9 Random tricks

24 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version `2025-06-25 09:25`

Outline

9 Random tricks

Uses of Randomness

- Since it is likely that $\mathsf{BPP} = \mathsf{P}$, we focus on the more fine-grained benefits of randomization:
  - simpler algorithms (with same performance)
  - improving performance (but not jumping from exponential to polytime)
  - improved robustness

- Here: Collection of examples illustrating different techniques
  - fingerprinting / hashing
  - exploiting abundance of witnesses
  - random sampling

9.1 Hashing

Fingerprinting / Hashing

- Often have elements from huge universe $U = [0..u)$ of possible values, but only deal with few actual items $x_1, \ldots, x_n$ at one time.
  Think: $n \ll u$
- Fingerprinting can help to be more efficient in this case
  - fingerprints from $[0..m)$
  - $m \ll u$
  - *Hash Function* $h : U \to [0..m)$

Fingerprinting / Hashing

- Often have elements from huge universe $U = [0..u)$ of possible values, but only deal with few actual items $x_1, \ldots, x_n$ at one time.
  Think: $n \ll u$
- Fingerprinting can help to be more efficient in this case
  - fingerprints from $[0..m)$
  - $m \ll u$
  - *Hash Function* $h : U \to [0..m)$
- Classic Example: hash tables and Bloom filters

Uniform – Universal – Perfect

Randomness is essential for hashing to make any sense! Three very different

1. *uniform hashing assumption*: (optimistic, often roughly right in practice!)
   How good is hashing if input is "as nicely random" as possible?

Uniform – Universal – Perfect

Randomness is essential for hashing to make any sense! Three very different

1. *uniform hashing assumption*: (optimistic, often roughly right in practice!)
   How good is hashing if input is "as nicely random" as possible?

2. Since fixed h is prone to "algorithmic complexity attacks" (worst case inputs)
   $\rightsquigarrow$ *universal hashing*: pick h at random from class H of suitable functions

   universal class of hash functions

Uniform – Universal – Perfect

Randomness is essential for hashing to make any sense! Three very different

1. *uniform hashing assumption*: (optimistic, often roughly right in practice!)
   How good is hashing if input is "as nicely random" as possible?

2. Since fixed h is prone to "algorithmic complexity attacks" (worst case inputs)
   $\rightsquigarrow$ *universal hashing*: pick h at random from class H of suitable functions

   universal class of hash functions

3. For given keys, can construct collision-free hash function
   $\rightsquigarrow$ *perfect hashing*

Uniform Hashing – Balls into Bins

Theorem 9.1

If we throw n balls into n bins, then the *fullest bin* has $O(\log n / \log\log n)$ balls w.h.p. ◀

Universal Hashing – Efficient Randomized Hashing

Definition 9.2 (Universal Family)

Let $\mathcal{H}$ be a set of hash functions from U to R with $|R| = m$ and $|U| \geq m$. Assume $h \in \mathcal{H}$ is chosen uniformly at random.
Then $\mathcal{H}$ is called a *universal* if

$$\forall x_1, x_2 \in U \quad : \quad x_1 \neq x_2 \quad \rightarrow \quad \mathbb{P}\big[h(x_1) = h(x_2)\big] \;\leq\; \frac{1}{m}.$$

$\mathcal{H}$ is called *strongly universal* or *pairwise independent* if

$$\forall x_1, x_2 \in U, y_1, y_2 \in R \quad : \quad x_1 \neq x_2 \quad \rightarrow \quad \mathbb{P}\big[h(x_1) = y_1 \wedge h(x_2) = y_2\big] \;\leq\; \frac{1}{m^2}.$$

How good is universal hashing?

9.2 Perfect Hashing

Perfect Hashing: Random Sampling

9.3 Primality Testing

Abundance of Witnesses: Primality Testing

Theorem 9.3 (Fermat's Little Theorem)

For p a prime and $a \in [1..p-1]$ holds

$$a^{p-1} \equiv 1 \pmod{p}$$

Theorem 9.4 (Euler's Criterion)

Let $p > 2$ an odd number.

$$p \text{ prime} \iff \forall a \in \mathbb{Z}_p \setminus \{0\} : a^{\frac{p-1}{2}} \bmod p \in \{1, p-1\}$$

Theorem 9.5 (Number of Witnesses)

For every odd $n \in \mathbb{N}$, $(n-1)/2$ odd, we have:

1. If n is prime then $a^{(n-1)/2} \bmod n \in \{1, n-1\}$, for all $a \in \{1, \ldots, n-1\}$.
2. If n is not prime then $a^{(n-1)/2} \bmod n \notin \{1, n-1\}$ for *at least half* of the elements in $\{1, \ldots, n-1\}$.

Simple Solovay-Strassen Primality Test

Input: an odd number n with $(n-1)/2$ odd.

1. Choose a random $a \in \{1, 2, \ldots, n-1\}$.
2. Compute $A := a^{(n-1)/2} \bmod n$.
3. If $A \in \{1, n-1\}$ then output "n probably prime" (reject);
4. otherwise output "n not prime" (accept).

Theorem 9.6 (Correctness)

The simple Solovay-Strassen algorithm is a polynomial OSE-MC algorithm to detect composite numbers n with $n \bmod 4 = 3$. ◀

Corollary 9.7

For positive integers n with $n \bmod 4 = 3$ the simple Solovay-Strassen algorithm provides a polynomial TSE-MC algorithm to detect prime numbers. ◀

Sampling Primes

RANDOMPRIME(ℓ, k) Input: $\ell, k \in \mathbb{N}, \ell \geq 3$.

1. Set X :="not found yet"; $I := 0$;
2. `while` X = "not found yet" and $I < 2\ell^2$ `do`
   - generate random bit string $a_1, a_2, \ldots, a_{\ell-2}$ and
   - compute $n := 2^{\ell-1} + \sum_{i=1}^{\ell-2} a_i \cdot 2^i + 1$

     // This way n becomes a random, odd number of length ℓ
   - Realize k independent runs of Solovay-Strassen-algorithm on n;
   - `if` at least one output says "$n \notin PRIMES$" `then` $I := I + 1$

     `else` X :="PN found"; `output` n;
3. `if` $I = 2 \cdot \ell^2$ `then output` "no PN found".

Theorem 9.8 (Correctness of RandomPrime)

Algorithm RandomPrime(l, l) is a polynomial (in l) TSE-MC algorithm to generate random prime numbers of length l. ◀

9.4 Schöning's Satisfiability

$\rightsquigarrow$ Focus on practical benefits of randomization

Randomized approaches can be grouped into categories:

1. Coping with adversarial inputs
   Randomized Quicksort, randomized BSTs, Treaps, skip lists
2. Abundance of Witnesses
   Solovay-Strassen primality test
3. Fingerprinting
   universal hashing
4. Random Sampling
   Perfect hashing, Schöning's 3SAT algorithm, Karger's Min-Cut algorithm
5. LP Relaxation & Randomized Rounding
   Set-Cover Approximation (next chapter)

Warmup: A randomized 2SAT algorithm

```
procedure localSearch2SAT(ϕ, confidence):
    k = number of variables of ϕ
    Choose assignment α ∈ {0,1}^k uniformly at random.
    for j = 1, ..., confidence · 2k^2
        if α fulfills ϕ return "ϕ satisfiable"
        Arbitrarily choose a clause C = ℓ_1 ∨ ℓ_2 that is not satisfied under α.
        Choose ℓ from {ℓ_1, ℓ_2} uniformly at random.
        α = assignment obtained by negating ℓ.
    return "ϕ probably not satisfiable"
```

Theorem 9.9 (localSearch2SAT is OSE-MC for 2SAT)

Let ϕ be a 2SAT formula.

1. If ϕ is unsatisfiable, `localSearch2SAT` always returns "probably not satisfiable".
2. If ϕ is satisfiable, `localSearch2SAT` returns "satisfiable" with probability at least $1 - 2^{-\mathit{confidence}}$.

Schöning's Randomized 3SAT Algorithm

procedure Schöning3SAT(ϕ, *confidence*):
    k = number of variables in ϕ
    for $i = 1, \ldots, confidence \cdot 24 \left\lceil \sqrt{k} \left(\frac{4}{3}\right)^k \right\rceil$ **do**
        Choose assignment $\alpha \in \{0,1\}^k$ uniformly at random.
        for $j = 1, \ldots, 3k$ **do**
            if α fulfills ϕ **return** "ϕ satisfiable"
            Arbitrarily choose a clause $C = \ell_1 \vee \ell_2 \vee \ell_3$ that is not satisfied under α.
            Choose ℓ from $\{\ell_1, \ell_2, \ell_3\}$ uniformly at random.
            α = assignment obtained by negating ℓ.
    return "ϕ probably not satisfiable"

Theorem 9.10 (Schöning3SAT is OSE-MC for 2SAT)

Let ϕ be a 3SAT formula with n clauses over k variables.

1. If ϕ is unsatisfiable, `Schöning3SAT` always returns "probably not satisfiable".
2. If ϕ is satisfiable, `Schöning3SAT` returns "satisfiable" with probability $\geq 1 - 2^{-confidence}$.
3. `Schöning3SAT` runs in time $O\left(confidence \cdot k^{3/2} \left(\frac{4}{3}\right)^k n\right)$.

9.5 Karger's Cuts

Smart probability amplification: Karger's Min-Cut

Definition 9.11 (Min-Cut)

Given: A (multi)graph $G = (V, E, c)$, where $c : E \to \mathbb{N}$ is the multiplicity of an edge
Feasible Solutions: cuts of G, i. e., $M(G) = \{(V_1, V_2) \ : \ V_1 \cup V_2 = V \ \wedge \ V_1 \cap V_2 = \emptyset\}$,
Goal: Minimize
Costs: $\sum_{e \in C(V_1, V_2)} c(e)$, where $C(V_1, V_2) = \big\{\{u, v\} \in E \ : \ u \in V_1 \wedge v \in V_2\big\}$.

Random Contraction

```
procedure contractionMinCut(G = (V, E, c))
    Set label(v) := {v} for every vertex v ∈ V.
    while G has more than 2 vertices
        Choose random edge e = {x, y} ∈ E.
        G := Contract(G, e).
        Set label(z) := label(x) ∪ label(y) for z the vertex resulting from x and y.
    Let G = ({u, v}, E′, c′); return (label(u), label(v)) with cost c′({u, v}).
```

Theorem 9.12 (contractionMinCut correct with some probability)

`contractionMinCut` is a polytime randomized algorithm that finds a minimal cut for a given multigraph G with n vertices with probability $\geq 2/\big(n(n-1)\big)$. ◀

Lemma 9.13 (Threshold for contractionMinCut)

Let $l : \mathbb{N} \to \mathbb{N}$ a monotonic, increasing function with $1 \leq l(n) \leq n$. If we stop `contractionMinCut` whenever G only has $l(n)$ vertices and determine for the resulting graph G/F deterministically a minimal cut, then we need time in

$$O(n^2 + l(n)^3)$$

and we find a minimal cut for G with probability at least

$$\frac{\binom{l(n)}{2}}{\binom{n}{2}}$$

Karger's Min-Cut Improved

```
procedure KargerSteinMinCut(G(V, E, c))
    n = |V|
    if n ≥ 6
        compute minimal cut deterministically
    else
        h = ⌈1 + n/√2⌉
        G/F_1 = Contract random edges in G until h nodes left
        (C_1, cost_1) = KargerSteinMinCut(G/F_1)
        G/F_2 = Contract random edges in G until h nodes left
        (C_2, cost_2) = KargerSteinMinCut(G/F_2)
        if cost_1 < cost_2 return (C_1, cost_1) else C_2, cost_2)
```

Theorem 9.14 (`KargerSteinMinCut` beats deterministic min-cut)

`KargerSteinMinCut` runs in time $O(n^2 \cdot \log(n))$ and finds a minimal cut with probability $\Omega(\frac{1}{\log(n)})$.