9 Random Tricks

25 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version 2025-07-01 01:07 H

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 – Balls Into Bins

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 uses

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

Uniform Hashing Assumption:
When n elements $x_1, \ldots, x_n$ are inserted, for their *hash sequence* $h(x_1), \ldots, h(x_n)$, all m^n possible values are **equally likely**.

$\rightsquigarrow$ behavior of data structure completely **independent** of $x_1, \ldots, x_n$!

$\rightsquigarrow$ might as well forget data!

Balls into bins model (a.k.a. balanced allocations)

- throw n balls into m bins

⚠ Literature usually swaps n and m!

- each ball picks bin *i.i.d. uniformly* at random
- classic abstract model to study randomized algorithms
  - For hashing, effectively the best imaginable case tends to be a bit optimistic!
  - but: data in applications often not far from this

A Paradox?

- X_i: Number of balls in bin i:

$\rightsquigarrow$ $X_1 \overset{\mathcal{D}}{=} \cdots \overset{\mathcal{D}}{=} X_m \overset{\mathcal{D}}{=} \mathrm{Bin}(n, \frac{1}{m})$

$\rightsquigarrow$ All X_i concentrated around expectation $\frac{n}{m}$ (Chernoff!)

actually, just shows $X_i = n/m \pm n^{0.501}$

Consider $\boxed{m = n}$ $\rightsquigarrow$ $\mathbb{E}[X_i] = 1$

- But also: expected number of *empty* bins:

$$\begin{aligned}\mathbb{E}[\#i \text{ with } X_i = 0] &= \sum_{i=1}^{m} \mathbb{P}[X_i = 0] \\ &= m \cdot \left(1 - \frac{1}{m}\right)^n \qquad (m = n,\ (1 + 1/n)^n \approx e) \\ &= n \cdot e(1 \pm O(n^{-1}))\end{aligned}$$

$\rightsquigarrow$ In expectation, $\frac{1}{e}$ fraction (37%) of bins empty!
How does that fit together with $\mathbb{E}[X_i] = 1$? Which expectation should we expect?

Birthday Paradox

- Let's consider a different question to approach this . . .
- ***Birthday 'Paradox':***
  How many people does it take to likely have two people with the same birthday?
- In balls-into-bins language: What n makes it likely that $\exists j \in [m] : X_j \geq 2$?
  Compute counter-probability: $\mathbb{P}[\max X_j \leq 1]$

$$1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdots \left(1 - \frac{n-1}{m}\right) = e^{-\frac{1}{m}} \cdot e^{-\frac{2}{m}} \cdots e^{-\frac{n-1}{m}} \cdot \left(1 \pm O\left(\left(\frac{n}{m}\right)^2\right)\right)$$

$$= e^{-\frac{n^2}{2m} \pm O(\frac{n}{m})} \qquad (\tfrac{n}{m} \to 0)$$

Taylor series $e^x = 1 + x \pm O(x^2)$ as $x \to 0$

⇝ Only for $\boxed{n = \Theta(\sqrt{m})}$ nontrivial probability

- $\mathbb{P}[\max X_j \leq 1] = \frac{1}{2}$ for $n \approx \sqrt{2m \ln(2)}$, so for $m = 365$ days, need $n \approx 22.49$ people

⇝ *Can't expect to see **all** bins close to **expected** occupancy.*

Fullest Bin

Theorem 9.1

If we throw n balls into n bins, then w.h.p., the *fullest bin* has $O\left(\frac{\log n}{\log\log n}\right)$ balls.

Proof:

Fullest Bin [2]

Proof (cont.):

Fullest Bin – Consequences

- Closer analysis shows for $n = \alpha m$, constant α ("load factor"),

$$\max X_j \;=\; \frac{\ln n}{\ln(\ln(n)/\alpha)} \cdot \bigl(1 + o(1)\bigr) \quad \text{w.h.p.}$$

What can we learn from this?

1. Under *uniform hashing assumption*, even **worst case** of chaining hashing cost beats BST.
2. ... but not by much.
3. Expected costs aren't fully informative for hashing;
   (big difference between expected average case and expected worst case)

Biggest caveat: uniform hashing assumption!

$\rightsquigarrow$ ... we'll come back to that

- Cool trick: *Power of 2 choices*
  Assume ***two*** candidate bins per ball (hash functions), take less loaded bin

$\rightsquigarrow$ $\max X_j \;=\; \ln\ln n / \ln 2 \pm O(1)$ (!) analysis more technical; details in *Mitzenmacher & Upfal*

Coupon Collector

- Balls into bins nicely models other situations worth memorizing
- ***Coupon Collector Problem:***
  How many (wrapped) packs do I need to buy to get all collectibles?
- Balls-into-bins: What n makes it likely that $\forall j : X_j \ge 1$?
  - Define S_i as the number of balls to get from i empty bins to $i-1$ empty bins.

  $\rightsquigarrow$ $S = S_m + S_{m-1} + \cdots + S_1$ is the total number of balls for coupon collector
  - $S_i \stackrel{\mathcal{D}}{=} \mathrm{Geo}(p_i)$ where $p_i = \dfrac{i}{m}$ $\rightsquigarrow$ $\mathbb{E}[S_i] = \dfrac{1}{p_i} = \dfrac{m}{i}$
  - $\mathbb{E}[S] = \displaystyle\sum_{i=1}^{m} \mathbb{E}[S_i] = m \sum_{i=1}^{m} \frac{1}{i} = mH_m = m \ln m \pm O(m)$
- Can similarly show $\mathrm{Var}[S] = \Theta(m^2)$
  (since S_i are independent, stdev is linear + using $\mathrm{Var}[S_i] = \dfrac{1-p_i}{p_i^2}$)

  $\rightsquigarrow$ $\sigma[S] = \Theta(m) = o(\mathbb{E}[S])$, so S converges in probability to $\mathbb{E}[S]$ (Chebyshev)

9.2 Universal Hashing

Randomized Hashing

- Balls-into-bins model is worryingly optimistic.
  - Assumes that chosen bins $B_1, \ldots, B_n \in [m]$ are *mutually independent*.
  - ⇝ Assumes both that input is not adversarial **and** that hash functions work well.

⇝ To replace the assumption about the input by explicit randomization, would need a *fully random hash function* $h : [n] \to [m]$

- if we were to uniformly choose from m^n possibilities
  we'd need to store $\lg(m^n) = n \lg m$ bits just for h
- (even if we did so, how to efficiently *evaluate* h then is unclear)
- ↯ too expensive

⇝ Pick h at random, but from a smaller class $\mathcal{H}$ of "convenient" functions

Universal Hashing

What's a convenient class?

Definition 9.2 (Universal Family)

Let $\mathcal{H}$ be a set of hash functions from U to $[m]$ and $|U| \ge m$.
Assume $h \in \mathcal{H}$ is chosen uniformly at random.

(a) Then $\mathcal{H}$ is called a *universal* if

$$\forall x_1, x_2 \in U \quad : \quad x_1 \ne x_2 \implies \mathbb{P}\big[h(x_1) = h(x_2)\big] \le \frac{1}{m}.$$

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

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

◀

- strong universal implies universal
- In the following, always assume (uniformly) **random** $h \in \mathcal{H}$.
- by contrast, $x_1, \ldots, x_n$ may be chosen adversarially (but all distinct) from $[u]$

Examples of universal families

$$h_{ab}(x) = \big(a \cdot x + b \bmod p\big) \bmod m \qquad p \text{ prime}, p \geq m$$
$$h_a(x) = \big(a \cdot x \bmod 2^k\big) \operatorname{div} 2^{k-\ell} \qquad u = 2^k, m = 2^\ell$$

- $\mathcal{H}_1 = \{h_{ab} : a \in [1..p), b \in [0..p)\}$ is universal
- $\mathcal{H}_0 = \{h_{ab} : a \in [0..p), b \in [0..p)\}$ is strongly universal
- $\mathcal{H}_2 = \{h_a : a \in [1..2^k), a \text{ odd}\}$ is universal

How good is universal hashing?

9.3 Perfect Hashing

Perfect Hashing: Random Sampling

A hash function $h : [u] \to [m]$ is called

- *perfect* for a set $\mathcal{X} = \{x_1, \ldots, x_n\} \subset [u]$ if $i \neq j$ implies $h(x_i) \neq h(x_j)$
- *minimal* for set $\mathcal{X} = \{x_1, \ldots, x_n\} \subset [u]$ if $m = n$

Perfect Hashing

- only possible for $n \leq m$
- stringent requirement $\rightsquigarrow$ here focus on static $\mathcal{X}$
  - carefully chosen variants with partial rebuilding allow insertion and deletion in $O(1)$ amortized expected time
- further requirements
  1. Hash function must be fast to evaluate (ideally $O(1)$ time)
  2. Hash function must be small to store (ideally $O(n)$ space)
  3. should be fast to compute given $\mathcal{X}$ (ideally $O(n)$ time)
  4. Have small m (ideally $m = \Theta(n)$)

9.4 Primality Testing

Abundance of Witnesses

- Suppose $L \in \mathsf{NP}$ and all of the following are true:
  - alleged certificate must be easy to check
    trivially in polytime; often very fast
  - for $x \in L$, there are **many** certificates that show $x \in L$
    not generally true, but sometimes!

⇝ Conceivable that a randomized algorithm succeeds:
  - Guess a random certificate string
  - Check if it decides the problem

Primality Testing

Testing if a given number is *prime* is one of the oldest algorithmic questions.

Factorizing products of large prime numbers seems very hard
much of cryptography builds on this being intractable!

Complexity of Primality Testing and Factorization

- PRIMES:
  - **Given:** Integer n in binary encoding
  - **Goal:** Check if n is a prime number
- INTEGERFACTORIZATION:
  - **Given:** Integer n in binary encoding
  - **Goal:** Find nontrivial factors $n = m_1 \cdot m_2, 2 \le m_1, m_2 < n$ or determine "n prime"
- If n is composite, a factorization is a certificate for *non-primality* $\rightsquigarrow$ PRIMES $\in$ co-NP
- we will show PRIMES $\in$ co-RP $\subset$ BPP
- Major theoretical breakthrough: PRIMES $\in$ P *Agrawal, Kayal, and Saxena (2004)*
- This is not known for INTEGERFACTORIZATION!
  - However, Shor's algorithm can factor integers on a (theoretical) quantum computer

Does PRIMES have abundance of witnesses?

Primality Testing: Fermat's Little Theorem

Theorem 9.3 (Fermat's Little Theorem)

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

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

◄

Primality Testing: Second Attempt

Theorem 9.4 (Euler's Criterion)

Let $p > 2$ an odd number.

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

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 $a \in \{1, \dots, n-1\}$.
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 $\{1, \dots, n-1\}$. ◀

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. ◀

9.5 Schöning's Satisfiability

⇝ 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 = ℓ1 ∨ ℓ2 that is not satisfied under α.
        Choose ℓ from {ℓ1, ℓ2} 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 $1 - 2^{-confidence}$.

Schöning's Randomized 3SAT Algorithm

```
procedure Schöning3SAT(ϕ, confidence):
    k = number of variables in ϕ
    for i = 1, ..., confidence · 24⌈√k (4/3)^k⌉ do
        Choose assignment α ∈ {0,1}^k uniformly at random.
        for j = 1, ..., 3k do
            if α fulfills ϕ return "ϕ satisfiable"
            Arbitrarily choose a clause C = ℓ1 ∨ ℓ2 ∨ ℓ3 that is not satisfied under α.
            Choose ℓ from {ℓ1, ℓ2, ℓ3} 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 $O\left(confidence \cdot k^{3/2}\left(\frac{4}{3}\right)^k n\right)$.

9.6 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 $n \geq 6$
        compute minimal cut deterministically
    else
        $h = \lceil 1 + \frac{n}{\sqrt{2}} \rceil$
        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 $\Omega(\frac{1}{\log(n)})$.