Computationally Hard Problems

Design of Randomized Algorithms: Fingerprinting

Carsten Witt

Institut for Matematik og Computer Science
Danmarks Tekniske Universitet

Fall 2016

Overview

Today:

- Two problems that are not hard in the classical sense (aka. $\mathcal{NP}$-hard)
- but where it is harder to design deterministic than randomized algorithms
- or the best deterministic algorithm is even outperformed by a randomized one.

Fingerprinting

- Fingerprinting is used to approximately compare objects.
- Let A and B two **large** objects (strings, files, images, ...).
- Compute small "fingerprints" $\mathrm{fp}(A)$ and $\mathrm{fp}(B)$ and compare them.
- The fingerprint should fulfill the condition

$$A = B \implies \mathrm{fp}(A) = \mathrm{fp}(B)$$

- ... but not the other way around. It can happen that

$$[\mathrm{fp}(A) = \mathrm{fp}(B)] \wedge [A \neq B]$$

Fingerprinting

$$[\mathrm{fp}(A) = \mathrm{fp}(B)] \wedge [A \neq B] \tag{1}$$

The fingerprinting function fp should be chosen such that the situation in (1) rarely occurs.

We describe a randomized fingerprinting function and analyze how likely it is that the situation (1) occurs.

This method can also be applied when an adversary would like to make the fingerprints $\mathrm{fp}(A)$, $\mathrm{fp}(B)$ of different objects A, B identical.

Fingerprinting

We consider the problem at bit-level.

Problem [EqualStrings]
Input: Two strings $\boldsymbol{x} = x_1 x_2 \cdots x_n$ and $\boldsymbol{y} = y_1 y_2 \cdots y_n$ of equal length over the same alphabet $\Sigma = \{0, 1\}$.
Output for the decision version: YES if the strings are equal $(\boldsymbol{y} = \boldsymbol{x})$ and NO otherwise.

A bit string $\boldsymbol{x} = x_1 x_2 \cdots x_n$ can be interpreted as a natural number X in binary representation:

$$X = x_1 + 2x_2 + 4x_3 + \cdots + 2^{n-1} x_n = \sum_{i=1}^{n} x_i 2^{i-1} .$$

Choice of Fingerprinting Function

$$\mathrm{fp}_q(\boldsymbol{x}) = X \bmod q$$

where

$$X = \sum_{i=1}^{n} x_i 2^{i-1} \ .$$

and q is a prime number.

Implementation note: when computing X, apply modulo-operation after every arithmetic operation to avoid large numbers.

Facts

- We compare $\mathrm{fp}_q(\boldsymbol{x})$ and $\mathrm{fp}_q(\boldsymbol{y})$
- Only $\log_2(q)$ bits are compared (and have to be transmitted) when using fp_q.
- if $\mathrm{fp}_q(\boldsymbol{x}) \neq \mathrm{fp}_q(\boldsymbol{y})$ then $\boldsymbol{x} \neq \boldsymbol{y}$.
- Can happen that $\boldsymbol{x} \neq \boldsymbol{y}$ and $\mathrm{fp}_q(\boldsymbol{x}) = \mathrm{fp}_q(\boldsymbol{y})$.
- Hence we have a one-sided error.
- If $\boldsymbol{x}$ or $\boldsymbol{y}$ can be chosen by an adversary who knows q then he can force an error.

$\mathrm{fp}_q(\boldsymbol{x})$ is also called a *check sum function*.

Controlling the Error Probability

An error occurs if

$$X \bmod q = Y \bmod q \quad \text{and} \quad X \neq Y \ .$$

equivalently:

$$q \text{ divides } |X - Y| \text{ and } |X - Y| \neq 0$$

How many primes q with this property exist?

Fact: Let $R \leq 2^n$. Then the number of primes q that divide R is at most n.

Fact: Let $t \in \mathbb{N}$. Then the number $\pi(t)$ of prime numbers less than t is **asymptotically** $t/\ln(t)$.

Controlling the Error Probability

Let $R \geq |X|, |Y|$, then $R \geq |X - Y|$.

Choose $R := 2^n$ and choose t "slightly larger" than n.

There are $\pi(t) \sim t/\ln(t)$ primes $< t$

At most n of these primes can divide R (hence at most n of these can divide $|X - Y|$).

Thus the chance to pick such a "bad" prime is

$$\boldsymbol{P}[\,\text{fail}\,] \leq \frac{n}{\pi(t)} \sim \frac{n \ln(t)}{t}$$

Controlling the Error Probability

Choose $t = Cn\ln(Cn)$ for some (large) C.

Then $\boldsymbol{P}\left[\,\text{fail}\,\right] = \frac{n}{\pi(t)} = O\left(\frac{n\ln(Cn\ln(Cn))}{Cn\ln(Cn)}\right) = O\left(\frac{1}{C}\right)$

Summary: If q is chosen uniformly at random from the primes in $[2, Cn\ln(Cn)]$ for some constant C then

$$\boldsymbol{P}\left[\,\mathrm{fp}_q(\boldsymbol{x}) = \mathrm{fp}_q(\boldsymbol{y}) \mid X \neq Y\,\right] = O\left(\frac{1}{C}\right) .$$

The algorithm is a Monte Carlo algorithm with success probability $1 - O(1/C)$.

Summary of Fingerprinting

- The designer chooses the error probability $1/C$.
- This determines t.
- Choose a prime less than t and use it for fingerprinting.
- With probability $1 - O(1/C)$ this will give the right result.

A More General Application of Fingerprinting

Problem [PatternMatching]
Input: Two strings $\boldsymbol{x} = x_1 x_2 \cdots x_n$ and $\boldsymbol{y} = y_1 y_2 \cdots y_m$ over the same alphabet Σ with $m \leq n$.
Output for the decision version: YES if $\boldsymbol{y}$ is a substring of $\boldsymbol{x}$, and NO otherwise.
Output for the optimizing version: A position in $\boldsymbol{x}$ (if any) such that $\boldsymbol{y}$ is a substring of $\boldsymbol{x}$ from that position.

Facts:

- Naive algorithm solves this in $O(nm)$ time.
- Complicated deterministic algorithms (e. g. by Knuth/Morris/Pratt) with time $O(n+m)$ available.
- Fingerprinting will yield an easy Las-Vegas algorithm.

Outline of the Algorithm

Let $\boldsymbol{x}_j = x_j x_{j+1} \cdots x_{j+m-1}$ be the substring of length m of $\boldsymbol{x}$ starting at position j.

The aim is to compare $\boldsymbol{y}$ and $\boldsymbol{x}_j$ for possibly all $j = 1, 2, \ldots, n - m + 1$.

This is done randomized by comparing the fingerprints $\mathrm{fp}_q(\boldsymbol{y})$ and $\mathrm{fp}_q(\boldsymbol{x}_j)$ for an appropriate prime q.

If fingerprints match for some j, we have *probably* found a solution.

However, there might be false positives again. We have to turn down the error probability for all $n - m + 1$ possible applications of fingerprinting together.

Controlling the Error Probability

As above, we get for fixed position j:

$$\boldsymbol{P}\left[\,\mathrm{fp}_q(\boldsymbol{y}) = \mathrm{fp}_q(\boldsymbol{x}_j) \mid \boldsymbol{y} \neq \boldsymbol{x}_j\,\right] \leq \frac{m}{\pi(t)} = O\left(\frac{m\ln(t)}{t}\right) .$$

At most n applications $\Rightarrow$ total failure probability $O\left(\frac{nm\ln(t)}{t}\right)$ (Lemma A.5 in notes, "Union Bound").

Setting $t = n^2 m \ln(n^2 m)$ gives:

$$\boldsymbol{P}\left[\,\exists j \colon \mathrm{fp}_q(\boldsymbol{y}) = \mathrm{fp}_q(\boldsymbol{x}_j) \mid \boldsymbol{y} \neq \boldsymbol{x}_j\,\right] = O\left(\frac{1}{n}\right) .$$

This is a Monte-Carlo algorithm with success prob. $1 - O(1/n)$.

Runtime and Las Vegas

The above approach runs in deterministic time $O(n+m)$ if we compute $\mathrm{fp}_q(\boldsymbol{x}_{j+1})$ from $\mathrm{fp}_q(\boldsymbol{x}_j)$ in constant time.

This is possible using basic modulo arithmetic (exercise).

Las-Vegas algorithm

- Run the above Monte-Carlo algorithm.
- If match found at position j, compare $\boldsymbol{x}_j$ and $\boldsymbol{y}$ deterministically in time $O(m)$.
- If false positive revealed, run naive algorithm with time $O(nm)$.

Having prob. $O(1/n)$ for false positives, the expected running time is

$$O\left(\left(1-\frac{1}{n}\right)(n+m)+\left(\frac{1}{n}\right)nm\right)=O(n+m)\ .$$

Computationally Hard Problems

Design of Randomized Algorithms: Primality Testing

Carsten Witt

Institut for Matematik og Computer Science
Danmarks Tekniske Universitet

Fall 2016

Prime Number Tests

- 3 is prime.
- 12 is composite $(3 \cdot 4)$.
- 37 is prime.
- 347 is prime.
- 3447 is composite.
- $33455667687843519803434902543543012334732498223424 79$ is ???prime.

Motivation

- Definition: A natural number q is prime if it only is divisible by 1 and itself. 2 is the smallest prime number.
- Prime numbers are fundamental for public key cryptographic systems such as RSA.
- Security is based on the **assumption** that it is hard to factor a large composite number, especially a product of only two large primes.
- These numbers typically have ≥ 600 decimal digits (2048 bits).
- To check a 600-digit number N for primality naively means dividing by all primes less than $\sqrt{N}$.
- There are approx. $\sqrt{N}/\ln(\sqrt{N})$ such primes.
- For $N \sim 10^{600}$ this means $1.4 \cdot 10^{297}$ tests.

Solovay-Strassen Algorithm

- In 1977 Solovay and Strassen invented a probabilistic primality test.
- It is a Monte-Carlo algorithm.
- It has a fixed running time.
- If the number under consideration is prime, the algorithm will correctly detect this.
- If the number is composite the algorithm might incorrectly say "prime" with probability less than $1/2$.
- For 25 years, randomization was the only approach to a polynomial-time primality test. Only in 2002, Agrawal, Kayal and Saxena proved: PRIMES is in P.
- Randomized algorithms are still the most efficient primality testers.

Legendre Symbol

The algorithm uses two number-theoretic concepts.

- Let p be a prime number and let a be any natural number. The *Legendre symbol* $\left[\frac{a}{p}\right]$ is defined by

$$\left[\frac{a}{p}\right] := a^{\frac{p-1}{2}} \bmod p \tag{2}$$

- The value always is $1 \bmod p$, $-1 \bmod p$ or $0 \bmod p$.
- Examples

$$\begin{aligned}
\left[\frac{7}{17}\right] &= 7^8 \bmod 17 = 5764801 \bmod 17 = 16 \bmod 17 = -1 \\
\left[\frac{8}{17}\right] &= 8^8 \bmod 17 = 16777216 \bmod 17 = 1 \\
\left[\frac{34}{17}\right] &= 34^8 \bmod 17 = 1785793904896 \bmod 17 = 0
\end{aligned}$$

Legendre Symbol

The number-theoretical meaning of the Legendre $\left[\frac{a}{p}\right]$ symbol is as follows:

- It is 0 if p divides a.
- It is 1 if a is a quadratic residue modulo p, that is if there is a number x such that $a = x^2 \bmod p$.
- It is -1 in all other cases.

The Legendre symbol is only defined for primes p.

If p is not a prime, the formula $\left[\frac{a}{p}\right] := a^{\frac{p-1}{2}} \bmod p$ can be evaluated, but it might no longer give the number-theoretic meaning; in particular, the result might be different from -1, 0 and 1.

Jacobi Symbol

- Let n be an odd number with prime factorization

$$n = p_1^{k_1} \cdot p_2^{k_2} \cdots p_s^{k_s}.$$

Let a be a natural number. The *Jacobi symbol* $\left[\frac{a}{n}\right]_J$ is defined by the product of powers of the individual Legendre symbols (now with index L)

$$\left[\frac{a}{n}\right]_J := \prod_{j=1}^{s} \left[\frac{a}{p_j}\right]_L^{k_j}$$

- Here is an example

$$\left[\frac{72}{19845}\right]_J = \left[\frac{72}{3^4 \cdot 5 \cdot 7^2}\right]_J = \left[\frac{72}{3}\right]_L^4 \cdot \left[\frac{72}{5}\right]_L \cdot \left[\frac{72}{7}\right]_L^2 = (0) \cdot (-1) \cdot (1) = 0$$

Computing the Legendre Symbol

$$\left[\frac{a}{p}\right]_L := a^{\frac{p-1}{2}} \bmod p \tag{3}$$

Tricks

- Avoid large numbers by taking the modulus after every arithmetic operation.
- Use fast exponentiation (repeated squaring).

Computing the Jacobi Symbol

$$\left[\frac{a}{n}\right]_J := \prod_{j=1}^{s} \left[\frac{a}{p_j}\right]_L^{k_j} \tag{4}$$

Do we have to know the prime factors of n to compute $\left[\frac{a}{n}\right]_J$?

No: There is a set of rules to manipulate the expression and make the numbers smaller.

Computing the Jacobi Symbol

I-1 $\left[\frac{ab}{n}\right]_J = \left[\frac{a}{n}\right]_J \left[\frac{b}{n}\right]_J$

I-2 If $a \equiv b \bmod n$ then
$\left[\frac{a}{n}\right]_J = \left[\frac{b}{n}\right]_J$

I-3 If a and n are odd and $\gcd(n, a) = 1$ then
$\left[\frac{a}{n}\right]_J = (-1)^{\frac{a-1}{2}\frac{n-1}{2}} \left[\frac{n}{a}\right]_J$

I-4 $\left[\frac{1}{n}\right]_J = 1$

I-5 $\left[\frac{2}{n}\right]_J = \begin{cases} -1 & \text{if } n \equiv 3 \bmod 8 \text{ or } n \equiv 5 \bmod 8 \\ 1 & \text{if } n \equiv 1 \bmod 8 \text{ or } n \equiv 7 \bmod 8 \end{cases}$

Algorithm: always apply I-2 if possible. Otherwise use I-1 to factor out powers of 2 from "numerator", treat these with I-5 and apply I-3 to the rest. Continue until rest can be treated using I-4 or I-5.

Computing the Jacobi Symbol

Example

$$
\begin{aligned}
\left[\frac{191}{279}\right]_J &= (-1)\left[\frac{279}{191}\right]_J && \text{by I-3, since } \frac{278}{2}\frac{190}{2} \text{ is odd} \\
&= (-1)\left[\frac{88}{191}\right]_J && \text{by I-2} \\
&= (-1)\left[\frac{2}{191}\right]_J^3 \left[\frac{11}{191}\right]_J && \text{by I-1} \\
&= (-1)(+1)^3\left[\frac{11}{191}\right]_J && \text{by I-5 } (191 \equiv 7 \bmod 8) \\
&= (-1)^2\left[\frac{191}{11}\right]_J && \text{by I-3 } \left(\frac{10}{2}\frac{190}{2} \text{ odd}\right) \\
&= \left[\frac{4}{11}\right]_J && \text{by I-2} \\
&= \left[\frac{2}{11}\right]_J^2 && \text{by I-1} \\
&= (-1)^2 && \text{by I-5 } (11 \equiv 3 \bmod 8) \\
&= 1
\end{aligned}
$$

Computing the Greatest Common Divisor

$\gcd(a, b)$ can be computed using Euclid's algorithm.

Rules:

- $\gcd(a, b) = \gcd(a \bmod b, b)$.
- $\gcd(a, b) = \gcd(b, a)$
- $\gcd(a, 0) = a$.

```
Euclid(a, b) {
  while(b != 0) {
    interchange(a, b)
    b := b mod a
  }
return(a)
}
```

Solovay-Strassen Algorithm

To check whether n is prime:

- Choose $a \in \{1, 2, \ldots, n-1\}$ uniformly at random.
- If $\gcd(n, a) \neq 1$ then output "n is composite" and stop.
- If $\gcd(n, a) = 1$ then
- compute $L_1 = \left[\frac{a}{n}\right]_L = a^{\frac{n-1}{2}} \bmod n$
- compute $L_2 = \left[\frac{a}{n}\right]_J$ using the rules.
- if $(L_1 = L_2)$
- then output "n is probably prime"
- else output "n is composite"

Note: Formally $a^{\frac{n-1}{2}} \bmod n$ can be computed even if n is not a prime.

Demo

Solovay-Strassen Algorithm

Why it works:

Theorem: Let n be a composite number. Let

$$U_n := \left\{a \mid \gcd(a, n) = 1 \wedge \left[\frac{a}{n}\right]_J = a^{\frac{n-1}{2}} \bmod n\right\}$$

be the set of numbers where using the "wrong" formula gives the correct result.

Then

$$|U_n| \leq \frac{1}{2}n$$

Informally: At least every second choice of a will disclose a non-prime.

Solovay-Strassen Algorithm

- An error probability of $1/2$ is way too large for a secure code.
- Run the algorithm k times; every time a new random test number a is chosen.
- If n is composite, this is detected with probability $1/2$ in each run.
- The probability for a composite n to survive k rounds is at most $(1/2)^k$.
- In 332 rounds the chance to let a composite number n pass the test is less than $(1/10)^{100}$.
- Only mathematicians believe that $(1/10)^{100} \neq 0$.

Solovay-Strassen, Time

Theorem: The Legendre symbol can be computed in time polynomial in the **length** of the binary (or decimal) representation of n and a.

The Jacobi symbol can be computed in time polynomial in the **length** of the binary representation of n and a.

The greatest common divisor can be computed in time polynomial in the **length** of the binary representation of n and a.

Altogether, the primality test runs in time polynomial in the **length** of the binary representation of n.