9 Random Tricks

25 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg

version `2025-07-08 00:07 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: $\mathbb{P}[\max X_j \leq 1]$

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

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

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

$\rightsquigarrow$ 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

$\rightsquigarrow$ *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"),

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

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
  - $S_i \overset{\mathcal{D}}{=} \mathrm{Geo}(p_i)$ 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)
- ϟ 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 $|U| \geq m$.
Assume $h \in \mathcal{H}$ is chosen uniformly at random.

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

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

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

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

◀

- 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

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

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

How good is universal hashing?

Theorem 9.3

Assign $x_1, \ldots, x_n \in [u]$ to bins $h(x_i) \in [m]$ using hash function h, uniformly chosen from a universal family of hash functions $\mathcal{H}$.

Let X_j be the load of bin $j \in [m]$.

Then $\mathbb{P}\left[\max X_j \geq \sqrt{2} \cdot \frac{n}{\sqrt{m}}\right] \leq \frac{1}{2}$.

Proof:

How good is universal hashing [2]

Proof:

So, how good is universal hashing?

- For $n = m$, fullest bin $\leq \sqrt{2n}$
- Much worse than $\Theta(\log n / \log \log n)$!
- Note that we only proved an upper bound, however
  - bound is tight in the worst case
    (if all we know is pairwise independence of hash values)
    $\rightsquigarrow$ exercises
  - for practical choices like $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2$ better bounds are proven
    (close to $O(n^{1/3})$ instead of $O(n^{1/2})$)
    but still far worse than uniform 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 $n \le m$
- 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 $m = \Theta(n)$)

Perfect Hashing: Simple, but space inefficient

Perfect Hashing: Two-tier solution

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!

$\rightsquigarrow$ Conceivable that a randomized algorithm succeeds:

  - Guess a random certificate string
  - Check if it decides the problem

Primality Testing

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

Trivial approach: test for all (primes) $p \le \sqrt{n}$ whether $p \mid n$

```
procedure sieveOfEratosthenes(n):
    isPrime[2..n] := true
    for i := 2, 3, ..., ⌊√n⌋
        if isPrime[i]
            for j = i, i+1, i+2, ..., ⌊n/i⌋
                isPrime[i · j] := false
    return {p ∈ [2..n] : isPrime[p]}

procedure isPrimeTrivial(n):
    P := sieveOfEratosthenes(⌊√n⌋)
    return ∀p ∈ P : p ∤ n
```

Running time:

- dominated by sieving primes up to $m = \lfloor \sqrt{n} \rfloor$
- $T(m) \le m + \sum_{\substack{p \le m \\ p \text{ prime}}} \frac{m}{p} \le m + m \sum_{p=1}^{m} \frac{1}{p}$

$\rightsquigarrow T(m) = O(m \log m)$

- closer analysis: actually $T(m) = O(m \log \log m)$

Space: $\sqrt{n}$ bits

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
  - n encoded in binary $\rightsquigarrow$ Sieve of Eratosthenes is pseudopolynomial
- 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
  - Indeed much of classic cryptography (RSA) builds on factoring being intractable
  - *Shor's algorithm* can factor integers on a (theoretical) quantum computer in polytime! (not clear whether or when this is a practical concern)

Does PRIMES have abundance of witnesses?

Primality Testing: Fermat's Little Theorem

Theorem 9.4 (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 9.5 (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 9.6 (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 9.7 (Correctness)

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

Corollary 9.8

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".

Sampling Primes – Analysis

Theorem 9.9 (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

Random Sampling

If a solution is tricky to construct in a target fashion,
but many solutions are known to exist, random sampling can help.

Generate random object according to simple procedure until solution found.

We've seen ideas of random sampling in perfect hashing.

Now: Use more aggressive sampling to find rare objects.

Warmup: 2SAT

Famously, 3SAT is NP-complete.

2SAT: Given CNF formula φ with $\leq$ **2** literals per clause; is φ satisfiable?

By contrast, 2SAT $\in P$

Idea: Any clause $(\ell_1 \vee \ell_2)$ is equivalent to the *implications* $\neg\ell_1 \to \ell_2$ and $\neg\ell_2 \to \ell_1$

$\rightsquigarrow$ Represent formula as *implication graph*:

- vertices = literals in φ
- edges = all implications equivalent to some clause

$\rightsquigarrow$ Can show: φ satisfiable $\iff$ no SCC contains both x_i and $\neg x_i$

strongly connected component

- SCCs computable in linear time
- indeed, if no strong component contains contradiction, topological sort of components allows to read off satisfying assignment

$\rightsquigarrow$ Basically, a solved problem . . . we will use it for demonstration purposes only

Warmup: A randomized 2SAT algorithm

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

Theorem 9.10 (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 satisfying assignment with probability at least $1 - 2^{-confidence}$.
3. localSearch2SAT runs in $O(confidence \cdot k^2 n)$ time.

Randomized 2SAT – Analysis

Proof:

Claims 1. and 3. are trivial. It remains to prove Claim 2.

localSearch2SAT starts with random $\alpha = \alpha_0$.
In iteration t, flip one variable in α_t to obtain α_{t+1}.
We will analyze a *simplified* random process W that never behaves worse than localSearch2SAT
$\rightsquigarrow$ obtain an upper bound on error probability.

φ satisfiable $\rightsquigarrow$ $\exists\ \alpha^*$ that satisfies φ.
W will stop iff $\alpha = \alpha^*$ (localSearch2SAT might stop sooner)

We measure W's progress via $X_t = k - d_H(\alpha_t, \alpha^*)$.
While localSearch2SAT starts at a random α, we let W start at $\alpha_0 = \neg\alpha^*$. So $X_0 = 0$.

$C = \ell_1 \vee \ell_2$ not satisfied $\rightsquigarrow$ α^* and α_t differ in one or both
in either case, flipping random one gets closer to α^* with prob. $\geq \frac{1}{2}$

Assume W makes correct flip with prob $= \frac{1}{2}$.
$\rightsquigarrow$ $\mathbb{P}[X_{t+1} = X_t \mathbf{+1} \mid X_t] = \frac{1}{2}$ and $\mathbb{P}[X_{t+1} = X_t \mathbf{-1} \mid X_t] = \frac{1}{2}$
(except $X_t = 0$, then always +1 and $X_t = k$, then terminate)

$(X_t)_{t\geq 0}$ is thus a *Markov process*.

Randomized 2SAT – Analysis [2]

Proof (cont.):

Let now y_i be the expected number of steps to reach state k from $X = i$.

$$\begin{aligned} y_k &= 0 \\ y_0 &= 1 + y_1 \\ y_i &= 1 + p_i \cdot y_{i+1} + q_i \cdot y_{i-1} \qquad q_i = 1 - p_i \qquad \text{for us } p_i = \frac{1}{2} \end{aligned}$$

Can solve this recurrence for general p_i by writing for $i \in [1..k)$:

$$p_i y_i + q_i y_i \;=\; y_i \;=\; 1 + p_i y_{i+1} + q_i y_{i-1}$$

rearrange to $p_i(y_{i+1} - y_i) = q_i(y_i - y_{i-1}) - 1$. Now divide by p_i.

$\rightsquigarrow$ Recurrence of *differences*: $\quad \dot{y}_i = \dfrac{q_i}{p_i}\dot{y}_{i-1} - \dfrac{1}{p_i}$

Write $\dot{y} = y_{i+1} - y_i$ and abbreviate $a_i = q_i/p_i$ and $b_i = -1/p_i$:

$$\begin{aligned} \dot{y}_i &= a_i \dot{y}_{i-1} + b_i \qquad (1 \le i \le k-1) \\ \dot{y}_0 &= y_1 - y_0 = -1 \end{aligned}$$

Randomized 2SAT – Analysis [3]

Proof (cont.):

Recurrences 101: Telescoping recurrence! Can solve this in full generality:

$$\rightsquigarrow \quad \boxed{\dot{y}_i \;=\; \left(\prod_{j=1}^{i} a_j\right) \cdot \dot{y}_0 + \sum_{j=1}^{i} \left(\prod_{k=j+1}^{i} a_k\right) b_j}$$

Moreover: Telescoping sum $\displaystyle\sum_{j=0}^{i-1} \dot{y}_j \;=\; y_i - y_0 \;\rightsquigarrow\; y_0 \;=\; y_k - \sum_{j=0}^{k-1} \dot{y}_j$

We have $p = q = \frac{1}{2}$, so $a_i = 1$ and $b_i = -2 \;\rightsquigarrow\; \dot{y}_i = \dot{y}_0 + -2i = -2i - 1$

$$\rightsquigarrow \quad y_0 \;=\; \sum_{j=0}^{k-1} (2j+1) \;=\; k^2$$

$\rightsquigarrow$ W reaches α^* after $y_0 = k^2$ expected iterations

$\rightsquigarrow$ Expected #iterations for localSearch2SAT to reach α^* is $\le k^2$

$\mathbb{P}[\text{localSearch2SAT unsuccessful after } 2k^2 \text{ iterations}] \le \frac{1}{2}$ (Markov)

Treat *confidence* $\cdot\, 2k^2$ iterations as *confidence* repetitions of independent attempts of $2k^2$ each.
Probability that none successful $\le 2^{-\mathit{confidence}}$. ∎

From 2SAT to 3SAT

- Let's try the same on 3SAT. What changes?
- Key argument in 2SAT
  - fixing one clause had probability $\ge \frac{1}{2}$ to move closer to α^*
  - for 3SAT, this is only $\ge \frac{1}{3}$ (worst case: 2 out of 3 literals already correct)

$\rightsquigarrow$ same analysis gives expected iterations to reach y_0

$$y_0 = y_k - \sum_{j=0}^{k-1} \dot{y}_j \qquad \text{with} \qquad \dot{y}_i = \left(\prod_{j=1}^{i} a_j\right) \cdot \dot{y}_0 + \sum_{j=1}^{i} \left(\prod_{k=j+1}^{i} a_k\right) b_j$$

but with $p = \frac{1}{3}$, $q = \frac{2}{3}$, so $\boldsymbol{a_i = 2}$ and $b = -3$

$$\rightsquigarrow \dot{y}_i = 2^i \cdot (-1) + \sum_{j=1}^{i} 2^{i-j} \cdot (-3) = -2^i - 3 \sum_{k=0}^{i-1} 2^k = -2^i - 3(2^i - 1) = -4 \cdot 2^i + 3$$

$$\rightsquigarrow y_0 = -\sum_{j=0}^{k-1} (-4 \cdot 2^i + 3) \ge 4 \cdot (\mathbf{2^k} - 1)$$

$\rightsquigarrow$ Worse than deterministic brute force!

Local Search with Restarts

- Problem first attempt: Over time, more likely to move *away* from α^*
  - Need $\approx 2^k$ expected time to move k steps closer to α^*
  - Won't cut it for large k
- But we assume here that we start with $\neg\alpha^*$
  whereas actual random α might be (much) closer!

$\rightsquigarrow$ Keep local search for small improvements,
but restart overall method many times, to hopefully start close to α^* some time

Schöning's Randomized 3SAT Algorithm

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

Theorem 9.11 (Schöning3SAT is OSE-MC for 3SAT)

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

1. Schöning3SAT is a OSE-MC for 3SAT.
2. To be correct with probability $\geq 1 - 2^{\mathit{confidence}}$, it runs in time $O\left(\mathit{confidence} \cdot \left(\frac{4}{3}\right)^k k^{3/2} n\right)$

Schöning3SAT is OSE-MC for 3SAT

Proof:

2. follows immediately from standard OSE-MC probability amplification.

Also obvious: φ unsatisfiable $\rightsquigarrow$ Schöning3SAT returns `PROBABLY_NOT_SATISFIABLE`.

It remains to show: $\exists \alpha^*$ that satisfies $\varphi \rightsquigarrow \mathbb{P}[\text{Schöning3SAT returns } \alpha^*] \ge \frac{1}{2}$

one run of outer loop

Claim: $q \;:=\; \mathbb{P}[\text{local search finds } \alpha^*] \;\ge\; \dfrac{1}{12\sqrt{k}}\left(\dfrac{3}{4}\right)^k$

Proof:

$X = k - d_G(\alpha^*, \alpha)$ #variables correctly assigned in random α $\rightsquigarrow$ $X_0 \overset{\mathcal{D}}{=} \mathrm{Bin}(k, \frac{1}{2})$

Conditional on X, need local search to climb $u = k - X$ steps up to succeed.

We keep trying for $3k$ steps, but will only consider first $3u$ of them.

If at least $2u$ of these are up-steps, we succeed no matter which ones are up steps.

Pessimistically, assume up-step with prob $= \frac{1}{3}$.

$$q_u \;=\; \mathbb{P}[\ge 2u \text{ up in } 3u \text{ steps}] \;\ge\; \mathbb{P}[= 2u \text{ up in } 3u \text{ steps}] \;=\; \binom{3u}{u}\left(\frac{1}{3}\right)^{2u}\left(\frac{2}{3}\right)^{u} \;\ge\; \frac{c}{\sqrt{u}}2^{-u}$$

Stirling-Robbins Inequality : $n! = e^{r}_{n} \cdot \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ with $\frac{1}{12n+1} < r_n < \frac{1}{12n}$ $\rightsquigarrow$ $1 \le e^{r_n} \le 2$

$$\rightsquigarrow \quad \binom{3u}{u} \;=\; \frac{(3u)!}{u!(2u)!} \;\ge\; \frac{c}{\sqrt{u}} \cdot \frac{3^{3u}}{2^{2u}} \qquad \text{with} \qquad c \;=\; \frac{\sqrt{3}}{8\sqrt{2\pi}} \;\approx\; 0.086 \;>\; \frac{1}{12}$$

Schöning3SAT is OSE-MC for 3SAT [2]

Proof (Theorem 9.11 cont.):

Proof (Claim cont.):

$$q = \sum_{x=0}^{k} \mathbb{P}[X = x] \cdot q_{k-x} = \sum_{u=0}^{k} \mathbb{P}[X = k-u] \cdot q_u \geq \frac{1}{2^k} + \sum_{u=1}^{k} \binom{k}{k-u} \left(\frac{1}{2}\right)^k \cdot q_u$$

$$\geq \frac{1}{2^k} + \sum_{u=1}^{k} \binom{k}{u} \left(\frac{1}{2}\right)^k \frac{c}{\sqrt{u}} 2^{-u} \geq \frac{1}{2^k} + \frac{c}{\sqrt{k}} \left(\frac{1}{2}\right)^k \left[\underbrace{\sum_{u=\mathbf{0}}^{k} \binom{k}{u} \left(\frac{1}{2}\right)^u \mathbf{1^{k-u}}}_{A} - \binom{k}{0} \cdot 1 \right]$$

A is an instance of binomial theorem $\sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k} = (a+b)^n$

$$q \geq \frac{1}{2^k} + \frac{c}{\sqrt{k}} \left(\frac{1}{2}\right)^k \left[\left(\frac{1}{2} + 1\right)^k - 1 \right] \geq \frac{c}{\sqrt{k}} \left(\frac{3}{4}\right)^k \geq \frac{1}{12\sqrt{k}} \left(\frac{3}{4}\right)^k$$

Expected number of independent repetitions before success: $\frac{1}{q}$.

Schöning3SAT runs $2 \cdot \frac{1}{q} = 24\sqrt{k}\left(\frac{4}{3}\right)^k$ repetitions. $\rightsquigarrow$ Success prob $\geq \frac{1}{2}$.

9.6 Karger's Cuts

THIS SECTION WILL BE SKIPPED AND IS NOT PART OF THE EXAM MATERIAL.

Smart probability amplification: Karger's Min-Cut

Definition 9.12 (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., $M(G) = \{(V_1, V_2) \;:\; V_1 \cup V_2 = V \;\land\; V_1 \cap V_2 = \emptyset\}$,
Goal: Minimize
Costs: $\sum_{e \in C(V_1,V_2)} c(e)$, where $C(V_1, V_2) = \big\{\{u, v\} \in E \;:\; u \in V_1 \land v \in V_2\big\}$.

Random Contraction

procedure contractionMinCut$(G = (V, E, c))$
  Set $label(v) := \{v\}$ **for** every vertex $v \in V$.
  while G has more than 2 vertices
    Choose random edge $e = \{x, y\} \in E$.
    G := Contract(G, e).
    Set $label(z) := label(x) \cup label(y)$ **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 9.13 (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 9.14 (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 9.15 (`KargerSteinMinCut` beats deterministic min-cut)

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