Algorithmic Adventures

From Knowledge to Magic

Book by Juraj Hromkovič, ETH Zurich
Slides by Tom Verhoeff, TU Eindhoven

Quotation

Chance favours only those whose spirit has been prepared already, those unprepared cannot see the hand stretched out to them by fortune.

Louis Pasteur

Changing the Requirements to Make Them Tractable

For an **optimization problem** we might be satisfied with a good **approximation** of the optimum (within some acceptable factor)

For some NP-hard problems, there are good polynomial approximation algorithms, for others not

Alternatively, we might accept an **unreliable** answer (within some acceptable confidence interval)

Randomized algorithms can be quick and reliable, though not 100%

Randomness as Concept

- Unpredictable: not predictable by an algorithm (?)
- Nondeterministic: fundamentally undetermined/open
- Stochastic: following mathematical axioms of probability theory
- Chaotic: extremely sensitive to initial conditions
- Incompressible: without shorter algorithmic description

Democritos believed that *randomness is the unknown*,
Nature is fundamentally determined.

Epicures claimed that *randomness is objective*,
it is the proper nature of events.

Two Styles of Randomization in Algorithms

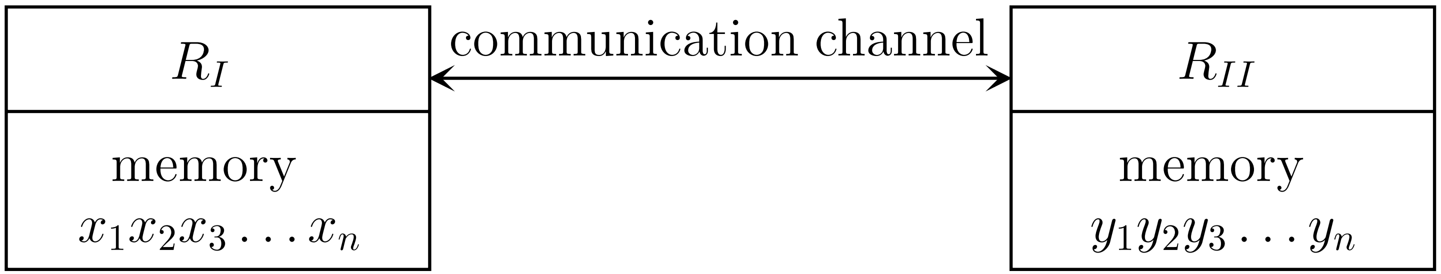
1. Algorithm may make random choices (flip a coin) at any moment
2. Algorithm randomly chooses a *deterministic* algorithm from a set

Random = according to some prescribed **probability distribution**

N.B. Probability distribution determines nature of randomness

E.g. Uniform distribution $\neq$ Normal distribution

Bit-String Equality Problem



Input: two n-bit strings in separate locations:
$x_1x_2x_3 \ldots x_n$ and $y_1y_2y_3 \ldots y_n$

Output: whether the strings are equal

Cost: communication between the two locations

Naïve approach: send n bits to other party and compare bitwise

1 TB: $n \approx 2^{43} \approx 10^{13}$ bits

Randomized Communication Protocol WITNESS

$$\mathbf{Number}(x) := \sum_{i=1}^{n} 2^{n-i} \cdot x_i$$

$$\mathbf{PRIM}(m) := \{\, p \text{ is a prime} \mid p \leq m \,\}$$

1. R_I chooses* random $p \in \mathsf{PRIM}(n^2)$
2. R_I computes† $s := \mathsf{Number}(x) \bmod p$

   R_I sends s and p to R_{II}
3. R_{II} computes† $q = \mathsf{Number}(y) \bmod p$

   R_{II} outputs "equal" if $q = s$, and else outputs "unequal"

*This is not so easy and needs special care

†This can be done in $\mathcal{O}(n)$ time

WITNESS: Communication Cost

- $0 \leq \text{Number}(x) < 2^n$
- $0 \leq p, s \leq n^2$
- Binary representation of p and s uses $\leq \left\lceil \log_2 n^2 \right\rceil \leq 2 \cdot \left\lceil \log_2 n \right\rceil$ bits
- **Total communication cost**: $4 \cdot \lceil \log_2 n \rceil$ bits
- Huge savings for large n: $4 \cdot \lceil \log_2 n \rceil \ll n$
- 1 TB: $n \approx 2^{43} \approx 10^{13} \Rightarrow$ communicate $4 \cdot 43 = 172$ bits

WITNESS: Reliability (Definitions)

When the protocol says "unequal", it is always correct:

$$\text{Number}(x) = \text{Number}(y) \;\Rightarrow\; \text{Number}(x) \bmod p = \text{Number}(y) \bmod p$$
$$s \neq q \;\Rightarrow\; \text{Number}(x) \neq \text{Number}(y)$$

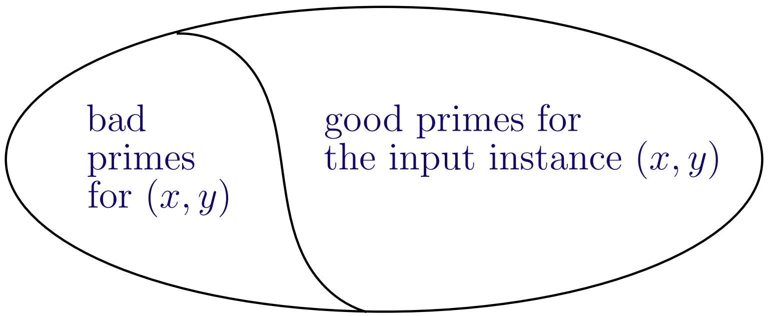
One-sided error possible: protocol could say "equal" erroneously

N.B. Operation '... mod p' throws away information

p is called **good/bad for** (x, y) when it gives right/wrong answer

$$\mathbf{Error}_{\text{WITNESS}}(x, y) \;:=\; \frac{\text{the number of bad primes for } (x, y)}{\text{Prim}(n^2)}$$

where $\mathbf{Prim}(m) := |\text{PRIM}(m)|$



WITNESS: Reliability Analysis

Prime Number Theorem: $\text{Prim}(m) \approx \dfrac{m}{\ln m}$

For $n \geq 9$: $\text{Prim}\left(n^2\right) > \dfrac{n^2}{\ln n^2} = \dfrac{n^2}{2 \ln n}$

Define: $\mathbf{Dif}(x, y) := \text{Number}(x) - \text{Number}(y)$

$$
\begin{aligned}
p \text{ is bad for } (x, y) \quad &\Leftrightarrow \quad x \neq y \text{ and } \text{Number}(x) \bmod p = \text{Number}(y) \bmod p \\
&\Leftrightarrow \quad x \neq y \text{ and } (\text{Number}(x) - \text{Number}(y)) \bmod p = 0 \\
&\Leftrightarrow \quad x \neq y \text{ and } p \text{ divides } \text{Dif}(x, y)
\end{aligned}
$$

Fundamental Theorem of Arithmetic: each positive integer has a unique prime factorization (apart from reordering factors)

$2^n > \text{Dif}(x, y) = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k} \geq 2^{e_1 + e_2 + \cdots + e_k} \geq 2^k$, hence $k < n$

$\text{Error}_{\text{WITNESS}}(x, y) < \dfrac{n}{n^2 / \ln n^2} \leq \dfrac{2 \ln n}{n}$

$n = 2^{43} \Rightarrow \text{Error} < 6.8 \times 10^{-12}$

Paradigms for Randomized Algorithms

- Foiling an adversary
- Random sampling
- Abundance of witnesses (cf. string equality)
- Fingerprinting and hashing (cf. string equality)
- Random re-ordering, load balancing
- ...

Derandomization: Eliminate randomness, preserve good properties

Las Vegas Algorithms versus Monte Carlo Algorithms

Las Vegas: Answer *always* correct; *probabilistic* runtime

Monte Carlo: Answer *probably* correct; *deterministic* runtime

Combination also possible

Improve Reliability by Repeated Execution

Independent repetitions: multiply error probability

Error probability decreases exponentially with number of repetitions

10 cycles of WITNESS for 1 TB :

- Cost $= 10 \cdot 172 = 1720$ bits communicated
- Error probability $< \left(\frac{2 \log 2^{43}}{2^{43}}\right)^{10} < 2.1 \times 10^{-112}$

Randomization in Sorting and Finding

Input: array of N elements, and an order relation

QuickSort sorts: running time expected $\mathcal{O}(N \log N)$, worst $\mathcal{O}\left(N^2\right)$

QuickFind finds median: running time expected $\mathcal{O}(N)$, worst $\mathcal{O}\left(N^2\right)$

Algorithm:

1. Pick a random *pivot* value P from the array
2. Partition the array into two parts: elements $\leq P$ and those $> P$
3. QuickSort: recursively apply to both parts
4. QuickFind: recursively apply to part known to contain the median

N.B. There exist deterministic $\mathcal{O}(N \log N)$ sorting and $\mathcal{O}(N)$ median algorithms

Randomized Volume Estimation and Counting

Packing puzzles can be solved recursively by **backtracking**

This gives rise to a search tree with all partial solutions

Estimate size of search tree:

1. Construct a random root path in the search tree
2. Assume that search tree is *uniform* with fan-outs as on this path
3. Calculate size of this uniform implied tree
4. Take average over multiple samples

Randomized On-line Scheduling Algorithm

See Chapter 10.3

Practical Problems with Randomization

How to analyse randomness, what distribution? **Statistical tests**

Human subjects are bad at creating/assessing randomness

Exploit **natural phenomena** (white noise, radioactive decay, . . .)
See: `random.org`

Need for **reproducibility**: *seeding*

Need for good **statistical properties**

Cryptographic protocols need unpredictability

N.B. Good statistical properties $\neq$ Unpredictability

It is notoriously hard to generate random events/numbers by software:
Pseudo Random Number Generator (PRNG)

Linear Congruential Generator (LCG):

$$X_{n+1} = (aX_n + c) \bmod m$$

for appropriate fixed integers a, c, m; X_0 is seed
LCG is *periodic*, and predictable after one sample (if a, c, m known)

Guideline: keep number of samples < *square root of the period*

Mersenne Twister: seeded, period $2^{19937} - 1 \approx 43 \times 10^{6000}$
Predictable after 624 samples

See: `en.wikipedia.org/wiki/Mersenne_twister`

Application of Randomization in Games

Three sources of uncertainty in game playing (can be mixed):

1. **Combinatorial**: full information, large number of combinations

   Monte Carlo methods for the board game Go: random game play

2. **Stochastic**: fortune, neutral interfering daemon

   Markov Decision Processes, deterministic optimal play

3. **Strategic**: hidden information, adversary with secrets

   Randomization guarantees unpredictability, prevents being exploited

Role of **variance**: N repetitions reduce **standard deviation** by $\frac{1}{\sqrt{N}}$

Summary

- Even exact algorithms are not 100% reliable when executed on real hardware, because hardware is inherently unreliable
- The longer the run time of a program, the higher the probability that something goes wrong, physically
- Sacrificing exactness, by using randomization, can lead to very efficient and still highly reliable algorithms
- Two techniques illustrated with bit-string equality protocol:
  1. Exploit an abundance of witnesses
  2. Repeat random computation to increase success probability