Timing: One "step" per microsecond (μs).

"μs = microsecond; "ms" = millisecond.

Age of Universe $\approx 1.3 \times 10^{10}$ years $\approx 4 \times 10^{17}$ μs.

Hamiltonian Cycle Problem

A *graph* consists of nodes some of which are connected by edges. A *Hamiltonian cycle* is a path that visits each node exactly once, returning to the starting node.



Hamiltonian Cycle Problem: Given an input x describing a graph, decide if there is a Hamiltonian cycle in the graph.

It is easy to verify any solution to the Hamiltonian cycle problem in polynomial time. (Feasible).

The best known algorithm to discover a Hamiltonian cycle (or even to decide if there exists one) require exponential time! (Infeasible)

Exhaustive search takes requires $n! > 2^n$ trials.

NP — Nondeterministic Polynomial Time

Recall P is the set of algorithmic problems that can be answered in "polynomial-time".

The class NP, "*non-deterministic polynomial time*" contains those problems which can be verified in polynomial time.

More precisely, NP is the class of problems which require Yes/No answers and for which, if the answer is "Yes" it can be verified in polynomial time given appropriate information.

Examples:

- Is integer x composite?
- Does graph G have a Hamiltonian cycle?

P =? NP

Open Question: Does $P = NP$? That is, if a problem has feasibly verifiable Yes answers, must the problem be feasibly solvable? (Taking "feasible" as "polynomial time".)

Loosely speaking: Is discovery as easy as verification?

More examples of problems in NP:

- Travelling Salesman Problem.
- Graph Colorability with 3 colors.
- Bin Packing.
- Satisfiability.
- many, many more.

NP-completeness All the above problems† are "NP-complete". If any of them are in polynomial time, then every NP problem is in polynomial time.

For example, if there is a polynomial-time algorithm for Hamiltonian cycle, then there is a polynomial-time algorithm for factoring integers.

† The compositeness checking problem is not (known to be) NP-complete.

Cryptography Based on RSA

Setting: Two people, Alice and Bob, wish to communicate with each other. Alice wants to let Bob and anyone else send her messages that only Alice decode. What this will mean is that any malicious eavesdropper would have to solve an infeasible problem.

As a side effect, Alice will also be able send messages to Bob or anyone else in such a way that they can be sure the message came from Alice. This means that any would-be forger would need to solve an infeasible problem.

Alice's public and private keys

Alice chooses a large integer N which is a product of two large primes p and q, $N = pq$.

Alice also finds two numbers d and e such that

$$e \cdot d \equiv (p-1)(q-1) \pmod{N}.$$

Alice publicizes: N and d. "The public key."

Alice keeps secret: p, q and e. "The private key."

Bob can send a secret message to Alice

Suppose Bob wants to send a private message to Alice. Bob chooses his message to be some integer $x \in \{0, \ldots, N-1\}$, that is, some value $x \bmod N$.

Bob computes $y = x^d \bmod N$. (Feasible!) and sends y to Alice over an insecure communication channel.

Alice computes $y^e \bmod N$. It turns out that this is always equal to x! Alice is the only one who can do this since e is secret, and finding e is as hard as factoring N.

x is the plaintext (the message). y is the encrypted message.

Sending Certified Messages

By reversing the process, Alice can send a message to Bob (or anyone else) so that Bob can be sure that only Alice could have encrypted it:

1. Bailey, Borwein, Plouffe D. Bailey,P. Borwein, S. Plouffe, "On the Rapid Computation of Various Polylogarithmic Constants." Math. Comput. 66, 903-913, 1997. (Also available on the web.)

2. M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H.Freeman, 1979.

3. Bruce Schneier, Applied Cryptography, 2nd edition, John Wiley & Sons, 1995.