Network Optimization by Randomization

Summer Semester 2011
TU Berlin
In This Lecture

- A randomized algorithm more efficient than any deterministic algorithm
- Generating random numbers
- Some probability concepts will be used but a review shall be presented during the next lecture
A (Very) Efficient Randomized Algorithm

**Problem:** Consider two versions $x$ and $y$ of a database stored on two remote computers $C_x$ and $C_y$. Design an (efficient) communication protocol between the two computers to determine whether the two versions are identical. Assume that $x = x_1x_2 \ldots x_n$ and $y = y_1y_2 \ldots y_n$, with $x_i, y_i \in \{0, 1\}$.

- If $n$ is *small* then the solution is simple: just send $x$ to $C_y$ and check. Running time?

- What about if $n$ is *large*, e.g., $n = 10^{15}$ (petabit order)?

*Note:* Every deterministic protocol must exchange (at least) $n$ bits in the worst case.

- For every efficient compression scheme (i.e., exchanging less than $n$ bits) there exists a worst-case input for which the protocol would fail; recall the Pigeonhole Principle.

- Efficient schemes can work but under additional info (e.g., there exists only one “1” in $x$ and $y$)
A (Very) Efficient Randomized Algorithm (contd.)

**RandProtocol**($x, y$)

At $C_x$:
- choose *uniformly* a prime $p$ from $[2, n^2]$
- $s = N(x) \mod p$
- send $s$ and $p$ to $C_y$ in binary

At $C_y$:
- $q = N(y) \mod p$
- if $q \neq s$ then ‘$x \neq y$’
- else ‘$x = y$’

Communication cost:
$$2 \lceil \log_2 n^2 \rceil \leq 4 \lceil \log_2 n \rceil \quad (\approx 200 \text{ bits for } n = 10^{15}!)$$

Too good to be true? Recall from the first lecture that there is a price to pay, i.e., the randomized algorithm may fail with some (hopefully tolerable) probability.

E.g., $x = 01111, y = 10110, N(x) = 15, N(y) = 22, p = 7$
Note that: $N(x) \mod p = 15 \mod 7 = 1$
$N(y) \mod p = 22 \mod 7 = 1$

$\Rightarrow$ The protocol fails!
Analyzing the Probability of Failure

Let \( \text{PRIM}(n^2) = \{ p : p \text{ is prime, } p \leq n^2 \} = A \cup B \), where

\[
A = \{ \text{“bad” primes (for } (x, y) \text{)} \}
\]

\[
B = \{ \text{“good” primes} \}
\]

\( p \) is a bad prime if \( N(x) \mod p = N(y) \mod p \) and \( x \neq y \)

\[
\Rightarrow p \mid |N(x) - N(y)|
\]

\[
\Rightarrow p \mid p_1^{i_1}p_2^{i_2} \ldots p_k^{i_k}, \text{ where } p_1 < p_2 < \ldots < p_k \text{ are primes}
\]

\[
\Rightarrow A = \{p_1, p_2, \ldots, p_k\}
\]

If \( |A| \geq n \) then \( |N(x) - N(y)| \geq p_1p_2 \ldots p_n > 2^n \), contradiction.

Therefore \( |A| \leq n - 1 \)

\[
\Rightarrow P(\text{failure}) = \frac{|A|}{|\text{PRIM}(n^2)|} \leq \frac{n-1}{n^2} \leq \frac{2 \log n}{n} \quad (\approx 10^{-14} \text{ for } n = 10^{15})
\]

Here we used the Prime Number Theorem, i.e.,

- \( \lim_{n \to \infty} \frac{|\text{PRIM}(n)|}{n \log n} = 1 \)

- \( |\text{PRIM}(n)| > \frac{n}{\log n} \) for \( n > 67 \)

How to get even smaller values for \( P(\text{failure}) \)?
Analyzing the Main Choices of the Algorithm

Recall: “... choose uniformly a prime $p$ from $[2, n^2]$”

1. What about choosing a prime $p$ from $[2, n]$ or $[2, n^3]$ or $[2, n^{10}]$?

2. More interestingly but challenging: Why uniformly?

Let $\text{PRIM}(25) = \{2, 3, 5, 7, 11, 13, 17, 19, 23\}$ and assume a geometric choosing of a prime, i.e.,

$$P(\text{choice} = n_i) = \left(\frac{1}{2}\right)^i \text{ for } i = 1, \ldots, 8 \text{ and } P(\text{choice} = n_9) = 0.01 \text{ (for normalization)}$$

If $A = \{5, 13, 19\}$ then $P(\text{failure}) = \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^6 + \left(\frac{1}{2}\right)^8$

In general:

- If $A$ (the set of “bad” primes) has small values then $P(\text{failure})$ is large.
- If $A$ has large values then $P(\text{failure})$ is small.

Note: The uniform choosing guarantees small failure probabilities for all (including worst-case) inputs.
Generating Random Numbers

- The need for random numbers
  - Simulate network behavior (e.g., inter-arrival times between packets), Monte-Carlo simulation, cryptography

- Physical Methods
  - Tossing coins, playing roulette. But how “random” is a coin toss? (not too much!)
  - Using physical phenomenons whose unpredictability is related to the laws of Quantum Mechanics (e.g., radioactive decay, thermal noise, radio frequency noise; check www.random.org)

- Computational Methods
  - “Anyone who considers arithmetical methods of producing random numbers is, of course, in a state of sin.” (John Von Neumann, 1951)
  - Pseudo-random Number Generators: algorithms which generate sequences of “random” numbers
Perception of Randomness

Paradox of Randomness: Toss a fair coin for 23 times and consider the outcomes:

- 0000000000000000
- 01101010000010011110011
- 11011110011101011111011

Is any of the sequences more “random” than the rest? (they can all occur with the same probability, i.e., \( \frac{1}{2^{23}} \))

The perception of randomness is influenced by our intuition, which is related to the structure of our visual-cognitive apparatus.

Note: The second sequence is the binary representation of \( \sqrt{2} - 1! \) (...is our intuition wrong?)

The source of the paradox: unexpected non-random outcome from a random experiment.
Formalizing (Defining) Randomness

Is there a way to formalize randomness?

- Axiomatic Probability Theory deliberately avoids defining whether a sequence of outcomes is random or not.

- Formalisms evolved by combining notions foreign to probability theory: algorithms and computability.

Consider the Cantor set of infinite binary sequences

$$2^\omega = \left\{ x_1x_2x_3 \ldots : x_i \in \{0, 1\} \right\}$$

- An idealization (model) of finite sequences
- Easier to work with one (but large) object than with many (but finite) objects

A Naive Definition: A sequence \( x \in 2^\omega \) is random if it satisfies the Law of Large Numbers, i.e.,

$$\lim_{n \to \infty} \frac{|\{i \leq n : x_i = 1\}|}{n} = \frac{1}{2}$$

What about the sequence \( x = 0101010101010101 \ldots \)?
Randomness as *Frequency Stability*

*Definition (von Mises 1919):* A sequence \( x \in 2^\omega \) is random if it is a collective (German: *kollectiv*), i.e., if it has two properties

1. \( \lim_{n \to \infty} \frac{f_n}{n} = \frac{1}{2} \)

\[ f_n : = |\{i \leq n : x_i = 1\}| \]

2. Every subsequence \( x_{n_1}x_{n_2}x_{n_3} \ldots \) of \( x \) with \( n_1 < n_2 < n_3 < \ldots \), obtained from a *reasonable* selection rule from \( x \), satisfies property 1.

More formally, \( x_n \) is selected if \( \phi(x_1, x_2, \ldots, x_{n-1}) = 1 \) where \( \phi : \{0, 1\}^* \to 0, 1 \) is an admissible partial function.

For instance:

- Choose \( x_n \) if \( x_{n-1} = 0 \).
- Choose \( x_n \) if \( n \) is prime.

*Kamke’s (counter)argument:* If arbitrary selection rules are allowed then collectives don’t exist: Let the subsequences \( x_{n_1}x_{n_2}x_{n_3} \ldots \) with \( n_1 < n_2 < \ldots \). Then there exists a subsequence such that \( x_{n_k} = 1 \).
Randomness as *Frequency Stability* (contd.)

What is wrong with Kamke’s (counter)argument?

- It’s non-constructive; note that no selection rule is specified
- Nevertheless, the argument created the need for a rigorous formalization of selection rules.

*Theorem (Wald, 1937)* If the set of selection rules is countable then collectives do exist. Moreover, the set of collectives has the cardinality of the continuum.

The problem of the class of selection rules still open...

*Definition (Church, 1940)* The class of selection rules is the class of computable Turing functions; the corresponding collectives are called *Church-stochastic*.

Recall that a function \( f : \mathbb{N} \to \mathbb{N} \) is Turing computable if there exists a Turing machine \( TM \) such that

- there exists an encoding \( e : \{0, 1\}^* \to \mathbb{N} \)
- for every \( n \) in the domain of \( f \) there exists an encoding \( w \) with \( e(w) = n \), \( TM \) stops on \( w \), and \( TM(w) = f(n) \)
Randomness as *Frequency Stability* (contd.)

*Theorem (Ville 1939)* There exist collectives which are

1. Church-stochastic, and
2. Every prefix contains more 1’s than 0’s (albeit \( \lim_{n} \frac{f_n}{n} = \frac{1}{2} \))

This states that the Mises-Wald-Church definitions of randomness are unsatisfactory.

Further definition, e.g., Kolmogorov-Loveland stochasticity, by restricting Church’s definition to computable non-monotonic.

... and further arguments that the definition is unsatisfactory (Lambalgen-Shen 1989): same result as Ville but for Kolmogorov-Loveland collectives.
Randomness as Incompressibility. Algorithmic Complexity

The idea: a sequence is “irregular” if it cannot be described with less characters than the sequence itself.

**Definition (Kolmogorov-Chaitin)** Given an universal Turing machine $U$, the algorithmic complexity of a sequence $x$ is given by

$$K_U(x) = \min \{|w| : U(w) = x\}$$

Note that $K_U(x)$ is the size of the smallest input program $w$ that, when fed to $U$, outputs $x$ and stops.

Recall that an universal Turing machine can simulate the behavior of any other Turing machine.

**Theorem (Kolmogorov)** For every Turing machine $T$

$$K_U(x) \leq K_T(x) + c_{UT}$$

In other words $U$ “nearly” has the property of the shortest code amongst all Turing machines.
Randomness as Incompressibility (contd.)

Definition (Kolmogorov) A sequence \( x = x_1x_2 \ldots \in 2^\omega \) is incompressible (chaotic) if there exists a constant \( c \) such that

\[
K(x(n)) \geq n - c
\]

for all \( n \), where \( x(n) = x_1x_2 \ldots x_n \).

Unfortunately, no such sequences exist!

Theorem (Martin-Löf) If \( f : \mathbb{N} \rightarrow \mathbb{N} \) is a computable function satisfying

\[
\sum_{n=1}^{\infty} 2^{-f(n)} = \infty \quad \text{(e.g.,} f(n) = \log n) \]

then for any binary sequence \( x \in 2^\omega \) it is the case that

\[
K(x(n)) < n - f(n)
\]

for infinitely many values of \( n \).

Solution (Uchain): restriction to prefix-free Turing machines (i.e., if \( TM \) halts on \( p \) then \( TM \) does not halt on \( pq \))

Alternative randomness definition by Martin-Löf based on typical sequences; equivalent to Uchain’s incompressibility definition.
Pseudo-Random Number Generators

- **Goal:** Imitate sequences of random numbers as closely as possible

- **Desirable properties:** Fast, low complexity, reproducibility

- General procedure:
  1. Generate a long sequence according to the discrete Uniform\((a, b)\) distribution with \(a << b\)
  2. Transform to a continuous Uniform\((0, 1)\) distribution
  3. Transform to a desirable distribution (e.g., Normal, Geometric)
Middle-Square Method (von Neumann, 1949)

\[ x_0 = \text{seed}; \quad //4\text{-digit integer number} \]
\[ i = 0; \]
\[ \text{while}(1) \]
\[ \quad y = x_i^2; \quad //\text{if } y \text{ has less than 8 digits add leading } 0\text{'s} \]
\[ \quad x_{i+1} = \text{middle 4 digits of } x_i; \]
\[ \quad i = i + 1; \]
\[ \text{end} \]

Shortcomings:

1. Sensitive to the seed
   E.g., \( x_0 = 2345 \Rightarrow \{2345, 4990, 9001, 180, 324, 1049, 1004, 80, 64, 40, \ldots\} \)
   all following \( x_i \)'s less than 100 (degeneration effect)

2. May have very short period
   E.g., \( x_0 = 2100 \Rightarrow \{2100, 4100, 8100, 6100, 2100, 4100, \ldots\} \)

3. High serial correlation of \( x_{i+1} \) and \( x_i \)

\textit{Problem:} How to get from a generated sequence to Uniform(0, 1)?
Linear Congruential Generator (LCG) Method

\[ x_0 = \text{seed}; \]
\[ i = 0; \]
\[ \text{while}(1) \]
\[ x_{i+1} = (ax_i + c) \mod m; \]
\[ i = i + 1; \]
\[ \text{end} \]

Used in the runtime libraries of many compilers, but very sensitive to the choice of \((\text{seed}, a, c, m)\)

E.g., \((\text{seed} = 7, a = 7, c = 7, m = 10) \Rightarrow \{7, 6, 9, 0, 7, 6, \ldots\}\)

\((\text{seed} = 8, a = 2, c = 5, m = 13) \Rightarrow \{8, 8, 8, \ldots\}\) (period = 1)

A good choice for \(m\) is \(2^k\); division (generally a complex operation) reduces to shifting

Theorem (Greenberger, 1961): LCG produces sequence with maximal period (cycle) if

1. \((c, m) = 1\) \((c\text{ and } m\text{ are relatively prime})\)
2. \(a \equiv 1 \pmod{p}\) for each prime divisor \(p\) of \(m\)
3. \(a \equiv 1 \pmod{4}\) if \(m\) is a multiple of 4

Also, serial correlation is small enough for appropriate values of \(m\) and \(a\).
Simulating Discrete Uniform Distributions with a Fair Coin

Encode “head” by 0 and “tail” by 1, and let $x = x_1x_2\ldots$ a sequence of 0’s and 1’s. Assume that

$$P(x_i = 0) = P(x_i = 1) = \frac{1}{2}$$

- Uniform distribution over $n$-bit integers

  procedure Uniform(1, $2^n$)
  for $i = 1$ to $n$
    $x_i$ is outcome of $i^{th}$ coin toss;
  end
  output $x_1x_2\ldots x_n$;

- Uniform distribution over \{1, 2, \ldots, n\}

  procedure Uniform(1, $n$)
  if \(n == 2^k\) then call Uniform(1, $2^k$) and halt;
  \(k = \lceil \log_2 n \rceil\);
  while(1)
    $x = $ Uniform(1, $2^k$);
    if ($x \leq n$) output $x$ and halt;
  end
Correctness of Uniform(1,n)

Let \( x \) be a sample of Uniform(1, n). All we need to check is whether

\[
P(x = i \mid x \leq n) = \frac{1}{n} \text{ for all } i \in \{1, 2, \ldots, n\}
\]

Use the formula for conditional probability

\[
P(x = i \mid x \leq n) = \frac{P(x = i \text{ and } x \leq n)}{P(x \leq n)} = \frac{P(x = i)}{P(x \leq n)} = \frac{1/2^k}{n/2^k} = \frac{1}{n}
\]

Note that the probability of halting at each iteration step is

\[
p := P(x \leq n) = \frac{n}{2^k} > \frac{1}{2}, \text{ since } k - 1 < \log_2 n \leq k \Rightarrow 2^{k-1} < n \leq 2^k
\]

What about the expected number of coin tosses? Denote \( N \) the number of iterations till the procedure halts and observe that

\[
P(N = i) = p(1 - p)^{i-1} \text{ (i.e., Geometric distribution)}
\]

Then \( E[N] = \sum_{i \geq 1} iP(N = i) = \sum_{i \geq 1} ip(1 - p)^{i-1} = \frac{1}{p} \leq 2 \)

So the expected number of coin tosses is at most \( 2[\log_2 n] \).